Prioritätsbasierte Taskverteilung in selbstorganisierenden Systemen

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von

Eric Hutter

aus Gießen

Frankfurt 2022
(D30)

vom Fachbereich Informatik und Mathematik der
Johann Wolfgang Goethe-Universität als Dissertation angenommen.

Dekan:	Prof. Dr. Martin Möller
Gutachter:	Prof. Dr. Uwe Brinkschulte
	Prof. Dr. Lars Hedrich
	Prof. Dr. Roman Obermaisser
Datum der Disputation:	23.06.2022

Zusammenfassung

Die allgemein steigende Komplexität technischer Systeme macht sich auch in eingebetteten Systemen bemerkbar. Außerdem schrumpfen die Strukturgrößen der eingesetzten Komponenten, was wiederum die Auftrittswahrscheinlichkeit verschiedener Effekte erhöht, die zu Fehlern und Ausfällen dieser Komponenten und damit der Gesamtsysteme führen können. Da in vielen Anwendungsbereichen ferner Sicherheitsanforderungen eingehalten werden müssen, sind zur Gewährleistung der Zuverlässigkeit flexible Redundanzkonzepte nötig.

Ein Forschungsgebiet, das sich mit Methoden zur Beherrschung der Systemkomplexität befasst, ist das *Organic Computing*. In dessen Rahmen werden Konzepte erforscht, um Eigenschaften und Organisationsprinzipien, wie sie in natürlichen Systemen beobachtet werden können, auf technische Systeme zu übertragen. Hierbei sind insbesondere sogenannte *Selbst-X-Eigenschaften* wie *Selbstorganisation*, *Selbstkonfiguration* und *Selbstheilung* von Bedeutung.

Eine konkrete Ausprägung dieses Forschungszweigs ist das *künstliche Hormonsystem* (*artificial hormone system, AHS*). Hierbei handelt es sich um eine Middleware für verteilte Systeme, welche es ermöglicht, die *Tasks* des Systems selbstständig auf seine Rechenknoten (*Prozessorelemente, PEs*) zu verteilen und insbesondere Ausfälle einzelner Tasks oder ganzer PEs automatisch zu kompensieren, indem die betroffenen Tasks auf andere PEs migriert werden. Hierbei existiert keine zentrale Instanz, welche die Taskverteilung steuert und somit einen Single-Point-of-Failure darstellen könnte. Entsprechend kann das AHS aufgrund seiner automatischen (Re)konfiguration der Tasks als selbstkonfigurierend und selbstheilend bezeichnet werden, was insbesondere die Zuverlässigkeit des realisierten Systems erhöht. Die Dauer der Selbstkonfiguration und Selbstheilung unterliegt zudem harten Zeitschranken, was den Einsatz des AHS auch in Echtzeitsystemen erlaubt.

Das AHS nimmt jedoch an, dass alle Tasks – unabhängig von der von ihnen realisierten Funktionalität – gleichwertig sind, zudem werden alle Tasks beim Systemstart in einer zufälligen Reihenfolge auf die einzelnen PEs verteilt. Häufig sind die in einem System auszuführenden Tasks jedoch für das Gesamtsystem von unterschiedlicher Wichtigkeit oder müssen gar in einer bestimmten Reihenfolge gestartet werden.

Um den genannten Eigenschaften Rechnung zu tragen, liefert diese Dissertation gegenüber dem aktuellen Stand der Forschung folgende Beiträge:

- Zunächst werden die bisher bekannten Zeitschranken des AHS genauer betrachtet und verfeinert.

- Anschließend wird das AHS durch die Einführung von *Zuteilungsprioritäten* erweitert: Mithilfe dieser Prioritäten kann eine Reihenfolge definiert werden, in welcher die Tasks beim Start des Systems auf die PEs verteilt (und damit gestartet) beziehungsweise in welcher betroffene Tasks nach einem Ausfall auf andere PEs migriert werden.

- Die Zeitschranken dieser AHS-Erweiterung werden im Detail analysiert.
- Durch die Priorisierung von Tasks ist es möglich, implizit Teilmengen von Tasks zu definieren, die ausgeführt werden sollen, falls die Rechenkapazitäten des Systems nach einer bestimmten Anzahl von PE-Ausfällen nicht mehr ausreichen, um *alle* Tasks auszuführen: Das ursprüngliche AHS ohne Prioritätenunterstützung kann in solchen *Überlastsituationen* keinerlei Garantien geben, sondern führt vielmehr eine nicht vorhersagbare Teilmenge aller Tasks aus. Die im Rahmen dieser Dissertation entwickelten Erweiterungen erlauben es dagegen, das System in Überlastsituationen automatisch und kontrolliert zu degradieren, sodass eine (anhand der Prioritäten definierte) Teilmenge der wichtigsten Systemfunktionalitäten lauffähig bleibt.
- Überlastsituationen werden daher im Detail betrachtet und analysiert. In solchen müssen nach dem Ausfall von PEs gegebenenfalls Tasks niedriger Priorität gestoppt werden, um auf den funktionsfähig verbleibenden PEs hinreichend viel Rechenkapazität zu schaffen, um Tasks *höherer* Priorität ausführen zu können und das System so in einen wohldefinierten Zustand zu überführen. Die Entscheidung, in welcher Reihenfolge hierbei Tasks gestoppt werden, wird von einer *Task-Dropping-Strategie* getroffen, die entsprechend einen großen Einfluss auf die Dauer einer solchen Selbstheilung des Systems nimmt.
- Es werden zwei verschiedene Task-Dropping-Strategien entwickelt und im Detail analysiert: die *naive* Task-Dropping-Strategie, welche alle niedrigprioren Tasks auf einmal stoppt (weshalb unter Umständen *alle* aktuell im System vorhandenen Tasks gestoppt werden und anschließend die meisten dieser Tasks neu zugeteilt werden müssen), sowie das *Eager Task Dropping*, das inkrementell vorgeht und in mehreren Phasen jeweils höchstens eine Task pro PE stoppt. Im direkten Vergleich zeigt sich, dass vom Eager Task Dropping fast immer weniger Tasks gestoppt werden als von der naiven Strategie, was einen deutlich schnelleren Abschluss der Selbstheilung ermöglicht. Lediglich in wenigen Sonderfällen ist die naive Strategie dem Eager Task Dropping überlegen.
- Es wird detailliert gezeigt, dass die entwickelte AHS-Erweiterung auch in Überlastsituationen die Einhaltung bestimmter harter Zeitschranken garantieren kann, was den Einsatz des erweiterten AHS in Echtzeitsystemen erlaubt.
- Alle theoretisch hergeleiteten Zeitschranken werden durch umfassende Evaluationen vollumfänglich bestätigt.

Die Dissertation schließt mit einem Vergleich des erweiterten, prioritätsbasierten AHS mit verschiedenen verwandten Konzepten, um dessen Vorteile gegenüber dem Stand der Forschung herauszuarbeiten sowie zukünftige vertiefende Forschung zu motivieren.

Abstract

The generally rising complexity of technical systems is also noticeable in embedded systems. Additionally, the used components' feature size is shrinking, which in turn increases the probability of occurrence of various effects that can cause faults and failures of these components or the entire system. Since safety requirements must be met in many application areas, flexible redundancy concepts are necessary to ensure reliability.

Organic Computing is a research field dealing with methods to control system complexity. It focuses on concepts to transfer properties and organizational principles that are observable in natural systems to technical systems. In this context, so-called *self-x properties* such as *self-organization*, *self-configuration* and *self-healing* are of particular importance.

A particular manifestation of this branch of research is the *artificial hormone system* (*AHS*). The AHS is a middleware for distributed systems that allows to distribute a system's tasks autonomously to its computing nodes (*processor elements, PEs*) and especially to compensate failures of individual tasks or whole PEs automatically by migrating the affected tasks to other PEs. In this process, no central instance exists that controls the task distribution and would thus pose a single point of failure. Accordingly, the AHS can be described as self-configuring and self-healing due to its automatic task (re)configuration, increasing in particular the realized system's reliability. In addition, the time required for self-configuration and self-healing is limited by hard time bounds, allowing the use of the AHS in real-time systems.

However, the AHS assumes that all tasks are of the same importance for the system regardless of their respective functionality and distributes them in a random order to the individual PEs during system startup. Often, though, the tasks to be executed in a system are of varying importance to the overall system or are even required to be started in a particular sequence.

In order to accommodate the aforementioned properties, this dissertation improves upon the state of the art in the following ways:

- Initially, the AHS' time bounds known so far are examined more closely and refined.
- Afterwards, the AHS is extended by introducing *assignment priorities*: By utilizing these priorities, an order can be defined in which all tasks are distributed (and thus started) during the system's startup or affected tasks are migrated to other PEs after a failure, respectively.
- The AHS extension's time bounds are analyzed in detail.
- Prioritization of tasks allows to implicitly define task subsets that shall be executed if the system's computational capacities are no longer sufficient to execute *all* tasks after a certain number of PE failures occurred: The original

AHS without priority support cannot provide any guarantees in such *overload situations*, but rather executes an unpredictable subset of all tasks. In contrast, the extensions developed in the scope of this dissertation allow automatic and controlled system degradation in overload situations, keeping a subset of the most important system functionality defined by the tasks' priorities running.

- Consequentially, overload situations are examined and analyzed in detail. In such situations, tasks of low priority might need to be stopped after PE failures in order to free up sufficient capacities on the remaining PEs to execute tasks of *higher* priority, thus transitioning the system into a well-defined state. The decision in which order tasks are stopped in this process is made by a *task dropping strategy* that correspondingly has a great influence on the duration of such a self-healing of the system.

- Two different task dropping strategies are developed and analyzed in detail: the *naive* task dropping strategy that stops all tasks of low priority at once (which might require stopping all tasks currently running in the system and reassigning most of them afterwards), and the *eager task dropping* which operates incrementally in multiple phases by stopping no more than one task per PE and phase. A direct comparison shows that eager task dropping almost always stops fewer tasks than the naive strategy, allowing a faster completion of the self-healing. Only in a few special cases is the naive strategy superior to the eager task dropping.

- It is shown in detail that the developed AHS extension can also guarantee dedicated hard time bounds in overload situations, allowing the extended AHS' use in real-time systems.

- All theoretically derived time bounds are fully confirmed by comprehensive evaluations.

The dissertation concludes with a comparison of the extended, priority-based AHS with different related concepts in order to highlight its advantages over the state of the art and to motivate future research.

Inhaltsverzeichnis

Zusammenfassung i

Abstract iii

Abbildungsverzeichnis x

Verzeichnis von Sätzen, Definitionen und Beispielen xiii

Tabellenverzeichnis xv

Notationsverzeichnis xvi

Abkürzungsverzeichnis xviii

1. Einleitung 1
- 1.1. Motivation: Alltägliche Komplexität . . . 1
- 1.2. Aktuelle Trends . . . 3
- 1.3. Neue Herausforderungen . . . 4
- 1.4. Lösungsansätze . . . 6
- 1.5. Zielsetzung dieser Dissertation . . . 7
- 1.6. Aufbau dieser Dissertation . . . 7
- 1.7. Publikationen . . . 8

2. Mathematische Grundlagen 9
- 2.1. Natürliche und ganze Zahlen . . . 9
- 2.2. Iverson-Klammer . . . 9
  - 2.2.1. Anwendungen . . . 9
- 2.3. Auf- und Abrundungsfunktion . . . 10
- 2.4. Multimengen . . . 11

3. Organic Computing 15
- 3.1. Geschichte . . . 15
- 3.2. Grundgedanken . . . 15
- 3.3. Selbst-X-Eigenschaften . . . 16
- 3.4. Abgrenzung zum Autonomic Computing . . . 16
- 3.5. Zusammenfassung . . . 17

4. Das Künstliche Hormonsystem 19
- 4.1. Hintergrund . . . 19
- 4.2. Funktionsweise . . . 20
  - 4.2.1. Nomenklatur . . . 21
  - 4.2.2. Hormonbasierte Regelkreise . . . 21

4.2.3. Spezielle Hormontypen . . . 23
4.2.4. Offermechanismus . . . 24
4.2.5. Verhalten bei PE-Ausfällen . . . 26
4.3. Eigenschaften des AHS . . . 26
4.4. Verbesserungen des AHS . . . 27
4.4.1. Acceleratorschranken . . . 28
4.4.2. Leichtgewichtige Hormonauswertung . . . 29
4.4.3. Aggressive Task-Decision-Strategie . . . 32
4.5. Implementierungen des AHS . . . 37
4.5.1. Hormonsimulator . . . 37
4.5.2. AHS-Middleware . . . 37
4.5.3. Verwendung im Rahmen der Evaluationen . . . 39

5. Neue Zeitschranken für die aggressive Task-Decision-Strategie 41
5.1. Annahmen über das AHS . . . 41
5.2. Bisherige Zeitschranken . . . 42
5.2.1. Selbstkonfiguration . . . 42
5.2.2. Selbstoptimierung . . . 43
5.2.3. Selbstheilung . . . 43
5.3. Erkennung eines PE-Ausfalls . . . 44
5.4. Taskdiagramme und deren Semantik . . . 46
5.5. Widerlegung der bisherigen Zeitschranken . . . 50
5.5.1. Experimente . . . 50
5.5.2. Erklärung der Zeitschrankendegeneration . . . 55
5.6. Herleitung neuer Zeitschranken . . . 58
5.6.1. Verallgemeinerung . . . 60
5.7. Abschließende Bemerkungen . . . 62

6. Prioritätsbasierte Taskverteilung 65
6.1. Motivation . . . 65
6.2. Vorbereitungen . . . 65
6.3. Entscheidungsstrategie . . . 66
6.3.1. Umsetzung . . . 67
6.3.2. Behandlung von Überlastsituationen . . . 67
6.3.3. Implementierung des Hormonzyklus . . . 68
6.3.4. Erkennung von Überlastsituationen . . . 74
6.3.5. Besonderheiten . . . 76
6.4. Zeitschranken . . . 76
6.4.1. Selbstkonfiguration . . . 77
6.4.2. Selbstoptimierung . . . 81
6.4.3. Selbstheilung . . . 82
6.5. Zusammenfassung . . . 84

7. Evaluation der prioritätsbasierten Taskverteilung 87

7.1. Motivation: Prioritäten bei Verwendung der aggressiven Strategie . . . 88
7.1.1. Betrachtung der Zuteilungsreihenfolge . . . 89
7.1.2. Betrachtung der Dauer der Selbstkonfiguration . . . 91
7.2. Spezialfall: Alle Prioritäten verschieden ($|P| = m$) . . . 93
7.2.1. 20 Tasks verschiedener Prioritäten . . . 93
7.2.2. 64 Tasks verschiedener Prioritäten auf 16 PEs . . . 99
7.2.3. 256 Tasks verschiedener Prioritäten auf 64 PEs . . . 99
7.2.4. Diskussion der Szenarien mit $|P| = m$ . . . 101
7.3. Spezialfall: Nur ein Prioritätslevel ($|P| = 1$) . . . 101
7.3.1. 10 Tasks, 3 PEs . . . 102
7.3.2. 20 Tasks, 5 PEs . . . 102
7.3.3. 64 Tasks auf 16 PEs sowie 256 Tasks auf 64 PEs . . . 103
7.3.4. Diskussion der Szenarien mit $|P| = 1$ . . . 103
7.4. Weniger Prioritätslevel als Tasks ($1 < |P| < m$) . . . 104
7.4.1. $i = 4$ Tasks pro Prioritätslevel . . . 105
7.4.2. $i = 8$ Tasks pro Prioritätslevel . . . 109
7.4.3. $i = 16$ Tasks pro Prioritätslevel . . . 112
7.4.4. $i = 16$ Tasks pro Prioritätslevel ohne Acceleratoren . . . 114
7.4.5. Diskussion der Szenarien mit weniger Prioritäten als Tasks . . . 117
7.5. Zusammenfassung der Resultate . . . 119
7.6. Vergleich mit der AHS-Middleware . . . 119

8. Selbstheilung in Überlastsituationen **123**
8.1. Szenario . . . 123
8.2. Annahmen . . . 124
8.3. Erkennung von Überlastsituationen . . . 126
8.4. Allgemeine Lemmata . . . 128
8.5. Naives Task Dropping . . . 129
8.5.1. Worst-Case-Analyse . . . 130
8.5.2. Average-Case-Analyse . . . 131
8.6. Eager Task Dropping . . . 135
8.6.1. Definitionen . . . 141
8.6.2. Lemmata . . . 142
8.6.3. Worst-Case-Analyse . . . 145
8.6.4. Verallgemeinerung der Zeitschranke . . . 183
8.7. Vergleich der Task-Dropping-Strategien . . . 185
8.7.1. Vergleich der Anzahl gestoppter Tasks . . . 185
8.7.2. Vergleich der Selbstheilungsdauer . . . 188
8.8. Zusammenfassung . . . 192

9. Evaluation der Selbstheilung in Überlastsituationen **195**
9.1. Erkennung von Überlastsituationen . . . 195
9.2. Evaluation der naiven Task-Dropping-Strategie . . . 197

9.2.1. Evaluation der Zeitschranke gemäß Satz 8.11 (alle Prioritäten verschieden) . . . 198
9.2.2. Evaluation der verallgemeinerten Zeitschranke gemäß Satz 8.12 200
9.3. Evaluation des Eager Task Dropping . . . 202
9.3.1. Evaluation der Zeitschranke gemäß Satz 8.37 (alle Prioritäten verschieden) . . . 202
9.3.2. Evaluation der verallgemeinerten Zeitschranke gemäß Satz 8.51 205
9.4. Vergleich der Strategien . . . 208
9.4.1. Sonderfälle $m = 1$ sowie $v = 1$ . . . 208
9.4.2. Worst-Case-Szenario für beide Strategien . . . 209
9.4.3. Worst-Case-Szenario der naiven Strategie . . . 209
9.4.4. Mehrere PE-Ausfälle nacheinander . . . 211
9.5. Abschließende Bemerkungen . . . 216

10. Verwandte Konzepte und Arbeiten **217**
10.1. Offline-Methoden . . . 217
10.1.1. AutoKonf . . . 218
10.1.2. ScOSA . . . 219
10.2. Online-Methoden . . . 221
10.2.1. Multiagentensysteme . . . 221
10.2.2. Taskverteilung in Sensornetzen . . . 223
10.3. Abschließende Betrachtungen . . . 227

11. Zusammenfassung und Ausblick **229**

Anhang **233**

A. Evaluation der Selbstheilung in Unterlastsituationen **235**
A.1. Vergleich der Fälle $v = 1$ und $v > 1$ . . . 235
A.1.1. Erste Selbstheilung mit $v > 1$ . . . 236
A.1.2. Zweite Selbstheilung mit $v = 1$ . . . 238
A.1.3. Diskussion der Ergebnisse . . . 239
A.2. Selbstheilung mit weniger Prioritäten als ausgefallene Tasks . . . 239
A.3. Betrachtung der unteren Schranke der Erkennung eines PE-Ausfalls . 242
A.4. Vergleich mit der AHS-Middleware . . . 242
A.5. Zusammenfassung . . . 244

B. Average-Case-Analyse der naiven Task-Dropping-Strategie **247**
B.1. Mathematische Grundlagen . . . 247
B.1.1. Steigende und fallende Faktorielle . . . 247
B.1.2. Binomialkoeffizienten . . . 248
B.2. Annahmen . . . 250
B.3. Erwartete Anzahl Taskstopps . . . 251

B.4. Erwartete Dauer der Selbstheilung 254
B.4.1. Vergleich der Schranken 263
B.5. Vergleich mit dem Worst Case 266
B.5.1. Anzahl gestoppter Tasks 266
B.5.2. Dauer der Selbstheilung 269

C. Lemmata **275**

D. Alternativer Beweis von Satz B.15 **281**
D.1. Anzahl der Überschreitungen 281
D.2. Beta-Binomialverteilung 287
D.3. Anwendung auf ACET der naiven Task-Dropping-Stragegie 288
D.3.1. Direkte Herleitung über die Beta-Binomialverteilung 289
D.3.2. Herleitung über die Anzahl der Überschreitungen 290
D.4. Zusammenfassung 291

E. Übersicht über die wichtigsten Zeitschranken **293**
E.1. Selbstkonfiguration 293
E.2. Selbstoptimierung 293
E.3. Selbstheilung 293
E.3.1. In Unterlastsituationen 293
E.3.2. In Überlastsituationen 294

L. Literatur **297**

Abbildungsverzeichnis

1.1. Ladegerät „PowerPort Atom PD 2“ der Firma Anker 2

4.1. Middleware in einem verteilten System 19
4.2. Hormonbasierte Regelkreise 22
4.3. Schema eines Hormonzyklus 29
4.4. Synchronitätsproblem bei abweichender Hormonzykluszeit 30
4.5. Schema eines Hormonzyklus mit Selbstsynchronisation 31
4.6. Zeitverhalten der selbstsynchronisierenden Implementierung 31
4.7. Vereinfachtes Schema eines Hormonzyklus 32
4.8. Aggressive Task-Decision-Strategie 34
4.9. Hormonsimulator 38
4.10. Nachbarschaftsverhältnisse für ein 3×3-Gitter im Hormonsimulator . 38

5.1. Darstellung einer Taskübernahme zu diskreten Zeitpunkten 47
5.2. Darstellung eines Taskverlusts zu diskreten Zeitpunkten 48
5.3. Beispiel einer Selbstkonfiguration 49
5.4. Selbstkonfiguration einer einzigen, zehn mal zuteilbaren Task auf zwei identischen PEs, ohne Lastsuppressoren 51
5.5. Selbstkonfiguration einer einzigen, zehn mal zuteilbaren Task auf einem PE, ohne Lastsuppressoren 52
5.6. Selbstkonfiguration einer einzigen, zehn mal zuteilbaren Task auf zwei PEs, mit Lastsuppressoren 53
5.7. Selbstkonfiguration zehn einfach zuteilbarer Tasks auf drei PEs, mit Lastsuppressoren 54
5.8. Selbstkonfiguration einer einzigen, zehn mal zuteilbaren Task auf drei PEs, mit Lastsuppressoren 54

6.1. Prioritätsbasierte Task-Decision-Strategie 69
6.2. Unterlastsituation, fälschlicherweise als Überlastsituation erkannt . . . 75
6.3. Korrekte Erkennung einer Unterlastsituation 76

7.1. Überblick über die Evaluationen der Selbstkonfiguration 86
7.2. Evaluationsszenario 1: 10 Tasks verschiedener Prioritäten auf 3 PEs . 89
7.3. Selbstkonfiguration für Szenario 1 90
7.4. Tasks in Evaluationsszenario 2.1 93
7.5. Selbstkonfiguration für Szenario 2.1 auf zwei gleichartigen PEs 95
7.6. Selbstkonfiguration für Szenario 2.1 auf zwei unterschiedlichen PEs . . 96
7.7. Selbstkonfiguration für Szenario 2.1 auf einem einzigen PE 97
7.8. Konstellation aus fünf PEs 98
7.9. Selbstkonfiguration für Szenario 2.1 auf fünf PEs 98
7.10. Tasks in Evaluationsszenario 2.2 99
7.11. Selbstkonfiguration für Szenario 2.2 auf 16 PEs 100

7.12. Tasks in Evaluationsszenario 2.3 101
7.13. Selbstkonfiguration für $m = 10$ Tasks identischer Priorität auf 3 PEs . 103
7.14. Selbstkonfiguration für $m = 20$ Tasks identischer Priorität auf 5 PEs . 104
7.15. Abbildungsschema der Prioritäten auf $|P| = \lceil m/i \rceil$ Prioritätslevel . . . 105
7.16. Selbstkonfiguration für $m = 10$ und $i = 4$ auf 3 PEs 106
7.17. Selbstkonfiguration für $m = 20$ und $i = 4$ auf 5 PEs 107
7.18. Selbstkonfiguration für $m = 64$ und $i = 4$ auf 16 PEs 107
7.19. Selbstkonfiguration für $m = 20$ und $i = 8$ auf 5 PEs 111
7.20. Selbstkonfiguration für $m = 64$ und $i = 8$ auf 16 PEs 111
7.21. Selbstkonfiguration für $m = 256$ und $i = 8$ auf 64 PEs 113
7.22. Selbstkonfiguration für $m = 64$ und $i = 16$ auf 16 PEs 115
7.23. Selbstkonfiguration für $m = 256$ und $i = 16$ auf 64 PEs 116
7.24. Selbstkonfiguration ohne Acceleratoren für $m = 64$, $i = 16$ auf 16 PEs 117
7.25. Evaluationsszenario: Vergleich von Hormonsimulator und Middleware . 120
7.26. Selbstkonfiguration für das Szenario aus Abbildung 7.25 121

8.1. Für Überlastsituationen verwendetes Modell 125
8.2. Worst-Case-Szenario der naiven Task-Dropping-Strategie 132
8.3. Beispiel für Selbstheilung bei Eager Task Dropping 137
8.4. Beispiel für Selbstheilung mit zwei Phasen 138
8.5. WCET der Eager-Task-Dropping-Strategie 154
8.6. WCET der Eager-Task-Dropping-Strategie für fixiertes m 155
8.7. WCET der Eager-Task-Dropping-Strategie für fixiertes v 156
8.8. Motivation der PE- und phasenorientierten Sichtweise auf die Selbstheilung in Überlastsituationen 158
8.9. PE- und phasenorientierte Sichtweise auf die Selbstheilung in Überlastsituationen 163
8.10. Falsche Resultate bei phasenorientierter Sichtweise für $v < m - 1$ gemäß Gleichung (8.5) 164
8.11. Korrekte Resultate bei phasenorientierter Sichtweise für $v < m - 1$ gemäß Gleichung (8.4) 165
8.12. WCET der Eager-Task-Dropping-Strategie für fixiertes m inklusive asymptotischer Schranken 181
8.13. WCET der Eager-Task-Dropping-Strategie für fixiertes v inklusive asymptotischer Schranken 182
8.14. Quotient der gestoppten Tasks für die naive Task-Dropping-Strategie sowie das Eager Task Dropping 187
8.15. Quotient der Selbstheilungsdauern im Worst Case 189
8.16. Quotient der Selbstheilungsdauern im Average- und Worst Case 191

9.1. Überblick: Evaluationen der Selbstheilung in Überlastsituationen . . . 194
9.2. Erkennung einer Überlastsituation 196
9.3. Selbstheilung in einer Überlastsituation für $m = 5$ und $v = 1$ bei Verwendung der naiven Task-Dropping-Strategie 198

9.4. Selbstheilung in einer Überlastsituation für $m = 5$ und $v = 3$ bei Verwendung der naiven Task-Dropping-Strategie 199
9.5. Selbstheilung in einer Überlastsituation für $m = 5$, $v = 3$ und $|P| = 5$ bei Verwendung der naiven Task-Dropping-Strategie 201
9.6. Selbstheilung in einer Überlastsituation für $m = 4$ und $v = 5$ bei Verwendung des Eager Task Dropping 203
9.7. Selbstheilung in einer Überlastsituation für $m = 3$ und $v = 2$ bei Verwendung des Eager Task Dropping 204
9.8. Selbstheilung in einer Überlastsituation für $m = 5$ und $v = 1$ bei Verwendung des Eager Task Dropping 205
9.9. Selbstheilung in einer Überlastsituation für $m = 4$, $v = 8$ und $|P| = 12$ bei Verwendung des Eager Task Dropping 206
9.10. Selbstheilung in einer Überlastsituation für $m = 4$, $v = 8$ und $|P| = 4$ bei Verwendung des Eager Task Dropping 207
9.11. Selbstheilung in einer Überlastsituation für $m = 1$ und $v = 6$ 209
9.12. Selbstheilung in einer Überlastsituation für $m = 8$ und $v = 4$ 210
9.13. Selbstheilung in einer Überlastsituation für $m = 4$ und $v = 4$ 212
9.14. Selbstheilungen in Überlastsituationen für $m = 5$ und $v = 6, \ldots, 1$ . . 214

A.1. Überblick: Evaluationen der Selbstheilung in Unterlastsituationen . . . 234
A.2. Zeitlicher Verlauf zweier Selbstheilungen mit $v = 2$ bzw. $v = 1$ verbleibenden PEs sowie $|P_f| = m_f$ 237
A.3. Zeitlicher Verlauf zweier Selbstheilungen mit $v = 7$ verbleibenden PEs sowie $|P_f| < m_f$ 240
A.4. Zeitlicher Verlauf der Selbstheilung nach einem PE-Ausfall zu verschiedenen Zeitpunkten 243
A.5. Zeitlicher Verlauf einer Selbstheilung in der AHS-Middleware 245

B.1. In $\text{ACET}_{\text{naiv}}^{\max} - \text{ACET}_{\text{naiv}}^{\min}$ auftretender Term $c_{m,v}$ 266
B.2. Quotient $Q(m)$ der gestoppten Tasks im Worst- und Average Case . . 267
B.3. Wahrscheinlichkeitsverteilung der Anzahl gestoppter Tasks 268
B.4. Quotient $Q(m, v)$ der Selbstheilungsdauer der naiven Task-Dropping-Strategie in Worst- und Average Case 270
B.5. Untere und obere Schranke des Quotienten $Q(m, v)$ für kleine Werte von v 271

Verzeichnis von Sätzen, Definitionen und Beispielen

2.1 Notation 9
2.2 Notation 9
2.3 Definition 9
2.4 Bemerkung 9
2.5 Definition 10
2.6 Definition 10
2.7 Bemerkung 10
2.8 Lemma 10
2.9 Definition 11
2.10 Bemerkung 11
2.11 Bemerkung 11
2.12 Definition 11
2.13 Definition 12
2.14 Definition 12
2.15 Definition 12
2.16 Definition 12
2.17 Definition 12
2.18 Definition 13
2.19 Konvention 13

5.1 Annahme 41
5.2 Annahme 41
5.3 Beispiel 42
5.4 Annahme 42
5.5 Satz 43
5.6 Satz 43
5.7 Definition 43
5.8 Definition 43
5.9 Satz 43
5.10 Definition 44
5.11 Satz 44
5.12 Beispiel 55
5.13 Notation 58
5.14 Definition 58
5.15 Beispiel 58
5.16 Beispiel 59
5.17 Beispiel 59
5.18 Satz 60
5.19 Korollar 61
5.20 Korollar 61
5.21 Satz 62
5.22 Satz 62

6.1 Definition 65
6.2 Definition 66
6.3 Definition 66
6.4 Strategie 66
6.5 Annahme 68
6.6 Satz 77
6.7 Satz 77
6.8 Bemerkung 78
6.9 Satz 79
6.10 Satz 79
6.11 Definition 80
6.12 Satz 81
6.13 Bemerkung 81
6.14 Satz 82
6.15 Satz 82
6.16 Bemerkung 83
6.17 Korollar 83
6.18 Definition 83
6.19 Satz 83

8.1 Definition 123
8.2 Definition 123
8.3 Bemerkung 124
8.4 Satz 126
8.5 Definition 128
8.6 Bemerkung 129
8.7 Lemma 129
8.8 Lemma 129
8.9 Korollar 129
8.10 Strategie 129
8.11 Satz 130
8.12 Satz 131
8.13 Strategie 135
8.14 Strategie 135
8.15 Definition 135

8.16 Satz . . . 136
8.17 Lemma . . . 140
8.18 Satz . . . 140
8.19 Bemerkung . . . 141
8.20 Definition . . . 141
8.21 Beispiel . . . 141
8.22 Definition . . . 141
8.23 Bemerkung . . . 142
8.24 Beispiel . . . 142
8.25 Definition . . . 142
8.26 Beispiel . . . 142
8.27 Lemma . . . 143
8.28 Lemma . . . 144
8.29 Bemerkung . . . 144
8.30 Lemma . . . 144
8.31 Beispiel . . . 145
8.32 Beispiel . . . 146
8.33 Satz . . . 146
8.34 Beispiel . . . 149
8.35 Bemerkung . . . 150
8.36 Bemerkung . . . 151
8.37 Satz . . . 152
8.38 Satz . . . 158
8.39 Lemma . . . 161
8.40 Satz . . . 161
8.41 Bemerkung . . . 162
8.42 Satz . . . 164
8.43 Beispiel . . . 165
8.44 Konvention . . . 167
8.45 Bemerkung . . . 171
8.46 Satz . . . 176
8.47 Bemerkung . . . 176
8.48 Satz . . . 177
8.49 Satz . . . 178
8.50 Bemerkung . . . 179
8.51 Satz . . . 183
8.52 Bemerkung . . . 184
8.53 Satz . . . 185

B.1 Definition . . . 247
B.2 Bemerkung . . . 247
B.3 Definition . . . 247
B.4 Bemerkung . . . 247
B.5 Bemerkung . . . 247
B.6 Definition . . . 248
B.7 Bemerkung . . . 248
B.8 Bemerkung . . . 248
B.9 Lemma . . . 248
B.10 Lemma . . . 248
B.11 Lemma . . . 249
B.12 Lemma . . . 250
B.13 Annahme . . . 250
B.14 Bemerkung . . . 250
B.15 Satz . . . 251
B.16 Satz . . . 255
B.17 Bemerkung . . . 257
B.18 Satz . . . 258
B.19 Korollar . . . 263
B.20 Satz . . . 263
B.21 Satz . . . 272

C.1 Lemma . . . 275
C.2 Lemma . . . 276
C.3 Lemma . . . 277
C.4 Lemma . . . 278

D.1 Satz . . . 281
D.2 Satz . . . 287
D.3 Definition . . . 287
D.4 Satz . . . 288
D.5 Satz . . . 288
D.6 Satz . . . 288
D.7 Beispiel . . . 290

Tabellenverzeichnis

1.1. Vergleich von USB-C-Ladegerät und Apollo Guidance Computer . . . 3

4.1. Untertypen von Suppressoren . . . 23
4.2. Untertypen von Acceleratoren . . . 24

5.1. Ergebnisse der Experimente: Selbstkonfiguration von m Taskinstanzen 55

7.1. Reihenfolge der Taskzuteilung in Evaluationsszenario 1 . . . 89
7.2. Zuteilungsdauer der Tasks aus Evaluationsszenario 1 . . . 92
7.3. Zuteilungsdauer der Prioritätslevel für $m = 64$ und $i = 4$ . . . 108
7.4. Zuteilungsdauer der Prioritätslevel für $m = 256$ und $i = 4$ . . . 110
7.5. Zuteilungsdauer der Prioritätslevel für $m = 64$ und $i = 8$ . . . 112
7.6. Zuteilungsdauer der Prioritätslevel für $m = 256$ und $i = 8$ . . . 114
7.7. Zuteilungsdauer der Prioritätslevel für $m = 256$ und $i = 16$ . . . 115

8.1. Sonderfälle mit $\mathrm{ACET}^{\mathrm{min}}_{\mathrm{naiv}}/\mathrm{WCET}_{\mathrm{eager}} < 1$ . . . 191

9.1. Dauern der Selbstheilungen aus Abbildung 9.14 . . . 213

A.1. Zuteilungsdauern der Tasks gemäß Abbildung A.2 . . . 238
A.2. Zuteilungsdauern der Prioritätslevel gemäß Abbildung A.3 . . . 241

Notationsverzeichnis

Die folgende Tabelle soll als Referenz für die wichtigsten im Rahmen dieser Dissertation verwendeten Notationen dienen. Die angegebenen Seitenzahlen beziehen sich dabei auf die Einführung bzw. Definition, sofern eine solche explizit gegeben wird.

Symbol	Bedeutung	Seite
$\mathbb{N}$	Natürliche Zahlen (inklusive Null)	9
$\mathbb{N}^+$	Positive natürliche Zahlen (ohne Null)	9
$[A]$	Iverson-Klammer (sofern A eine Aussage ist, die entweder wahr oder falsch sein kann)	9
$\lfloor x \rfloor$	Abrundungs- bzw. Floorfunktion	10
$\lceil x \rceil$	Aufrundungs- bzw. Ceilingfunktion	10
$\{\!\vert \cdot \vert\!\}$	Multimenge	11
$M \subseteqq N$	M ist induzierte Teilmultimenge von M	12
$\mathrm{supp}(M)$	Reduzierte Grundmenge (*support*) einer Multimenge M	12
$n^{\overline{k}}$	Steigende Faktorielle	247
$n^{\underline{k}}$	Fallende Faktorielle	247
T, T_i	Task	–
$\mathcal{T}$	Taskmenge oder -multimenge	58
$\mathcal{T}(T)$	Anzahl Zuteilungen von Task T	58
PE_i	PE mit Index i	21
$\mathrm{PE}_\times$	Ausgefallenes PE	43
$E_{i,j}$	Statischer Eignungswert für Task T_i auf PE_j (ggf. modifizierte Notation, z.B. E_T für Task T ohne Angabe eines spezifischen PEs)	21
Em	Modifizierter Eignungswert	
A	Accelerator. Mögliche Untertypen sind beispielsweise Ao Organaccelerator As Stay-Accelerator Am Monitoring-Accelerator	24
S	Suppressor. Mögliche Untertypen sind beispielsweise Sa Übernahmesuppressor (*acquisition suppressor*) Sl Lastsuppressor (*load suppressor*) Sm Monitoring-Suppressor	23

Symbol	Bedeutung	Seite
$\mathcal{H}$	Für $\mathcal{H} \in \{Em, A, S\}$: gesendetes oder empfangenes Hormon	21
$\mathcal{H}_{i,j}^{k,l}$	Hormon, gesendet von Task T_i auf PE_j an Task T_k auf PE_l	21
$\mathcal{H}_{i,j}$	Hormon, gesendet von Task T_i auf PE_j; Empfängertask und -PE nicht spezifiziert	21
$\mathcal{H}^{k,l}$	Hormon, empfangen von Task T_k auf PE_l; sendende Task und sendendes PE nicht spezifiziert	21
a	Zeit, die *höchstens* benötigt wird, um den Ausfall von $\text{PE}_\times$ zu bemerken	43
$a_{\min}$	Zeit, die *mindestens* benötigt wird, um den Ausfall von $\text{PE}_\times$ zu bemerken	44
m	Anzahl Tasks bzw. Taskinstanzen: • Gesamtzahl der im System vorhandenen oder zuzuteilenden Tasks bzw. Taskinstanzen • Anzahl Tasks pro PE bei einer Selbstheilung in einer Überlastsituation	–
m_f	Anzahl ausgefallener Tasks (bei einer Selbstheilung ohne Überlastsituation)	43
v	Anzahl verbleibender PEs bei einer Selbstheilung in einer Überlastsituation (nach dem Ausfall von $\text{PE}_\times$)	123
$\text{prio}(T)$	Priorität einer Task T	65
P	Menge von Taskprioritäten	65
t	Zeit	–
$t_\times$	Zeitpunkt des Ausfalls von $\text{PE}_\times$	123
$T_{\max}$	Zum aktuellen Zeitpunkt höchstpriore Task, die keinem PE zugeteilt ist	128
H_k	k-te harmonische Zahl, $H_k \coloneqq \sum_{i=1}^{k} i^{-1}$	177

Abkürzungsverzeichnis

AHS Artificial Hormone System, künstliches Hormonsystem

ACET Average Case Execution Time

PE Processing Element, Prozessorelement

WCET Worst Case Execution Time

1 Einleitung

1.1. Motivation: Alltägliche Komplexität

Komplexität ist in heutigen technischen Systemen allgegenwärtig. Das Notebook, auf dem diese Zeilen entstanden, wird über eine USB-C-Schnittstelle geladen. Das zugehörige Netzteil kommuniziert mit daran angeschlossenen Geräten, um beispielsweise zu erfragen, welche Spannung es liefern soll – beispielsweise 20, 15, 9 oder 5 Volt. Dies geschieht gemäß dem Standard „USB Power Delivery“ [USB]. Diese Standardisierung macht es möglich, statt des Notebooks beispielsweise ein Smartphone zu laden – es ist kein separates Netzteil erforderlich.

Dieses Beispiel zeigt, wie die Computertechnologie das moderne Leben durchdrungen hat: Computer finden sich heute in den unscheinbarsten Geräten, beispielsweise im erwähnten Netzteil in Form eines Mikrocontrollers. Der eigentliche Zweck des Netzteils – das Laden eines Akkumulators – kann nur aufgrund des Zusammenspiels vieler einzelner Rechnersysteme erfüllt werden: Auch das angeschlossene Gerät muss aktiv mit dem Netzteil die angeforderte Spannung aushandeln, eine entsprechende Ladeelektronik den Ladevorgang überwachen.

Es findet sich also Komplexität an den unscheinbarsten Orten, eingebettete Systeme sind allgegenwärtig und häufig nicht mehr wahrzunehmen. *Wie* komplex heutige eingebettete Systeme sind, die selbst scheinbar einfachste Aufgaben durchführen, soll im Folgenden genauer betrachtet werden.

Abbildung 1.1 zeigt das Ladegerät „PowerPort Atom PD 2“ der Firma Anker [Ank]. Es ist ähnlich zum eingangs erwähnten Netzteils, unterstützt jedoch das Laden zweier Geräte gleichzeitig. Der im Netzteil verwendete Mikrocontroller ist ein „CYPD4225“ der Firma Cypress [Cha19]. Diese Tatsache wird in [Hel20] genutzt, um dessen Fähigkeiten gemäß des zugehörigen Datenblatts [Cyp19] mit dem *Apollo Guidance Computer* (AGC) zu vergleichen. Dieser wurde in den Raumfähren des Apollo-Projekts genutzt: Ein Exemplar befand sich im Kommandomodul (*command and service module*, CSM), ein zweites im Landemodul (*lunar module*, LM) [Min08]. Folglich waren zwei solcher AGC mit dafür verantwortlich, dass im Jahre 1969 mit Apollo 11 die ersten Menschen den Mond betreten konnten – die Sinnhaftigkeit des Vergleichs eines modernen *Ladegeräts* mit über 50 Jahre alten Raumfahrtcomputern scheint daher zunächst fragwürdig.

Stellt man jedoch die Leistungsdaten des modernen Mikrocontrollers und jene des AGC gegenüber, wie in [Hel20] getan und in Tabelle 1.1 wiedergegeben, zeigen sich höchst interessante Resultate: Der Mikrocontroller des Ladegeräts hat eine fast 47-fach höhere Taktfrequenz als der AGC, doppelt so viel Arbeitsspeicher und die knapp 1,8-fache Menge an Programmspeicher. Hinzu kommt, dass der AGC für die meisten Operationen mehr als einen Taktzyklus benötigt – eine Addition dauert für 15 bzw. 31 Bit breite Operanden 24 bzw. 36 Taktzyklen [Hel20], während der moderne Mikrocontroller eine dreistufige Pipeline verwendet [Arm] und beide Operationen in

Abbildung 1.1: Ladegerät „PowerPort Atom PD 2“ der Firma Anker

jeweils einem einzigen Taktzyklus durchführen kann. Berücksichtigt man zudem dessen deutlich höheren Takt, so kann er in derselben Zeit ungefähr die 1123- bzw. 1685-fache Anzahl Additionen ausführen, sofern keine Pipeline-Stalls auftreten.[1] Selbst wenn *jede* Instruktion einen vollständigen Flush der Pipeline auslösen würde, sind die Faktoren mit 375 bzw. 562 immer noch beeindruckend.

Unter Zuhilfenahme weiterer Annahmen[2] wird in [Hel20] schließlich resümiert, dass der moderne Mikrocontroller höchstwahrscheinlich ausreichen würde, um den AGC zu ersetzen. Die Quelle merkt jedoch an, dass dies selbstverständlich keinen fairen Vergleich darstellt – der Mikrocontroller ist weder für Raumfahrtaufgaben konzipiert noch zertifiziert und würde daher unter den Einflüssen kosmischer Strahlung mit hoher Wahrscheinlichkeit häufig falsche Ergebnisse liefern oder gar komplett ausfallen. Auch besitzt der Mikrocontroller nur 30 General-Purpose-I/Os (GPIOs), was möglicherweise für die Steuerung einer Raumfähre nicht ausreicht.

Im Vergleich zu den komplexen Aufgaben, die der AGC im Apollo-Projekt erfüllte, wirkt die Funktion, welche der moderne Mikrocontroller im betrachteten Ladegerät erfüllt, geradezu trivial. Dieser Vergleich macht jedoch umso mehr deutlich, welche Komplexität sich in heutigen technischen Systemen verbirgt, selbst wenn deren Aufgaben augenscheinlich sehr einfach sind.

An dieser Stelle sei dahingestellt, ob die vom verwendeten Mikrocontroller bereitgestellte Rechenleistung im Ladegerät tatsächlich benötigt wird oder dieser vielmehr

[1] Während der AGC eine einzige Addition durchführt, vergehen 1125 bzw. 1687 Taktzyklen des Mikrocontrollers. Nach drei Zyklen ist die Pipeline gefüllt und die erste Addition abgearbeitet; in jedem der folgenden 1122 bzw. 1684 Zyklen kann dann eine weitere Addition fertiggestellt werden.

[2] Beispielsweise unterstützt der Prozessorkern des Mikrocontrollers im Gegensatz zum AGC keine Ganzzahldivision, folglich wird angenommen, dass ersterer eine solche in Software ausführen kann und dafür nicht länger benötigt als der AGC. Ferner nutzt der Mikrocontroller ein Instruktionsformat mit variabler Länge von 16 bzw. 32 Bit. Es wird daher angenommen, dass eine Übertragung des AGC-Programms auf den Befehlssatz des Mikrocontrollers aus nicht zu vielen 32-Bit-Instruktionen besteht, um noch in den Programmspeicher zu passen.

Tabelle 1.1: Vergleich eines USB-C-Ladegeräts mit dem Apollo Guidance Computer (AGC). Übersetzt von [Hel20].

	PowerPort Atom PD 2	**AGC**
Funktion	Ein oder zwei Notebooks bzw. Smartphones laden	• Großteil des Weges zum Mond fliegen (CSM) • Auf dem Mond landen (LM) • Vom Mond abheben (LM) • Zurück zur Erde fliegen (CSM)
Mikrochip	Cypress CYPD4225	Diskrete Bauteile
Taktfrequenz	48 MHz	1024 kHz = 1,024 MHz
RAM	8 KB	2048 15-Bit-Worte; inklusive Paritätsbits: 4 KB
Programm-speicher	128 KB Flash	36864 15-Bit-Worte; inklusive Paritätsbits: 72 KB
Instruktionssatz	32 Bit, ARM Cortex-M0	16 Bit, akkumulatorbasiert

gewählt wurde, weil er die günstigste am Markt verfügbare Option darstellte. Nichtsdestotrotz stellt sich die Frage, wie sich *komplexe* Aufgaben technisch umsetzen lassen, wenn heute bereits scheinbar *einfache* Probleme mit einem Hardwareaufwand einhergehen, der mit dem zu einer *Mondlandung* nötigen vergleichbar ist.

Zunächst wollen wir im folgenden Abschnitt jedoch weitere gegenwärtige Trends betrachten, die für heutige technische Systeme von Bedeutung sind.

1.2. Aktuelle Trends

Dass eingebettete Systeme heute allgegenwärtig sind, ist nur möglich, weil die Halbleitertechnologie seit der Entwicklung des ersten integrierten Schaltkreises im Jahre 1958 enorme Fortschritte gemacht hat: Verglichen mit der Technologie, die dem Apollo-Programm zur Verfügung stand, ist bereits die Leistungsfähigkeit *einfacher* moderner Mikrocontroller nahezu astronomisch hoch.

Wurde der 1978 eingeführte „Urvater“ der x86-Architektur, der Intel 8086, beispielsweise noch in einem 3 µm-Prozess gefertigt [Int], so kommt für seinen 2019 eingeführten Urenkel „Core i9-10980XE“ ein 14 nm-Prozess zum Einsatz [Int19]. Dies ist nur ein Beispiel für die allgemein schrumpfenden Strukturgrößen in der Halbleiterfertigung.

Ein weitere Entwicklung betrifft die Anzahl der Kerne pro Prozessor. Im Vergleich zu seinem Urahnen, der maximal einen Thread gleichzeitig bearbeiten kann, enthält der oben erwähnte Core i9 18 Kerne und kann 36 Threads gleichzeitig bearbeiten.

Das Topmodell der AMD-Epyc-Serverprozessoren schafft es gar auf 64 Kerne bzw. 128 Threads [Adv20].

Dieser Trend zu immer höherer Parallelität zeigt sich jedoch auch in eingebetteten Systemen: Viele technische Systeme werden heute durch viele einzelne Mikrocontroller gesteuert, die kooperieren und ein verteiltes System bilden. Als Beispiel hierzu seien moderne Automobile genannt, die durch bis zu 150 verschiedene Steuergeräte (*electronic control units*, ECUs) gesteuert werden [Win19], auch wenn zur Umsetzung autonomer Fahrfunktionen sowie komplexer Infotainmentsysteme mitunter eine Trendumkehr erkennbar ist, wie Teslas Modell 3 zeigt [Kum20]. Auch AUTOSAR[1] [AUT] als de-facto-Standard für automotive Softwarearchitekturen unterstützt Mikrocontroller mit mehreren Kernen, was dem zuvor erwähnten Trend mehrkerniger Mikroprozessoren Rechnung trägt.

1.3. Neue Herausforderungen

Dennoch lassen sich Probleme nicht allein dadurch lösen, dass technische Systeme aus immer mehr und immer leistungsfähigeren Mikroprozessoren aufgebaut werden: Die Systeme müssen auch entsprechend programmiert werden, um ein reibungsloses Zusammenspiel aller Komponenten zu gewährleisten. Es ist klar, dass die Vielzahl der an heutigen eingebetteten Systemen beteiligten Komponenten daher zusätzliche Komplexität mit sich bringt, die es zu beherrschen gilt.

Zudem unterliegen viele eingebettete Systeme hohen Sicherheitsanforderungen sowie Echtzeitbedingungen. Im Automobilbereich seien hierzu beispielsweise die Anforderungen der ISO 26262 [ISO18] genannt. Folglich reicht es nicht aus, die bloße Kooperation der einzelnen Komponenten sicherzustellen: Auch die Möglichkeit des Ausfalls einzelner Komponenten muss berücksichtigt werden, sodass auch ein solcher die Sicherheitsanforderungen nicht verletzt.

Ein für den Automobilbereich denkbares Redundanzkonzept könnte daher das Vorhalten eines dedizierten Backups für jede sicherheitskritische ECUs darstellen. Neuere Entwicklungen wie das AutoKonf-Projekt[2] [Orl+19; Osz+18] versuchen jedoch, die mit einem solchen Ansatz einhergehenden Kosten zu reduzieren, indem sich stattdessen mehrere verschiedene ECUs eine einzige generische Backup-ECU teilen. Im Falle des AutoKonf-Projekts wird die Kosteneinsparung jedoch durch eine verminderte Ausfallsicherheit erkauft: Fallen mehrere ECUs gleichzeitig aus, so stehen unter Umständen nicht genügend Backup-ECUs zur Verfügung, um alle Systemfunktionen am Laufen zu halten.

Möchte man also sowohl die Kosten senken als auch eine möglichst hohe Ausfallsicherheit erreichen, so wird ein flexibleres Redundanzkonzept benötigt. Aufgrund der hohen Komponentenzahl existiert jedoch eine schier unvorstellbar große Zahl möglicher Kombinationen einzelner Ausfälle, die Vorausberechnung statischer Reaktionen auf

[1] **Aut**omotive **O**pen **S**ystem **Ar**chitecture

[2] **Auto**matisch re**konf**igurierbare Aktorikansteuerungen für ausfallsichere automatisierte Fahrfunktionen

bestimmte Ausfallszenarien ist daher nicht mehr realistisch. Als Resultat sind neue Wege nötig, um die Systemkomplexität beherrschbar zu halten und gleichzeitig das nötige Maß an Ausfallsicherheit zu erreichen.

Mit dem zuvor benannten Trend schrumpfender Strukturgrößen geht jedoch insbesondere einher, dass transiente sowie permanente Fehler einzelner Bauteile immer wahrscheinlicher werden und der Ausfall von Komponenten daher in Zukunft eher als Regel- denn als Ausnahmefall zu betrachten ist.

Als Beispiel seien an dieser Stelle nur einige der möglichen Probleme in integrierten Schaltungen genannt:

Single-Event-Effekte (SEE) werden durch Strahlung hervorgerufen, beispielsweise durch Alphastrahlung.[1] Ein SEE stellt an sich noch keinen Fehler dar, sondern beschreibt lediglich die physikalischen Konsequenzen, die aus dem Strahlungseinfall resultieren. Führen diese zu einem Fehler, beispielsweise einem Bit-Flip, so spricht man von einem *Single Event Upset* (SEU). Hierbei steigt die Wahrscheinlichkeit für SEUs mit sinkenden Strukturgrößen [Ibe15].

Es ist leicht einzusehen, dass SEUs sich unterschiedlich auswirken können: Beispielsweise könnte ein Bit-Flip im Programmspeicher eines Systems zu einem fehlerhaften Programmablauf oder gar einem Softwareabsturz (und damit einem Ausfall) führen. Ein Bit-Flip in einem Prozessorregister dagegen könnte beispielsweise den Operanden einer arithmetischen Operation verändern, was ein falsches Ergebnis der Operation zur Folge hätte. Ebenso ist es jedoch auch möglich, dass ein Bit-Flip keinerlei Konsequenzen im System hat (beispielsweise, wenn der durch den Bit-Flip korrumpierte Wert nicht mehr gelesen wird).

Elektromigration (EM) ist ein Resultat des Stromflusses durch einen Leiter. Die fließenden Elektronen übertragen aufgrund von Stößen ihren Impuls auf die Metallionen des stromführenden Leiters. Ist die hierdurch hervorgerufene Kraft auf die Ionen groß genug, tritt ein Materialtransport innerhalb des Leiters von der Kathode zur Anode ein. Als Konsequenz können Leiterbahnen unterbrochen werden, wenn sämtliches Leiterbahnmaterial abtransportiert wurde, oder Kurzschlüsse auftreten, wenn zwei Leiterbahnen durch angelagertes Material leitfähig verbunden werden. Folglich kann Elektromigration zu permanenten Ausfällen einer integrierten Schaltung führen. Wann Elektromigration zu einem solchen Ausfall führt, ist vom Ausmaß des Materialtransports und dem Schaltungsaufbau abhängig, folglich sind Zeitspannen von wenigen Wochen bis hin zu mehreren Jahren möglich. Generell lässt sich jedoch sagen, dass der Materialtransport mit wachsender Stromdichte im Leiter zunimmt. Diese nimmt jedoch insbesondere mit sinkenden Strukturgrößen zu, weshalb Elektromigration in heutigen integrierten Schaltungen einen nicht zu unterschätzenden Effekt darstellt [LT18].

Thermische Probleme. Die Verkleinerung der Strukturgrößen integrierter Schaltungen geht mit einer Vergrößerung der auftretenden Leckströme einher. Als Resultat

[1] Alphastrahlung besteht aus Alphateilchen (Heliumkernen) und wirkt ionisierend [Ibe15].

steigt der Energieverbrauch der Schaltung [Kim+03]. Da es bei kleineren Strukturgrößen zudem möglich ist, auf derselben Fläche mehr Transistoren unterzubringen, steigt die pro Fläche abzuführende Wärme an. Zudem ist die thermische Verlustleistung nicht homogen auf einem Chip verteilt: Die Leckströme können aufgrund von Prozessabweichungen innerhalb des Chips stark variieren, zudem produzieren Bereiche mit hoher Schaltaktivität mehr Abwärme als weniger aktive Bereiche. Als Resultat entstehen im Chip Bereiche hoher Temperatur, sogenannte *Hotspots*, deren Position nur schwer vorhergesagt werden kann. Zur Vermeidung solcher Hotspots kann es daher unter Umständen nötig sein, die Taktrate des Chips temporär zu verringern, um seinen Energieverbrauch zu senken, was sich jedoch nachteilig auf die Rechenleistung auswirkt [LV06]. Ebenso begünstigen hohe Temperaturen den Effekt der Elektromigration, was sich wiederum negativ auf die Zuverlässigkeit der integrierten Schaltung auswirkt [LT18].

Insbesondere bei der 3D-Integration integrierter Schaltungen verschlimmern sich diese Probleme, da es noch schwieriger wird, die anfallende Abwärme abzuführen. Als Resultat müssen thermische Effekte beim Design der Schaltung besonders beachtet werden [LV06; Jai+10].

Folglich erhöhen die beschriebenen Effekte den Bedarf für ein flexibles Redundanzkonzept, um Komponentenausfälle kompensieren zu können.

1.4. Lösungsansätze

Einen Ansatz zur Beherrschung der wachsenden Komplexität technischer Systeme stellt das *Organic Computing* dar. Dieses versucht, Organisationsprinzipien, wie sie natürlichen Systemen zugrunde liegen, auf technische Systeme zu übertragen und wird in Kapitel 3 im genauer beschrieben.

Für diese Dissertation relevant ist im Kontext des Organic Computing insbesondere das *künstliche Hormonsystem* (*artificial hormone system*, AHS), welches in Kapitel 4 im Detail vorgestellt wird.

An dieser Stelle sei daher lediglich erwähnt, dass das AHS eine Middleware darstellt, die zur Verteilung von Tasks in verteilten eingebetteten Systemen eingesetzt werden kann. Dabei ist es insbesondere möglich, die zur Verteilung aller Tasks nötige Zeit durch harte Zeitschranken nach oben hin zu beschränken, was den Einsatz des AHS in Echtzeitsystemen ermöglicht. Ferner kann ein mithilfe des AHS entwickeltes System insbesondere temporäre oder permanente Komponenten- sowie Taskausfälle selbstständig erkennen und durch Neuverteilung der betroffenen Tasks ebenfalls echtzeitfähig kompensieren, sofern die Rechenleistung der verbleibenden Rechenknoten hierfür ausreicht.

Ist eine solche Neuverteilung jedoch nicht mehr möglich, weil zu viele Komponenten ausgefallen sind, so ist das Verhalten des AHS in seiner aktuellen Ausführung nicht definiert und es können folglich keinerlei Garantien mehr gegeben werden. Die Gewährleistung von Garantien auch in solchen Situationen bildet nun die Motivation der Forschungen, welche im Rahmen dieser Dissertation durchgeführt wurden.

1.5. Zielsetzung dieser Dissertation

Das Ziel dieser Dissertation ist es, einen Beitrag zur Beherrschung der steigenden Komplexität eingebetteter Systeme bei gleichzeitiger Erhöhung der Ausfallsicherheit zu leisten, indem Möglichkeiten zur prioritätsbasierten Taskverteilung in selbstorganisierten Systemen untersucht werden.

Konkret wird hierzu das AHS erweitert, um die Reihenfolge der Taskverteilung durch Prioritäten beeinflussen zu können. Dies ermöglicht zum einen den deterministischen Start eines auf Grundlage des AHS entwickelten Systems, sodass die wichtigsten Funktionen zuerst zugeteilt werden.

Zum anderen ermöglicht es eine solche Priorisierung, das System in *Überlastsituationen* automatisch zu degradieren: Eine solche Situation liegt vor, wenn durch Komponentenausfälle nicht mehr genug Kapazitäten bereitstehen, um *alle* Tasks im System zu verteilen. Die Zuweisung von Prioritäten ermöglicht es nun, automatisch und in deterministischer Reihenfolge unkritische Tasks zugunsten kritischer Tasks zu stoppen, um die wichtigsten Systemfunktionen funktionsfähig zu halten.

Hierbei ist die Zeit, welche benötigt wird, um auf eine Überlastsituation zu reagieren, stark von der zum Stopp der unkritischen Tasks konkret verfolgten Strategie abhängig. Daher werden im Laufe dieser Dissertation verschiedene solcher Strategien entwickelt und im Detail analysiert. Insbesondere werden harte Zeitschranken hergeleitet, deren Einhaltung in regulären Unterlastsituationen sowie in Überlastsituationen garantiert werden kann. Umfassende Evaluationen, die mithilfe einer Implementierung der prioritätsbasierten Erweiterung durchgeführt wurden, bestätigen die theoretisch hergeleiteten Zeitschranken.

1.6. Aufbau dieser Dissertation

Der Aufbau dieser Dissertation gestaltet sich wie folgt:

Kapitel 2 legt zunächst einige mathematische Grundlagen, die im weiteren Verlauf vorausgesetzt werden.

Kapitel 3 erläutert kurz die Hintergründe des *Organic Computing*, das zum Ziel hat, die Komplexität technischer Systeme zu verstehen und zu beherrschen.

Kapitel 4 stellt das AHS vor.

Kapitel 5 beleuchtet die Echtzeiteigenschaften des AHS im Detail und verfeinert die bisher bekannten Zeitschranken.

Kapitel 6 stellt eine Erweiterung des AHS vor, welche es ermöglicht, Tasks *prioritätsbasiert* im System zu verteilen.

Kapitel 7 evaluiert diese Erweiterung in Unterlastsituationen.

Kapitel 8 analysiert das Verhalten in Überlastsituationen. Hierbei werden insbesondere harte Zeitschranken hergeleitet, die in solchen Situationen garantiert werden können.

Kapitel 9 evaluiert diese Zeitschranken in Überlastsituationen.

Kapitel 10 liefert einen Überblick über verwandte Konzepte und Arbeiten.

Kapitel 11 fasst schließlich die Ergebnisse dieser Dissertation zusammen und motiviert weiterführende Forschungsthemen.

1.7. Publikationen

Wesentliche Ergebnisse dieser Dissertation wurden bereits in folgenden Publikationen veröffentlicht:

[HB20] Eric Hutter und Uwe Brinkschulte. „Towards a Priority-Based Task Distribution Strategy for an Artificial Hormone System“. In: *Architecture of Computing Systems – ARCS 2020*. Hrsg. von André Brinkmann u. a. Bd. 12155. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, S. 69–81.

Ausgezeichnet mit dem *Best Paper Award* der *33rd International Conference on Architecture of Computing Systems (ARCS) 2020*.

[Hut21] Eric Hutter. „Degrading Systems by Priority-Based Self-Healing Using an Artificial Hormone System“. In: *Organic Computing: Doctoral Dissertation Colloquium 2020*. Hrsg. von Sven Tomforde und Christian Krupitzer. Intelligent Embedded Systems 18. Kassel: Kassel University Press GmbH, 2021, S. 3–15.

[HB21a] Eric Hutter und Uwe Brinkschulte. „Evaluating a Priority-Based Task Distribution Strategy for an Artificial Hormone System“. In: *Architecture of Computing Systems – ARCS 2021*. Hrsg. von Christian Hochberger, Lars Bauer und Thilo Pionteck. Bd. 12800. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, S. 150–164.

[HB21b] Eric Hutter und Uwe Brinkschulte. „Handling Assignment Priorities to Degrade Systems in Self-Organizing Task Distribution“. In: *2021 24th International Symposium on Real-Time Distributed Computing (ISORC)*. 2021 24th International Symposium on Real-Time Distributed Computing (ISORC). Daegu, South Korea: IEEE, Juni 2021, S. 132–140.

Ausgezeichnet mit dem *Best Student Paper Award* des *2021 24th International Symposium on Real-Time Distributed Computing (ISORC)*.

2 Mathematische Grundlagen

In diesem Kapitel sollen einige mathematische Grundlagen dargelegt werden. Insbesondere werden allgemeine Definitionen und Lemmata betrachtet, die in späteren Kapiteln vorausgesetzt werden.

2.1. Natürliche und ganze Zahlen

Um Missverständnissen vorzubeugen, sollen zunächst folgende Notationen für die Mengen der natürlichen und ganzen Zahlen eingeführt werden:

Notation 2.1. Es bezeichne $\mathbb{N} := \{0, 1, 2, 3, \dots\}$ die Menge der *natürlichen Zahlen*. Diese enthält insbesondere die Null.

Im Gegensatz dazu bezeichne $\mathbb{N}^+ := \mathbb{N} \setminus \{0\}$ die Menge der *positiven natürlichen Zahlen*.

Natürliche Zahlen, $\mathbb{N}$

Positive natürliche Zahlen, $\mathbb{N}^+$

Notation 2.2. Es bezeichne $\mathbb{Z} := \{\dots, -2, -1, 0, 1, 2, \dots\}$ die Menge der *ganzen Zahlen*.

Ganze Zahlen, $\mathbb{Z}$

2.2. Iverson-Klammer

Definition 2.3. Sei A eine Aussage, die entweder wahr oder falsch sein kann. Dann gelte

$$[A] := \begin{cases} 0, & A \text{ falsch}, \\ 1, & A \text{ wahr}. \end{cases}$$

Bemerkung 2.4. Diese Notation geht ursprünglich auf Kenneth E. Iverson zurück, wurde von Donald E. Knuth zur hier gezeigten Form verallgemeinert [Knu92] und findet unter anderem in [GKP94] Anwendung. Daher wird sie im Folgenden als *Iverson-Klammer* bezeichnet.

Iverson-Klammer, $[A]$

2.2.1. Anwendungen

Die Verwendung von Iverson-Klammern ermöglicht eine elegante Notation verschiedener Sachverhalte. Zwei für diese Dissertation relevante Anwendungen der Iverson-Klammer wollen wir nun kurz betrachten.

Fallunterscheidungen Mithilfe von Iverson-Klammern lassen sich Fallunterscheidungen kompakt darstellen. Beispielsweise ist

$$f(x,y) \coloneqq \begin{cases} x+b, & y<1 \\ x+a+b, & y=1 \\ 2x+b, & y>1 \end{cases}$$

identisch zu

$$f(x,y) \coloneqq x + x \cdot [y>1] + a \cdot [y=1] + b.$$

Verwendung in Summentermen Auch in Summentermen kann die Verwendung von Iverson-Klammern hilfreich sein, um Terme kompakt darzustellen. Beispielsweise entspricht der Term

$$\sum_{i \in \mathbb{N}} \left[i^2 < n \right]$$

der Anzahl aller $i \in \mathbb{N}$, für die $i^2 < n$ gilt.

2.3. Auf- und Abrundungsfunktion

An verschiedenen Stellen dieser Arbeit werden Auf- und Abrundungsfunktionen verwendet, die wie folgt definiert seien:

Aufrundungsfunktion, Ceilingfunktion, $\lceil x \rceil$

Definition 2.5. Die *Aufrundungs-* oder *Ceilingfunktion* sei für alle $x \in \mathbb{R}$ definiert durch

$$\lceil x \rceil \coloneqq \min\{z \in \mathbb{Z} \mid z \geq x\}.$$

Abrundungsfunktion, Floorfunktion, $\lfloor x \rfloor$

Definition 2.6. Die *Abrundungs-* oder *Floorfunktion* sei für alle $x \in \mathbb{R}$ definiert durch

$$\lfloor x \rfloor \coloneqq \max\{z \in \mathbb{Z} \mid z \leq x\}.$$

Bemerkung 2.7. Diese Notationen sowie die Namen *Floor-* und *Ceilingfunktion* gehen ebenfalls auf Kenneth E. Iverson zurück [GKP94, Kap. 3.1].

Zudem werden zur Umwandlung zwischen Auf- und Abrundungsfunktion folgende Äquivalenzen vorausgesetzt:

Lemma 2.8. Für $n \in \mathbb{Z}$ und $m \in \mathbb{N}^+$ gelten

$$\left\lceil \frac{n}{m} \right\rceil = \left\lfloor \frac{n-1}{m} \right\rfloor + 1 \tag{2.1}$$

sowie

$$\left\lfloor \frac{n}{m} \right\rfloor = \left\lceil \frac{n+1}{m} \right\rceil - 1. \tag{2.2}$$

Beweis. Für $n \in \mathbb{Z}$ und $m \in \mathbb{N}^+$ gilt nach [GKP94, ex. 3.12]

$$\left\lceil \frac{n}{m} \right\rceil = \left\lfloor \frac{n+m-1}{m} \right\rfloor.$$

Damit folgt Gleichung (2.1):

$$\left\lceil \frac{n}{m} \right\rceil = \left\lfloor \frac{n+m-1}{m} \right\rfloor = \left\lfloor \frac{n-1}{m} + 1 \right\rfloor = \left\lfloor \frac{n-1}{m} \right\rfloor + 1.$$

Aus dieser folgt nun Gleichung (2.2):

$$\left\lfloor \frac{n}{m} \right\rfloor = \left\lfloor \frac{(n+1)-1}{m} \right\rfloor + 1 - 1 = \left\lceil \frac{n+1}{m} \right\rceil - 1. \qquad \square$$

2.4. Multimengen

Um Systeme, die mehrfach zuteilbare Tasks umfassen, formal beschreiben zu können, sollen im Folgenden Notationen für Multimengen eingeführt werden. Die im Rahmen dieser Dissertation verwendete Notation orientiert sich an verschiedenen Notationen aus [Cal01] sowie [GJ11] und soll im Folgenden kurz eingeführt werden.

Universum
Multimenge

Definition 2.9. Es sei U eine Menge, die *Universum* genannt werde.
Eine *Multimenge* M über U sei eine Abbildung $M : U \to \mathbb{N}$. Für jedes $x \in U$ gebe $M(x)$ an, wie oft x in M enthalten ist.

$\{\!|\cdot|\!\}$

Bemerkung 2.10. Soll eine Multimenge durch Auflistung ihrer Elemente angegeben werden, so werden diese in $\{\!|\cdot|\!\}$ eingeschlossen, um den Unterschied zu einer Menge deutlich zu machen.

> **Beispiel**
>
> Es sei $U := \{1, 2, 3\}$. Dann kann die Multimenge M über U mit
>
> $$M(1) = 2, \quad M(2) = 0, \quad M(3) = 3$$
>
> geschrieben werden als $\{\!|1, 1, 3, 3, 3|\!\}$.

Bemerkung 2.11. Eine Menge $M \subseteq U$ kann als eine Multimenge M' aufgefasst werden, für die

$$M'(x) = \begin{cases} 1, & x \in M \\ 0, & x \notin M \end{cases}$$

für alle $x \in U$ gilt; M' entspricht dabei gerade der charakteristischen Funktion von M.

Teilmultimenge, $\subseteq$

Definition 2.12. Eine Multimenge M heiße *Teilmultimenge* einer Multimenge N, kurz $M \subseteq N$, falls $M(x) \leq N(x)$ für alle $x \in U$ gilt.

Im Kontext dieser Dissertation werden gelegentlich Teilmultimengen benötigt, die jedes Element ihrer Obermultimenge entweder genauso oft wie diese oder gar nicht enthalten. Solche speziellen Teilmultimengen werden im Folgenden *induzierte* Teilmultimengen genannt:

Induzierte Teilmultimenge, $\subseteqq$

Definition 2.13. Eine Multimenge M heiße *induzierte Teilmultimenge* einer Multimenge N, kurz $M \subseteqq N$, falls für alle $x \in U$ entweder $M(x) = N(x)$ oder $M(x) = 0$ gilt.

> **Beispiel**
>
> Sei $N \coloneqq \{\!|1, 2, 2, 3, 3, 3|\!\}$ eine Multimenge über $U \coloneqq \{1, 2, 3\}$.
> Für $M \coloneqq \{\!|1, 2, 3|\!\}$ gilt $M \subseteq N$, aber nicht $M \subseteqq N$, da $M(2) \neq N(2)$ sowie $M(3) \neq N(3)$.
> Für $M' \coloneqq \{\!|1, 2, 2|\!\}$ gelten dagegen sowohl $M' \subseteq N$ als auch $M' \subseteqq N$.

Element von, $\in$, $\notin$

Definition 2.14. Sei M eine Multimenge über einem Universum U. Ein $x \in U$ heiße *Element von* M (kurz: $x \in M$), wenn $M(x) > 0$. Analog bedeute $x \notin M$, dass $M(x) = 0$.

Kardinalität, $|M|$

Definition 2.15. Sei M eine Multimenge über einem Universum U.
Dann sei die *Kardinalität* $|M|$ von M definiert durch

$$|M| \coloneqq \sum_{x \in U} M(x) = \sum_{x \in M} M(x).$$

Reduzierte Grundmenge, supp

Definition 2.16. Es sei M eine Multimenge. Dann bezeichne die *(reduzierte) Grundmenge* (*support*) $\operatorname{supp}(M)$ eine Menge, die genau alle Elemente aus U enthält, die mindestens einmal in M enthalten sind:

$$\operatorname{supp}(M) \coloneqq \{x \in U \mid M(x) > 0\}$$

> **Beispiel**
>
> Sei $M \coloneqq \{\!|1, 1, 1, 2, 3, 4, 4|\!\}$. Dann gilt $\operatorname{supp}(M) = \{1, 2, 3, 4\}$.

Disjunkt

Definition 2.17. Seien M, N Multimengen über einem Universum U.
Dann heißen M und N genau dann *disjunkt*, wenn $\operatorname{supp}(M) \cap \operatorname{supp}(N) = \emptyset$.

> **Bemerkung**
>
> Anschaulich bedeutet dies: M und N heißen disjunkt, wenn es kein Element des Universums U gibt, das sowohl in M als auch in N enthalten ist.

Definition 2.18. Sei M eine Multimenge über einem Universum U. Sei $N \subseteq U$ eine *Menge*.

Mengendifferenz, $M \setminus N$

Dann sei die *Mengendifferenz* $M \setminus N$ über U eine Multimenge, die durch die Abbildung

$$(M \setminus N)(x) \coloneqq \begin{cases} M(x), & x \notin N \\ 0, & x \in N \end{cases}$$

für alle $x \in U$ definiert ist.

Beispiel

Sei $M \coloneqq \{\!|1, 1, 2, 3, 4, 4, 4|\!\}$ und $N \coloneqq \{1, 2\}$. Dann gilt $M \setminus N = \{\!|3, 4, 4, 4|\!\}$.

Im Rahmen dieser Arbeit wird zumeist eine Multimenge $\mathcal{T}$ aller Taskinstanzen eines Systems betrachtet. Daher sei der Einfachheit halber folgende Konvention vereinbart:

Konvention 2.19. Falls für eine Multimenge auf die Angabe eines Universums verzichtet wird, gelte implizit, dass U der Menge $\{T_i \mid i \in \mathbb{N}\}$ aller möglichen Tasks T_i entspricht (bzw. gegebenenfalls der Menge $\{T\}$, falls nur eine einzige Task betrachtet wird und daher keine Unterscheidung verschiedener Tasks nötig ist).

3 Organic Computing

Das bereits zuvor erwähnte *Organic Computing* stellt eine Strömung innerhalb der Informatik dar, die zum Ziel hat, Wege zu erforschen, um die stetig steigende Komplexität technischer Systeme beherrschen zu können. Hierbei wird das Prinzip der Selbstorganisation, wie es in vielen natürlichen Systemen in verschiedenen Ausprägungen beobachtet werden kann, auf technische Systeme übertragen.

Zunächst wollen wir kurz die Geschichte des Organic Computing betrachten.

3.1. Geschichte

Der Begriff *Organic Computing* geht auf einen 2002 abgehaltenen Workshop der Fachgruppe *Architektur von Computer-Systemen* (ARCS) der Gesellschaft für Informatik (GI) zurück, in dem versucht wurde, Zukunftstrends in der Informatik vorherzusagen [TSM17; MT17]. Die so entstandenen Resultate wurden 2003 in einem gemeinsamen Positionspapier der GI und der Informationstechnischen Gesellschaft im VDE (ITG) veröffentlicht [All+03].

Im Kontext des von 2005 bis 2011 laufenden DFG-Schwerpunktprogramms 1183 „Organic Computing“ [Deua] wurden schließlich verschiedene Forschungsprojekte gefördert. Zur Mitte des Schwerpunktprogramms wurden erste Ergebnisse einiger dieser Projekte in [Wür08] vorgestellt, weitere folgten zum Ende des Programms in [Mül11].

3.2. Grundgedanken

Im Folgenden wollen wir nun einige Aspekte des Organic Computing kurz vorstellen. Hierbei gehen wir jedoch nur auf dessen wesentliche Ideen ein, die nötig sind, um das in Kapitel 4 beschriebene *künstliche Hormonsystem* sowie die darauf aufbauende und im Rahmen dieser Dissertation vorgestellte Forschung in den Kontext des Organic Computing einordnen zu können.

Wie bereits erwähnt, soll Organic Computing helfen, komplexe technische Systeme beherrschen zu können. Hierzu sollen Systeme so entwickelt werden, dass diese sich selbstständig an geänderte Betriebsumstände anpassen können. Konkret lässt sich Organic Computing dadurch charakterisieren, dass Entscheidungen, die traditionell von einem Systementwickler zur Designzeit eines Systems getroffen wurden, in die Laufzeit des Systems verlagert werden – und damit unter die Kontrolle des Systems [TSM17; MT17]. Als Resultat dieser Verlagerung ist das System in der Lage, viele Entscheidungen (teilweise) autonom zu treffen, es kann als *selbstorganisierend* bezeichnet werden.

3.3. Selbst-X-Eigenschaften

Im Rahmen des Organic Computing lässt sich das Systemverhalten mithilfe verschiedener sogenannter *Selbst-X-Eigenschaften* charakterisieren. Die im Rahmen dieser Dissertation wichtigsten solchen Eigenschaften lassen sich nach [TSM17] wie folgt definieren:

Selbstkonfiguration Das System konfiguriert sich in Bezug auf seine Parameter selbstständig, um ein höheres Ziel zu bewerkstelligen.

Selbstheilung Das System kann auftretende Fehler und Ausfälle selbstständig erkennen, diagnostizieren und heilen.

Selbstoptimierung Das System analysiert seine Entscheidungen und versucht, bessere Lösungen zu erreichen.[1]

Daneben existiert eine Vielzahl weiterer Selbst-X-Eigenschaften, für zusätzliche Informationen sei auf die angegebene Literatur verwiesen. Es lässt sich jedoch festhalten, dass die genannten Eigenschaften alle auch in verschiedensten natürlichen Systemen zu beobachten sind – entsprechend groß ist die Zahl möglicher Inspirationsquellen, um solche Selbst-X-Eigenschaften in technischen Systemen umzusetzen.

3.4. Abgrenzung zum Autonomic Computing

Neben ihrer Bedeutung für das Organic Computing sind Selbst-X-Eigenschaften auch im Rahmen des *Autonomic Computing* von hoher Relevanz. Diese Strömung kam 2003 ungefähr zeitgleich mit dem Organic Computing auf. Konkret bezieht sich der Begriff *Autonomic* dabei auf das (autonome) Nervensystem [KC03]. Wie beim Organic Computing liegt hier also die Übertragung eines natürlichen Sachverhalts auf technische Systeme zugrunde.

Für das Autonomic Computing ist neben den in Abschnitt 3.3 genannten Eigenschaften der Selbstkonfiguration, -heilung und -optimierung auch der *Selbstschutz* des Systems von Bedeutung. Diese Eigenschaft beschreibt, dass ein System sich automatisch gegen bösartige Angriffe oder kaskadierende Fehler schützt und ferner mögliche Probleme so früh wie möglich vorhersieht, erkennt und proaktiv Gegenmaßnahmen einleitet [KC03]. Der Vollständigkeit halber sei an dieser Stelle betont, dass der Selbstschutz eines Systems auch Bestandteil des Organic Computing ist [TSM17], jedoch außerhalb des Fokus dieser Dissertation liegt und daher im besagten Abschnitt nicht separat betrachtet wurde.

[1] In [TSM17] wird als Bezeichnung dieser Eigenschaft stattdessen *Selbstverbesserung* bevorzugt, da in realen Anwendungsszenarien oft keine globale Optimierung möglich ist, weshalb lediglich versucht werden kann, *möglichst gute* Konfigurationen zu finden. Aus Gründen der Konsistenz zu früheren Publikationen bezüglich des in Kapitel 4 vorgestellten künstlichen Hormonsystems wird im Rahmen dieser Dissertation dennoch von *Selbstoptimierung* gesprochen.

Auf den ersten Blick mögen Organic- und Autonomic-Computing-Systeme daher ähnliche Eigenschaften aufweisen, im Detail lassen sich dennoch Unterschiede ausmachen, welche die Strömungen zumindest teilweise voneinander abgrenzen können.

Autonomic Computing wurde ursprünglich von IBM vorangetrieben, um automatisierte Lösungen für Rechenzentren zu entwickeln: Durch Selbstkonfiguration und verwandte Eigenschaften sollte die wachsende Komplexität beherrschbar gehalten werden. Hierbei legt Autonomic Computing den Fokus vor allem auf selbstverwaltete Individualsysteme und ist stark vom natürlichen Nervensystem inspiriert. Im Kontext des Organic Computing dagegen stammen die auf technische Systeme übertragenen Verhaltensmuster meist aus einem breiteren Spektrum natürlicher Vorbilder und der Fokus liegt nicht zwangsläufig nur auf den Systemen an sich, sondern auch auf dem generellen Verständnis der Mechanismen der Selbstorganisation [MT17, Kap. 9.6].

Nichtsdestotrotz sind sich Autonomic und Organic Computing sehr ähnlich. Trotz ihrer oben wiedergegebenen Unterscheidungsmerkmale bezeichnen Müller-Schloer und Tomforde beide Strömungen dennoch als komplementäre Forschungsrichtungen, die insbesondere einem gegenseitigen Ideenaustausch unterliegen [MT17, Kap. 9.6].

Ähnlich resümieren Lalanda et al. aus Sicht des Autonomic Computing: Häufig seien die Grenzen zwischen beiden Strömungen schwer zu ziehen [LMD13, Kap. 2.5.3.4].

3.5. Zusammenfassung

Nach diesem kurzen Exkurs über die Hintergründe des Organic Computing wollen wir uns im nächsten Kapitel dem bereits erwähnten *künstlichen Hormonsystem* widmen, das aufgrund seiner im natürlichen Hormonsystem gefundenen Inspiration als konkrete Ausprägung des Organic Computing zu betrachten ist.

Hierbei sind insbesondere die in Abschnitt 3.3 genannten Selbst-X-Eigenschaften von Relevanz, die vom künstlichen Hormonsystem realisiert werden. Im weiteren Verlauf dieser Dissertation wird betrachtet, welche Garantien bezüglich dieser Eigenschaften gegeben werden können und welchen Einschränkungen diese unterliegen.

4 Das Künstliche Hormonsystem

Die im Rahmen dieser Dissertation vorgestellten Ansätze basieren auf dem künstlichen Hormonsystem, das in diesem Kapitel im Detail vorgestellt werden soll.

Die Abschnitte 4.1 bis einschließlich 4.3 basieren im Wesentlichen auf [BPvR08; vRBP11b]; für nähere Informationen sei daher auf diese Publikationen verwiesen.

4.1. Hintergrund

AHS

Das künstliche Hormonsystem (*artificial hormone system*, *AHS*) ist eine Middleware zur Zuteilung von Tasks an die Rechenknoten eines verteilten Systems. Im Allgemeinen stellt eine solche Middleware den Tasks eine einheitliche Schnittstelle samt Kommunikationsmechanismen bereit, sodass die Tasks miteinander interagieren können, als liefen sie auf einem Einzelsystem und nicht auf möglicherweise verschiedenen Rechenknoten eines verteilten Systems, wie in Abbildung 4.1 skizziert. Im Kontext des AHS werden die verschiedenen Rechenknoten allgemein als *Prozessorelemente* (*processing elements*, *PEs*) bezeichnet.

Prozessorelement, PE

Das AHS ist insbesondere als vollständig dezentrale Middleware konzipiert, die keinen *Single Point of Failure* beinhaltet: Im Falle des Ausfalls eines PEs wird dieser automatisch bemerkt und die betroffenen Tasks werden auf funktionsfähige PEs migriert, sofern deren Kapazität ausreicht, diese Tasks zusätzlich zu übernehmen. In der Folge sind Systeme, die mithilfe des AHS entwickelt wurden, besonders tolerant gegenüber Ausfällen von PEs.

Zur Realisierung dieses Konzepts bedient sich das AHS biologisch inspirierter Prinzipien: Wie der Name *künstliches Hormonsystem* bereits suggeriert, überträgt das AHS die Grundkonzepte des Hormonsystems der Säugetiere auf technische Systeme: Im biologischen Vorbild werden chemische Botenstoffe, sogenannte *Hormone*, von Drüsen an benachbarte Zellen *(parakrin)* oder über den Blutkreislauf an den gesamten Organismus *(endokrin)* ausgeschüttet. Die Wirkung, die ein Hormon entfaltet, hängt dabei nur von der empfangenden Zelle ab; durch das Wechselspiel antagonistischer

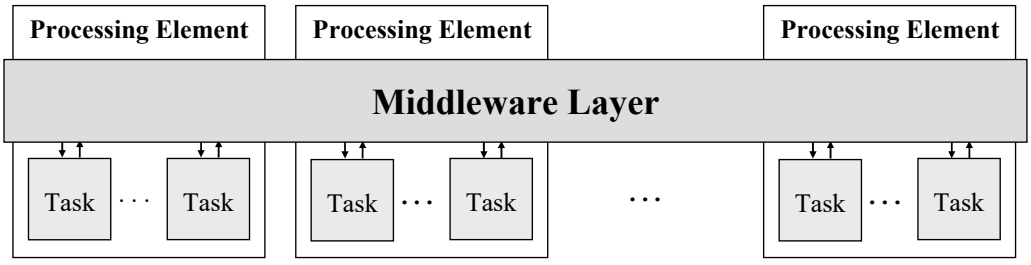


Abbildung 4.1: Middleware in einem verteilten System [vRBP11b]

Hormone werden innerhalb des Organismus so dezentrale Regelkreise aufgebaut, welche die verschiedenen Organfunktionen regulieren [KR21].

Ebenso wie das natürliche Hormonsystem baut das AHS vollständig dezentrale Regelkreise auf, um die einzelnen Tasks auf die PEs zu verteilen. Fällt ein PE aus, so ist es nicht mehr in der Lage, Hormone auszuschütten, was erkannt werden kann. In der Folge werden die vom Ausfall betroffenen Tasks durch die Regelkreise automatisch neuen PEs zugeteilt.

Im Gegensatz zum natürlichen Hormonsystem verwendet das AHS zur Realisierung der Regelkreise jedoch keine chemischen Botenstoffe, sondern bildet diese vielmehr durch kurze digitale Nachrichten nach, die über ein Kommunikationsnetzwerk übertragen und wie ihre natürlichen Vorbilder *Hormone* genannt werden.

4.2. Funktionsweise

Auf jedem PE des verteilten Systems läuft eine Instanz des AHS. Diese führt insbesondere für jede Task des Systems einen Regelkreis aus, der durch den Austausch von drei Grundtypen von Hormonen realisiert wird:

Eignungswerte (*eager values*) E
: geben die Eignung eines PEs für eine bestimmte Task an: Beispielsweise kann ein PE, das über eine Gleitkommaeinheit verfügt, eine Task, die Gleitkommaoperationen durchführt, effizienter ausführen als ein PE, das solche Operationen in Software durchführen muss. Diese unterschiedliche Eignung der PEs kann durch die Festlegung ihrer Eignungswerte für die Task modelliert werden, sodass diese vom AHS nach Möglichkeit auf dem besser geeigneten PE zugeteilt wird.

Suppressoren S
: hemmen die Ausführung von Tasks auf PEs: Suppressoren werden von Eignungswerten subtrahiert. Insbesondere werden Suppressoren verwendet, um zu steuern, wie viele Instanzen einer Task im System zugeteilt werden. Des Weiteren können verschiedene Untertypen von Suppressoren verwendet werden, um die Zuteilung weiterer Tasks auf bereits ausgelasteten PEs zu verhindern.

Acceleratoren A
: wirken antagonistisch zu Suppressoren und begünstigen die Ausführung von Tasks auf PEs: Acceleratoren werden zu Eignungswerten addiert. Ist eine Task auf einem PE zugeteilt, so werden insbesondere Acceleratoren ausgeschüttet, um die Ausführung kooperierender Tasks auf benachbarten PEs zu begünstigen. Dies reduziert die Kommunikationsdistanzen und führt zur Bildung *virtueller Organe*. Außerdem können Acceleratoren genutzt werden, um günstige PE-Zustände zu modellieren (z.B. geringe Auslastung) und so die Zuteilung von Tasks auf diesen PEs zu begünstigen.

4.2.1. Nomenklatur

Bevor mit der Beschreibung der Funktionsweise des AHS fortgefahren wird, soll zunächst die im Folgenden verwendete Nomenklatur präzisiert werden.

Tasks werden mit T gekennzeichnet und zur Unterscheidung gegebenenfalls mit einer Zahl bzw. einem lateinischen Buchstaben versehen, z.B. T_1 oder T_i. Mengen von Tasks werden dagegen mit $\mathcal{T}$ gekennzeichnet, z.B. $\mathcal{T}$ oder $\mathcal{T}_1$.

Um Hormone genauer zu spezifizieren, werden deren Sendertask und -PE sowie Empfängertask und -PE gegebenenfalls als Sub- bzw. Superskript aufgeführt. Konkret bezeichne

$\mathcal{H}_{i,j}^{k,l}$ ein Hormon $\mathcal{H}$, das von Task T_i, die auf PE_j zugeteilt ist, an Task T_k auf PE_l gesendet wird. Das Hormon ist jedoch lediglich an die auf PE_l zugeteilten Instanzen von T_k *adressiert*; die Task muss auf diesem PE jedoch nicht zwangsläufig zugeteilt sein.

$\mathcal{H}_{i,j}$ ein Hormon $\mathcal{H}$, das von der auf PE_j zugeteilten Task T_i an einen nicht weiter spezifizierten Adressaten gesendet wird sowie

$\mathcal{H}^{i,j}$ ein Hormon $\mathcal{H}$, das auf PE_j für Task T_i von einem nicht weiter spezifizierten Sender empfangen wurde.

Hierbei sei $\mathcal{H}$ als Platzhalter für einen konkreten Hormontyp zu verstehen. Existiert im System nur eine einzige Task T, so kann statt eines Index i auch die Task selbst angegeben werden, wie beispielsweise im Hormon $\mathcal{H}_{T,j}$.

4.2.2. Hormonbasierte Regelkreise

Mithilfe dieser Nomenklatur soll nun die Funktionsweise der hormonbasierten Regelkreise erläutert werden, die in Abbildung 4.2 schematisch dargestellt sind.

Besagte Abbildung zeigt die auf einem PE_j ausgeführten Regelkreise: PE_j besitzt für jede Task T_i aus der Menge $\mathcal{T}$ aller im System vorhandener Tasks einen statischen Eignungswert $E_{i,j}$, der die Eignung von PE_j für T_i beschreibt. Zu diesem Wert werden alle für T_i empfangenen Acceleratoren $A^{i,j}$ addiert, während alle für T_i empfangenen Suppressoren $S^{i,j}$ subtrahiert werden. Das Ergebnis ist der modifizierte Eignungswert $Em_{i,j}$. Dieser wird zum einen an alle übrigen PEs gesendet und zum anderen mit allen von diesen empfangenen modifizierten Eignungswerten $Em^{i,j}$ verglichen. Ist der gesendete modifizierte Eignungswert positiv und größer als alle empfangenen modifizierten Eignungswerte, so ist PE_j das momentan bestgeeignete PE für die Ausführung von T_i und darf die Task übernehmen. Bei Gleichstand zwischen mehreren PEs wird ein zweites Kriterium herangezogen, um eine eindeutige Entscheidung zu erhalten; in den derzeitigen Implementierungen des AHS beispielsweise gewinnt in einem solchen Fall das PE mit dem kleinsten Index.

Neben den modifizierten Eignungswerten, die jedes PE für jede Task sendet, schüttet ein PE für alle auf ihm ausgeführten Tasks Suppressoren und Acceleratoren aus.

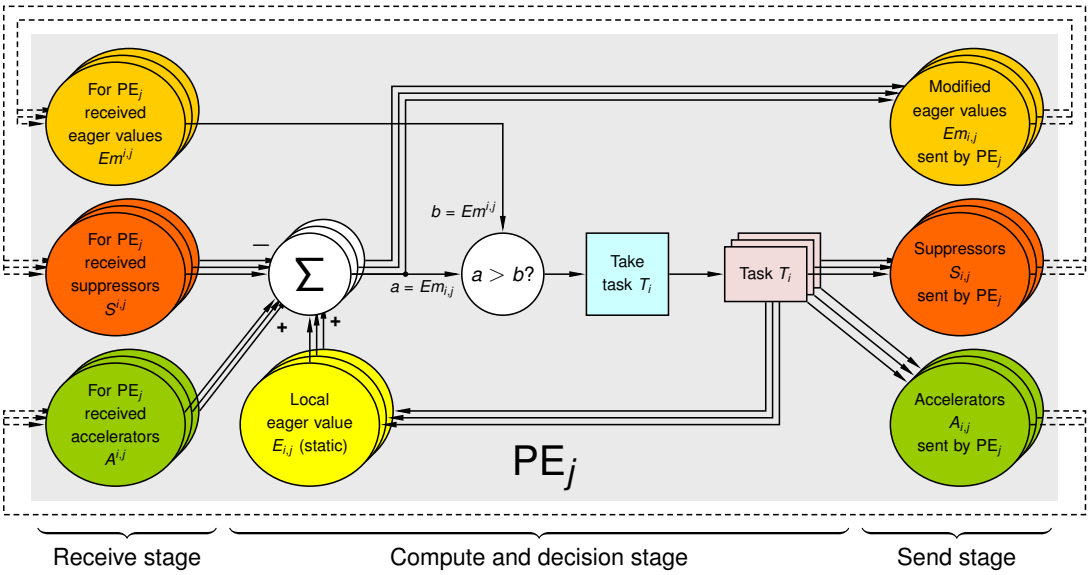


Abbildung 4.2: Hormonbasierte Regelkreise nach [HBF20] mit angepasster Nomenklatur

Suppressoren werden hierbei global an alle PEs gesendet und verhindern, dass eine Task T_i unbegrenzt oft zugeteilt wird: Mit Erreichen einer bestimmten Anzahl im gesamten System zugeteilter Instanzen von T_i werden die modifizierten Eignungswerte für T_i auf allen PEs durch die resultierenden Suppressoren auf Null reduziert, was die Übernahme weiterer Instanzen von T_i verhindert.

Im Gegensatz zu den global gesendeten Suppressoren werden Acceleratoren vom ausführenden PE nur an sich selbst sowie an im Kommunikationsnetz *benachbarte* PEs gesendet, um deren modifizierte Eignungswerte für Tasks zu erhöhen, die als mit T_i *verwandt* gekennzeichnet sind. Dies sind im Regelfall alle Tasks, mit denen T_i häufig Nachrichten austauscht. Als Resultat werden diese Tasks bevorzugt in der Nachbarschaft von PE$_j$ zugeteilt, was zur Bildung von Clustern verwandter Tasks, sogenannter *virtueller Organe*, führt und die Kommunikationsdistanzen zwischen häufig miteinander kommunizierenden Tasks damit gering hält.

Hormonzyklus, Zyklus, Hormonzykluszeit

Jeder Durchlauf dieser dezentralen Regelkreise wird als *Hormonzyklus* oder kurz *Zyklus* bezeichnet. Die Dauer eines Hormonzyklus, die auch *Hormonzykluszeit* genannt wird, ist eine Konstante und nur nach unten hin unter anderem durch die Eigenschaften des verwendeten Kommunikationsmediums beschränkt, für genauere Informationen sei an dieser Stelle auf [vRBP11b] verwiesen.

Es sei jedoch insbesondere darauf hingewiesen, dass jedes PE in jedem Hormonzyklus höchstens eine Taskinstanz übernimmt: Könnten mehrere Tasks pro PE und Zyklus übernommen werden, so könnten bereits alle Tasks vergeben sein, bevor die gesendeten Acceleratoren ihre Wirkung entfalten konnten. Als Resultat würde das Ziel nicht erreicht, Cluster verwandter Tasks auf benachbarten PEs zu bilden.

Tabelle 4.1: Untertypen von Suppressoren

Symbol und Bezeichnung		Beschreibung
Sa	Übernahmesuppressor (*acquisition suppressor*)	Wird an alle PEs gesendet und beschränkt die Anzahl der Instanzen, die von einer Task T im gesamten System maximal zugeteilt werden können.
Sl	Lastsuppressor (*load suppressor*)	Wird von einem PE an sich selbst gesendet und modelliert die Auslastung, die eine Task auf dem PE hervorruft. Als Resultat beschränkt dieser Suppressor die Anzahl Tasks, die das PE maximal ausführen kann.
Sm	Monitoring-Suppressor (*monitoring suppressor*)	Wird von einem PE an sich selbst gesendet und modelliert beispielsweise Szenarien wie eine hohe Temperatur oder ein geringes Energiebudget, um zusätzliche Taskübernahmen auf diesem PE zu verhindern.

4.2.3. Spezielle Hormontypen

Die bisherige Beschreibung des AHS lieferte nur einen Überblick über dessen grundlegende Konzepte; bei genauerer Betrachtung lassen sich Suppressoren und Acceleratoren weiter unterteilen.

Tabelle 4.1 listet hierzu die wichtigsten Untertypen von Suppressoren auf. Die bisher beschriebene Verwendung von Suppressoren zur Beschränkung der Übernahmehäufigkeit einer bestimmten Task im gesamten System entspricht hierbei den *Übernahmesuppressoren Sa*. Im Gegensatz dazu stehen die von einem PE an sich selbst gesendeten Last- und Monitoring-Suppressoren *Sl* bzw. *Sm*: Erstere modellieren die von übernommenen Tasks hervorgerufene Systemlast und beschränken so die Anzahl von Tasks, die ein einzelnes PE übernehmen kann, während letztere von einer lokalen Monitoring-Instanz verwendet werden können, um degradierende PE-Zustände wie steigende Temperatur oder ein geringes Energiebudget zu kennzeichnen und so die Anzahl zusätzlicher Taskübernahmen auf dem PE zu beschränken.

Tabelle 4.2 gibt einen Überblick über die wichtigsten Untertypen von Acceleratoren. Bisher wurde vor allem der *Organaccelerator Ao* beschrieben, der zur Bildung virtueller Organe beitragen soll, die aus verwandten Tasks bestehen, welche auf benachbarten PEs zugeteilt wurden. Monitoring-Acceleratoren *Am* dagegen wirken antagonistisch zu Monitoring-Suppressoren und können wie diese von einer lokalen Monitoring-Instanz verwendet werden, um Umstände zu modellieren, die sich positiv auf das PE auswirken, wie beispielsweise ein durch hohe Sonneneinstrahlung auf Solarzellen erhöhtes Energiebudget, und somit die Übernahme weiterer Tasks auf dem PE zu begünstigen. Stay-Acceleratoren *As* schließlich treten in Zusammenhang mit dem Offermechanismus des AHS auf, der im Folgenden beschrieben wird.

Tabelle 4.2: Untertypen von Acceleratoren

Symbol und Bezeichnung		Beschreibung
Ao	Organaccelerator (*organ accelerator*)	Wird an alle benachbarten PEs gesendet, um die Zuteilung verwandter Tasks auf diesen zu begünstigen und so zur Bildung virtueller Organe beizutragen.
As	Stay-Accelerator (*stay accelerator*)	Wird von einem PE während einer Offerphase an sich selbst gesendet als Anreiz, eine Task *nicht* auf ein anderes PE zu migrieren, um eine solche Migration nur dann durchzuführen, wenn ein anderes PE *deutlich* besser zur Ausführung der Task geeignet ist.
Am	Monitoring-Accelerator (*monitoring accelerator*)	Wird von einem PE an sich selbst gesendet und stellt den Antagonisten des Monitoring-Suppressors dar. Beispielsweise können Szenarien wie ein großes Energiebudget modelliert werden, um zusätzliche Taskübernahmen zu begünstigen.

4.2.4. Offermechanismus

Das bisher beschriebene Verhalten des AHS umfasst die Verteilung der im System vorhandenen Tasks auf die einzelnen PEs. Sofern die Übernahmesuppressoren stark genug gewählt wurden, um nach (ein- oder mehrfacher) Übernahme einer Task T_i die modifizierten Eignungswerte für T_i auf allen PEs auf Null sinken zu lassen, wird sich ein stabiler Systemzustand einstellen, sobald alle Taskinstanzen im System zugeteilt worden sind und folglich *alle* gesendeten modifizierten Eignungswerte den Wert Null haben.

Um sich dennoch zur Laufzeit flexibel an ändernde Betriebsumstände anpassen zu können, unterstützt das AHS den sogenannten *Offermechanismus*. Hierbei werden bereits zugeteilte Tasks in bestimmten Zeitabständen im Rahmen von *Offerphasen* anderen PEs zur Migration angeboten: Konkret wird ein PE_j, das eine Task T_i ausführt, während einer Offerphase aufhören, den Übernahmesuppressor $Sa_{i,j}$ zu senden und stattdessen wieder einen modifizierten Eignungswert $Em_{i,j} > 0$ senden. Der weggefallene Übernahmesuppressor ermöglicht es anderen PEs nun, ihrerseits einen positiven modifizierten Eignungswert für T_i zu senden. Falls es ein PE_k gibt, dessen modifizierter Eignungswert $Em_{i,k}$ größer ist als alle von anderen PEs für T_i empfangenen modifizierte Eignungswerte, so darf PE_k die Task T_i übernehmen und in Folge einen Übernahmesuppressor $Sa_{i,k} > 0$ senden. Im darauffolgenden Hormonzyklus wird PE_j nun einerseits aufgrund des von PE_k gesendeten modifizierten Eignungswerts bemerken, dass PE_k zum aktuellen Zeitpunkt besser zur Ausführung von T_i geeignet ist. Andererseits wird $Em_{i,j}$ durch den empfangenen Übernahmesuppressor auf Null reduziert

werden, weshalb PE_j die Ausführung von T_i stoppt. Als Resultat wurde T_i von PE_j zu PE_k migriert, da dieses, wie durch seinen modifizierten Eignungswert angezeigt wurde, besser zur Ausführung von T_i geeignet war.

Sendet dagegen kein anderes PE während einer Offerphase für T_i einen modifizierten Eignungswert, der größer als jener von PE_j ist,[1] so wird die Offerphase nach einer festgelegten Zeitspanne beendet, indem PE_j wieder einen Übernahmesuppressor für T_i sendet und so die modifizierten Eignungswerte für T_i auf PE_j sowie allen übrigen PEs wieder auf Null gesenkt werden.

Mögliche Situationen, die dazu führen, dass ein PE_k im Rahmen einer Offerphase besser für eine Task T_i geeignet ist, als das PE_j, auf dem T_i ursprünglich zugeteilt wurde, sind beispielsweise

Veränderte Betriebssituationen:
: Beispielsweise könnte sich PE_j aufgrund seiner Auslastung stark erwärmt haben (was in einem erhöhten Monitoring-Suppressor resultiert), während PE_k aufgrund hoher Sonneneinstrahlung ein höheres Energiebudget besitzt (was durch einen erhöhten Monitoring-Accelerator ausgedrückt wird).

Einfluss virtueller Organe:
: Seit der Zuteilung von T_i auf PE_j können in der Nachbarschaft von PE_k mit T_i verwandte Tasks zugeteilt worden sein (durch erstmalige Zuteilung oder Migration auf ein anderes PE), die Organacceleratoren für T_i auf PE_k aussenden und so die Migration von T_i auf PE_k begünstigen.

Eine Taskmigration ist jedoch immer mit Kosten verbunden: Zum einen ist es im Allgemeinen zumeist nötig, neben der eigentlichen Task auch ihren inneren Zustand zu migrieren, zum anderen zieht der Start der migrierten Task auf dem neuen PE unter Umständen auch eine aufwendige Initialisierung nach sich. Daher ist es sinnvoll, eine Task nur dann von einem PE_j auf ein PE_k zu migrieren, wenn letzteres für die Ausführung von T_i *deutlich* besser geeignet ist.

Ebenfalls sollten Oszillationen im Rahmen des Offermechanismus verhindert werden: Aufgrund der von T_i an sich selbst gesendeten Lastsuppressoren könnte $Em_{i,j} < Em_{i,k}$ gelten, weshalb T_i in einer Offerphase auf PE_k migriert wird. In der nächsten Offerphase könnte aufgrund der nun verlagerten PE-Auslastung jedoch $Em_{i,j} > Em_{i,k}$ gelten, weshalb T_i wieder auf PE_j zurück migriert würde. Dieses Oszillationsschema könnte sich nun in jeder Offerphase fortsetzen.

Aus diesen Gründen wird während einer Offerphase vom anbietenden PE_j ein *Stay-Accelerator* $As_{i,j}$ an sich selbst gesendet. Dieser hebt den modifizierten Eignungswert $Em_{i,j}$ an, weshalb T_i nur dann migriert wird, wenn der von einem anderen PE_k gesendete modifizierte Eignungswert $Em_{i,k}$ mindestens um den Wert von $As_{i,j}$ höher ist als $Em_{i,j}$. Als Resultat werden – sofern der Wert von $As_{i,j}$ groß genug gewählt wurde – zum einen Migrationskosten nur dann in Kauf genommen, wenn die Migration

[1] oder wird ein solcher zwar von einem PE_k gesendet, die Task von diesem jedoch nicht übernommen, weshalb kein Suppressor gesendet wird, der den modifizierten Eignungswert von PE_j auf Null reduziert

eine hinreichend große Verbesserung der Taskverteilungsgüte im System nach sich zieht und zum anderen werden Oszillationen vermieden.

4.2.5. Verhalten bei PE-Ausfällen

Ein wesentlicher Aspekt des AHS ist, dass aufgrund der vollständig dezentralen Regelkreise keine zentrale Instanz existiert, die Einfluss auf die Verteilung der Tasks nimmt. Dies erhöht die Zuverlässigkeit des Gesamtsystems gegenüber PE-Ausfällen: Fällt ein PE aus, so wird es keine Hormone mehr senden, insbesondere also auch keine Übernahmesuppressoren für die auf ihm zugeteilten Tasks. Folglich kann der Ausfall eines PEs durch das Ausbleiben von ihm gesendeter Suppressoren erkannt werden.

Als Konsequenz werden die modifizierten Eignungswerte aller zusammen mit dem PE ausgefallenen Tasks jedoch auf allen anderen PEs ansteigen, weshalb diese die im System nun fehlenden Tasks automatisch erneut zuteilen und den Ausfall des PEs so kompensieren, sofern die PEs die ausgefallenen Tasks prinzipiell ausführen können und ihre Kapazitäten hierzu auch ausreichen.

Solange also ausreichend Kapazitäten verfügbar sind und jede Task prinzipiell auf jedem PE ausgeführt werden kann, ist es daher stets möglich, den Ausfall eines beliebigen PEs durch die verbleibenden PEs zu kompensieren, weshalb kein Single Point of Failure existiert.

4.3. Eigenschaften des AHS

Wie zuvor beschrieben, ist das AHS inspiriert vom biologischen Hormonsystem der Säugetiere und imitiert in einem begrenzten Rahmen dessen Verhalten im Kontext technischer Systeme.

Folglich ist seine Adaption natürlicher Organisationsprinzipien auf technische Systeme als Ansatz zur Komplexitätsbeherrschung im Kontext des Organic Computing zu betrachten: Klassischerweise wäre die Verteilung von Tasks auf die Rechenknoten eines verteilten Systems eine Aufgabe gewesen, die zur Designzeit durchgeführt worden wäre, die resultierende Verteilung wäre zur Laufzeit des Systems entsprechend statisch gewesen. Als Beispiel hierzu sei die bereits in Kapitel 1 erwähnte AUTOSAR-Initiative für automotive Architekturen genannt, von der eine solche statische Zuordnung der Tasks auf die vorhandenen Steuergeräte vorgesehen wird [FB15].

Bei Verwendung des AHS wird die Entscheidung über diese Zuordnung dagegen in die Laufzeit verlagert, was es insbesondere ermöglicht, auf geänderte Betriebsumstände des Systems zu reagieren und unter anderem Ausfälle einzelner Rechenknoten bzw. PEs automatisch zu kompensieren. Auch dieser Aspekt der Verlagerung von Entscheidungen aus der Designzeit in die Laufzeit des Systems ist ein charakteristischer Aspekt des Organic Computing [MT17; TSM17].

Als Konsequenz weisen mithilfe des AHS realisierte Systeme verschiedene Selbst-X-Eigenschaften auf:

Selbstorganisation:

Es nimmt keine zentrale Instanz Einfluss auf die Taskverteilung; diese wird von den PEs selbstständig durch die Realisierung dezentraler Regelkreise durchgeführt.

Selbstkonfiguration:
Das System findet selbstständig eine initiale Taskverteilung, die sich automatisch ergibt, sobald alle Taskinstanzen auf die PEs verteilt wurden. Hierbei können insbesondere die Betriebsumstände des Systems (Temperatur, Energiebudget etc.) beachtet sowie Beziehungen zwischen den Tasks berücksichtigt werden (Bildung virtueller Organe).

Selbstoptimierung:
Die initiale Taskverteilung ist nicht statisch, vielmehr kann sich das System zur Laufzeit durch Taskmigrationen im Rahmen des Offermechanismus selbstständig an geänderte Betriebsumstände anpassen und so stetig verbessern.

Selbstheilung:
Der Ausfall einzelner PEs wird von den verbleibenden PEs selbstständig erkannt und kann ebenfalls selbstständig kompensiert werden, es existiert kein *Single Point of Failure*.

Des Weiteren eignet sich das AHS für den Einsatz in Echtzeitsystemen, da zum einen die durch den Austausch der Hormone erzeugte Kommunikationslast beschränkt ist [BPvR08] und zum anderen harte Zeitschranken für die Selbstkonfigurations- und -optimierungsfähigkeiten des AHS sowie – mit Einschränkungen – auch dessen Selbstheilungsfähigkeiten garantiert werden können. Diese Zeitschranken werden in Kapitel 5 im Detail betrachtet.

Dennoch sei darauf hingewiesen, dass die vom AHS hervorgerufene Kommunikationslast trotz ihrer prinzipiellen Beschränktheit in hinreichend großen Skalierungen von Tasks oder PEs dennoch jene Kapazität übersteigen kann, die vom in der Praxis verwendeten Kommunikationsmedium bereitgestellt wird. Im Rahmen dieser Dissertation wird daher angenommen, dass das verwendete Kommunikationsmedium diese Last stets bewältigen kann, um das AHS in der hier beschriebenen Form einsetzen zu können. Nichtsdestotrotz sei erwähnt, dass Varianten des AHS existieren, welche die Kommunikationslast in großen Skalierungen durch Einführung von Hierarchiestufen deutlich verringern; für nähere Informationen hierzu sei auf [Lun19; LB18] verwiesen.

4.4. Verbesserungen des AHS

Im Laufe der Zeit wurden verschiedene Erweiterungen gegenüber dem „klassischen“ AHS, wie es in den vorigen Abschnitten beschrieben wurde, entwickelt, um es in verschiedenen Aspekten zu verbessern. Die im Kontext dieser Dissertation relevanten Erweiterungen werden im Folgenden erläutert.

4.4.1. Acceleratorschranken

Gemäß der vorigen Beschreibung berechnet sich der modifizierte Eignungswert einer Task T_i auf einem PE_j zu

$$Em_{i,j} = E_{i,j} - \sum S^{i,j} + \sum A^{i,j},$$

vom statischen Eignungswert $E_{i,j}$ werden also alle empfangenen Suppressoren subtrahiert und alle empfangenen Acceleratoren addiert.[1]

In dieser Form neigt das AHS jedoch zu Instabilitäten, die durch Acceleratoren hervorgerufen werden: Wenn ein PE beispielsweise voll ausgelastet ist und sein modifizierter Eignungswert durch Lastsuppressoren daher auf Null reduziert wurde, so können von benachbarten PEs gesendete Acceleratoren diesen dennoch wieder auf einen positiven Wert anheben, was zu zusätzlichen Taskübernahmen führen kann. Sind die Acceleratoren zu stark gewählt, kann es gar zu unbegrenzten Taskzuteilungen kommen.

Um diese Probleme zu lösen, wurden in [Bri13] Erweiterungen des AHS vorgeschlagen, welche die Berechnungsvorschrift des modifizierten Eignungswerts verändern, sodass Acceleratoren zwar ihre Fähigkeit zur Bildung virtueller Organe beibehalten, die Systemstabilität jedoch nicht mehr gefährden können.

Für diese Dissertation relevant ist die Methode der *Acceleratorschranken*. Hierbei werden Acceleratoren nur dann in die Berechnung des modifizierten Eignungswerts einbezogen, wenn dieser bereits ohne deren Einfluss positiv ist:

$$Em_{i,j} = \begin{cases} E_{i,j} - \sum S^{i,j}, & E_{i,j} \leq \sum S^{i,j} \\ E_{i,j} - \sum S^{i,j} + \sum A^{i,j}, & E_{i,j} > \sum S^{i,j}. \end{cases}$$

Als Konsequenz können Acceleratoren einen modifizierten Eignungswert $Em_{i,j} = 0$ nicht mehr auf einen positiven (und damit zur Übernahme von T_i qualifizierenden) Wert anheben. Insbesondere ist die Anzahl der Taskübernahmen auf einem PE dann nur abhängig vom Verhältnis aus statischen Eignungswerten und Suppressoren und die Stabilität des Systems kann garantiert werden, sofern die Suppressoren größer als Null sind; für Details sei auf [Bri13] verwiesen.

Bei Verwendung von Acceleratorschranken lässt sich ebenfalls berechnen, wie oft eine Task T_i im gesamten System höchstens zugeteilt werden kann:

Hierzu sei

$$E_{\max,i} \coloneqq \max_j \{E_{i,j}\} \qquad \text{der größtmögliche statische Eignungswert für } T_i \text{ und}$$

$$S^i_{\min} \coloneqq \min_j \{S_{i,j}\} \qquad \text{der kleinstmögliche Suppressor für } T_i.$$

[1] Für die Berechnung des modifizierten Eignungswerts wird jedoch stets eine auf Null gesättigte Subtraktion verwendet, sodass $Em_{i,j} \geq 0$ gilt.

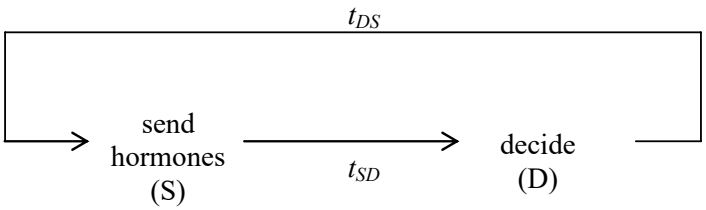


Abbildung 4.3: Schema eines Hormonzyklus [vRBP11a]

Ferner gebe $\mathcal{T}(T_i)$ an, wie oft T_i im gesamten System zugeteilt werden kann. Dann gilt

$$\mathcal{T}(T_i) \leq \left\lceil \frac{E_{\max,i}}{S^i_{\min}} \right\rceil,$$

für Details zur Herleitung dieser Schranke sei erneut auf [Bri13] verwiesen.

4.4.2. Leichtgewichtige Hormonauswertung

Eine weitere für diese Dissertation relevante Verbesserung betrifft die sogenannte *leichtgewichtige* Implementierung des AHS.

Um diese zu verstehen, wird im Folgenden jedoch zunächst die ursprüngliche *asynchrone* Implementierung des AHS kurz umrissen.

4.4.2.1. Asynchrone Implementierung

Während Abbildung 4.2 den Ablauf eines Hormonzyklus in die Phasen *Receive*, *Compute and decide* und *Send* unterteilte, bietet es sich für eine reale Implementierung an, Hormone in einem Hintergrundthread asynchron zum restlichen Hormonzyklus zu empfangen. In Konsequenz lässt sich eine Implementierung dann, wie in Abbildung 4.3 gezeigt, in die beiden aufeinander folgenden Phasen *Send* (S) und *Decide* (D) unterteilen.

Die einzelnen Phasen werden durch die beiden Verzögerungen t_{SD} und t_{DS} voneinander getrennt, welche die Länge des Hormonzyklus bestimmen. Die Dauer dieser Verzögerungen unterliegt gewissen Beschränkungen, die im Wesentlichen vom verwendeten Kommunikationsmedium abhängig sind. Für genaue Analysen diesbezüglich sei auf [BPvR08] verwiesen.

Implementiert man das AHS auf diese Weise, so laufen die Hormonzyklen der einzelnen PEs asynchron voneinander ab. Entsprechend werden die Hormonzyklen verschiedener PEs daher leicht unterschiedliche Dauern aufweisen, was zu Synchronitätsproblemen führen kann: Bei ungünstiger Phasenlage der Zyklen zweier PEs zueinander kann insbesondere der in Abbildung 4.4 gezeigte Fall eintreten. Hier existieren zwei PEs PE_i und PE_j, wobei letzteres eine minimal längere Hormonzykluszeit als PE_i aufweist. Als Resultat der Phasenlage empfängt PE_i im dargestellten Hormonzyklus *keine* Hormone von PE_j, während PE_j in einem einzigen Hormonzyklus *zweimal* Hormone von PE_i empfängt.

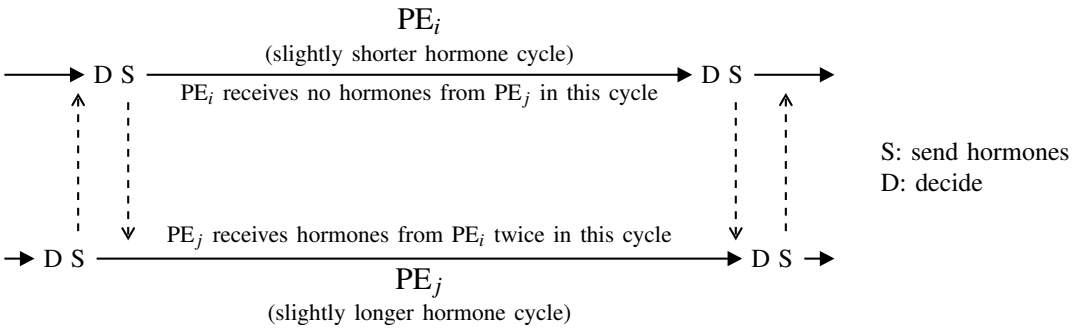


Abbildung 4.4: Synchronitätsproblem bei abweichender Hormonzykluszeit nach [vRBP11a] mit angepasster Nomenklatur

Damit dieser Effekt nicht zu Problemen führt, ist es nötig, für jede Task das zuletzt empfangene Hormon jedes Typs von jedem PE zu speichern, um doppelt empfangene Hormone zu bemerken und bei ausbleibenden Hormonen auf den zuletzt empfangenen Wert zurückzufallen. Insbesondere für eingebettete Systeme mit wenig Ressourcen ist die hierfür nötige Speichermenge jedoch nicht zu vernachlässigen.

4.4.2.2. Selbstsynchronisierende Implementierung

Aus diesen Gründen wird in [vRBP11a] eine vereinfachte, *leichtgewichtige* Implementierung des AHS vorgestellt, welche die Hormonzyklen der verschiedenen PEs automatisch synchronisiert.

Hierzu wird zunächst die Verzögerung zwischen den Phasen *Decide* und *Send* angepasst, wie in Abbildung 4.5 dargestellt: Wird während des Wartens auf den Ablauf der Zeit t_{DS} ein Hormon empfangen, so wird sofort der nächste Hormonzyklus begonnen. Das chronologisch erste PE wird daher den Ablauf der Zeit t_{DS} vollständig abwarten und danach seine Hormone senden. Der Empfang dieser Hormone wird alle chronologisch späteren PEs veranlassen, ebenfalls ihre Hormone zu senden, auch wenn die Zeitspanne t_{DS} auf ihnen noch nicht verstrichen ist. In der Folge synchronisieren sich die Hormonzyklen aller PEs automatisch, das AHS wird *selbstsynchronisierend*.

Diese Selbstsynchronisation hat jedoch auch einen Einfluss auf das Startverhalten jedes einzelnen PEs. Dies ist ebenfalls in Abbildung 4.5 dargestellt: Startet ein PE seinen ersten Hormonzyklus, so wartet es zunächst, bis die Zeitspanne $t_{DS} + t_{SD}$ verstrichen ist oder ein Hormon empfangen wurde. Dies stellt einen korrekt synchronisierten Systemstart sicher.

Des Weiteren muss ein Spezialfall betrachtet werden: Wird ein PE gestartet, während andere PEs gerade ihre Hormone senden, so ist es wahrscheinlich, dass das neue PE einen Teil der gesendeten Hormone verpasst hat. Aus diesem Grund darf es in seinem ersten Zyklus zwar bereits seinerseits Hormone senden, jedoch noch keine Tasks übernehmen. Für weitere Details zu diesen Sonderfällen sei auf [vRBP11a] verwiesen.

Im Vergleich zur asynchronen Implementierung unterliegt die selbstsynchronisierende

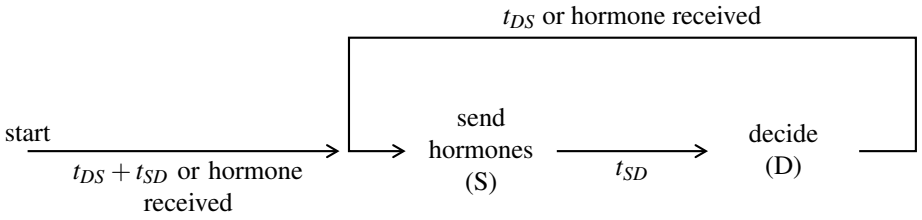


Abbildung 4.5: Schema eines Hormonzyklus mit Selbstsynchronisation nach [vRBP11a], ein im Original vorhandener Druckfehler wurde behoben

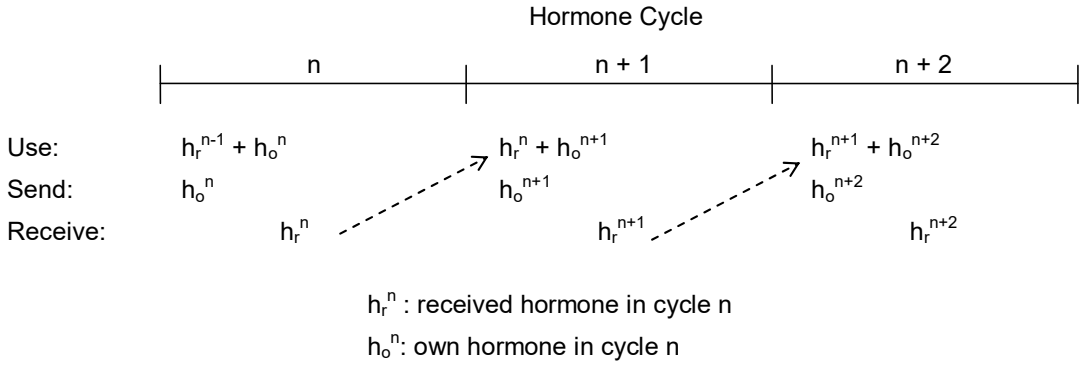


Abbildung 4.6: Zeitverhalten der selbstsynchronisierenden Implementierung [vRBP11a]

Implementierung zudem anderen Einschränkungen bezüglich der Wahl von t_{SD} und t_{DS}, für genaue Betrachtungen sei erneut auf besagte Publikation verwiesen.

Als Resultat des selbstsynchronisierenden Systemverhaltens können die in Abschnitt 4.4.2.1 beschriebenen Synchronitätsprobleme nicht mehr auftreten: In jedem Hormonzyklus werden die Hormone jedes PEs genau einmal empfangen. Daher ist es nicht mehr nötig, für jede Task alle Hormone jedes Typs von jedem PE vorzuhalten.

Vielmehr reicht es aus, für jede Task

- den höchsten empfangenen modifizierten Eignungswert sowie
- die Summe der empfangenen Suppressoren und Acceleratoren

jeweils für den aktuellen sowie den *vorherigen* Hormonzyklus vorzuhalten.

Konkret werden zur Berechnung des modifizierten Eignungswertes in Hormonzyklus n die eigenen Suppressoren und Acceleratoren aus Hormonzyklus n sowie die von anderen PEs empfangenen Hormone aus dem vergangenen Hormonzyklus $n-1$ verwendet, wie in Abbildung 4.6 skizziert. Anschaulich bedeutet dies, dass Suppressoren und Acceleratoren erst mit einem Zyklus Verzögerung wirken, in Zyklus n von PE_i gesendete Suppressoren bzw. Acceleratoren sich also erst in Zyklus $n+1$ in den von anderen PEs gesendeten empfangenen modifizierten Eignungswerten bemerkbar machen.

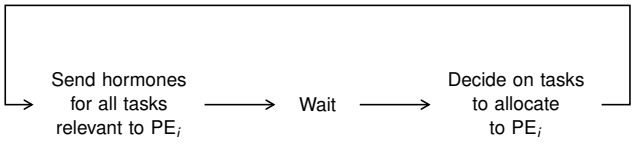


Abbildung 4.7: Vereinfachtes Schema eines Hormonzyklus nach [BP12] mit angepasster Nomenklatur

Der Grund hierfür ist, dass zum Zeitpunkt der Berechnung des modifizierten Eignungswerts die relevanten Suppressoren und Acceleratoren der anderen PEs aus demselben Zyklus noch nicht empfangen werden konnten. Die lokalen Suppressoren und Acceleratoren liegen dagegen bereits vor, weshalb sie problemlos verwendet werden können; die Verwendung der lokalen Hormone aus dem vorherigen Hormonzyklus würde zudem Verzögerungen der Taskverteilung verursachen, für Details sei auf [vRBP11a] verwiesen.[1]

Mithilfe dieser selbstsynchronisierenden oder *leichtgewichtigen* Hormonauswertung muss pro Task folglich nur noch eine *konstante* Anzahl *aggregierter* Hormone vorgehalten werden, was den Speicherbedarf des AHS pro Task deutlich senkt: Zuvor war dieser für Eignungswerte und Suppressoren linear in der Anzahl der PEs und für Acceleratoren linear im Produkt aus den Anzahlen benachbarter PEs und verwandter Tasks [vRBP11a]. Des Weiteren erleichtert das selbstsynchronisierende Verhalten viele Analysen, weshalb in dieser Dissertation stets die leichtgewichtige Hormonauswertung vorausgesetzt wird.

4.4.3. Aggressive Task-Decision-Strategie

Task-Decision-Strategie

Wie in Abschnitt 4.2.2 erläutert, darf jedes PE pro Hormonzyklus höchstens eine Task übernehmen. Die Entscheidung, *welche* Task in einem Hormonzyklus übernommen werden soll, wird von einer *Task-Decision-Strategie* getroffen. Abbildung 4.7 zeigt ein im Vergleich zu früheren Abbildungen vereinfachtes Schema eines Hormonzyklus; der Abschnitt „Decide on tasks to allocate to PE$_i$“ entspricht dabei gerade dem Aufgabenbereich der konkret verwendeten Task-Decision-Strategie. Deren Verhalten nimmt dabei auch direkt Einfluss darauf, wie schnell die einzelnen Tasks im System vergeben werden können und damit auf die garantierbaren Zeitschranken.

Bei Nutzung der im AHS ursprünglich implementierten Task-Decision-Strategie überprüft jedes PE in jedem Hormonzyklus für höchstens *eine* Task, ob es diese gewonnen hat. Ist dies nicht der Fall, übernimmt es in diesem Zyklus *keine* Task. Als Resultat lässt sich zeigen, dass die Selbstkonfiguration bei m zuzuteilenden Tasks nach höchstens m^2 Hormonzyklen beendet ist (bzw. $2m - 1$ Hormonzyklen bei Verwendung einer leicht modifizierten Task-Decision-Strategie); für nähere Informationen sei auf [BPvR08] verwiesen.

[1] Wie wir in Abschnitt 4.4.3 und insbesondere in Kapitel 5 sehen werden, führt die Verwendung *fremder* Hormone aus dem vorherigen Hormonzyklus unter bestimmten Voraussetzungen jedoch trotzdem zu Verzögerungen der Taskzuteilung.

Im Gegensatz dazu überprüft jedes PE bei Verwendung der in [BP12] eingeführten Task-Decision-Strategie in jedem Zyklus sukzessive *alle* Tasks, bis es entweder eine Task gefunden hat, die von ihm gewonnen wurde und daher übernommen werden darf, oder alle Tasks erfolglos durchlaufen wurden. Die daher als *aggressiv* bezeichnete Strategie bildet die Grundlage für die in Kapitel 6 vorgestellte *prioritätsbasierte* Task-Decision-Strategie, weshalb sie im Folgenden genauer betrachtet werden soll.

4.4.3.1. Funktionsweise

Abbildung 4.8 gibt einen Überblick über die Funktionsweise der Strategie, die im Folgenden genauer beschrieben wird.

Auf jedem PE sind alle $|\text{supp}(\mathcal{T})| := m$ verschiedenen Tasks intern in einer Liste abgelegt:[1]

$$T_1, T_2, \ldots, T_m, \bot;$$

hierbei markiert $\bot$ das Ende der Liste.

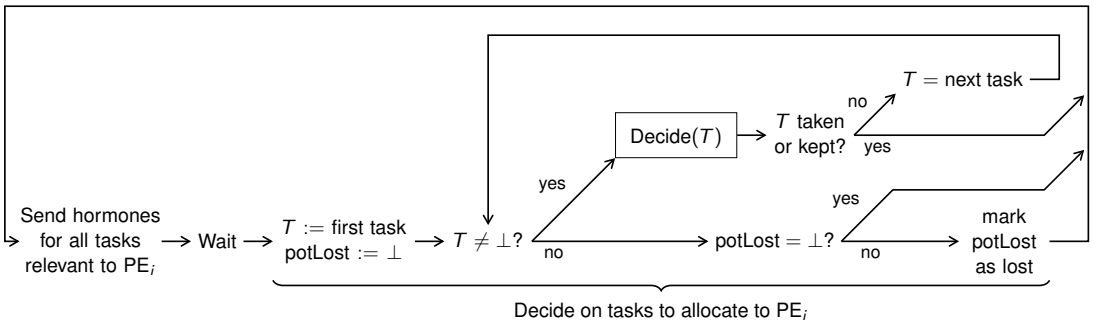
Nachdem ein PE für die auf ihm relevanten Tasks Hormone ausgesendet sowie auf das Eintreffen von Hormonen anderer PEs gewartet hat, beginnt es damit, aktiv nach einer gewonnenen Task zu suchen, wie in Abbildung 4.8a dargestellt. Hierzu werden zunächst zwei Initialisierungen vorgenommen: Die Variable T referenziert die gerade betrachtete Task, die mit der ersten Task der Liste initialisiert wird. Die Variable potLost dagegen wird mit $\bot$ initialisiert und kann am Ende des Hormonzyklus unter Umständen auf eine Task zeigen, die im Rahmen einer Offerphase an ein anderes PE verloren wurde.

Solange noch nicht das Ende der Taskliste erreicht wurde, entscheidet die Unterprozedur Decide(T) über die Übernahme von T. Wurde T dabei übernommen oder im Rahmen eines Offers behalten, da das PE weiterhin das bestgeeignete für T ist, so ist der Hormonzyklus beendet, anderenfalls wird mit der nächsten Task fortgefahren.

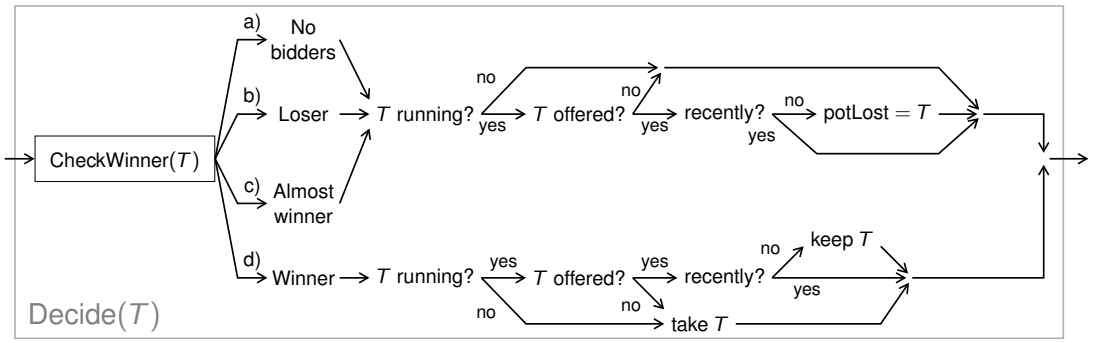
Wurden alle Tasks betrachtet, ohne eine Task zu übernehmen oder im Rahmen eines Offers zu behalten, so wird geprüft, ob potLost $= \bot$ gilt. Ist dies der Fall, so wurde in diesem Zyklus keine Task im Rahmen eines Offers an ein anderes PE verloren und der Hormonzyklus ist gleichfalls beendet. Zeigt potLost dagegen auf eine Task T_i, so wurde T_i an ein anderes PE_k verloren. In diesem Fall wird T_i mit einer *Lose-Pending*-Markierung versehen. Diese bewirkt, dass T_i gestoppt wird, sobald der modifizierte Eignungswert für T_i in einem der nächsten Hormonzyklen durch von PE_k empfangene Übernahmesuppressoren auf Null fällt. Der hierdurch *verzögerte* Stopp von T_i ist nötig, um sicherzustellen, dass die Task während ihrer Migration auf PE_k ununterbrochen ausgeführt wird.

Schließlich soll nun die Funktionsweise der in Abbildung 4.8b dargestellten Decide-Unterprozedur erläutert werden.

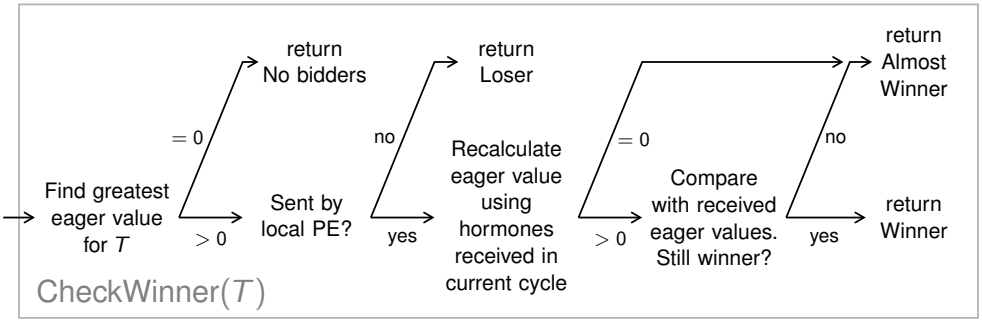
[1] Konkret müssen auf jedem PE nur die $m' \leq m$ für dieses tatsächlich relevanten Tasks betrachtet werden; o.B.d.A. wird im Folgenden angenommen, dass jede Task für jedes PE relevant ist.



(a) Hormonzyklus



(b) Unterprozedur „Decide(T)“



(c) Unterprozedur „CheckWinner(T)“

Abbildung 4.8: Aggressive Task-Decision-Strategie

Zu Beginn dieser werden im Rahmen einer weiteren Unterprozedur CheckWinner (Abbildung 4.8c) alle für die Task T empfangenen modifizierten Eignungswerte mit dem lokalen modifizierten Eignungswert verglichen. Dabei sind vier Fälle zu unterscheiden:

a) **No bidders.** Alle modifizierten Eignungswerte für T sind Null, es hat sich also kein PE um die Ausführung von T beworben.

b) **Loser.** Der größte empfangene modifizierte Eignungswert für T ist größer als Null, wurde jedoch nicht vom lokalen PE_j gesendet, sondern von einem PE_k mit $j \neq k$. Basierend auf den vorliegenden Informationen muss PE_j daher annehmen, dass PE_k die Task T in diesem Hormonzyklus übernehmen *könnte*.

c) **Almost Winner.** Dieser Spezialfall kann eintreten, weil Suppressoren bei Verwendung der leichtgewichtigen Hormonauswertung erst mit einem Hormonzyklus Verzögerung wirken (siehe Abschnitt 4.4.2.2): Jedes PE sendet seinen modifizierten Eignungswert zu Beginn eines Hormonzyklus und wartet danach auf den Empfang von Hormonen anderer PEs. Hat im vorigen Zyklus ein PE_j eine Task T übernommen, so sendet es im aktuellen Zyklus einen Übernahmesuppressor $Sa_{T,j}$ sowie einen durch diesen verringerten modifizierten Eignungswert $Em_{T,j} \geq 0$.

   Im nächsten Zyklus kann es daher passieren, dass ein PE_k ($j \neq k$) einen modifizierten Eignungswert $Em_{T,k} > 0$ sendet, der es zur Übernahme von T qualifizieren würde, während der erst danach empfangene Übernahmesuppressor $Sa^{T,j}$ das PE aber wieder disqualifiziert. PE_k ist daher in der Regel gerade das PE, welches im *vorigen* Zyklus den *zweithöchsten* modifizierten Eignungswert für T gesendet hatte.

   Wie die Abbildung zeigt, wird der *Almost-Winner*-Zustand eingenommen, wenn das lokale PE mit seinem gesendeten modifizierten Eignungswert (der fremde Suppressoren und Acceleratoren aus dem *vorherigen* Zyklus enthielt) die Task T gewonnen hätte, eine Neuberechnung mit den nun vorliegenden, im aktuellen Zyklus empfangenen Hormonen jedoch einen modifizierten Eignungswert ergibt, der Null oder kleiner als ein in diesem Zyklus von einem anderen PE empfangener Eignungswert ist. Ist der neu berechnete Eignungswert zwar verändert, aber immer noch der höchste unter allen empfangenen Eignungswerten, so wird stattdessen der *Winner*-Zustand eingenommen.

   Es sei angemerkt, dass alle übrigen PEs (auch PE_j) für T aufgrund des empfangenen Eignungswertes $Em_{T,k}$ den *Loser*-Zustand einnehmen, da sie fälschlicherweise davon ausgehen, dass T in diesem Zyklus von PE_k übernommen werden *könnte*.

d) **Winner.** Das lokale PE hat unter allen PEs den höchsten modifizierten Eignungswert für T gesendet und darf T daher in diesem Zyklus übernehmen.

Wie nun in Abbildung 4.8b zu sehen ist, führen die Fälle a) – c) für die aggressive Strategie zu identischem Verhalten. Sie sind dennoch als getrennte Fälle aufgeführt, da sie zum einen Resultate grundlegend verschiedener Sachverhalte darstellen und für die in Kapitel 6 vorgestellte prioritätsbasierte Task-Decision-Strategie von entscheidender

Bedeutung sind. Zum anderen ist insbesondere der Almost-Winner-Zustand für die Analyse der Zeitschranken der aggressiven Strategie in Kapitel 5 von besonderer Relevanz. Daher wurden alle Fälle zum Zwecke der besseren Vergleichbarkeit auch für die aggressive Strategie getrennt aufgeführt.

Wurde T gewonnen, so müssen einige weitere Fälle unterschieden werden:

- Falls T nicht bereits auf dem lokalen PE zugeteilt ist, so wird die Task übernommen.
- Ist T dagegen zwar bereits auf dem lokalen PE_i zugeteilt, wird aber gerade nicht im Rahmen einer Offerphase anderen PEs angeboten, so wird eine zusätzliche Instanz von T übernommen.
- Ist T auf dem lokalen PE zugeteilt und befindet sich die Task gerade in einer Offerphase, so wird T behalten und die Offerphase beendet, sofern seit dem Beginn der Offerphase eine bestimmte Anzahl Hormonzyklen vergangen ist; andernfalls wird keine Aktion durchgeführt. Dies ist notwendig, um den anderen PEs hinreichend viel Zeit zu geben, auf den weggefallenen Übernahmesuppressor für T mit einer Erhöhung ihrer modifizierten Eignungswerte reagieren zu können.[1]

Wurde T dagegen *nicht* gewonnen, so wird potLost auf T gesetzt, falls T auf dem lokalen PE zugeteilt ist und gerade im Rahmen einer Offerphase anderen PEs angeboten wird: Eines dieser PEs eignet sich zum aktuellen Zeitpunkt besser zur Ausführung von T, was durch den empfangenen modifizierten Eignungswert für T signalisiert wurde. Hierbei wird jedoch ebenfalls darauf geachtet, dass potLost nur dann gesetzt wird, wenn T nicht erst seit Kurzem angeboten wird.

Wie zuvor beschrieben, wird potLost am Ende des Hormonzyklus mit einer *Lose-Pending*-Markierung versehen, welche zu einem verzögerten Stopp der entsprechenden Task führt, sobald aufgrund empfangener Übernahmesuppressoren erkannt wurde, dass die Migration der Task auf ein anderes PE abgeschlossen ist.

Zum Abschluss der Beschreibung der aggressiven Task-Decision-Strategie sei an dieser Stelle auf zwei ihrer Besonderheiten hingewiesen:

- Werden mehrere Tasks gleichzeitig angeboten und an andere PEs verloren, so kann pro Hormonzyklus höchstens eine davon gestoppt werden, da Taskstopps über die potLost-Variable realisiert werden. Wenn mehr als eine potenziell verlorene Task existiert, überschreibt die zuletzt betrachtete alle zuvor betrachteten Tasks, was in einer Verzögerung der Taskstopps resultieren kann.

[1] Mit dem Beginn einer Offerphase wird das anbietende PE_j für T keinen Übernahmesuppressor $Sa_{T,j}$ mehr senden und sich stattdessen mit einem modifizierten Eignungswert $Em_{T,j}$ darum bewerben, T weiterhin ausführen zu dürfen. Da Suppressoren jedoch verzögert wirken, werden alle übrigen PEs im nächsten Hormonzyklus einen modifizierten Eignungswert senden, der $Sa_{T,j}$ noch enthält. Folglich ist es den PEs erst im übernächsten Zyklus möglich, auf den weggefallenen Übernahmesuppressor mit einem erhöhten modifizierten Eignungswert zu reagieren. Als Resultat muss PE_j eine erneute Entscheidung über T herauszögern, bis die übrigen PEs tatsächlich eine Chance hatten, einen erhöhten modifizierten Eignungswert zu senden, um einen fairen Ablauf der Offerphase sicherzustellen.

- Eine angebotene und auf ein anderes PE migrierte Task kann auf ihrem ursprünglichen PE erst gestoppt werden, wenn in einem Hormonzyklus keine andere Task übernommen oder behalten wird, da die potLost-Variable ansonsten nicht überprüft wird.

Diese beiden Besonderheiten können also dazu führen, dass Tasks, die auf ein anderes PE migriert werden, länger als unbedingt nötig auf beiden PEs gleichzeitig ausgeführt werden. Die in Kapitel 6 vorgestellte *prioritätsbasierte* Task-Decision-Strategie wird dieses Verhalten ändern und migrierte Tasks sofort stoppen, sobald sichergestellt ist, dass die Task tatsächlich von ihrem neuen PE gestartet wurde.

Bevor die Zeitschranken, deren Einhaltung bei Verwendung der aggressiven Task-Decision-Strategie garantiert werden kann, in Kapitel 5 detailliert analysiert werden, soll zunächst noch ein kurzer Überblick über die bestehenden Implementierungen des AHS gegeben werden.

4.5. Implementierungen des AHS

Zum aktuellen Zeitpunkt existieren zwei Implementierungen des AHS, die im Folgenden kurz vorgestellt werden sollen.

4.5.1. Hormonsimulator

Der ursprünglich in [BPvR07] vorgestellte Hormonsimulator ist in C++ unter Verwendung der Microsoft Foundation Classes (MFC) geschrieben und simuliert die Taskverteilung des AHS. Jedes PE wird durch einen eigenen Thread simuliert und in der grafischen Oberfläche zusammen mit den auf ihm zugeteilten Tasks visualisiert, wie in Abbildung 4.9 gezeigt. Die auf einem PE zugeteilten Tasks werden hierbei nur simuliert, führen aber keinen Code aus.

Die einzelnen PEs sind in einem zweidimensionalen Gitter angeordnet und werden mit ihrer Gitterposition (x, y) bezeichnet, beispielsweise befindet sich $\mathrm{PE}_{(4,1)}$ in besagter Abbildung oben rechts und $\mathrm{PE}_{(1,4)}$ unten links. Dabei gelten zwei PEs als benachbart, wenn sie im Gitter horizontal, vertikal oder diagonal aneinandergrenzen. Diese Nachbarschaftsverhältnisse sind in Abbildung 4.10 für ein 3×3-Gitter visualisiert: Beispielsweise ist $\mathrm{PE}_{(2,2)}$ mit allen umliegenden PEs benachbart.

Die Bildung virtueller Organe als Resultat dieser Nachbarschaftsverhältnisse lässt sich in Abbildung 4.9 ebenfalls erkennen: Es existieren zwanzig Tasks, von denen je fünf miteinander verwandt sind. Jedes Cluster aus verwandten Tasks wurde aufgrund der an benachbarte PEs ausgesendeten Acceleratoren auf je vier PEs verteilt.

4.5.2. AHS-Middleware

Neben dem Hormonsimulator existiert eine Implementierung der AHS-Middleware, die in ANSI-C geschrieben ist und ursprünglich in [vRB10] vorgestellt wurde. Im Gegensatz zum Simulator führt die Middleware den Programmcode jeder von ihr übernommenen

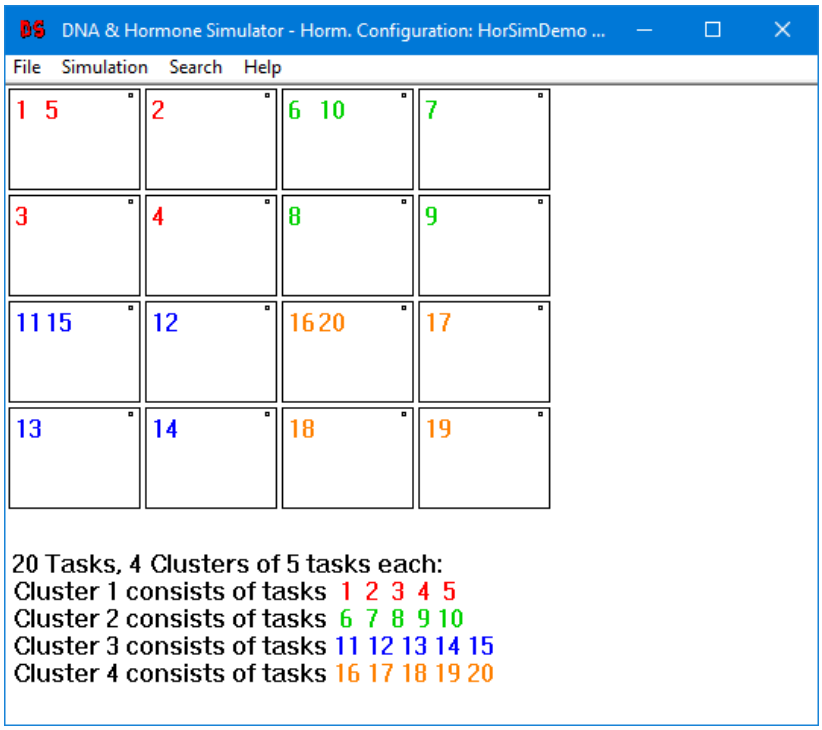


Abbildung 4.9: Hormonsimulator

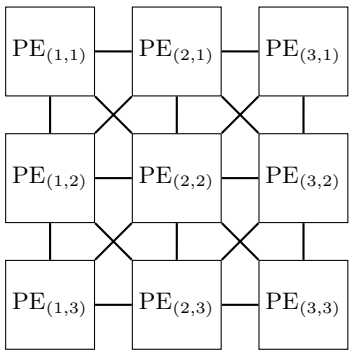


Abbildung 4.10: Nachbarschaftsverhältnisse für ein 3×3-Gitter im Hormonsimulator: Benachbarte PEs sind jeweils durch eine Linie miteinander verbunden.

Task in einem eigenen Thread aus, sodass damit reale Systeme aufgebaut werden können.

Die Middleware ist hierbei als statische Bibliothek konzipiert. Das zugrunde liegende Betriebssystem sowie das verwendete Kommunikationsmedium werden nur durch Abstraktionsschichten angesprochen, was die Portabilität der Implementierung erhöht und es so ermöglicht, die Middleware auf verschiedene Plattformen zu portieren – von Mikrocontrollern bis hin zu Desktopcomputern. Zum aktuellen Zeitpunkt existieren Portierungen der Middleware auf

- Microsoft Windows unter Verwendung von UDP/IP [vRB10; vRBP11b; Bri16] und CAN (Controller Area Network),
- Linux unter Verwendung von UDP/IP [Bri16], CAN sowie CAN FD (CAN with Flexible Data-Rate) [BHF19; HBF20],
- Apple macOS unter Verwendung von UDP/IP, CAN sowie CAN FD,
- ein minimales Echtzeitbetriebssystem, das auf einem typischen Automotive-Mikrocontroller mit 40 kB RAM läuft und CAN zur Kommunikation nutzt [BF19],
- die AUTOSAR Classic Platform auf einem Automotive-Mikrocontroller mit 128 kB RAM unter Verwendung von CAN [BHF19; HBF20].

Jede auf einem beliebigen Rechenknoten ausgeführte Instanz der Middleware stellt ein PE des AHS dar. Zu Testzwecken kann es insbesondere hilfreich sein, auf einem einzelnen PC mehrere Instanzen der Middleware (und somit PEs) auszuführen. Im Gegensatz zur vorgegebenen Gittertopologie des Hormonsimulators können mithilfe der Middleware beliebige PEs als benachbart definiert und so beliebige Netzwerktopologien umgesetzt werden. In der Folge werden zur Kennzeichnung der einzelnen PEs innerhalb der Middleware ganzzahlige Indizes verwendet.

4.5.3. Verwendung im Rahmen der Evaluationen

Alle im Rahmen dieser Dissertation entwickelten Erweiterungen des AHS wurden sowohl im Simulator als auch in der Middleware implementiert und getestet. Hierbei zeigte sich stets identisches Verhalten. Sofern in einzelnen Evaluationen nicht explizit auf die Verwendung der Middleware verwiesen wird, wurden diese im Simulator durchgeführt. Dies hat vor allem praktische Gründe: Mithilfe zeitgesteuerter Ereignisse können Experimente innerhalb des Hormonsimulators automatisch gesteuert werden, was mit der Middleware nicht möglich ist. Ferner ermöglicht es der Simulator, eine bestimmte initiale Taskverteilung zu fixieren. Dies überspringt die Selbstkonfiguration des Systems und erleichtert es so insbesondere, Worst-Case-Szenarien zu evaluieren. In der Middleware wird dagegen stets eine initiale Selbstkonfiguration durchgeführt, die jedoch nicht notwendigerweise in der für die Evaluation gewünschten Worst-Case-Taskverteilung resultiert.

5 Neue Zeitschranken für die aggressive Task-Decision-Strategie

Im vorigen Kapitel wurde mehrfach erwähnt, dass das AHS für den Einsatz in Echtzeitsystemen geeignet ist, da es die Einhaltung harter Zeitschranken garantieren kann, ohne diese jedoch konkret anzugeben.

Dieses Kapitel befasst sich daher detailliert mit diesen Zeitschranken. Insbesondere werden jene Schranken betrachtet, die von der aggressiven Task-Decision-Strategie für die Selbstkonfiguration und -heilung garantiert werden können.

Nach einem Überblick über die bisher publizierten Zeitschranken der Strategie wird gezeigt, dass sie eine Ungenauigkeit enthalten und unter bestimmten Voraussetzungen nicht korrekt sind. Daher werden im Anschluss neue Zeitschranken für die aggressive Task-Decision-Strategie hergeleitet.

Später wird sich zudem zeigen, dass das zugrundeliegende Problem, das in den Herleitungen der bisherigen Zeitschranken nicht betrachtet wurde, auch für die Analyse der in Kapitel 6 eingeführten *prioritätsbasierten* Task-Decision-Strategie von entscheidender Bedeutung ist.

5.1. Annahmen über das AHS

Zunächst sollen an dieser Stelle jedoch einige allgemeine Annahmen über das AHS getroffen werden, die grundlegend für die Analysen in dieser Dissertation sind und daher von nun an stets vorausgesetzt werden:

Annahme 5.1. Es wird die leichtgewichtige Hormonauswertung verwendet, wie sie in Abschnitt 4.4.2 beschrieben wurde.

Als Konsequenz kann daher insbesondere angenommen werden, dass sich die Hormonzyklen aller PEs automatisch synchronisieren.

Annahme 5.2. Es kommen Acceleratorschranken zum Einsatz, wie in Abschnitt 4.4.1 beschrieben.

Diese Annahme hat zum einen den ebenfalls in Abschnitt 4.4.1 beschriebenen Effekt der Stabilisierung des Systems zufolge, da Acceleratorschranken effektiv die Häufigkeit der Übernahme jeder einzelnen Task beschränken.

Zusätzlich ist jedoch ein anderer Effekt von Bedeutung, der essenziell für viele der folgenden Betrachtungen ist: Kommen *keine* Acceleratorschranken zum Einsatz, so kann es Tasks geben, die nur dann zugeteilt werden können, wenn mit ihnen verwandte Tasks bereits zugeteilt sind und Acceleratoren an benachbarte PEs senden. Dies wird in folgendem Beispiel illustriert:

Beispiel 5.3

Es existieren zwei Tasks T_1 und T_2, die miteinander verwandt sind. T_1 habe auf allen PEs einen statischen Eignungswert $E_1 = 1$, T_2 habe dagegen einen statischen Eignungswert $E_2 = 0$. Ist T_1 auf einem PE_i zugeteilt, werde aufgrund der Verwandtschaft zu T_2 ein Organaccelerator $Ao_{1,i}^{2,\Phi_i} > 0$ für T_2 an alle zu PE_i benachbarte PEs Φ_i gesendet. Zudem kommen *keine* Acceleratorschranken zum Einsatz.

T_2 kann in diesem Szenario nur dann auf einem PE instanziiert werden, wenn T_1 bereits auf einem benachbarten PE zugeteilt ist, da ansonsten kein PE einen modifizierten Eignungswert $Em_2 > 0$ senden kann und $Em_2 = 0$ nicht zur Übernahme von T_2 berechtigt.

Erst der Empfang des Organaccelerators $Ao^{2,j} > 0$ ermöglicht es einem PE_j, einen positiven – und damit zur Übernahme von T_2 qualifizierenden – modifizierten Eignungswert $Em_{2,j} > 0$ für T_2 zu senden.

Hätte auch T_1 einen statischen Eignungswert von Null, so könnten sogar weder T_1 noch T_2 jemals instanziiert werden.

Die Verwendung von Acceleratorschranken schließt die Zuteilung von T_2 in jedem Fall aus: Wie in Kapitel 4.4.1 beschrieben, werden Acceleratoren in diesem Fall nur dann in die Berechnung des modifizierten Eignungswertes Em einbezogen, wenn dieser bereits *ohne* den Einfluss der Acceleratoren positiv ist. Für T_2 würde daher niemals ein $Em_2 > 0$ gesendet, weshalb keine Zuteilung der Task geschehen könnte.

Die Verwendung von Acceleratorschranken stellt daher insbesondere sicher, dass keine Kausalketten unter den Tasks existieren, alle Tasks also unabhängig voneinander zugeteilt werden können.

Die folgende Annahme ist daher eine logische Konsequenz aus dieser Folgerung:

Annahme 5.4. Es existiert keine Task T_i, für deren statischen Eignungswert $E_i = 0$ auf *allen* PEs gilt.

Gibt es eine solche Task, so kann sie aufgrund der Acceleratorschranken niemals zugeteilt werden, weshalb sie ohne Beschränkung der Allgemeinheit als nicht existent betrachtet werden kann.

5.2. Bisherige Zeitschranken

Nun sollen die bisher publizierten Zeitschranken des AHS bei Verwendung der aggressiven Task-Decision-Strategie aufgeführt werden, wie sie in [BP12] beschrieben sind; wir nehmen im Folgenden jedoch vereinfachend an, dass jede im System vorhandene Task von jedem PE ausgeführt werden kann. Die hier gegebenen Beweise der Zeitschranken nutzen dieselben Argumentationsketten wie in besagter Publikation.

5.2.1. Selbstkonfiguration

Die Selbstkonfiguration ist beendet, sobald im System alle Tasks vergeben sind, was der Fall ist, sobald alle modifizierten Eignungswerte durch Übernahmesuppressoren

auf Null gesunken sind. Für die Dauer der Selbstkonfiguration gilt dann:

Satz 5.5. Es bezeichne m die Anzahl der im System zu verteilenden Tasks. Ferner finden währenddessen weder Selbstoptimierung noch Selbstheilung statt und es gebe genug PEs, um alle m Tasks verteilen zu können.
Dann dauert die Selbstkonfiguration höchstens m Hormonzyklen.

Beweis. In jedem Hormonzyklus wird im gesamten System mindestens eine Task zugeteilt, da die aggressive Strategie aktiv auf jedem PE nach einer gewonnenen Task sucht. Da folglich stets ein PE existiert, das den höchsten modifizierten Eignungswert für eine Task gesendet hat, folgt die Aussage. □

5.2.2. Selbstoptimierung

Die Dauer der Selbstoptimierung lässt sich dagegen wie folgt abschätzen:

Satz 5.6. Es bezeichne m die Anzahl der im System vorhandenen Tasks.
Dann dauert die Selbstoptimierung höchstens m Hormonzyklen.

Beweis. Im Worst Case beginnen zur gleichen Zeit Offerphasen für *alle* Tasks. Dann entspricht die Selbstoptimierung jedoch gerade einer Selbstkonfiguration aller m Tasks, woraus die Aussage folgt. □

5.2.3. Selbstheilung

Wie in Kapitel 4.2.5 beschrieben, kann das AHS im Rahmen seiner Selbstheilungsfähigkeiten den Ausfall eines beliebigen PEs kompensieren, solange die Kapazität der verbleibenden PEs ausreicht, um alle betroffenen Tasks zu übernehmen. Die Dauer einer solchen Selbstheilung lässt sich ebenfalls nach oben hin abschätzen, zunächst seien jedoch folgende Definitionen vereinbart:

Definition 5.7. Es bezeichne $\mathrm{PE}_\times$ das ausgefallene PE. **$\mathrm{PE}_\times$**

Definition 5.8. Es bezeichne a die Zeit (gemessen in Hormonzyklen), die *höchstens* benötigt wird, um den Ausfall von $\mathrm{PE}_\times$ zu erkennen. ***a***

Mit diesen Vorkehrungen lässt sich die Dauer einer Selbstheilung nun wie folgt abschätzen:

Satz 5.9. Es bezeichne m_f die Anzahl der Tasks, die auf $\mathrm{PE}_\times$ zugeteilt waren. Ferner haben die verbleibenden PEs genügend Kapazitäten, um alle m_f Tasks zusätzlich zu übernehmen und es finden während der Selbstheilung weder Selbstkonfiguration, Selbstoptimierung noch eine weitere Selbstheilung statt.
Dann dauert die Selbstheilung höchstens $m_f + a$ Hormonzyklen.

Beweis. Nach der Erkennung des Ausfalls von $\mathrm{PE}_\times$ entspricht die Selbstheilung einer Selbstkonfiguration mit m_f Tasks, womit die Aussage folgt. □

Bei genauerer Betrachtung fällt auf, dass die Zeitschranken für Selbstoptimierung und Selbstheilung eng mit der Zeitschranke der Selbstkonfiguration zusammenhängen. Kann also gezeigt werden, dass die Zeitschranke der Selbstkonfiguration unter bestimmten Bedingungen nicht eingehalten wird, so sind die anderen beiden Zeitschranken davon ebenso betroffen. Bevor dies jedoch gezeigt wird, soll im nächsten Abschnitt die Konstante a genauer betrachtet werden.

5.3. Erkennung eines PE-Ausfalls

Die Konstante a, welche die Zeit angibt, die zur Erkennung eines PE-Ausfalls höchstens benötigt wird, wird im Laufe dieser Dissertation noch oft auftreten. Des Weiteren wird an einigen Stellen auch die Zeit $a_{\min}$ benötigt, die wie folgt definiert sei:

$a_{\min}$

Definition 5.10. Es bezeichne $a_{\min}$ die Zeit (gemessen in Hormonzyklen), nach welcher der Ausfall von $\mathrm{PE}_\times$ *frühestens* von anderen PEs erkannt werden kann.

Streng genommen wird ein PE-Ausfall jedoch nicht *explizit* erkannt: Vielmehr führt das plötzliche Fehlen vom ausgefallenen $\mathrm{PE}_\times$ gesendeter Übernahmesuppressoren dazu, dass die modifizierten Eignungswerte der verbleibenden PEs für die betroffenen Tasks auf einen positiven Wert steigen, sofern ihre Kapazitäten hierfür ausreichend sind. Als Resultat werden die ursprünglich auf $\mathrm{PE}_\times$ zugeteilten Tasks nun erneut zugeteilt, was den Ausfall von $\mathrm{PE}_\times$ kompensiert, ohne explizit als solcher erkannt worden zu sein.

Entsprechend kann a insbesondere auch als Zeitspanne betrachtet werden, nach deren Verstreichen ein empfangener Übernahmesuppressor (oder ein beliebiges anderes Hormon) als ungültig angesehen wird. Für die in dieser Dissertation verwendete leichtgewichtige Hormonauswertung lassen sich a und $a_{\min}$ wie folgt beschränken:[1]

Satz 5.11. Es werde die leichtgewichtige Hormonauswertung verwendet, zudem finden zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$ weder Selbstkonfiguration noch Selbstheilung oder Selbstoptimierung statt.

Dann gelten $a_{\min} = 1$ und $a = 2$: Der Ausfall von $\mathrm{PE}_\times$ wird also frühestens nach einem, spätestens jedoch nach zwei Hormonzyklen von den anderen PEs erkannt.

Beweis. Wie zuvor beschrieben, wird der Ausfall von $\mathrm{PE}_\times$ nicht explizit erkannt, sondern äußert sich im Anstieg der modifizierten Eignungswerte der verbleibenden PEs als Resultat des weggefallenen Übernahmesuppressors von $\mathrm{PE}_\times$. Folglich muss bestimmt

[1] Die leichtgewichtige Hormonauswertung speichert nur die höchsten empfangenen Eignungswerte bzw. die Summe der empfangenen Suppressoren und Acceleratoren für jede Task, jeweils für den aktuellen als auch den vorherigen Hormonzyklus. Demgegenüber steht die in Kapitel 4.4.2.1 vorgestellte asynchrone Implementierung, die das zuletzt empfangene Hormon jedes Typs zusammen mit dem zugehörigen Empfangszeitstempel speichert. Folglich kann der Verfallszeitpunkt a der Hormone in diesem Fall beliebig festgesetzt werden, während dies für die leichtgewichtige Hormonauswertung nicht möglich ist: Da empfangene Hormone beim Empfang sofort aggregiert und nicht einzeln gespeichert werden, lassen sich auch keine Empfangszeitstempel speichern, weshalb a hier nicht variiert werden kann.

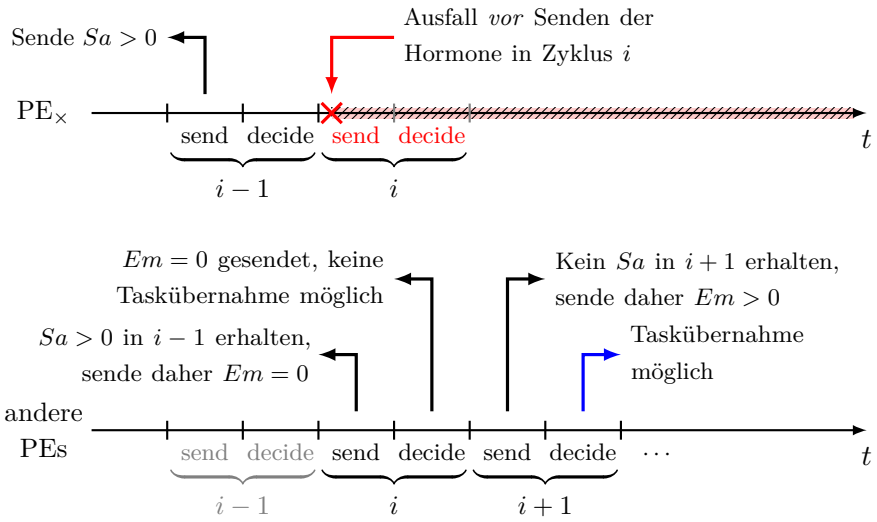
werden, wann die modifizierten Eignungswerte der verbleibenden PEs frühestens bzw. spätestens ansteigen können.

Hierzu falle $\mathrm{PE}_\times$ im i-ten Hormonzyklus aus. Dann können zwei Fälle unterschieden werden:

a) $\mathrm{PE}_\times$ fällt aus, *bevor* es seine Hormone in Zyklus i senden konnte.

In diesem Fall senden die verbleibenden PEs in Zyklus i aufgrund der in Zyklus $i-1$ empfangenen und verzögert wirkenden Suppressoren modifizierte Eignungswerte $Em = 0$, die nicht zur Übernahme der ausgefallenen Tasks qualifizieren; in der Decide-Phase wird jedoch der Ausfall von $\mathrm{PE}_\times$ bemerkt, da in Zyklus i kein Übernahmesuppressor von $\mathrm{PE}_\times$ empfangen wurde. In Zyklus $i+1$ können die PEs nun einen modifizierten Eignungswert $Em > 0$ senden, folglich ist in diesem Zyklus die erste Taskübernahme möglich. Die Erkennung des Ausfalls von $\mathrm{PE}_\times$ dauerte mit Zyklus i also genau einen Hormonzyklus.

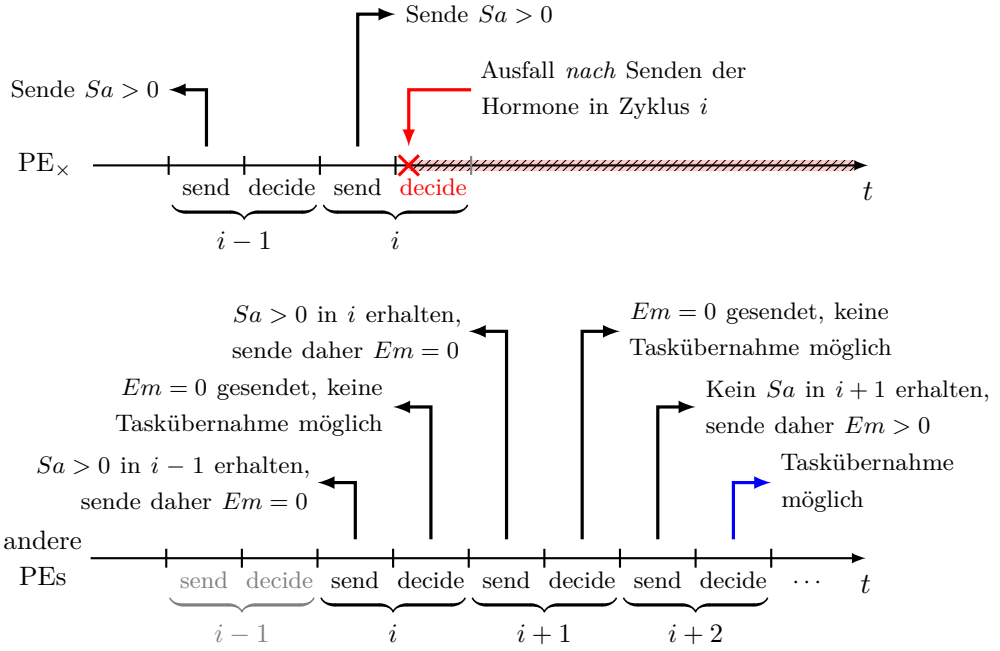
Dies ist in folgender Grafik skizziert (der Übersicht halber sind die Zeiten t_{SD} und t_{DS} nicht dargestellt):



b) $\mathrm{PE}_\times$ fällt aus, *nachdem* es seine Hormone in Zyklus i gesendet hat.

Das Verhalten der verbleibenden PEs in Zyklus i ist ähnlich zu Fall a), es werden jedoch Suppressoren von $\mathrm{PE}_\times$ empfangen. Daher wird auch in Zyklus $i+1$ ein modifizierter Eignungswert $Em = 0$ gesendet; in der anschließenden Decide-Phase wird jedoch der Ausfall von $\mathrm{PE}_\times$ bemerkt, da in diesem Zyklus keine Übernahmesuppressoren von $\mathrm{PE}_\times$ empfangen wurden. Daher ist Zyklus $i+2$ der erste Zyklus, in dem positive modifizierte Eignungswerte gesendet werden und folglich Tasks neu zugeteilt werden können. Die Erkennung des Ausfalls von $\mathrm{PE}_\times$ dauerte mit den Zyklen i und $i+1$ also zwei Hormonzyklen.

Dies ist ähnlich zu Fall a) in folgender Grafik skizziert:



Offensichtlich stellt Fall a) mit einem Hormonzyklus den Best Case und b) mit zwei Hormonzyklen den Worst Case dar, woraus die Aussage folgt. □

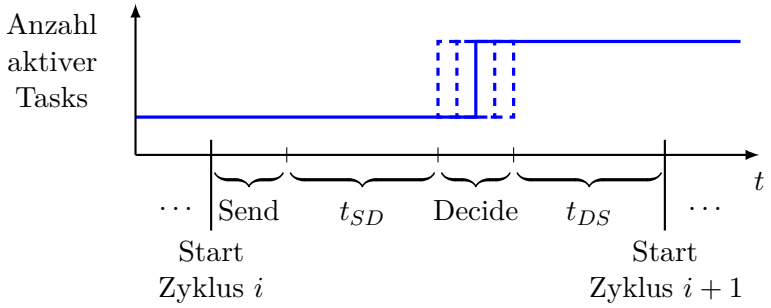
Obwohl für die Konstanten a und $a_{\min}$ nun konkrete Werte bestimmt werden konnten, werden diese im Verlauf dieser Dissertation dennoch meistens als symbolische Konstanten aufgefasst, um zu betonen, dass die Resultate sich auch auf andere Implementierungen des AHS-Hormonzyklus als die leichtgewichtige Hormonauswertung übertragen lassen.

5.4. Taskdiagramme und deren Semantik

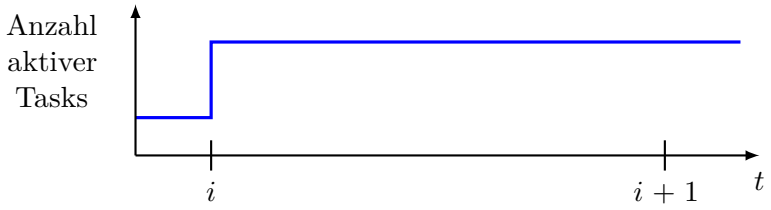
Im Folgenden soll zunächst experimentell gezeigt werden, dass die zuvor wiedergegebenen Zeitschranken der Selbstkonfiguration – und damit verbunden auch der Selbstoptimierung und Selbstheilung – unter bestimmten Voraussetzungen nicht korrekt sind.

In Vorbereitung auf diese Experimente sollen in diesem Abschnitt daher Diagramme eingeführt werden, welche den zeitlichen Verlauf der Anzahl im System aktiver Tasks darstellen sowie deren Semantik erläutert werden.

Möchte man den zeitlichen Verlauf der im System aktiven Tasks grafisch aufbereiten, so stellt sich das Problem, dass Hormonzyklen nicht atomar sind: Jeder Hormonzyklus besteht, wie in Kapitel 4.4.2 beschrieben, aus Sende- und Entscheidungsphase sowie den Wartezeiten t_{SD} zwischen Senden und Entscheiden und t_{DS} zwischen Entscheiden und Senden; das Empfangen läuft asynchron im Hintergrund ab. Dies gilt sowohl für den Hormonsimulator als auch für die Implementierung des AHS als Middleware.



(a) Taskübernahme zu einem beliebigen Zeitpunkt während der Decide-Phase des i-ten Hormonzyklus



(b) Taskübernahme nach (a) zum diskreten Zeitpunkt $t = i$

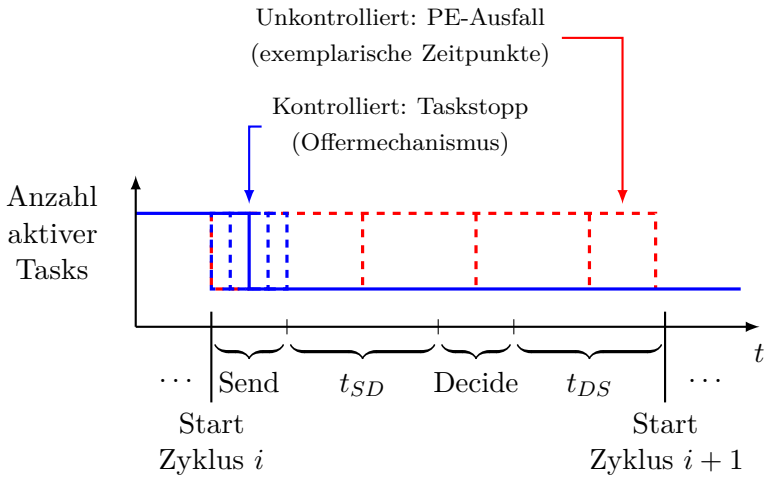
Abbildung 5.1: Darstellung einer Taskübernahme zu diskreten Zeitpunkten

Eine Taskübernahme kann nur während der Entscheidungsphase stattfinden; der exakte Zeitpunkt innerhalb dieser Phase kann jedoch im Allgemeinen nicht genauer spezifiziert werden. In Abbildung 5.1a ist eine solche Taskübernahme skizziert. Für die Darstellung in dieser Dissertation werden Taskübernahmen wie in Abbildung 5.1b dargestellt: Hormonzyklen werden zum Zwecke der Darstellung als atomare Einheiten aufgefasst; wird während des i-ten Hormonzyklus eine Task übernommen, so wird diese Übernahme zum diskreten Zeitpunkt $t = i$ dargestellt.

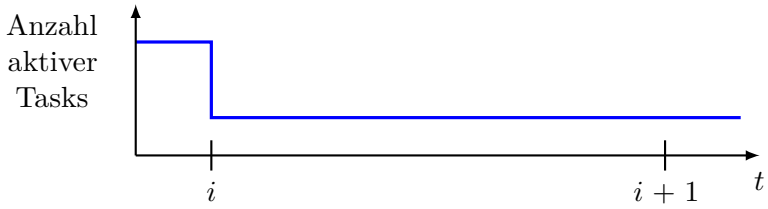
Im Gegensatz zu Taskübernahmen können Taskverluste aus mehreren Gründen geschehen:

- Kontrollierte Taskverluste geschehen vor allem durch Taskmigrationen im Rahmen des Offermechanismus. Der Stopp einer Task, die auf ein anderes PE migriert wurde, erfolgt dabei zu einem im Allgemeinen nicht genauer spezifizierbaren Zeitpunkt während der *Sendephase*: Wie in Kapitel 4.4.3 beschrieben, werden durch Migration verlorene Tasks erst dann gestoppt, wenn der modifizierte Eignungswert des bisherigen PEs durch vom neuen PE empfangene Übernahmesuppressoren auf Null gesenkt wurde. Dies wird von den in Kapitel 4.5 vorgestellten Implementierungen des AHS jedoch in der Sendephase während der Berechnung der neuen modifizierten Eignungswerte bemerkt, weshalb eine auf ein anderes PE migrierte Task in dieser Phase gestoppt wird.

  Aufgrund des verzögerten Stopps der Task wird sichergestellt, dass eine migrierte Task ununterbrochen ausgeführt wird. Als Konsequenz kann eine Task daher unter



(a) Taskverlust zu einem beliebigen Zeitpunkt während des i-ten Hormonzyklus



(b) Taskverlust nach (a) zum diskreten Zeitpunkt $t = i$

Abbildung 5.2: Darstellung eines Taskverlusts zu diskreten Zeitpunkten

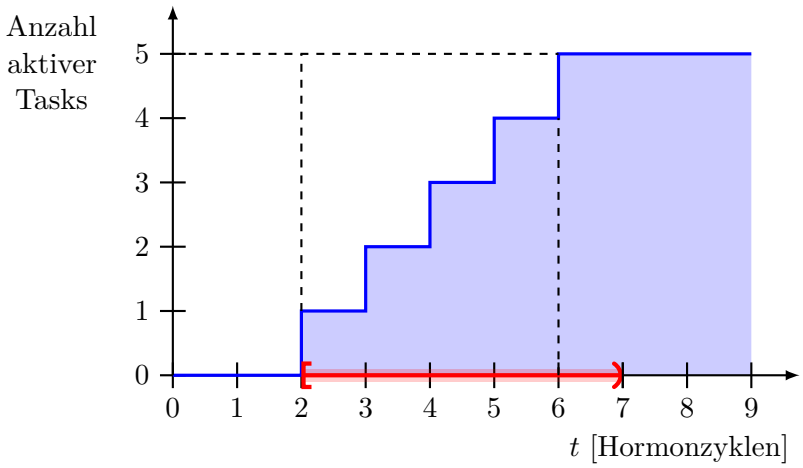


Abbildung 5.3: Beispiel einer Selbstkonfiguration für $m = 5$ Tasks. Die zur Selbstkonfiguration zählenden Hormonzyklen sind rot markiert.

Umständen mehrere Zyklen lang sowohl auf ihrem bisherigen als auch auf dem neuen PE ausgeführt werden, da Übernahme und Stopp nicht notwendigerweise im selben Hormonzyklus geschehen müssen.

- Unkontrollierte Taskverluste geschehen vor allem durch Ausfälle von PEs; diese können jedoch zu beliebigen Zeitpunkten ausfallen.

In Abbildung 5.2a sind diese beiden Fälle skizziert, während Abbildung 5.2b illustriert, wie Taskverluste im Rahmen dieser Dissertation dargestellt werden: Auch hier wird jeder Hormonzyklus als atomar angesehen, ein Taskverlust während des i-ten Hormonzyklus wird zum Zeitpunkt $t = i$ dargestellt.

Anders interpretiert gibt die zu einem diskreten Zeitpunkt $t = i$ dargestellte Anzahl im System aktiver Tasks an, wie viele Tasks *am Ende* des i-ten Hormonzyklus im System aktiv waren.

In Abbildung 5.3 ist schließlich ein Taskdiagramm abgebildet, das den beispielhaften Verlauf einer Selbstkonfiguration für $m = 5$ Tasks darstellt.

Wie in Kapitel 4.4.2 beschrieben dient Hormonzyklus 0 der Synchronisation der PEs. In Zyklus 1 werden zwar bereits Hormone ausgetauscht, es darf aber noch kein PE eine Task übernehmen. Aus diesem Grund zählen diese beiden Hormonzyklen noch nicht zur Selbstkonfiguration, diese beginnt erst mit der Übernahme der ersten Task in Hormonzyklus 2 und endet mit der Übernahme der fünften Task in Zyklus 6. Folglich erstreckt sich die Selbstkonfiguration über die Zyklen 2 bis 6, was einer Gesamtzahl von 5 Hormonzyklen entspricht und sich mit Satz 5.5 deckt.

5.5. Widerlegung der bisherigen Zeitschranken

In diesem Abschnitt soll nun gezeigt werden, dass die bisher bekannten Zeitschranken unter bestimmten Voraussetzungen nicht eingehalten werden.

5.5.1. Experimente

In Satz 5.5 ist gemäß [BP12] angegeben, dass die Selbstkonfiguration von m zu vergebenden Tasks unter Verwendung der aggressiven Entscheidungsstrategie nach höchstens m Hormonzyklen beendet ist.

Erlaubt man Tasks, die mehrfach zuteilbar sind (indem der Übernahmesuppressor entsprechend abgeschwächt wird, siehe Kapitel 4.4.1), so kann diese Zeitschranke jedoch degenerieren.

Um dies zu demonstrieren, wurden verschiedene Szenarien im Hormonsimulator untersucht. Mithilfe eines Python-Skripts wurden dessen Logdateien ausgewertet und unter Zuhilfenahme der Matplotlib-Bibliothek [Hun07] grafisch dargestellt. Auf eine Betrachtung der AHS-Middleware in diesen Szenarien wurde verzichtet, da deren Verhalten jeweils mit dem des Hormonsimulators vergleichbar war.

Eine Task mit 10 Instanzen, 2 PEs, keine Lastsuppressoren Abbildung 5.4 zeigt die Selbstkonfiguration für das erste betrachtete Szenario, das die bekannte Zeitschranke degeneriert:

- Es existiert eine einzige Task T, die aufgrund der Kombination aus Eignungswerten und Übernahmesuppressoren im System zehn mal zugeteilt werden kann.
- Es gibt zwei PEs $\mathrm{PE}_{(1,1)}$ und $\mathrm{PE}_{(2,1)}$, deren statische Eignungswerte für T identisch sind.
- Es werden keine Lastsuppressoren verwendet.

Die Anzahl der aktiven Taskinstanzen sind hierbei gestapelt nach den sie ausführenden PEs dargestellt. Im konkreten Szenario werden *alle* Instanzen von $\mathrm{PE}_{(1,1)}$ ausgeführt. Dies ist auf die fehlenden Lastsuppressoren bei identischen statischen Eignungswerten zurückzuführen: Beide PEs senden daher stets den identische modifizierte Eignungswerte für T. Um bei einem solchen Gleichstand zwischen mehreren PEs eine eindeutige Entscheidung sicherzustellen, wird in den derzeitigen Implementierungen des AHS das PE mit dem *kleinsten* Index zum Gewinner der Task erklärt. Im Falle des hier verwendeten Hormonsimulators wird zudem eine lexikographische Ordnung zugrunde gelegt, um die zweidimensionalen Indizes der PE-Gittertopologie zu ordnen. Folglich hat $\mathrm{PE}_{(1,1)}$ unter allen PEs den kleinsten Index und gewinnt daher jede Instanz der Task.

Nach Satz 5.5 dürfte die Selbstkonfiguration höchstens 10 Hormonzyklen benötigen und – da die erste Task zu $t = 2$ zugeteilt wird – am Ende von Zyklus 11 abgeschlossen sein. In der Realität wird die letzte Taskinstanz jedoch erst zu $t = 20$ zugeteilt; die gesamte Selbstkonfiguration benötigt daher 19 Hormonzyklen.

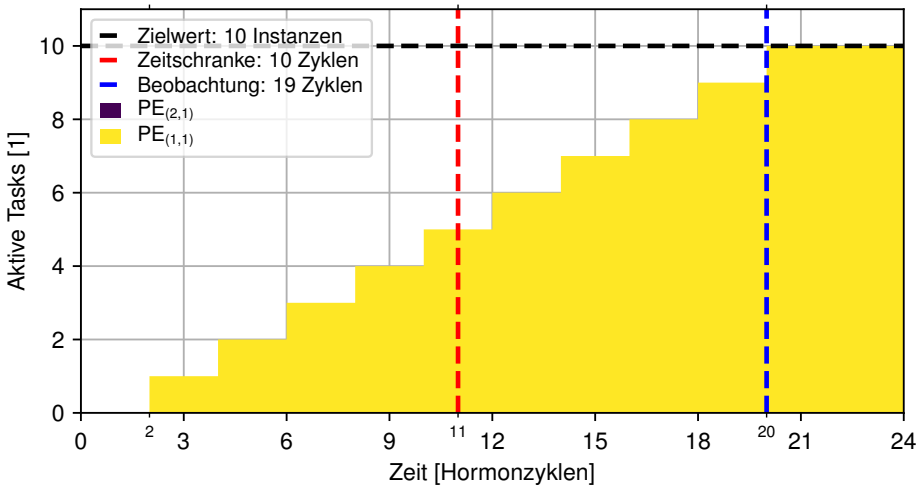


Abbildung 5.4: Selbstkonfiguration einer einzigen, zehn mal zuteilbaren Task auf zwei identischen PEs, ohne Lastsuppressoren

Dieses Beispiel widerlegt also die bekannte Zeitschranke von m Hormonzyklen zur Zuteilung von m Tasks bzw. Taskinstanzen.

Eine Task mit 10 Instanzen, 1 PE, keine Lastsuppressoren Wiederholt man das Experiment mit nur einem einzigen PE, so erhält man dagegen das in Abbildung 5.5 gezeigte Verhalten. In diesem Fall gilt die bekannte Zeitschranke offenbar weiterhin: Im vorherigen Experiment lag zwischen je zwei Zuteilungen einer Taskinstanz jeweils ein Zyklus, in dem keine Zuteilung stattfand. In diesem Szenario wird dagegen in jedem Hormonzyklus eine Taskinstanz zugeteilt.

Eine Task mit 10 Instanzen, 1 PE, Lastsuppressoren Eine Wiederholung des Experiments bei Verwendung von Lastsuppressoren (die so dimensioniert sind, dass sie die Anzahl zuteilbarer Instanzen von T nicht verändern) auf einem einzigen PE führt zum selben zeitlichen Verlauf der Anzahl aktiver Tasks, wie sie in Abbildung 5.5 dargestellt ist und bricht die bisherige Zeitschranke daher ebenfalls nicht.

Eine Task mit 10 Instanzen, 2 PEs, Lastsuppressoren Die Kombination aus zwei identischen PEs und Lastsuppressoren ist ebenfalls konform zur Zeitschranke von höchstens m Hormonzyklen, wie in Abbildung 5.6 dargestellt. Im Gegensatz zur Situation aus Abbildung 5.4 führen die verwendeten Lastsuppressoren zu einer abwechselnden und damit gleichmäßigen Übernahme der Taskinstanzen auf $\mathrm{PE}_{(1,1)}$ und $\mathrm{PE}_{(2,1)}$: Nach jeder Taskübernahme reduziert das zugehörige PE im nächsten Hormonzyklus seinen modifizierten Eignungswert, weshalb die nächste Taskinstanz vom jeweils anderen PE gewonnen und übernommen wird. Folglich sind zum Zeitpunkt $t = 11$ je fünf Taskinstanzen auf $\mathrm{PE}_{(1,1)}$ und $\mathrm{PE}_{(2,1)}$ zugeteilt, wie der nach den PEs gestapelte Graph

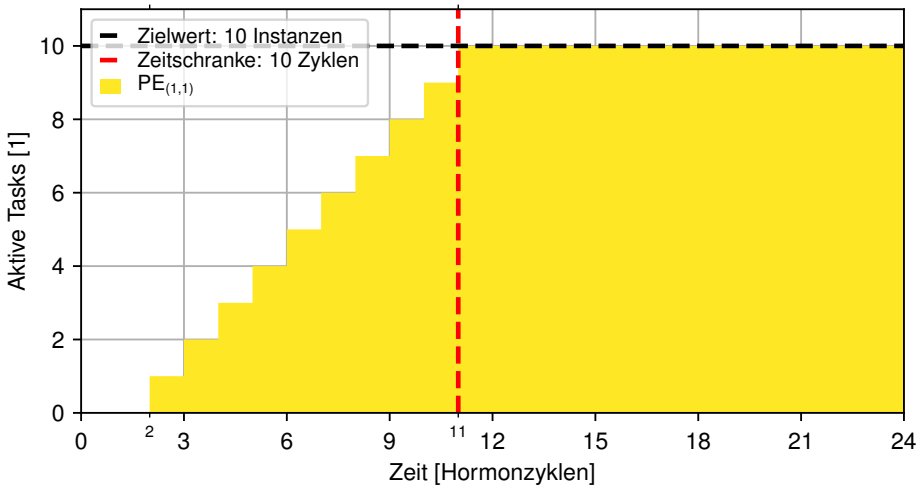


Abbildung 5.5: Selbstkonfiguration einer einzigen, zehn mal zuteilbaren Task auf einem PE, ohne Lastsuppressoren

zeigt.

Zehn Tasks mit je einer Instanz, 2 PEs Ein identisches Verhalten zeigt sich, wenn stattdessen *zehn* je *einfach zuteilbare* Tasks mit Lastsuppressoren auf zwei PEs verteilt werden. Entfallen die Lastsuppressoren dagegen, gleicht das Verhalten wieder jenem aus Abbildung 5.5, da $PE_{(1,1)}$ aufgrund seines kleineren Index alle Taskinstanzen gewinnt.

Zehn Tasks mit je einer Instanz, 1 PE Die Zuteilung von zehn je einfach zuteilbaren Tasks auf einem einzigen PE führt zu einem Verhalten, das jenem aus Abbildung 5.5 gleicht, unabhängig davon, ob Lastsuppressoren verwendet werden.

3 PEs Eine Wiederholung der Experimente mit drei PEs führt zu folgenden Ergebnissen:

- Die Selbstkonfiguration von zehn je einfach zuteilbaren Tasks ohne Lastsuppressoren führt zu einem Verhalten, das jenem aus Abbildung 5.5 gleicht: Alle Tasks werden innerhalb von 10 Hormonzyklen auf $PE_{(1,1)}$ zugeteilt.
- Werden dagegen Lastsuppressoren verwendet, ergibt sich das in Abbildung 5.7 gezeigte Verhalten. Dieses entspricht dem in Abbildung 5.6 gezeigten; der einzige Unterschied ist, dass die Tasks nun auf drei PEs verteilt werden. Auch hier wird die Zeitschranke von 10 Hormonzyklen nicht verletzt.
- Die Selbstkonfiguration einer einzigen, zehnfach zuteilbaren Task mit Lastsuppressoren zeigt dagegen das in Abbildung 5.8 dargestellte Verhalten: Die Taskinstanzen werden – wie im vorherigen Fall – aufgrund der Lastsuppressoren gleichmäßig auf

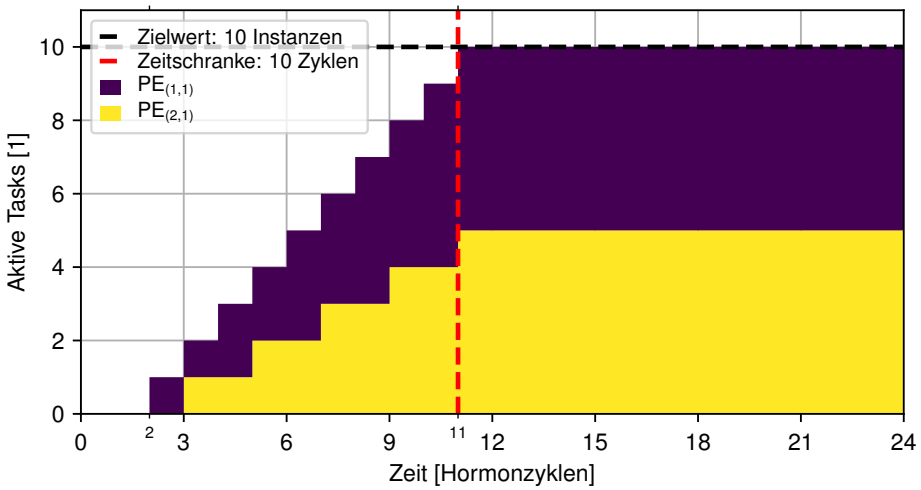


Abbildung 5.6: Selbstkonfiguration einer einzigen, zehn mal zuteilbaren Task auf zwei PEs, mit Lastsuppressoren

die verfügbaren PEs aufgeteilt, die Zeitschranke von 10 Hormonzyklen degeneriert jedoch zu 19 Hormonzyklen.

- Wird erneut auf Lastsuppressoren verzichtet, so ergibt sich wiederum ein zu Abbildung 5.4 identisches Verhalten: Alle Taskinstanzen werden von $PE_{(1,1)}$ übernommen und die Zeitschranke degeneriert zu 19 Hormonzyklen.

Zusammenfassung Tabelle 5.1 fasst die Ergebnisse der soeben beschriebenen Experimente zusammen. Die Spalte „Verteilung auf mehrere PEs“ gibt an, ob die einzelnen Tasks bzw. Taskinstanzen auf mehr als ein PE verteilt (✓) oder nur auf einem einzigen PE instanziiert wurden (✗). Die Spalte „Verhalten wie Abbildung“ verweist auf Abbildungen, welche den Verlauf der Anzahl aktiver Tasks für das jeweilige Szenario darstellen, auch wenn die zugehörige Legende oder Bezeichnung eigentlich einem anderen Szenario entspricht.

Zusammenfassend lassen sich mehrere Beobachtungen festhalten:

- Die Taskinstanzen werden nur dann auf mehrere PEs verteilt, wenn Lastsuppressoren eingesetzt werden.[1] Eine Erklärung für diesen Sachverhalt wurde bereits zuvor angedeutet: Ohne Verwendung von Lastsuppressoren reduzieren Taskübernahmen den gesendeten modifizierten Eignungswert niemals, weshalb stets ein Gleichstand der modifizierten Eignungswerte eintritt. Da ein solcher von den derzeitigen Implementierungen des AHS aufgelöst wird, indem das PE mit dem (lexikographisch) kleinsten Index zum Gewinner der Task erklärt wird, gewinnt $PE_{(1,1)}$ aufgrund seines unter allen PEs kleinsten Index *jede* Taskinstanz.

[1] Dieses Kriterium ist für Szenarien, die nur ein PE umfassen, nicht anwendbar.

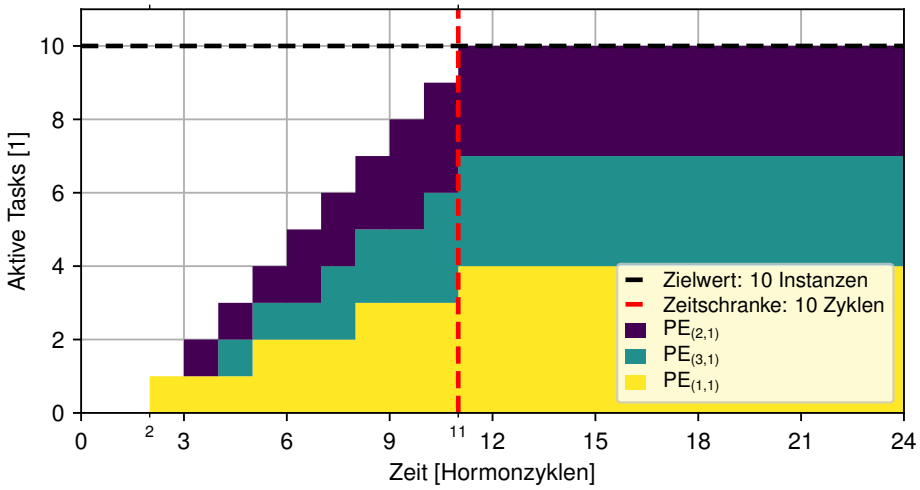


Abbildung 5.7: Selbstkonfiguration zehn einfach zuteilbarer Tasks auf drei PEs, mit Lastsuppressoren

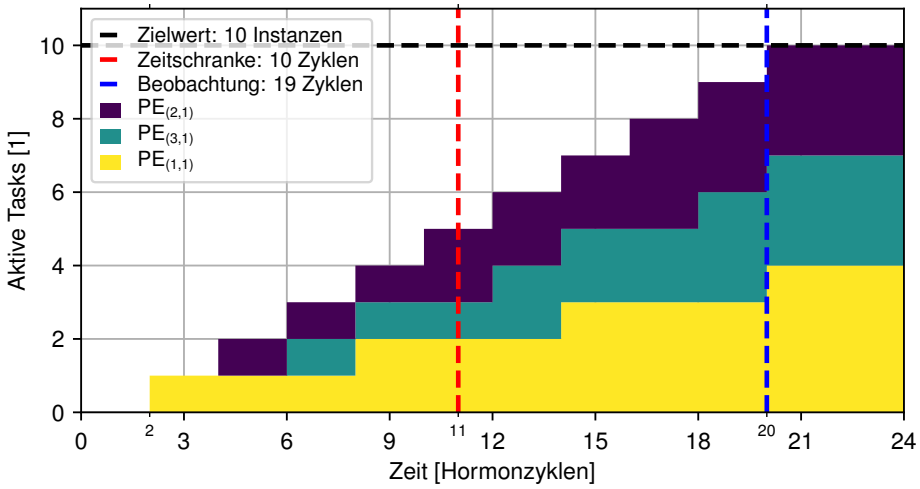


Abbildung 5.8: Selbstkonfiguration einer einzigen, zehn mal zuteilbaren Task auf drei PEs, mit Lastsuppressoren

Tabelle 5.1: Ergebnisse der einzelnen Experimente zur Degeneration der Zeitschranke der Selbstkonfiguration

Anzahl PEs	Anzahl Tasks	Anzahl Instanzen pro Task	Lastsuppressoren	Verteilung auf mehrere PEs	Benötigte Zyklen	Verhalten wie Abbildung
1	1	10	×	–	10	5.5
	1	10	✓	–	10	5.5
	10	1	×	–	10	5.5
	10	1	✓	–	10	5.5
2	1	10	×	×	**19**	5.4
	1	10	✓	✓	10	5.6
	10	1	×	×	10	5.5
	10	1	✓	✓	10	5.6
3	1	10	×	×	**19**	5.4
	1	10	✓	✓	**19**	5.8
	10	1	×	×	10	5.5
	10	1	✓	✓	10	5.7

- Die Zeitschranke von $m = 10$ Hormonzyklen wird nur in Szenarien verletzt, die eine einzige, jedoch mehrfach zuteilbare Task umfassen.
- Ebenfalls wird die Zeitschranke nur dann verletzt, wenn mehr als ein PE zur Verfügung steht.
- Eine einzige, zehnfach zuteilbare Task in Verbindung mit Lastsuppressoren hält die Zeitschranke von 10 Hormonzyklen ein, falls nur zwei PEs verwendet werden; in Anwesenheit von drei PEs werden dagegen 19 Hormonzyklen benötigt.

Als Resultat wurden in diesen Experimenten die bisherige Zeitschranke der Selbstkonfiguration sowie die darauf aufbauenden Zeitschranken der Selbstkonfiguration und Selbstheilung in bestimmten Szenarien widerlegt. Der nächste Abschnitt erklärt den diesen Beobachtungen zugrunde liegenden Effekt.

5.5.2. Erklärung der Zeitschrankendegeneration

Der Grund für die in den vorhergehenden Szenarien beobachtete Degeneration der Zeitschranke von m Hormonzyklen zur Zuteilung von m Tasks ist der in Kapitel 4.4.3 beschriebene *Almost-Winner*-Zustand, der auftritt, weil von einem PE gesendete Übernahmesuppressoren sich erst mit einem Zyklus Verzögerung in den von anderen PEs gesendeten modifizierten Eignungswerten bemerkbar machen. Als Resultat kann die Zeitschranke degenerieren, falls mehrfach zuteilbare Tasks erlaubt werden; konkret kann die Zuteilung der m Taskinstanzen dann $2m - 1$ Zyklen dauern, wie folgendes Beispiel demonstriert:

> **Beispiel 5.12**
>
> Es sei T_1 eine Task, die m mal zugeteilt werden kann, also beispielsweise auf al-

len PEs einen statischen Eignungswert $E_1 = m$ und einen Übernahmesuppressor $Sa_1 = 1$ besitze. Alle Acceleratoren und Lastsuppressoren seien auf Null gesetzt und werden daher im Folgenden ignoriert. Ferner existieren keine weiteren Tasks, es sind im gesamten System daher genau m Taskinstanzen zu vergeben. Zudem gebe es (mindestens) zwei PEs im System.

Für $m = 10$ entspricht dieses Szenario daher den aus Tabelle 5.1 bekannten Fällen „2 bzw. 3 PEs, 1 Task, 10 Instanzen, keine Lastsuppressoren“, deren (identisches) Verhalten in Abbildung 5.4 dargestellt ist. Allerdings wird im Folgenden der Einfachheit halber eine angepasste Nummerierung der Hormonzyklen verwendet, sodass die erste Task bereits in Zyklus 1 zugeteilt wird.

Im ersten Hormonzyklus sendet jedes PE_i einen Eignungswert $Em_{1,i} = m$ für T_1. Wegen des Gleichstands aller modifizierten Eignungswerte übernimmt daher PE_1 aufgrund seines kleinsten Index eine Instanz von T_1:

		$\mathbf{PE_1}$	$\mathbf{PE_2}$
Zyklus 1	**Send**	Sende $Em_{1,1} = m$	Sende $Em_{1,2} = m$
	Wait	Warte t_{SD} und empfange Hormone	
	Decide	*Winner*, übernehme T_1	*Loser*

Im zweiten Zyklus wird jedes PE_j mit $j > 1$ weiterhin einen Eignungswert $Em_{1,j} = m$ für T_1 senden, PE_1 wird dagegen einen durch den Übernahmesuppressor reduzierten Eignungswert $Em_{1,1} = m - 1$ senden. Erst im Anschluss bemerken alle übrigen PEs, dass sie T_1 aufgrund des mittlerweile empfangenen Übernahmesuppressors $Sa_{1,1}$ nicht gewonnen haben, weshalb sie in den Almost-Winner-Zustand wechseln und keine Instanz von T_1 übernehmen dürfen. PE_1 übernimmt jedoch auch keine weitere Instanz von T_1, da es die höheren Eignungswerte von den anderen PEs empfangen hat und in den Loser-Zustand wechselt. Folglich wird in diesem Zyklus keine Task zugeteilt:

Zyklus 2		$\mathbf{PE_1}$	$\mathbf{PE_2}$
	Send	Sende $Em_{1,1} = m - 1$, Sende $Sa_{1,1} = 1$	Sende $Em_{1,1} = m$
	Wait	Warte t_{SD} und empfange Hormone	
	Decide	*Loser*	*Almost winner*

Erst im dritten Zyklus wird wieder eine Instanz von T_1 zugeteilt, da nun alle PEs einen Eignungswert von $m - 1$ senden. Folglich gewinnt PE_1 die zweite Instanz von T_1:

Zyklus 3		$\mathbf{PE_1}$	$\mathbf{PE_2}$
	Send	Sende $Em_{1,1} = m - 1$, Sende $Sa_{1,1} = 1$	Sende $Em_{1,2} = m - 1$
	Wait	Warte t_{SD} und empfange Hormone	
	Decide	*Winner*, übernehme T_1	*Loser*

Dieser Zyklus entspricht also – bis auf den erhöhten Suppressor und die in gleichem Maße erniedrigten modifizierten Eignungswerte – dem ersten Zyklus; das Schema aus Zuteilung einer Taskinstanz gefolgt von einem Wartezyklus ohne Taskübernahme wiederholt sich insgesamt m mal, bis $Em_{1,i}$ für jedes PE_i auf Null gesunken ist.

Das Beispiel lässt sich prinzipiell auf Lastsuppressoren $Sl_1 > 0$ sowie PEs mit unterschiedlichen statischen Eignungswerten für T_1 verallgemeinern. In diesen Fällen werden jedoch – je nach konkret gewählten Hormonwerten – unter Umständen mehr als zwei PEs benötigt, um die Zeitschranke von m Zyklen zu degenerieren. Hierzu sei erneut auf Tabelle 5.1 verwiesen, aus der hervorgeht, dass die Degeneration der Zeitschranke für das Szenario „1 Task, 10 Instanzen, Lastsuppressoren“ für 2 PEs nicht eintritt, wohl aber für 3 PEs; der Grund ist in jedem Fall die Einnahme des Almost-Winner-Zustands, welcher die Übernahme der nächsten Taskinstanz um einen Hormonzyklus verzögert.

Die Zuteilung der m-ten Taskinstanz erfolgt im betrachteten Beispiel somit erst in Zyklus $2m - 1$. Der folgende Wartezyklus muss nicht beachtet werden, da zu diesem

Zeitpunkt bereits alle Tasks verteilt sind und die Selbstkonfiguration folglich nach $2m-1$ Zyklen beendet ist. Wie zuvor beschrieben, hätte die bisherige Zeitschranke hierfür jedoch höchstens m Hormonzyklen abgeschätzt. Der aufgrund der verzögert wirkenden Suppressoren eingenommene Almost-Winner-Zustand verhindert jedoch, dass zwei Instanzen *derselben* Task in *aufeinanderfolgenden* Hormonzyklen übernommen werden können.

5.6. Herleitung neuer Zeitschranken

Offensichtlich gilt die Zeitschranke aus Satz 5.5 in Anwesenheit mehrfach zuteilbarer Tasks also nicht. Wir möchten daher nun eine verfeinerte Zeitschranke für die Selbstkonfiguration herleiten, falls Tasks mehrfach zuteilbar sind; diese wird später als Grundlage neuer Zeitschranken für Selbstoptimierung und Selbstheilung dienen.

Hierbei wird folgende Notation zum Einsatz kommen:

Multimenge aller Tasks, $\mathcal{T}$, $\mathcal{T}(T)$

Notation 5.13. Es bezeichne $\mathcal{T}$ die Multimenge aller Tasks sowie $\mathcal{T}(T)$ die Anzahl möglicher Zuteilungen einer Task $T \in \mathcal{T}$.

Zunächst soll aber der Begriff *Wartezyklus* konkretisiert werden:

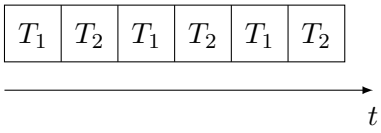
Wartezyklus

Definition 5.14. Ein Hormonzyklus heiße *Wartezyklus*, wenn in ihm im gesamten System keine Task zugeteilt wird, obwohl es Tasks bzw. Taskinstanzen gibt, die noch zugeteilt werden müssen und ausreichend Kapazitäten für deren Zuteilung existieren.

> **Bemerkung**
>
> Ein Zyklus, in dem keine Task zugeteilt wird, weil alle PEs voll ausgelastet sind, wird gemäß dieser Definition *nicht* als Wartezyklus bezeichnet.

Wie oben gesehen, wird die Zuteilung der $(i+1)$-ten Instanz einer Task T um mindestens einen Zyklus gegenüber der Zuteilung der i-ten Instanz verzögert. Diese Verzögerung führt zu einem Wartezyklus, wenn währenddessen keine andere Task im System zugeteilt wird. Dies wird im folgenden Beispiel verdeutlicht:

> **Beispiel 5.15**
>
> Dem AHS seien die Tasks T_1 mit $\mathcal{T}(T_1) = 3$ und T_2 mit $\mathcal{T}(T_2) = 3$ bekannt. Insbesondere gibt es dann $m := |\mathcal{T}| = 6$ zuzuteilende Taskinstanzen.
>
> Im Folgenden wird angenommen, dass pro Zyklus nur auf jeweils *einem* PE Tasks zugeteilt werden, was dem Worst Case entspricht.
>
> Eine wie folgt verschränkte Zuteilungsreihenfolge führt zu keinen Wartezyklen, es werden $m = 6$ Zyklen benötigt, um alle Taskinstanzen zuzuteilen:



Bemerkung

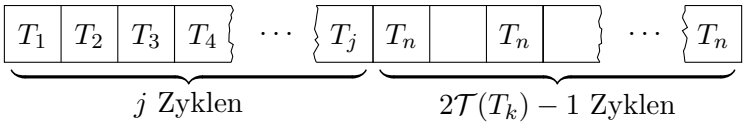
An dieser Stelle sei angemerkt, dass bereits in [Bri13] beschrieben wurde, dass Taskzuteilungen im Fall mehrfach zuteilbarer Tasks in einer verschränkten Reihenfolge geschehen. Der Grund für diese Tatsache wurde jedoch nicht genannt, ebenso wurden die Auswirkungen auf die Zeitschranken der in [BP12] eingeführten aggressiven Task-Decision-Strategie nicht genannt.

Allgemein lässt sich sagen, dass die Anzahl der Wartezyklen minimal wird, wenn die durch Einnahme des Almost-Winner-Zustands hervorgerufenen Verzögerungen durch eine verschränkte Taskzuteilung versteckt werden können. Im Worst Case ist jedoch keine solche Verschränkung möglich:

Beispiel 5.16

Es existieren j Tasks $T_1, \ldots, T_j$ mit $\mathcal{T}(T_i) = 1$ für $1 \leq i \leq j$ sowie eine Task T_k mit $\mathcal{T}(T_k)$ Zuteilungen.

Im Worst Case werden diese Tasks in folgender Reihenfolge zugeteilt:



Dies dauert für $j + \mathcal{T}(T_k) \coloneqq m$ Taskinstanzen insgesamt $m + \mathcal{T}(T_k) - 1 \leq 2m - 1$ Hormonzyklen.

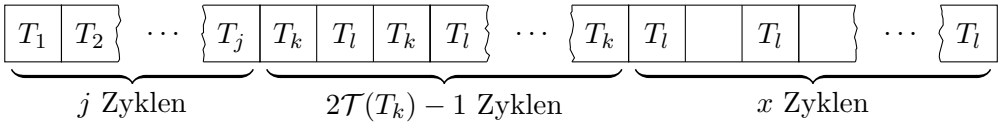
Insbesondere werden für $\mathcal{T}(T_k) = 1$ gerade m Zyklen benötigt, wie von der Zeitschranke der aggressiven Strategie vorhergesagt. Für $j = 0$ werden dagegen exakt $2m - 1$ Hormonzyklen benötigt.

Existiert mehr als eine mehrfach zuteilbare Task, so können die Verzögerungen dagegen durch verschränkte Taskzuteilungen versteckt werden:

Beispiel 5.17

Es existieren j Tasks $T_1, \ldots, T_j$ mit $\mathcal{T}(T_i) = 1$ für $1 \leq i \leq j$, eine Task T_k mit $\mathcal{T}(T_k)$ Zuteilungen sowie eine Task T_l mit $\mathcal{T}(T_l) \geq \mathcal{T}(T_k)$ Zuteilungen.

Im Worst Case werden diese Tasks in folgender Reihenfolge zugeteilt:



Solange Instanzen von mindestens zwei verschiedenen Tasks zu verteilen sind, entstehen keine Wartezyklen:

- Die aggressive Task-Decision-Strategie wird im gesamten System in jedem Hormonzyklus mindestens eine Taskinstanz zuteilen, sofern dies möglich ist; der Worst Case wird offensichtlich erreicht, wenn im gesamten System *höchstens* eine Taskinstanz pro Hormonzyklus zugeteilt wird.
- Der Almost-Winner-Zustand kann lediglich die Zuteilung einer weiteren Instanz *derselben* Task um einen Hormonzyklus verzögern.

Im Beispiel wurden während der Zuteilung der $\mathcal{T}(T_k)$ Instanzen von T_k gerade $(\mathcal{T}(T_k) - 1)$ Instanzen von T_l zugeteilt, weshalb

$$x = 2(\mathcal{T}(T_l) - (\mathcal{T}(T_k) - 1)) - 1 = 2\mathcal{T}(T_l) - 2\mathcal{T}(T_k) + 1$$

gilt. Insgesamt werden also

$$j + (2\mathcal{T}(T_k) - 1) + (2\mathcal{T}(T_l) - 2\mathcal{T}(T_k) + 1) = j + 2\mathcal{T}(T_l) \text{ Hormonzyklen}$$

benötigt: Alle Verzögerungen der Zuteilung von Instanzen der Task T_k wurden durch Zuteilungen von T_l effektiv verborgen.

Folglich sind Wartezyklen nur am Ende der Selbstkonfiguration möglich, da dann keine Tasks mehr zu verteilen sind, welche die auftretenden Verzögerungen verbergen könnten. Als Konsequenz entsteht die maximale Anzahl Wartezyklen, wenn die Task T_l mit den meisten zuzuteilenden Instanzen zuletzt übernommen wird.

5.6.1. Verallgemeinerung

Diese Überlegungen können nun verallgemeinert werden, um Zeitschranken herzuleiten, die auch in Anwesenheit mehrfach zuteilbarer Tasks korrekt sind.

5.6.1.1. Selbstkonfiguration

Für die Selbstkonfiguration folgt:

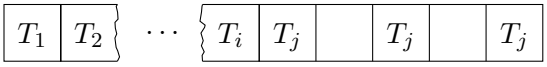
Satz 5.18. Es sei $\mathcal{T}$ die Multimenge aller Tasks. Es bezeichne $\mathcal{T}(T)$ die Anzahl der zuzuteilenden Instanzen der Task T, die sich aufgrund des Zusammenspiels der Werte von Eignungswerten und Suppressoren ergibt. Ferner bezeichne $m \coloneqq |\mathcal{T}|$ die Gesamtzahl aller zuzuteilenden Taskinstanzen. Es gebe zudem genug PEs, um alle m Taskinstanzen zu verteilen und es finden währenddessen weder Selbstoptimierung noch Selbstheilung statt.

Dann dauert die Selbstkonfiguration höchstens

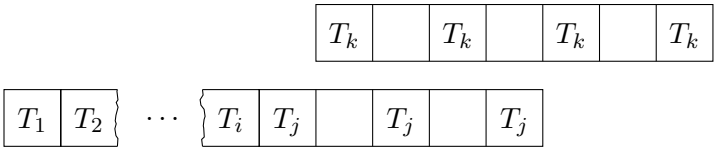
$$m + \max_{T \in \mathcal{T}} \{\mathcal{T}(T)\} - 1 \text{ Hormonzyklen.}$$

Beweis. O.B.d.A. gelte $\text{supp}(T) = \{T_1, \ldots, T_k\}$. Zudem bezeichne T_k die Task mit den meisten zuzuteilenden Instanzen aller Tasks aus $\mathcal{T}$.

Sei nun R eine Zuteilungsreihenfolge, die bei Zuteilung aller Tasks aus $\mathcal{T} \setminus \{T_k\}$ entsteht. Diese kann, wie in den vorigen Überlegungen hervorgeht, höchstens am Ende Wartezyklen enthalten:



Erweitert man R durch Hinzunahme von T_k, so ergibt sich eine Zuteilungsreihenfolge R' für $\mathcal{T}$. Hierbei wird T_k zuletzt zugeteilt und kann Wartezyklen in R mit Zuteilungen von T_k füllen, am Ende jedoch auch zusätzliche Wartezyklen hinzufügen:



Im Worst Case hatte R jedoch gar keine Wartezyklen, sondern umfasste gerade $(m - \mathcal{T}(T_k))$ Zyklen. In diesem Fall werden durch die Hinzunahme von T_k folglich $\mathcal{T}(T_k)$ Zyklen, in denen Taskinstanzen von T_k zugeteilt werden, sowie $\mathcal{T}(T_k) - 1$ Wartezyklen hinzugefügt.

Da T_k die Task mit den meisten zuzuteilenden Instanzen ist, kann es zudem keine Zuteilungsreihenfolge für $\mathcal{T}$ geben, die mehr Zyklen als die soeben konstruierte benötigt.

Damit folgt, dass die Selbstkonfiguration im Worst Case gerade

$$\begin{aligned}(m - \mathcal{T}(T_k)) + \mathcal{T}(T_k) - 1 + \mathcal{T}(T_k) &= m + \mathcal{T}(T_k) - 1 \\ &= m + \max_{T \in \mathcal{T}} \{\mathcal{T}(T)\} - 1\end{aligned}$$

Hormonzyklen dauert. □

Aus dieser Zeitschranke folgt als Spezialfall unmittelbar die aus Satz 5.5 bzw. [BP12] bekannte Zeitschranke, die aus Gründen der Korrektheit jedoch nur anwendbar ist, wenn es keine mehrfach zuteilbaren Tasks gibt:

Korollar 5.19. Es bezeichne $\mathcal{T}$die Multimenge aller Tasks. Ferner gelte $\mathcal{T}(T) = 1$ für alle $T \in \mathcal{T}$, es sind also keine mehrfach zuteilbaren Tasks vorhanden.[1]

Dann dauert die Selbstkonfiguration höchstens m Hormonzyklen.

Beweis. Folgt mit $\max_{T \in \mathcal{T}} \{\mathcal{T}(T)\} = 1$ unmittelbar aus Satz 5.18. □

Ferner folgt ein weiterer Spezialfall, der in Abschnitt 5.5.1 für $m = 10$ experimentell beobachtet wurde:

Korollar 5.20. Es gebe nur eine Task T mit $\mathcal{T}(T) = m$.

Dann dauert die Selbstkonfiguration höchstens $2m - 1$ Hormonzyklen.

[1] Die Multimenge aller Tasks kann in diesem Fall also als eine *Menge* aller Tasks angesehen werden.

Beweis. Folgt mit $\max_{T \in \mathcal{T}}\{\mathcal{T}(T)\} = m$ unmittelbar aus Satz 5.18. □

5.6.1.2. Selbstoptimierung

Unter Verwendung der neuen Zeitschranke ergibt sich für die Selbstoptimierung:

Satz 5.21. Es bezeichne $\mathcal{T}$ die Multimenge aller Tasks und $m := |\mathcal{T}|$ gebe die Anzahl der im System vorhandenen Taskinstanzen an.

Dann dauert die Selbstoptimierung höchstens

$$m + \max_{T \in \mathcal{T}} \{\mathcal{T}(T)\} - 1 \text{ Hormonzyklen.}$$

Beweis. Mit identischer Argumentation wie im Beweis von Satz 5.6: Im Worst Case beginnen zur gleichen Zeit Offerphasen für alle m Taskinstanzen. Das entspricht jedoch einer Selbstkonfiguration aller Taskinstanzen, woraus mit Satz 5.18 die Behauptung folgt. □

Offensichtlich folgen auch hier Spezialfälle, in denen die Selbstoptimierung höchstens m bzw. $2m - 1$ Hormonzyklen benötigt (wenn keine mehrfach zuteilbaren Tasks existieren bzw. nur eine einzige, m mal zuteilbare Task existiert).

5.6.1.3. Selbstheilung

Auch die Zeitschranke der Selbstheilung muss entsprechend angepasst werden:

Satz 5.22. Es bezeichnen $\text{PE}_\times$ das ausgefallene PE und $\mathcal{T}_f$ die Multimenge der zuvor auf diesem zugeteilten Taskinstanzen. Ferner haben die verbleibenden PEs genügend Kapazitäten, um alle $|\mathcal{T}_f| := m_f$ Taskinstanzen zusätzlich zu übernehmen und es finden während der Selbstheilung weder Selbstkonfiguration, Selbstoptimierung noch eine weitere Selbstheilung statt.

Dann dauert die Selbstheilung höchstens

$$m_f + \max_{T \in \mathcal{T}_f} \{\mathcal{T}(T)\} - 1 + a \text{ Hormonzyklen.}$$

Beweis. Mit identischer Argumentation wie im Beweis von Satz 5.9: Nach dem Erkennen des Ausfalls von $\text{PE}_\times$ entspricht die Selbstheilung gerade einer Selbstkonfiguration mit m_f Taskinstanzen und die Behauptung folgt mit Satz 5.18. □

Auch hier existieren Spezialfälle mit höchstens $m_f + a$ bzw. $2m_f - 1 + a$ Hormonzyklen.

5.7. Abschließende Bemerkungen

In diesem Kapitel wurde gezeigt, dass die bisher bekannten Zeitschranken der aggressiven Task-Decision-Strategie nicht korrekt sind, falls im System mehrfach zuteilbare Tasks existieren. Daher wurden verfeinerte Zeitschranken hergeleitet, die auch unter

diesen Umständen korrekt sind; falls keine mehrfach zuteilbaren Tasks existieren, so folgen die bisherigen Zeitschranken als Spezialfälle dieser neuen Schranken.

Da die Erfahrung zeigt, dass mehrfach zuteilbare Tasks nur höchst selten zum Einsatz kommen,[1] kann in den meisten Fällen dennoch auf die bisherigen Zeitschranken zurückgegriffen werden. Insbesondere wird die Existenz mehrfach zuteilbarer Tasks in späteren Kapiteln dieser Dissertation explizit ausgeschlossen werden.

Nichtsdestotrotz ist der Almost-Winner-Zustand als Grund für die beobachtete Degeneration der bisher bekannten Zeitschranken von entscheidender Bedeutung für die Analyse der neuartigen, *prioritätsbasierten* Task-Decision-Strategie, die auf der aggressiven Strategie aufbaut und im nächsten Kapitel vorgestellt wird.

[1] Als aktuell einziges bekanntes Beispiel für den praktischen Einsatz mehrfach zuteilbarer Tasks sei die beispielsweise in [HBF20] erwähnte *State-Transfer-Task* genannt, die redundant auf verschiedenen PEs vorgehalten wird, um die inneren Zustände anderer Tasks zu sichern, sodass deren Wiederherstellung nach Task- bzw. PE-Ausfällen möglich ist.

6 Prioritätsbasierte Taskverteilung

Wie bereits in Kapitel 1.5 erwähnt, befasst sich diese Dissertation primär mit einer prioritätsbasierten Task-Decision-Strategie, die im Folgenden vorgestellt werden soll. Einige der Resultate wurden bereits in [HB20] und [HB21a] veröffentlicht.

6.1. Motivation

Das AHS in seiner in Kapitel 4 beschriebenen Form garantiert zwar Zeitschranken, nach deren Ablauf alle Tasks in einem System verteilt sind, spezifiziert jedoch nicht die Reihenfolge der Taskverteilung. Als Resultat ist die Zuteilungsreihenfolge der Tasks nicht deterministisch. Folglich ist es insbesondere nicht möglich, abhängige Tasks in einer bestimmten Reihenfolge zuzuteilen.

Beispielsweise ist ein System vorstellbar, das unter anderem eine Task T_1 umfasst, welche die Führungsgröße für eine nachgeschaltete PID-Regler-Task T_2 liefert. Wird nun T_2 vor T_1 zugeteilt, so könnte der Regler durch Windup-Effekte bereits einen zu hohen Fehler aufintegriert haben, sodass sich das System in einem nicht funktionsfähigen Zustand befindet, obwohl alle Tasks zugeteilt sind. Im schlimmsten Fall werden sogar alle übrigen im System vorhandenen Tasks zwischen T_2 und T_1 zugeteilt.

Folglich erscheint es sinnvoll, das AHS dahingehend zu erweitern, dass deterministische Zuteilungsreihenfolgen möglich sind. Als Resultat ließen sich beispielsweise Windup-Effekte vermindern. Ebenfalls wäre es möglich, Latenzen beim Start bzw. der initialen Selbstkonfiguration eines Systems zu verstecken: Sobald die wichtigsten Tasks zugeteilt sind, ist das System unter Umständen bereits teilweise funktionsfähig und im Hintergrund können weitere unkritische Tasks zugeteilt werden.

Insbesondere würde eine *Priorisierung* der einzelnen Tasks es ermöglichen, die wichtigsten Systemfunktionen funktionsfähig zu halten, wenn zu viele Systemkomponenten ausgefallen sind, um eine *vollständige* Selbstheilung durchzuführen: In solchen *Überlastsituationen* könnten Tasks niedriger Priorität gestoppt werden, um Rechenkapazitäten für Tasks hoher Priorität zu schaffen.

6.2. Vorbereitungen

Aus den genannten Gründen ermöglicht es die im Rahmen dieser Dissertation entwickelte *prioritätsbasierte* Task-Decision-Strategie, jeder Task im System eine statische Priorität zuzuweisen:

Taskpriorität, Priorität, $\mathbf{prio}(T)$

Definition 6.1. Für eine Task $T \in \mathcal{T}$ bezeichne $\text{prio}(T) \in \mathbb{N}$ die statische *Taskpriorität* oder *Priorität* von T. Dabei habe T genau dann eine *höhere* Priorität als eine Task T', wenn $\text{prio}(T) > \text{prio}(T')$ gilt.

Prioritätslevel

Gelegentlich wird statt der Begriffe *Taskpriorität* oder *Priorität* auch von einem *Prioritätslevel* gesprochen, um zu betonen, dass Tasks höherer Prioritäten im System vor Tasks mit niedrigeren Prioritäten zugeteilt werden.

Menge aller Taskprioritäten, P

Die Menge aller im System vorhandenen Prioritäten wird mit P bezeichnet.

Im Folgenden wird zudem vorausgesetzt, dass jedes PE die Priorität jeder Task kennt – auch dann, wenn ein PE bestimmte Tasks nicht ausführen kann.

Zudem wollen wir nun den bereits mehrfach verwendeten Begriff einer Überlastsituation konkretisieren:

Überlastsituation

Definition 6.2. Eine *Überlastsituation* liegt vor, wenn eine Task T_i existiert, die im System nicht zugeteilt werden kann, weil jedes PE_j, das T_i prinzipiell ausführen *könnte*, aufgrund von Lastsuppressoren einen modifizierten Eignungswert $Em_{i,j} = 0$ für T_i sendet.

Analog seien Unterlastsituationen definiert:

Unterlastsituation

Definition 6.3. Eine *Unterlastsituation* liegt vor, wenn im System genügend Rechenkapazitäten bereitstehen, um alle Tasks im System verteilen zu können.

Im Kontext dieser Dissertation nehmen wir zudem an, dass das System sich standardmäßig in einer Unterlastsituation befindet und eine Überlastsituation frühestens nach dem Ausfall eines PEs vorliegt. Insbesondere stehen daher im Rahmen der initialen Selbstkonfiguration des Systems genügend Rechenkapazitäten bereit, um alle Tasks im System verteilen zu können.

6.3. Entscheidungsstrategie

Das Grundprinzip der prioritätsbasierten Task-Decision-Strategie lässt sich nun wie folgt beschreiben, um die Zuteilungsreihenfolge aller Tasks im System gemäß deren Priorität zu ordnen:

Strategie 6.4. Jedes PE verzögert die Zuteilung von Tasks, wenn es feststellt, dass ein anderes PE im selben Hormonzyklus eine Task höherer Priorität zuteilen *könnte*.

Diese Strategie garantiert, dass eine Task T_1 nur dann zugeteilt werden kann, wenn keine höherpriore Task T_2 existiert, die noch nicht zugeteilt ist – ansonsten würde sich (mindestens) ein PE um die Ausführung von T_2 bewerben. Dann müsste jedoch ein PE existieren, das für T_2 den höchsten modifizierten Eignungswert gesendet hat und T_2 zuteilen darf, was die anderen PEs erkennen und daher die Zuteilung von T_1 verzögern würden.

Wie man leicht sieht, stellt diese Strategie lediglich sicher, dass Tasks höherer Priorität vor Tasks niedrigerer Priorität zugeteilt werden. Die Reihenfolge der Zuteilung von Tasks *derselben* Priorität ist jedoch nicht spezifiziert.

6.3.1. Umsetzung

Im Folgenden soll die Umsetzung der prioritätsbasierten Strategie analog zur Beschreibung der aggressiven Task-Decision-Strategie aus Kapitel 4.4.3 erläutert werden.

Bei Verwendung der prioritätsbasierten Task-Decision-Strategie untersucht jedes PE in jedem Hormonzyklus alle Tasks – insbesondere auch solche, die es selbst nicht ausführen kann – in der Reihenfolge absteigender Prioritäten.

Wie im Fall der aggressiven Strategie werden hierzu alle $|\text{supp}(\mathcal{T})| \coloneqq k$ verschiedenen Tasks intern in einer Liste abgelegt:

$$T_{\pi(1)}, T_{\pi(2)}, \ldots, T_{\pi(k)}, \bot$$

Hierbei markiere $\bot$ das Ende der Liste und π sei eine Permutation, sodass

$$\text{prio}\left(T_{\pi(i)}\right) \geq \text{prio}\left(T_{\pi(j)}\right) \quad \text{für alle } 1 \leq i \leq j \leq k \text{ gilt,}$$

weshalb obige Liste also in absteigender Reihenfolge nach Prioritäten sortiert ist.

Wenn ein PE diese Liste durchläuft und dabei eine Task T findet, die es gewonnen hat, so darf es T nur dann übernehmen, wenn aus den empfangenen Hormonwerten *nicht* hervorgeht, dass ein anderes PE existiert, das eine höherpriore Task T' gewonnen hat (und diese daher im selben Hormonzyklus übernehmen *könnte*). Ist dies dennoch der Fall, so wird die Entscheidung über T für diesen Hormonzyklus beendet und auf einen folgenden Zyklus verschoben. Als Resultat kann es passieren, dass ein PE in einem Hormonzyklus keine Task zuteilt, auch wenn es (mindestens) eine Task gewonnen hat. Dies stellt sicher, dass Tasks höherer Priorität vor jenen niedrigerer Priorität zugeteilt werden.

Ähnlich zur aggressiven Strategie sucht jedes PE auch gemäß dieser Strategie aktiv nach einer Task, für die es den höchsten modifizierten Eignungswert gesendet hat.

6.3.2. Behandlung von Überlastsituationen

Neben der Beachtung von Taskprioritäten unterscheidet sich diese prioritätsbasierte Task-Decision-Strategie jedoch in einem weiteren zentralen Aspekt von der aggressiven Strategie: Wird festgestellt, dass für eine hochpriore Task T weder Eignungswerte noch Suppressoren empfangen wurden und daher eine Überlastsituation vorliegt,[1] so stoppt jedes PE niedrigpriore Tasks, sofern dies möglich ist, sodass es für T im nächsten Hormonzyklus einen positiven modifizierten Eignungswert senden kann. Ein solcher Stopp von Tasks zugunsten anderer Tasks war bisher im AHS nicht vorgesehen und basiert auf der folgenden fundamentalen Annahme:

[1] Wird für T ein Übernahmesuppressor $Sa_{T,i} > 0$ empfangen, so ist T auf PE_i zugeteilt. Wird für T ein modifizierter Eignungswert $Em_{T,i}$ empfangen, so befindet sich T entweder in einer Offerphase (und ist auf einem PE zugeteilt) oder ist nicht zugeteilt, kann jedoch auf PE_i zugeteilt werden. In beiden Fällen liegt keine Überlastsituation vor. Werden für T dagegen weder Suppressoren noch modifizierte Eignungswerte empfangen, so *kann* eine Überlastsituation vorliegen; die für eine korrekte Erkennung einer solchen nötigen Details werden in Abschnitt 6.3.4 beschrieben.

Annahme 6.5. Jede Task $T' \in \mathcal{T}$ darf jederzeit gestoppt werden, um Rechenkapazitäten für Tasks *höherer* Priorität bereitzustellen.

Der Stopp einer Task T' kann dabei permanent oder auch nur temporär sein: Falls genügend Rechenkapazitäten bereitstehen, kann T' zu einem späteren Zeitpunkt erneut zugeteilt werden. Dies ist denkbar, wenn mehr niedrigpriore Tasks zugunsten höherpriorer Tasks gestoppt wurden, als tatsächlich nötig gewesen wäre, um deren Zuteilung sicherzustellen.

Dass ein solcher *temporärer* Taskstopp auftreten kann, hängt mit der verteilten Natur des AHS zusammen. Stellt ein PE_i fest, dass eine hochpriore Task T existiert, die aufgrund des Vorliegens einer Überlastsituation auf keinem PE zugeteilt werden kann, so muss es *lokal* eine bestmögliche Entscheidung treffen, um sicherzustellen, dass T baldmöglichst im System zugeteilt werden kann: PE_i hat nur begrenzte Informationen über den Zustand des Gesamtsystems. Insbesondere kann es sich daher nicht darauf verlassen, dass andere PEs ebenfalls eine Aktion durchführen, um die Zuteilung von T zu ermöglichen: Diese können T unter Umständen überhaupt nicht ausführen, führen nur Tasks höherer Priorität aus (die nicht zugunsten von T gestoppt werden dürfen) oder können nicht genügend Tasks stoppen, um T ausführen zu können. Ferner würde eine Koordination der PEs untereinander zur Berechnung einer optimalen Auswahl zu stoppender Tasks zusätzliche Hormonzyklen benötigen, was die Zuteilung von T weiter verzögern würde.

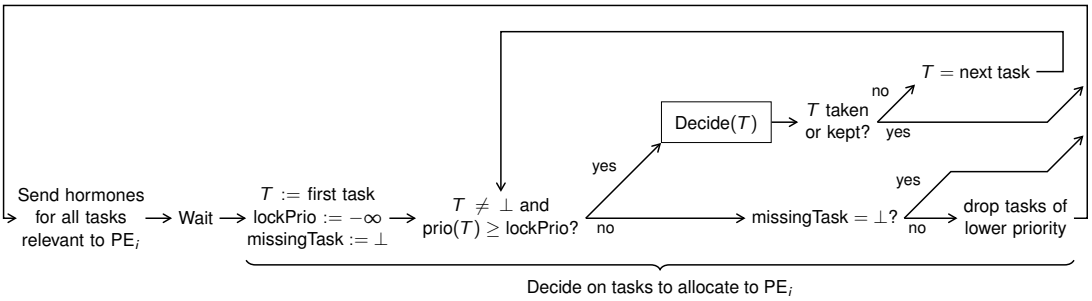
Aus diesen Gründen trifft jedes PE die aus seiner Sicht beste lokale Entscheidung, um eine Zuteilung von T baldmöglichst zu erlauben, auch wenn diese Entscheidung global betrachtet unter Umständen nicht optimal ist.

6.3.3. Implementierung des Hormonzyklus

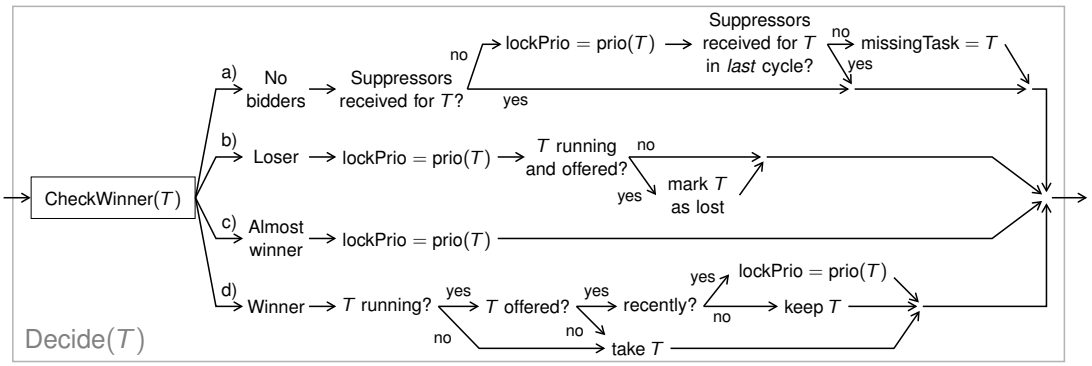
Die Implementierung eines Hormonzyklus bei Verwendung der prioritätsbasierten Task-Decision-Strategie ist nun in Abbildung 6.1 schematisch dargestellt, Abbildung 6.1a zeigt hierbei den Ablauf des gesamten Hormonzyklus.

Konkret werden nach dem Senden der Hormone sowie dem Warten auf den Empfang von Hormonen anderer PEs einige Initialisierungen durchgeführt: Die Variable missingTask soll bei Bedarf eine Referenz auf eine hochpriore Task speichern, die aufgrund einer Überlastsituation nicht im System zugeteilt werden kann. Diese wird mit $\perp$ initialisiert, da noch keine solche Task identifiziert wurde. Die Variable lockPrio dagegen soll die Zuweisung von Tasks auf ein bestimmtes Prioritätslevel beschränken: Wird bemerkt, dass ein anderes PE im aktuellen Zyklus eine Task T_1 übernehmen könnte, so darf in diesem Zyklus keine Task T_2 mit $\text{prio}(T_2) < \text{prio}(T_1)$ übernommen werden, was durch Setzen von lockPrio signalisiert wird. Zu Beginn liegen jedoch noch keine solchen Einschränkungen vor und lockPrio wird daher auf $-\infty$ gesetzt. Schließlich wird die Taskreferenz T auf die höchstpriore im System vorhandene Task gesetzt.

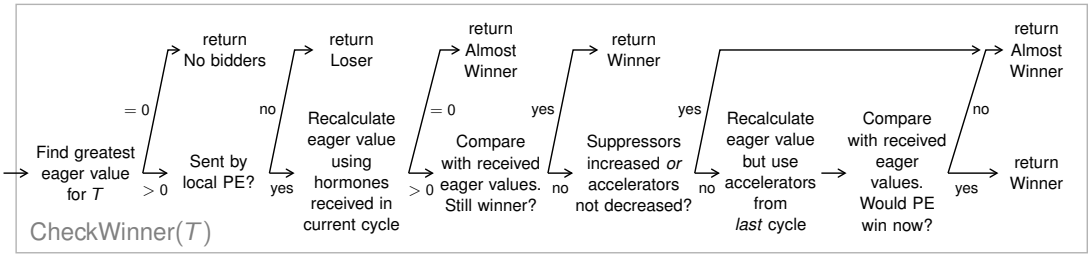
Ähnlich zur aggressiven Strategie werden im Anschluss an diese Initialisierungen alle Tasks in einer Schleife durchlaufen, die jedoch abgebrochen wird, sobald entweder alle Tasks betrachtet wurden ($T = \perp$) oder die Priorität von T kleiner als lockPrio ist. In



(a) Gesamter Hormonzyklus



(b) Unterprozedur „Decide(T)“



(c) Unterprozedur „CheckWinner(T)“

Abbildung 6.1: Prioritätsbasierte Task-Decision-Strategie

letzterem Fall könnte ein anderes PE eine Task der Priorität lockPrio übernehmen, weshalb in diesem Hormonzyklus keine Übernahme von Tasks mit niedrigerer Priorität erfolgen darf.

Wenn am Ende der Schleife missingTask gesetzt ist, so verweist diese Variable auf eine der höchstprioren Tasks, die im System nicht zugeteilt sind und aufgrund einer Überlastsituation auch nicht zugeteilt werden können. In diesem Fall wird jedes PE versuchen, hinreichend viele Tasks zu stoppen, um die Zuteilung der durch missingTask referenzierten Task im folgenden Hormonzyklus sicherzustellen. Die Entscheidung, *welche* Tasks in einem solchen Fall gestoppt werden, obliegt einer *Task-Dropping-Strategie*. Insbesondere sind die garantierbaren Zeitschranken in Überlastsituationen von der verwendeten Task-Dropping-Strategie abhängig. Kapitel 8 wird sich im Detail mit Überlastsituationen im Allgemeinen sowie konkreten Task-Dropping-Strategien befassen.

Wurden dagegen noch nicht alle Tasks betrachtet, so wird für Task T durch die Decide-Unterprozedur entschieden, ob diese auf dem lokalen PE übernommen werden soll. Ist dies der Fall, wird die Schleife ebenfalls beendet, andernfalls wird T auf die nächste zu betrachtende Task gesetzt (bzw. $\perp$, falls bereits alle Tasks betrachtet wurden).

Decide-Unterprozedur Wie im Fall der aggressiven Task-Decision-Strategie vergleicht die Decide-Unterprozedur auch hier für eine Task T zunächst über die CheckWinner-Unterprozedur den lokalen modifizierten Eignungswert mit den empfangenen Eignungswerten der übrigen PEs. Im Vergleich zur aggressiven Strategie, deren Ablauf für die Fälle *No bidders*, *Loser* und *Almost winner* identisch war (siehe Kapitel 4.4.3), müssen hier im Anschluss jedoch alle möglichen Fälle gesondert behandelt werden, wie in Abbildung 6.1b gezeigt:

a) **No bidders**: Wurden für T Übernahmesuppressoren empfangen, so muss T irgendwo im System zugeteilt sein. In diesem Fall kann mit der nächsten Task fortgefahren werden.

   Wurden dagegen keine Suppressoren empfangen, so ist T im System derzeit nicht zugeteilt.[1] In diesem Fall wird lockPrio auf die Priorität von T gesetzt, um zu verhindern, dass eine Task niedrigerer Priorität vor T zugeteilt wird.

   Falls jedoch auch im *vorherigen* Zyklus keine Suppressoren für T empfangen wurden, so liegt eine Überlastsituation vor. In diesem Fall wird missingTask auf T gesetzt, um durch den Stopp niederpriorer Tasks Kapazitäten für die Ausführung von T zu schaffen. Würde nicht auf den Empfang von Suppressoren im vorherigen Zyklus geprüft, könnte eine Unterlastsituation fälschlicherweise als Überlastsituation erkannt werden, wie später in Abschnitt 6.3.4 erläutert wird.

b) **Loser**: Es existiert ein PE, das einen höheren modifizierten Eignungswert für T gesendet hat, folglich *könnte* dieses PE T im aktuellen Hormonzyklus übernehmen.

[1] Wenn T sich in einer Offerphase befindet, wird das anbietende PE stattdessen einen positiven modifizierten Eignungswert für T senden, weshalb der Zustand *No bidders* nicht eingenommen wird.

Daher wird lockPrio auf die Priorität von T gesetzt, um die Übernahme von Tasks niedrigerer Priorität in diesem Zyklus zu verhindern. Falls T aktuell auf dem lokalen PE zugeteilt ist und im Rahmen einer Offerphase anderen PEs angeboten wird, so wird T für einen verzögerten Stopp markiert, der durchgeführt wird, sobald der lokale modifizierte Eignungswert durch von anderen PEs empfangene Suppressoren auf Null sinkt.[1]

c) **Almost Winner**: In diesem Fall wechseln alle übrigen PEs für T in den *Loser*-Zustand, da sie davon ausgehen müssen, dass T im aktuellen Zyklus von diesem PE übernommen werden könnte. Als Resultat werden sie entsprechend lockPrio setzen, weshalb auch das lokale PE dies tun muss, um eine mögliche Verletzung der Zuteilungsreihenfolge zu verhindern: Anderenfalls könnte das lokale PE in diesem Zyklus eine Task T' niedrigerer Priorität übernehmen, sofern T die letzte zugeteilte Task ihres Prioritätslevels war. Wenn T jedoch mehrfach zugeteilt werden kann, so könnte im nächsten Zyklus eine weitere Instanz von T zugeteilt werden, obwohl bereits eine Instanz der Task T' mit niedrigerer Priorität zugeteilt ist.

d) **Winner**: Das lokale PE hat für T unter allen PEs den höchsten modifizierten Eignungswert gesendet und darf T daher in diesem Zyklus übernehmen. Hierbei müssen jedoch verschiedene Fälle unterschieden werden:

   I) T ist noch nicht auf dem PE zugeteilt: In diesem Fall wird eine Instanz von T übernommen.

   II) T ist bereits auf dem PE zugeteilt. Dieser Fall kann in zwei verschiedenen Situationen auftreten:

      i) T wird im Rahmen einer Offerphase anderen PEs angeboten. Da das lokale PE im *Winner*-Zustand ist, ist es zum aktuellen Zeitpunkt immer noch das bestgeeignete PE für T.

      Ist seit dem Beginn der Offerphase eine bestimmte Zeit vergangen, die es den übrigen PEs ermöglichte, auf den Wegfall der Übernahmesuppressoren tatsächlich mit dem Senden eines positiven modifizierten Eignungswerts zu reagieren, so kann die Offerphase für T beendet werden.

      Anderenfalls darf über T noch keine Entscheidung getroffen werden. Die übrigen PEs müssen für T jedoch in den *Loser*-Zustand gewechselt sein und haben entsprechend lockPrio gesetzt. Folglich muss auch das lokale

[1] Dies stellt einen Unterschied zur aggressiven Strategie dar: Diese stoppt eine auf ein anderes PE migrierte Task nur, wenn im selben Hormonzyklus keine andere Task übernommen wurde, zudem wird pro Hormonzyklus höchstens *eine* migrierte Task gestoppt. Die prioritätsbasierte Strategie stoppt dagegen beliebig viele migrierte Tasks pro Hormonzyklus, auch wenn im selben Zyklus andere Tasks übernommen werden. Dies verkürzt die Zeitspanne, in der eine migrierte Task auf zwei PEs gleichzeitig ausgeführt wird, auf das absolut notwendige Minimum, wenn mehrere Tasks gleichzeitig auf andere PEs migriert werden.

PE ebenfalls lockPrio setzen, um zu verhindern, dass es in diesem Zyklus über die Übernahme von Tasks mit niedrigerer Priorität entscheiden kann (und somit unter Umständen gegenüber den anderen PEs einen Vorteil erhielte).

ii) Es soll eine weitere Instanz von T übernommen werden. Dies kann nur eintreten, wenn T den anderen PEs aktuell nicht angeboten wird, da ansonsten Fall (i) zum Tragen käme: Aufgrund der Implementierung des AHS können keine zusätzlichen Instanzen einer Task übernommen werden, während diese sich in einer Offerphase befindet.

In diesem Fall wird daher eine zusätzliche Instanz von T übernommen.

CheckWinner-Unterprozedur Zuletzt soll noch auf die in Abbildung 6.1c dargestellte CheckWinner-Unterprozedur eingegangen werden, da sie sich von der gleichnamigen Prozedur der aggressiven Strategie (Abbildung 4.8c) unterscheidet. Diese Änderung hängt damit zusammen, dass das AHS bisher keinen kontrollierten Stopp von Tasks zugunsten anderer Tasks vorsah, die prioritätsbasierte Task-Decision-Strategie in Überlastsituationen dagegen schon: Konkret führt der Stopp einer Task T' dazu, dass im nächsten Zyklus keine Acceleratoren mehr für eine mit T' verwandte Task T gesendet werden.

Dies kann jedoch – ähnlich wie erhöhte Lastsuppressoren infolge einer Übernahme von T im vorherigen Hormonzyklus – dazu führen, dass ein PE für T in den *Almost-Winner*-Zustand wechseln muss. Wurde T' jedoch gerade gestoppt, um in einer Überlastsituation die Übernahme einer höherprioren Task T zu ermöglichen, so kann dies daher zu einer Verzögerung der Zuteilung von T führen: Im Worst Case kann T daher erst *zwei* Zyklen nach dem Stopp von T' auf einem PE übernommen werden. Würde der *Almost-Winner*-Zustand in einem solchen Fall nicht eingenommen, könnte T dagegen bereits einen Zyklus nach dem Stopp von T' übernommen werden.

Aus diesem Grund wurde gegenüber der aggressiven Task-Decision-Strategie der Abschnitt ab „Suppressors increased *or* accelerators not decreased“ eingefügt: Wurden im Vergleich zum vorherigen Zyklus erhöhte Suppressoren oder keine weggefallenen Acceleratoren für T registriert (also eine Übernahme von T im vorherigen Zyklus auf einem PE oder kein Stopp einer mit T verwandten Task T' im vorherigen Zyklus), so wird wie gehabt der *Almost-Winner*-Zustand eingenommen.

Anderenfalls wird der lokale modifizierte Eignungswert erneut berechnet, wobei jedoch die Acceleratoren aus dem *vorherigen* Zyklus verwendet werden. Wenn dieser größer ist als alle von anderen PEs empfangenen modifizierten Eignungswerte, so wird der *Winner*-Zustand eingenommen, ansonsten der *Almost-Winner*-Zustand.

Die Entscheidung, den *Winner*-Zustand einzunehmen, obwohl das PE basierend auf den aktuellsten Hormonwerten eigentlich nicht zur Übernahme von T qualifiziert ist, verhindert das oben beschriebene Problem: Aufgrund von Taskstopps wegfallende Acceleratoren verzögern die Übernahme von Tasks nun nicht mehr unnötig. Da die übrigen PEs ohnehin in den *Loser*-Zustand wechseln, weil sie davon ausgehen, dass T in diesem Zyklus von einem anderen PE übernommen werden *könnte*, ist es daher

möglich, statt des *Almost-Winner*-Zustands den *Winner*-Zustand einzunehmen, ohne die Korrektheit der Taskverteilung zu gefährden.

Der einzige Nebeneffekt dieser Modifikation gegenüber der CheckWinner-Unterprozedur der aggressiven Task-Decision-Strategie ist ein unter Umständen anderes Verhalten bei der Organbildung, wenn sich die Summe der empfangenen Acceleratoren reduziert. Da die Organbildung jedoch ohnehin nur eine Approximation darstellt, ist dies zunächst nicht weiter problematisch. Zudem verringert sich die Summe der Acceleratoren jedoch ohnehin nur in drei Fällen:

a) Es fällt ein $\mathrm{PE}_{\times}$ aus, es liegt jedoch keine Überlastsituation vor.

   Wird davon ausgegangen, dass sich das System zum Zeitpunkt des Ausfalls in einem stabilen Zustand befand, also alle Tasks zugeteilt waren, so wird die Organbildung durch die modifizierte CheckWinner-Unterprozedur nicht beeinflusst: Mit dem Ausfall des PEs werden für seine Tasks keine Suppressoren mehr gesendet, ebenso werden keine Acceleratoren mehr an verwandte Tasks gesendet. Sobald nun ein anderes PE auf den Wegfall der Suppressoren mit dem Senden eines positiven modifizierten Eignungswerts reagiert, so sind in diesem auch die weggefallenen Acceleratoren bereits nicht mehr enthalten, weshalb die Organbildung sich wie gehabt verhält.

   Befand sich das System dagegen nicht in einem stabilen Zustand, so muss es sich zum Zeitpunkt des Ausfalls in einer Selbstkonfiguration oder -heilung befunden haben, es wurden also positive modifizierte Eignungswerte für (nicht ausgefallene) Tasks ausgetauscht. Zwar verhindert die angesprochene Modifikation der CheckWinner-Unterprozedur nun, dass die Taskzuteilung verzögert wird, wenn Acceleratoren durch den Ausfall verwandter Tasks weggefallen sind. Dafür kann in diesem Fall aber möglicherweise die Güte der Organbildung leiden. Da ein PE-Ausfall jedoch eine Notsituation darstellt, kann die Qualität der Organbildung aber als zweitrangig betrachtet werden, zumal im Rahmen der Selbstoptimierung später versucht werden kann, diese zu verbessern.

   Zudem ist zu erwarten, dass die Zuteilung der betroffenen Tasks trotzdem nicht auf *völlig* ungeeigneten PEs geschehen wird: Die Tasks werden durch die modifizierte CheckWinner-Unterprozedur auf denselben PEs zugeteilt, auf denen sie zugeteilt worden wären, wenn $\mathrm{PE}_{\times}$ nicht ausgefallen wäre. In der Vergangenheit waren diese also die bestgeeigneten PEs für die Tasks, weshalb zu erwarten ist, dass sie auch nach dem PE-Ausfall noch vergleichsweise gut geeignet sind. Zudem befinden sich aufgrund der bisherigen Organbildung in der Nachbarschaft möglicherweise noch andere verwandte Tasks, was ebenfalls die Vermutung nahe liegt, dass die Tasks dennoch auf vergleichsweise gut geeigneten PEs erfolgen wird.

b) Es fällt ein $\mathrm{PE}_{\times}$ aus, es liegt eine Überlastsituation vor.

   Ein solcher Ausfall stellt eine noch größere Notsituation als der vorherige Fall dar. Folglich kann eine bestmögliche Organbildung als zweitranging angesehen werden, da zunächst dafür gesorgt werden muss, dass die wichtigsten Tasks des

Systems weiterhin ausgeführt werden können. Wie zuvor beschrieben, kann die modifizierte CheckWinner-Unterprozedur in diesem Fall jedoch die Taskzuteilung um einen Hormonzyklus beschleunigen.

c) Es wird eine Task im Rahmen einer Offerphase gestoppt, nachdem sie auf ein anderes PE migriert wurde.

Im Gegensatz zu den vorigen Punkten stellt dies keine Notsituation dar. Die Qualität der Organbildung wird in diesem Fall jedoch nur dann beeinträchtigt, wenn sich gleichzeitig weitere Tasks in Offerphasen befinden oder Selbstkonfiguration bzw. -heilung stattfinden. In jedem dieser Szenarien lassen sich jedoch ähnliche Überlegungen wie in Fall a) anstellen, weshalb Tasks im Regelfall dennoch nicht auf *völlig* ungeeigneten PEs zugeteilt werden und zudem später im Rahmen der Selbstoptimierung versucht werden kann, suboptimale Zuteilungen zu verbessern.

6.3.4. Erkennung von Überlastsituationen

Wie zuvor beschrieben, wird eine Situation nur dann als Überlastsituation erkannt (Setzen von missingTask in Abbildung 6.1b), wenn im *No-bidders*-Zustand weder im aktuellen noch im vorherigen Hormonzyklus Übernahmesuppressoren empfangen wurden.

Die Überprüfung auf Suppressoren aus dem vorherigen Zyklus verzögert dabei die Erkennung einer Überlastsituation um einen Hormonzyklus. Dies ist nötig, damit eine Unterlastsituation nicht von einigen PEs fälschlicherweise als Überlastsituation erkannt wird. Abbildung 6.2 skizziert hierzu ein Szenario, in dem $\mathrm{PE}_\times$ in Hormonzyklus i nach dem Senden seiner Hormone ausfällt. PE_1 hat eine Auslastung von 100 %, weshalb es bereits aufgrund seiner eigenen Lastsuppressoren keine positiven modifizierten Eignungswerte für die zuvor auf $\mathrm{PE}_\times$ zugeteilten Tasks senden kann. PE_2 dagegen ist nicht vollständig ausgelastet, sendet aufgrund der von $\mathrm{PE}_\times$ empfangenen Übernahmesuppressoren jedoch ebenfalls keine positiven modifizierten Eignungswerte.

Aufgrund des Ausfalls werden in Zyklus $i+1$ keine Übernahmesuppressoren von $\mathrm{PE}_\times$ mehr empfangen. Nimmt PE_1 dies zum Anlass, die Situation als Überlastsituation zu klassifizieren und daher Tasks zu stoppen (damit es in Zyklus $i+2$ aufgrund der verringerten Last einen positiven modifizierten Eignungswert senden kann), so ist dies eine Fehlentscheidung: PE_2 ist nicht voll ausgelastet und sendet in Zyklus $i+2$ einen positiven modifizierten Eignungswert.

Aus diesem Grund darf ein PE eine Situation erst dann als Überlastsituation erkennen, wenn auch im *vorherigen* Zyklus kein Übernahmesuppressor empfangen wurde. Dies gibt den übrigen (nicht vollständig ausgelasteten) PEs eine Chance, auf den Ausfall von $\mathrm{PE}_\times$ mit dem Senden eines positiven modifizierten Eignungswerts zu reagieren und verhindert daher überflüssige Taskstopps. Das entsprechend geänderte Verhalten von PE_1 ist in Abbildung 6.3 dargestellt (das Verhalten von $\mathrm{PE}_\times$ und PE_2 ist identisch zu Abbildung 6.2 und daher nicht erneut abgebildet).

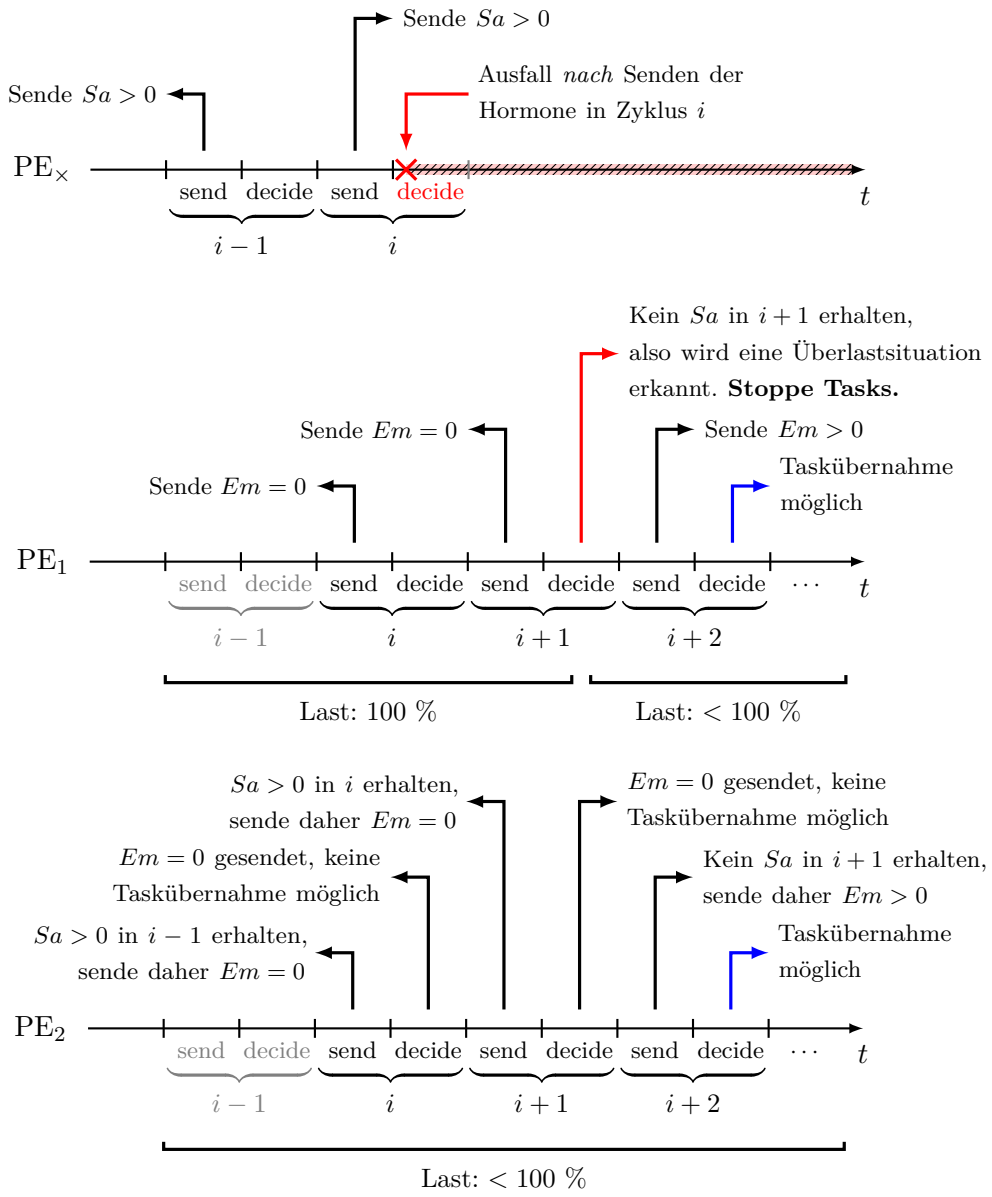


Abbildung 6.2: Unterlastsituation, fälschlicherweise als Überlastsituation erkannt

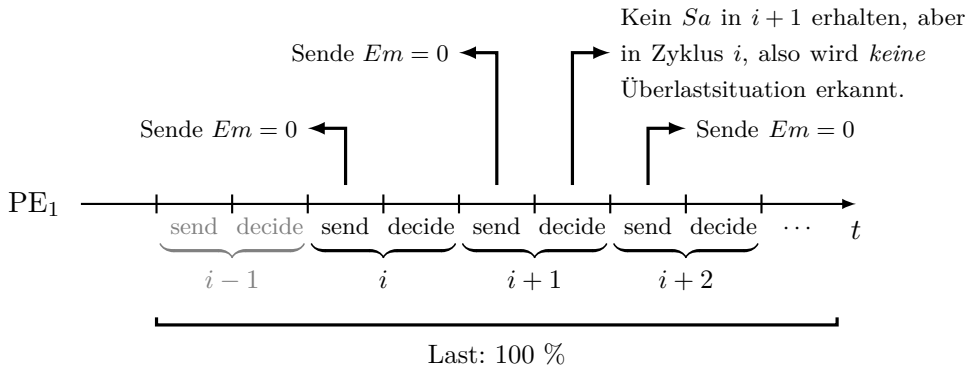


Abbildung 6.3: Geändertes Verhalten von PE_1 gegenüber Abbildung 6.2: Es erfolgt keine fälschliche Erkennung einer Überlastsituation mehr

6.3.5. Besonderheiten

Um die Beschreibung der Funktionsweise der prioritätsbasierten Task-Decision-Strategie abzuschließen, sei an dieser Stelle noch auf zwei Besonderheiten des Offermechanismus hingewiesen:

- Wie bereits erwähnt, werden auf andere PEs migrierte Tasks schneller gestoppt als bei Verwendung der aggressiven Task-Decision-Strategie, um die Zeitspanne, in der migrierte Tasks auf zwei PEs gleichzeitig ausgeführt werden, so kurz wie möglich zu halten.

- Während sich eine Task T in einer Offerphase befindet, kann keine Entscheidung über Tasks niedrigerer Priorität getroffen werden, der Offermechanismus verhindert also insbesondere deren Zuteilung.

  Dies ist das Resultat einer inhärenten Eigenschaft des AHS: Wird für T kein Übernahmesuppressor gesendet, so können die übrigen PEs nicht unterscheiden, ob T auf einem PE zugeteilt ist, das diese Task jedoch gerade zur Migration anbietet, oder ob T aktuell auf keinem PE zugeteilt ist.

  Andererseits berücksichtigt diese Besonderheit die Prioritäten der Tasks in besonderer Weise: Solange versucht wird, eine hochpriore Task auf ein besser geeignetes PE zu migrieren, werden keine niederprioren Tasks zugeteilt. Aus diesem Blickwinkel erscheint dieses Verhalten weniger als Einschränkung denn als Eigenschaft, die für eine zusätzliche Ordnung während der Task(um)verteilung sorgt. Daher wurden die Grundzüge des Offermechanismus unverändert gelassen, was die genannte Besonderheit zur Folge hat.

6.4. Zeitschranken

Nachdem die prioritätsbasierte Task-Decision-Strategie beschrieben wurde, sollen nun ihre Zeitschranken analysiert werden.

6.4.1. Selbstkonfiguration

Wie im Rahmen der Betrachtung der Zeitschranken der aggressiven Task-Decision-Strategie erläutert wurde, bauen die Zeitschranken der Selbstheilung und Selbstoptimierung auf jener der Selbstkonfiguration auf, weshalb diese zuerst betrachtet werden soll. Hierzu nehmen wir an, dass während der Selbstkonfiguration weder Selbstoptimierung noch eine parallele Selbstheilung stattfinden.

6.4.1.1. Identische Prioritäten

Zunächst ergibt sich, dass die prioritätsbasierte Task-Decision-Strategie bezüglich der Selbstkonfiguration denselben Zeitschranken unterliegt wie die aggressive Strategie, wenn alle Prioritäten identisch sind:

Satz 6.6. Die Zeitschranke der Selbstkonfiguration der *aggressiven* Task-Decision-Strategie gelten auch für die *prioritätsbasierte* Strategie, sofern alle Taskprioritäten identisch sind.

Insbesondere dauert die Selbstkonfiguration also höchstens

$$m + \max_{T \in \mathcal{T}} \{\mathcal{T}(T)\} - 1 \text{ Hormonzyklen},$$

wenn $\mathcal{T}$ die Multimenge aller Tasks und $m := |\mathcal{T}|$ die Gesamtzahl aller zuzuteilenden Taskinstanzen bezeichnet.

Beweis. Wenn alle Taskprioritäten identisch sind, ist die Bedingung $\text{prio}(T) \geq \text{lockPrio}$ stets erfüllt (siehe Abbildung 6.1a). Zudem existieren dann niemals Tasks geringerer Priorität, die gestoppt werden könnten, falls wegen des Vorliegens einer Überlastsituation $\text{missingTask} \neq \bot$ gilt.

In Bezug auf die Selbstkonfiguration verhalten sich aggressive und prioritätsbasierte Task-Decision-Strategien daher in diesem Fall identisch, weshalb die Herleitung der Zeitschranke für die aggressive Strategie (Satz 5.18) auch für die prioritätsbasierte Strategie gilt. □

6.4.1.2. Verschiedene Prioritäten

Lässt man zu, dass Tasks verschiedene Prioritäten haben, so muss die Zeitschranke dagegen verallgemeinert werden:

Satz 6.7. Es bezeichne $\mathcal{T}$ die Multimenge aller Tasks und $\mathcal{T}(T)$ für jedes $T \in \mathcal{T}$ die Anzahl der zuteilbaren Instanzen von T. Zudem bezeichne $m := |\mathcal{T}|$ die Gesamtzahl aller zuzuteilenden Taskinstanzen.

Ferner bezeichne P die Menge aller Prioritäten sowie $\mathcal{T}_p \subseteq \mathcal{T}$ die induzierte Teilmultimenge aller Tasks mit Priorität p, wobei alle $\mathcal{T}_p$ paarweise disjunkt sind.

Dann dauert die Selbstkonfiguration höchstens

$$m + \sum_{p \in P} \left(\max_{T \in \mathcal{T}_p} \{\mathcal{T}(T)\} \right) - 1 \text{ Hormonzyklen}.$$

Beweis. Zunächst sei darauf hingewiesen, dass alle Instanzen einer mehrfach zuteilbaren Task T zwangsläufig dieselbe Priorität besitzen.

Es werden alle Prioritätslevel nacheinander zugeteilt, wobei für die Zuteilung aller Tasks von Prioritätslevel i gemäß Satz 6.6 höchstens

$$|\mathcal{T}_i| + \max_{T \in \mathcal{T}_i}\{\mathcal{T}(T)\} - 1$$

Hormonzyklen benötigt werden.

Zusätzlich tritt jeweils ein Wartezyklus auf, bevor im Anschluss an die Zuteilung der letzten Task von Prioritätslevel i die Zuteilung der ersten Task von Prioritätslevel $i+1$ geschehen kann. Der Grund hierfür ist vergleichbar mit jenem für das Auftreten von Wartezyklen bei mehrfach zuteilbaren Tasks (siehe Kapitel 5): Sobald die letzte Taskinstanz T eines Prioritätslevels auf einem PE_j zugeteilt wurde, wird PE_j Übernahmesuppressoren für T an sich selbst sowie alle anderen PEs aussenden, um weitere Übernahmen von T zu verhindern. Da diese auf den anderen PEs jedoch erst mit einem Hormonzyklus Verzögerung wirken, werden diese PEs im nächsten Zyklus dennoch einen positiven modifizierten Eignungswert für T senden. Als Resultat wird ein PE_k für T in den *Almost-Winner*-Zustand wechseln; PE_j sowie alle übrigen PEs wechseln dagegen in den *Loser*-Zustand. Beide Zustände führen jedoch dazu, dass lockPrio auf $\mathrm{prio}(T)$ gesetzt wird, was die Zuteilung von Tasks mit niedrigerer Priorität verhindert. Da T die letzte zuzuteilende Taskinstanz von Prioritätslevel i war, ist der auf deren Zuteilung folgende Hormonzyklus ein Wartezyklus, da keine weitere Taskinstanz zugeteilt werden kann; erst im darauffolgenden Zyklus kann die Übernahme der ersten Task des nächsten Prioritätslevels erfolgen.

Die einzige Ausnahme hiervon bildet das letzte Prioritätslevel: Nachdem die letzte Taskinstanz dieses Levels zugeteilt wurde, sind alle Taskinstanzen vergeben und die Selbstkonfiguration somit beendet. In diesem Fall muss also kein zusätzlicher Wartezyklus berücksichtigt werden, da kein weiteres Prioritätslevel (und damit keine weitere zuzuteilende Task) existiert.

In der Summe ergeben sich daher

$$\begin{aligned}
&\sum_{p \in P} \left(|\mathcal{T}_p| + \max_{T \in \mathcal{T}_p}\{\mathcal{T}(T)\} - 1 + 1 \right) - 1 \\
&\quad = \sum_{p \in P} |\mathcal{T}_p| + \sum_{p \in P} \left(\max_{T \in \mathcal{T}_p}\{\mathcal{T}(T)\} \right) - 1 \\
&\quad = m + \sum_{p \in P} \left(\max_{T \in \mathcal{T}_p}\{\mathcal{T}(T)\} \right) - 1 \text{ Hormonzyklen.}
\end{aligned}$$

□

Bemerkung 6.8. Wenn nur ein einziges PE existiert, treten keine Wartezyklen auf, da kein anderes PE existiert, das aufgrund der verzögert wirkenden Suppressoren in den *Almost-Winner*-Zustand wechselt (siehe Kapitel 5.5.1 und 5.5.2).

Dies kann für einige Selbstheilungsszenarien berücksichtigt werden, um Zeitschranken besser abschätzen zu können. Im Falle der Selbstkonfiguration dagegen wird implizit

angenommen, dass mehr als ein PE zur Verfügung steht: Anderenfalls wäre bereits ein einiger PE-Ausfall kritisch für das Gesamtsystem und die Notwendigkeit einer *verteilten* Middleware wie dem AHS äußerst fragwürdig.

6.4.1.3. Keine mehrfach zuteilbaren Tasks

Lässt man keine mehrfach zuteilbaren Tasks zu, so vereinfachen sich die Zeitschranken:

Satz 6.9. Es bezeichne $\mathcal{T}$ die *Menge* aller Tasks, jede Task sei also genau einmal zuteilbar. Zudem bezeichne $m := |\mathcal{T}|$ die Gesamtzahl aller zuzuteilenden Tasks. Ferner bezeichne P die Menge aller Prioritäten.

Dann ist die Selbstkonfiguration nach höchstens $m + |P| - 1$ Hormonzyklen beendet.

Beweis. Es bezeichne $\mathcal{T}_i \subseteq \mathcal{T}$ die Teilmenge aller Tasks mit Priorität i.

Fasst man nun $\mathcal{T}$ als *Multimenge* mit $\mathcal{T}(T) = 1$ für alle $T \in \mathcal{T}$ auf, so vereinfacht sich die Abschätzung aus Satz 6.7 zu

$$\begin{aligned} & m + \sum_{p \in P} \left(\max_{T \in \mathcal{T}_p} \{\mathcal{T}(T)\} \right) - 1 \\ &= m + \sum_{p \in P} (1) - 1 \\ &= m + |P| - 1 \text{ Hormonzyklen.} \end{aligned}$$

□

6.4.1.4. Alle Prioritäten verschieden

Die Dauer der Selbstkonfiguration wird maximal, wenn alle Prioritäten verschieden sind: In diesem Fall besteht jedes Prioritätslevel aus nur einer einzigen Task und die Zeitschranke degeneriert zu $2m - 1$ Hormonzyklen:

Satz 6.10. Es bezeichne $\mathcal{T}$ die Multimenge aller Tasks und $\mathcal{T}(T)$ für jedes $T \in \mathcal{T}$ die Anzahl der zuteilbaren Instanzen von T. Zudem bezeichne $m := |\mathcal{T}|$ die Gesamtzahl aller zuzuteilenden Taskinstanzen.

Ferner bezeichne P die Menge aller Prioritäten sowie $\mathcal{T}_p \subseteq\!\!\!\subseteq \mathcal{T}$ die induzierte Teilmultimenge aller Tasks mit Priorität p, wobei alle $\mathcal{T}_p$ paarweise disjunkt sind.

Dann gilt: Wenn $|\mathrm{supp}(\mathcal{T}_p)| = 1$ für alle $p \in P$, jedes Prioritätslevel i also aus nur einer einzigen, aber möglicherweise mehrfach zuteilbaren Task T_i besteht, so dauert die Selbstkonfiguration höchstens $2m - 1$ Hormonzyklen.

Beweis. Gemäß Satz 6.7 dauert die Selbstkonfiguration höchstens

$$m + \sum_{p \in P} \left(\max_{T \in \mathcal{T}_p} \{\mathcal{T}(T)\} \right) - 1 \text{ Hormonzyklen.}$$

Da jedes Prioritätslevel aus genau einer Task besteht, ergibt sich

$$= m + \sum_{p \in P} \mathcal{T}(T_p) - 1$$
$$= 2m - 1. \qquad \square$$

Es sei darauf hingewiesen, dass dieses Resultat unabhängig davon gilt, ob mehrfach zuteilbare Tasks existieren (letzteres folgt mit $|P| = m$ auch aus Satz 6.9).

6.4.1.5. Verbot mehrfach zuteilbarer Tasks

Im weiteren Verlauf dieser Dissertation werden wir jedoch generell keine mehrfach zuteilbaren Tasks mehr erlauben: Wie zuvor gesehen, führt deren Betrachtung zu vielen Spezialfällen. Im Fall des „klassischen“ AHS ohne Prioritäten konnten diese Spezialfälle als Anlass verwendet werden, die Besonderheiten des *Almost-Winner*-Zustands genauer zu betrachten. Im Rahmen der soeben betrachteten prioritätsbasierten Selbstkonfiguration war es zudem vergleichsweise einfach möglich, mehrfach zuteilbare Tasks in die Betrachtungen einzuschließen und die Ergebnisse schließlich auf einfach zuteilbare Tasks zu spezialisieren.

Angesichts der Beobachtung, dass mehrfach zuteilbare Tasks im AHS erfahrungsgemäß nur höchst selten verwendet werden, wird aus Platzgründen auf eine weitere Betrachtung der sich ergebenden Spezialfälle verzichtet. Stattdessen wird im Folgenden stets vorausgesetzt, dass jede Task nur einfach zugeteilt werden kann.

6.4.1.6. Zuteilungsdauer

Die bisher betrachteten Zeitschranken bezogen sich auf die Zeit, welche zur Zuteilung *aller* Tasks benötigt wird. Aus Sicht des „klassischen“ AHS ohne Prioritäten war dies das einzig sinnvolle Maß, die Einführung von Prioritäten ermöglicht jedoch die Betrachtung einer weiteren Größe:

Konkret drückt eine Beziehung $\text{prio}(T_1) > \text{prio}(T_2)$ aus, dass die Task T_1 als „wichtiger“ als T_2 betrachtet werden kann. Es liegt daher nahe, eine Zeitspanne abzuschätzen, nach deren Ablauf garantiert werden kann, dass T_1 im System zugeteilt ist. Im Gegensatz zur gesamten Dauer der Selbstkonfiguration muss dabei keine Rücksicht auf die niedrigpriore Task T_2 genommen werden, weshalb auf diese Weise bessere Garantien für hochpriore Tasks gegeben werden können.

Hierzu führen wir den Begriff der *Zuteilungsdauer* ein. Deren exakte Definition unterscheidet sich im Detail jedoch für Selbstkonfiguration und Selbstheilung, im Folgenden sei daher zunächst die Definition im Kontext der Selbstkonfiguration angegeben:

Zuteilungsdauer (Selbstkonfig.), $\mathrm{D}(T)$

Definition 6.11. Die *Zuteilungsdauer* $\mathrm{D}(T)$ einer Task $T \in \mathcal{T}$ sei im Kontext der Selbstkonfiguration definiert als die Länge der Zeitspanne, nach deren Ablauf garantiert werden kann, dass T im Rahmen der Selbstkonfiguration einem PE zugeteilt wurde.

Nun ist es möglich, die bisherige Zeitschranke der vollständigen Selbstkonfiguration

aller Tasks zu verallgemeinern, um stattdessen die Zuteilungsdauer einzelner Tasks abzuschätzen:

Satz 6.12. Es bezeichne P die Menge aller Prioritäten sowie $\mathcal{T}_p \subseteq \mathcal{T}$ die Menge aller Tasks mit Priorität p.

Dann gilt

$$\mathrm{D}(T) \leq \sum_{\substack{p \in P \\ p \geq \mathrm{prio}(T)}} (|\mathcal{T}_p| + 1) - 1 \text{ Hormonzyklen.}$$

Beweis. Eine Task T wird spätestens als letzte Task ihres Prioritätslevels zugeteilt. Folglich kann die Selbstheilungsdauer als Dauer einer Selbstkonfiguration aller Tasks abgeschätzt werden, deren Priorität mindestens so hoch wie $\mathrm{prio}(T)$ ist.

Dies sind jedoch gerade alle Tasks der Vereinigungsmenge

$$M \coloneqq \bigcup_{\substack{p \in P \\ p \geq \mathrm{prio}(T)}} \mathcal{T}_p.$$

Gemäß Satz 6.10 benötigt die Zuteilung aller dieser Tasks jedoch höchstens $|M|+|P'|-1$ Hormonzyklen, wobei $|P'|$ die Anzahl der beteiligten Prioritäten darstellt. Damit ergibt sich jedoch

$$\begin{aligned}
\mathrm{D}(T) &\leq |M| + |P'| - 1 \\
&= \sum_{\substack{p \in P \\ p \geq \mathrm{prio}(T)}} |\mathcal{T}_p| + \sum_{\substack{p \in P \\ p \geq \mathrm{prio}(T)}} 1 \quad - 1 \\
&= \sum_{\substack{p \in P \\ p \geq \mathrm{prio}(T)}} (|\mathcal{T}_p| + 1) - 1 \text{ Hormonzyklen.}
\end{aligned}$$

□

Bemerkung 6.13. Da die Reihenfolge der Zuteilung von Tasks derselben Priorität nicht definiert ist (wie dies bei Verwendung der aggressiven Task-Decision-Strategie für *alle* Tasks der Fall ist), muss allen Tasks derselben Priorität dieselbe Zuteilungsdauer zugewiesen werden. Als Resultat kann Satz 6.12 daher auch dazu benutzt werden, um abzuschätzen, nach wie vielen Hormonzyklen alle Tasks eines bestimmten Prioritätslevels zugeteilt sind, statt dies für eine spezifische Task zu bestimmen.

Es sei ferner darauf hingewiesen, dass die Abschätzungen von Satz 6.10 und Satz 6.12 identisch sind, wenn die betrachtete Task T zum niedrigsten Prioritätslevel gehört: In diesem Fall entspricht die Zuteilungsdauer gerade der Dauer der gesamten Selbstkonfiguration.

6.4.2. Selbstoptimierung

Die zur Selbstoptimierung nötige Zeit kann auch bei Verwendung der prioritätsbasierten Task-Decision-Strategie beschränkt werden:

Satz 6.14. Es bezeichne m die Anzahl der im System vorhandenen Tasks, die alle nur einfach zuteilbar seien, sowie P die Menge ihrer Prioritäten.

Dann dauert die Selbstoptimierung höchstens

$$m + |P| - 1 \text{ Hormonzyklen.}$$

Beweis. Die Argumentation ist identisch zu jener aus dem Beweis von Satz 5.6, da die Grundprinzipien des Offermechanismus gegenüber der aggressiven Task-Decision-Strategie nicht verändert wurden:[1] Im Worst Case beginnen zur gleichen Zeit Offerphasen für alle Taskinstanzen, was einer vollständigen Selbstkonfiguration entspricht. Daher folgt mit Satz 6.9 die Behauptung. □

6.4.3. Selbstheilung

Wie für die aggressive Task-Decision-Strategie lässt sich die Dauer der Selbstheilung nach dem Ausfall eines PEs auch bei Verwendung der prioritätsbasierten Task-Decision-Strategie beschränken. Im Folgenden wird hierzu angenommen, dass eine Unterlastsituation vorliegt, die Kapazität der verbleibenden PEs also ausreicht, um alle ausgefallenen Tasks zu übernehmen. Kapitel 8 wird sich dagegen im Detail mit den Zeitschranken der Selbstheilung befassen, falls eine Überlastsituation vorliegt.

Für Unterlastsituationen lässt sich nun die Dauer der gesamten Selbstheilung beschränken. Die Zeitschranke der Selbstkonfiguration nahm an, dass mindestens zwei PEs existieren (siehe Bemerkung 6.8), da ansonsten keine Selbstheilung möglich ist. Im Rahmen der Selbstheilung an sich ist es jedoch sinnvoll, auch den Fall zu betrachten, dass nach einem Ausfall nur noch ein einziges PE verbleibt:

Satz 6.15. Es bezeichne m_f die Anzahl der Tasks, die zuvor auf dem ausgefallenen $\text{PE}_\times$ zugeteilt waren und $|P_f|$ die Menge der Prioritäten dieser Tasks. Es seien keine mehrfach zuteilbaren Tasks erlaubt. Ferner haben die $v \geq 1$ verbleibenden PEs genügend Kapazitäten, um alle m_f Tasks zusätzlich zu übernehmen und es finden während der Selbstheilung weder Selbstkonfiguration, Selbstoptimierung noch eine weitere Selbstheilung statt.

Dann dauert die Selbstheilung höchstens

$$m_f + (|P_f| - 1) \cdot [v > 1] + a \text{ Hormonzyklen,}$$

wobei a die zur Erkennung des Ausfalls höchstens notwendige Zeit beschreibt.

Beweis. Nach dem Erkennen des Ausfalls von $\text{PE}_\times$ nach spätestens a Hormonzyklen entspricht die Selbstheilung gerade einer Selbstkonfiguration mit m_f Taskinstanzen. Der Fall $v > 1$ folgt daher mit Satz 6.9. Bleibt dagegen nur noch ein einziges PE übrig,

[1] Wie zuvor angemerkt, können erfolgreich migrierte Tasks auf dem ursprünglichen PE schneller gestoppt werden als bei Verwendung der aggressiven Strategie. Zudem können keine Entscheidungen über eine Task T_1 getroffen werden, während sich eine Task T_2 mit $\text{prio}(T_2) > \text{prio}(T_1)$ in einer Offerphase befindet. Beide Unterschiede zur aggressiven Strategie beeinflussen jedoch die Gesamtdauer der Selbstoptimierung nicht.

so fallen gemäß Bemerkung 6.8 keine Wartezyklen an, woraus der Fall $v = 1$ folgt, da dann nach der Erkennung des Ausfalls lediglich m_f Hormonzyklen zur Verteilung von m_f Tasks benötigt werden. □

Bemerkung 6.16. Die Werte der Konstanten $a_{\min}$ und a sind nur von den Eigenschaften der leichtgewichtigen Hormonauswertung abhängig, weshalb Satz 5.11 auch für die prioritätsbasierte Task-Decision-Strategie gültig ist; folglich gelten weiterhin $a_{\min} = 1$ und $a = 2$.

Im Allgemeinen ist jedoch nur schwer vorhersagbar, welche (bzw. wie viele) Prioritäten die auf einem ausfallenden PE zugeteilten Tasks haben. Daher kann es unter Umständen sinnvoll sein, stattdessen anzunehmen, dass alle diese Prioritäten verschieden sind:

Korollar 6.17. Es gelten die Voraussetzungen aus Satz 6.15, alle auf $\mathrm{PE}_\times$ zugeteilten Tasks haben jedoch verschiedene Prioritäten.

Dann dauert die Selbstheilung höchstens

$$m_f + (m_f - 1) \cdot [v > 1] + a \text{ Hormonzyklen.}$$

Beweis. Folgt mit $|P_f| = m_f$ unmittelbar aus Satz 6.15. □

6.4.3.1. Zuteilungsdauer

Analog zur Selbstkonfiguration lässt sich auch für die Selbstheilung eine Zuteilungsdauer definieren, um nicht die Dauer einer gesamten Selbstheilung betrachten zu müssen, sondern für einzelne Tasks besser abschätzen zu können, wann diese wieder einem PE zugeteilt sind:

Zuteilungsdauer (Selbstheilung), $\mathrm{D}(T)$

Definition 6.18. Es bezeichne $\mathcal{T}_f$ die Menge der Tasks, die zuvor auf dem ausgefallenen $\mathrm{PE}_\times$ zugeteilt waren. Dann sei die *Zuteilungsdauer* $\mathrm{D}(T)$ einer Task $T \in \mathcal{T}_f$ im Kontext der Selbstheilung in einer Unterlastsituation definiert als die Länge der Zeitspanne, nach deren Ablauf garantiert werden kann, dass T aufgrund der Selbstheilung wieder einem PE zugeteilt ist.

Satz 6.19. Es bezeichne $\mathcal{T}_f$ die Menge der Tasks, die zuvor auf dem ausgefallenen $\mathrm{PE}_\times$ zugeteilt waren und es bezeichne P_f die Menge der Prioritäten dieser Tasks. Ferner seien alle Tasks nur einfach zuteilbar und es bezeichne

$$\mathcal{T}_{f,p} \coloneqq \{T \in \mathcal{T}_f \mid \mathrm{prio}(T) = p\}$$

die Teilmenge aller ausgefallenen Tasks mit Priorität p.

Zudem haben die $v \geq 1$ verbleibenden PEs genügend Kapazitäten, um alle $|\mathcal{T}_f|$ Tasks zusätzlich zu übernehmen und es finden während der Selbstheilung weder Selbstkonfiguration, Selbstoptimierung noch eine weitere Selbstheilung statt.

Dann gilt für alle $T \in \mathcal{T}_f$

$$\mathrm{D}(T) \leq \sum_{\substack{p \in P_f \\ p \geq \mathrm{prio}(T)}} (|\mathcal{T}_{f,p}| + [v > 1]) - [v > 1] + a.$$

Beweis. Folgt mit ähnlicher Argumentation wie in Satz 6.12 und Satz 6.15:
T wird im Rahmen der Selbstheilung spätestens als letzte Task ihres Prioritätslevels zugeteilt. Es sind zwei Fälle zu unterscheiden:

a) $v > 1$: Es fallen Wartezyklen an. In diesem Fall kann die Zuteilungsdauer von T wie in Satz 6.12 abgeschätzt werden, es muss jedoch zusätzlich die additive Konstante a berücksichtigt werden:

$$\begin{aligned} \mathrm{D}(T) &\leq \sum_{\substack{p \in P_f \\ p \geq \mathrm{prio}(T)}} (|\mathcal{T}_{f,p}| + 1) - 1 + a \\ &= \sum_{\substack{p \in P_f \\ p \geq \mathrm{prio}(T)}} (|\mathcal{T}_{f,p}| + [v > 1]) - [v > 1] + a. \end{aligned}$$

b) $v = 1$: Es fallen *keine* Wartezyklen an. In diesem Fall dauert die Zuteilung aller Tasks, die eine Priorität haben, die größer oder gleich $\mathrm{prio}(T)$ ist, daher höchstens

$$\sum_{\substack{p \in P_f \\ p \geq \mathrm{prio}(T)}} |\mathcal{T}_{f,p}| \text{ Hormonzyklen.} \tag{6.1}$$

Damit beträgt die Zuteilungsdauer von T jedoch höchstens

$$\begin{aligned} \mathrm{D}(T) &\leq \sum_{\substack{p \in P_f \\ p \geq \mathrm{prio}(T)}} |\mathcal{T}_{f,p}| + a \\ &= \sum_{\substack{p \in P_f \\ p \geq \mathrm{prio}(T)}} (|\mathcal{T}_{f,p}| + 0) - 0 + a \\ &= \sum_{\substack{p \in P_f \\ p \geq \mathrm{prio}(T)}} (|\mathcal{T}_{f,p}| + [v > 1]) - [v > 1] + a. \end{aligned}$$

□

Wie bereits zuvor angemerkt, kann im Allgemeinen nicht vorhergesagt werden, welche Tasks auf $\mathrm{PE}_\times$ zugeteilt sind, daher sind auch $\mathcal{T}_f$ und P_f im Allgemeinen nicht vorhersagbar. Folglich ist die Analyse der Zuteilungsdauer im Falle der Selbstheilung nur dann sinnvoll durchführbar, wenn die konkrete Taskmenge eines Systems sowie die maximalen Kapazitäten aller PEs bekannt sind und anhand dieser Informationen der Worst Case bestimmt werden kann.

6.5. Zusammenfassung

In diesem Kapitel wurde eine prioritätsbasierte Task-Decision-Strategie für das AHS vorgestellt. Diese ermöglicht es, Tasks statisch zu priorisieren, um die Reihenfolge der Taskzuteilung steuern zu können. Damit ist es insbesondere möglich, mithilfe der

Zuteilungsdauer den Zeitpunkt der Zuteilung bestimmter Tasks besser abzuschätzen als durch die Dauer einer vollständigen Selbstkonfiguration.

Vergleicht man die prioritätsbasierte Task-Decision-Strategie mit der bisher verwendeten aggressiven Strategie, so stellt man fest, dass beiden Strategien identische Zeitschranken zugrunde liegen, sofern alle Tasks die gleiche Priorität besitzen. Dies ist darauf zurückzuführen, dass beide Strategien im Kern ähnlich funktionieren.

Weist man den Tasks dagegen verschiedene Prioritäten zu, so führt dies zu einem schlechteren Zeitverhalten. Selbst im Worst-Case, in dem *alle* Tasks verschiedene Prioritäten haben, liegt zwischen den Zeitschranken beider Strategien jedoch ein Faktor, der kleiner als 2 ist;[1] wie im Fall der aggressiven Strategie werden für Selbstkonfiguration, Selbstoptimierung und Selbstheilung also weiterhin $\mathcal{O}(m)$ Hormonzyklen benötigt, wenn m die Anzahl (neu) zuzuteilender Tasks bezeichnet. Es sei an dieser Stelle darauf hingewiesen, dass in Anhang E ein Überblick über die wichtigsten hergeleiteten Zeitschranken zu finden ist.

Bisher wurde jedoch die Selbstheilung in *Überlastsituationen* nicht betrachtet. Kapitel 8 wird sich im Detail mit solchen Situationen befassen: In diesen ist es der prioritätsbasierten Task-Decision-Strategie im Gegensatz zur aggressiven Strategie möglich, zumindest die wichtigsten Tasks am Laufen zu halten. Wir werden sehen, dass auch bei Überlast noch die Einhaltung bestimmter Zeitschranken garantiert werden kann, was im Fall der aggressiven Task-Decision-Strategie nicht möglich ist.

Zunächst wollen wir uns im folgenden Kapitel jedoch damit befassen, die soeben hergeleiteten Zeitschranken im Rahmen einer Evaluation experimentell zu überprüfen.

[1] Für Selbstkonfiguration und Selbstoptimierung ergibt sich ein Faktor von $2 - 1/m$, für die Selbstheilung bei mehr als einem verbleibenden PE ein Faktor von $2 - (1 + a)/(m_f + a)$ und für die Selbstheilung bei genau *einem* verbleibenden PE unterliegen beide Strategien denselben Zeitschranken.

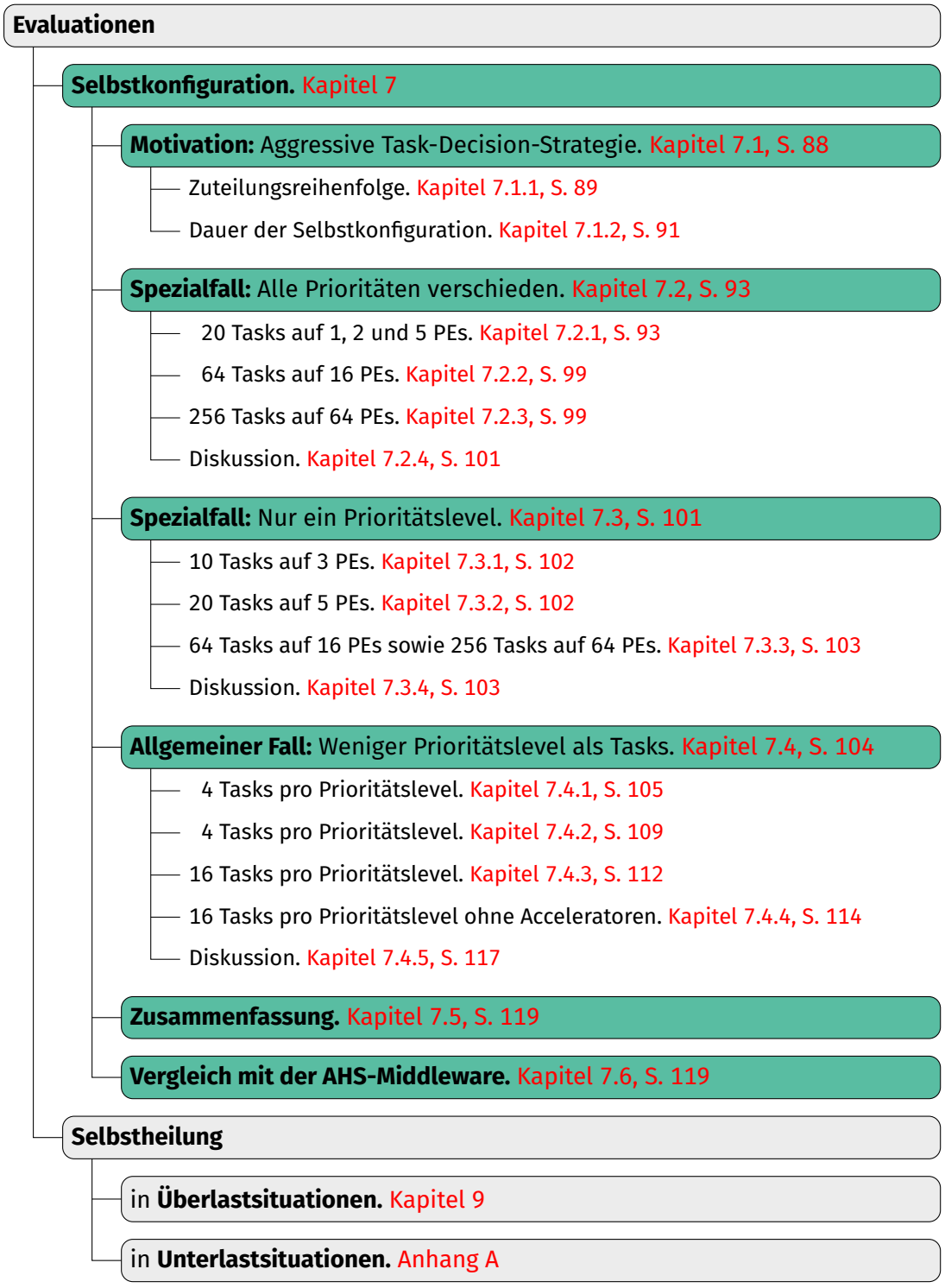


Abbildung 7.1: Überblick über die Evaluationen der Selbstkonfiguration

7 Evaluation der prioritätsbasierten Taskverteilung

In diesem Kapitel sollen das Verhalten sowie die theoretisch hergeleiteten Zeitschranken der prioritätsbasierten Task-Decision-Strategie experimentell überprüft werden. Im Allgemeinen existiert jedoch eine Vielzahl möglicher Systemkonfigurationen, da viele verschiedene Parameter variiert werden können, beispielsweise

- die Anzahl und die Nachbarschaftsverhältnisse der PEs,
- die Anzahl der Tasks und deren individuelle Prioritäten,
- die Verwandtschaftsverhältnisse der Tasks sowie
- die Hormonwerte der einzelnen PEs für die Tasks, aus denen sich unter anderem ergibt, wie viele Tasks jedes PEs höchstens übernehmen kann.

Daher wurde die Evaluation auf einige exemplarische Systemkonfigurationen beschränkt, die insbesondere verschiedene Skalierungen des künstlichen Hormonsystems abdecken.

Ferner wurde sich zunächst darauf beschränkt, primär die Selbstkonfiguration zu evaluieren, da dieser die elementarste Zeitschranke zugrunde liegt. Wie im vorigen Kapitel argumentiert wurde, basieren die für Selbstoptimierung und Selbstheilung hergeleiteten Zeitschranken auf jener der Selbstkonfiguration. Nach der Erkennung des Ausfalls eines PEs entspricht eine Selbstheilung gerade einer Selbstkonfiguration der mit dem PE ausgefallenen Tasks. Im Rahmen der Selbstoptimierung dagegen werden Tasks periodisch anderen PEs zur Migration angeboten. Dies entspricht aber wiederum einer Selbstkonfiguration der so angebotenen Tasks: Diese werden bei ihrer neuerlichen Übernahme entweder weiterhin auf ihrem ursprünglichen PE ausgeführt oder auf ein neues PE migriert; in beiden Fällen ist die Selbstoptimierung für diese Task beendet. Beginnt im Worst Case die Selbstoptimierung gleichzeitig für *alle* Tasks, entspricht dies jedoch gerade einer vollständigen Selbstkonfiguration. Folglich sind insbesondere die Zeitschranken von Selbstoptimierung und Selbstkonfiguration identisch, wie Satz 5.21 bzw. [BP12] für die aggressive und Satz 6.14 für die prioritätsbasierte Strategie feststellen.

Da die prioritätsbasierte Task-Decision-Strategie die grundlegenden Mechanismen von Selbstoptimierung und Selbstheilung von der aggressiven Strategie übernimmt, sind hier daher – abgesehen von der Unterstützung von Prioritäten – keine signifikanten Abweichungen im Verhalten zu erwarten, was auch in [HB21a] bestätigt werden konnte. Aus diesem Grund wurde im Rahmen dieser Dissertation auf eine Evaluation der Selbstoptimierung verzichtet. Stattdessen wird im Folgenden die für alle übrigen Mechanismen fundamentale Selbstkonfiguration betrachtet. In Anhang A wird zudem die Selbstheilung näher untersucht, um insbesondere den Einfluss der Erkennungszeit eines PE-Ausfalls miteinzubeziehen.

Wie bereits in Kapitel 4.5.3 erläutert, wurden die Evaluationen primär im Hormonsimulator durchgeführt, zur Überprüfung der Ergebnisse wird in Abschnitt 7.6 jedoch vergleichend auch die Middleware betrachtet. Folglich sind die PEs in allen Evaluationsszenarien in einem zweidimensionalen Gitter angeordnet, da dies die vom Simulator vorgesehene Topologie darstellt. Wie in Kapitel 4.5.1 erläutert sind je zwei PEs in dieser Topologie genau dann benachbart, wenn sie horizontal, vertikal oder diagonal aneinandergrenzen. Sofern nicht anders angegeben, waren alle im Rahmen der einzelnen Evaluationsszenarien verwendeten PEs jeweils gleichartig, wiesen also für dieselben Tasks auch identische Hormonwerte auf.

Des Weiteren wurden keine mehrfach zuteilbaren Tasks betrachtet, wie bereits in Kapitel 6.4.1.5 begründet wurde. Aufgrund der Verwendung von Acceleratorschranken (siehe Kapitel 4.4.1) konnten für die Evaluationen vergleichsweise starke Acceleratoren eingesetzt werden. Dies ermöglicht eine ausgeprägte Bildung virtueller Organe, ohne dabei jedoch die Systemstabilität negativ zu beeinträchtigen.

Den Voraussetzungen der hergeleiteten Zeitschranken entsprechend finden während der Selbstkonfiguration weder Selbstoptimierung noch Selbstheilung statt. Zudem sei vereinbart, dass eine Task T_i in diesem Kapitel der Einfachheit halber nur mit ihrem Index i bezeichnet werde.

Abbildung 7.1 gibt einen groben Überblick über die durchgeführten Evaluationen, die im Folgenden im Detail vorgestellt werden. Einige der Evaluationsergebnisse wurden bereits in [HB21a] veröffentlicht.

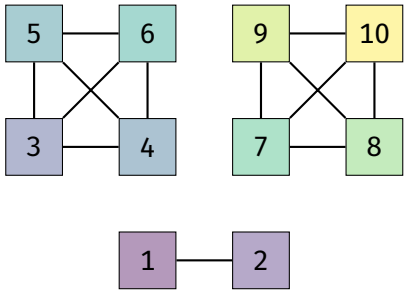
7.1. Motivation: Vergleich der aggressiven mit der prioritätsbasierten Task-Decision-Strategie

Zunächst soll ein Evaluationsszenario die Notwendigkeit einer prioritätsbasierten Task-Decision-Strategie motivieren. Hierzu wurde das in Abbildung 7.2 dargestellte Szenario einmal mit der aggressiven und einmal mit der prioritätsbasierten Task-Decision-Strategie simuliert. In diesem Szenario existieren $m = 10$ Tasks, die alle verschiedene Prioritäten besitzen. Jedes der drei gleichartigen PEs, die in einem zur Linientopologie degenerierten 1×3-Gitter angeordnet sind, kann maximal fünf Tasks übernehmen.

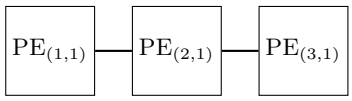
Diagramminterpretation

Der zeitliche Verlauf der Selbstkonfiguration ist in Abbildung 7.3 für beide Task-Decision-Strategien aufgetragen. Hierbei werden die Tasks stets von unten nach oben in der Reihenfolge *absteigender* Prioritäten gestapelt. Diese Darstellungsform ermöglicht eine schnelle qualitative Überprüfung der Zuteilungsreihenfolge: Die Tasks werden nur dann in der korrekten Reihenfolge absteigender Prioritäten zugeteilt, wenn der Farbbalken jeder neu zugeteilten Task von Beginn ihrer Zuteilung an auf derselben Höhe liegt, also nicht nachträglich durch die Zuteilung von Tasks niedrigerer Priorität nach oben geschoben wird.

Beim Vergleich der beiden Diagramme wird daher sofort ersichtlich, dass nur die prioritätsbasierte Task-Decision-Strategie alle Tasks in der korrekten Reihenfolge zuteilt, da hier alle Farbbalken ab dem jeweiligen Zuteilungszeitpunkt von links nach rechts durchgängig sind. Im Falle der aggressiven Task-Decision-Strategie ist dies nicht



(a) Tasks und ihre Verwandtschaftsbeziehungen. Jede Task i hat die Priorität $\text{prio}(i) = i$.



(b) PEs und ihre Nachbarschaftsbeziehungen

Abbildung 7.2: Evaluationsszenario 1: 10 Tasks verschiedener Prioritäten auf 3 PEs

Tabelle 7.1: Reihenfolge der Taskzuteilung in Evaluationsszenario 1 bei Verwendung der aggressiven Task-Decision-Strategie gemäß Abbildung 7.3a

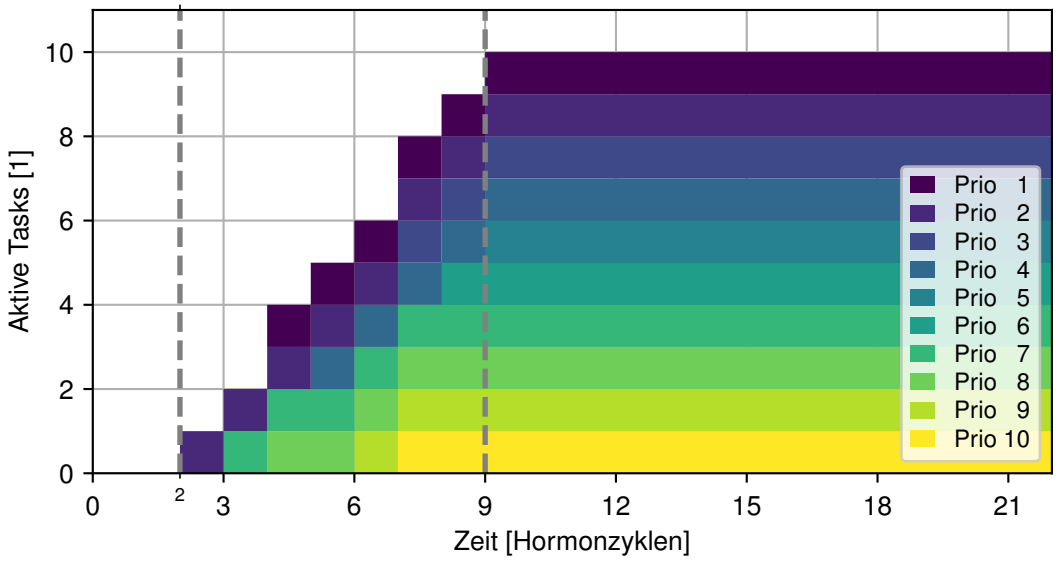
Hormonzyklus	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Zugeteilte Tasks	2	7	1, 8	4	9	3, 10	6	5

der Fall: Zur Einhaltung der korrekten Stapelungsreihenfolge müssen in der grafischen Darstellung häufig bestehende Farbbalken nachträglich zerteilt und nach oben geschoben werden, was eine Verletzung der Zuteilungsreihenfolge bedeutet, die anhand der Taskprioritäten festgelegt ist. Quantitativ lässt sich dagegen erkennen, dass es im Falle der prioritätsbasierten Strategie länger dauert, bis alle Tasks zugeteilt sind.

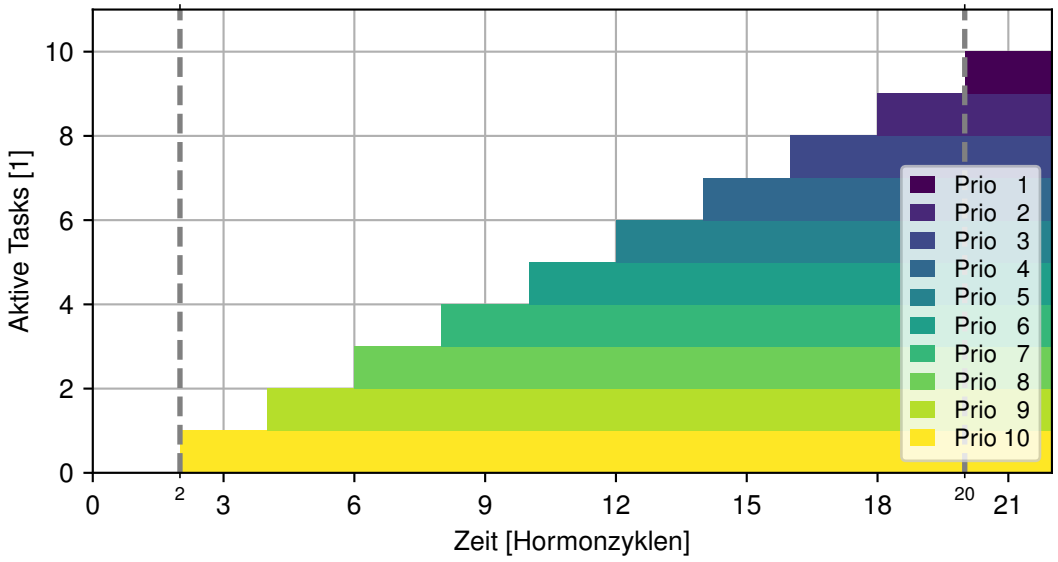
7.1.1. Betrachtung der Zuteilungsreihenfolge

Der Grund für die unterschiedliche Reihenfolge der Taskzuteilung ist, dass die Taskprioritäten von der aggressiven Strategie nicht beachtet werden. Als Resultat ist für diese Strategie keine Zuteilungsreihenfolge definiert. Im Falle des Simulators werden im konkreten Fall alle Tasks in der in Tabelle 7.1 gezeigten Reihenfolge zugeteilt. Diese entspricht nicht der durch die Taskprioritäten festgelegten Reihenfolge, zudem werden insbesondere mehrfach jeweils mehrere Tasks unterschiedlicher Priorität im selben Hormonzyklus zugeteilt. Im Gegensatz dazu werden die Tasks von der prioritätsbasierten Strategie erwartungsgemäß in der Reihenfolge absteigender Prioritäten zugeteilt, weshalb hier in keinem Hormonzyklus mehr als eine Task zugeteilt werden kann, da alle Tasks verschiedene Prioritäten besitzen.

Folglich ist zunächst festzuhalten, dass die priorirätsbasierte Strategie der aggressiven Strategie im qualitativen Aspekt der Beachtung der Taskprioritäten überlegen ist: Durch eine anhand der Prioritäten definierte deterministische Zuteilungsreihenfolge können beispielsweise die wichtigsten Tasks eines Systems nach dessen Start möglichst schnell zugeteilt werden, sodass das System bereits teilweise funktionsfähig ist, während die übrigen Tasks zugeteilt werden. Dieses Verhalten kann bei Verwendung der aggressiven



(a) Bei Verwendung der aggressiven Task-Decision-Strategie



(b) Bei Verwendung der prioritätsbasierten Task-Decision-Strategie

Abbildung 7.3: Zeitlicher Verlauf der Selbstkonfiguration für Evaluationsszenario 1 (Abbildung 7.2), gestapelt nach Taskprioritäten

Task-Decision-Strategie nicht realisiert werden.

Eine Schattenseite der prioritätsbasierten Strategie ist jedoch das bereits erwähnte schlechtere Zeitverhalten der Selbstkonfiguration, das wir nun mithilfe der hergeleiteten Zeitschranken beider Strategien untersuchen wollen.

7.1.2. Betrachtung der Dauer der Selbstkonfiguration

Aus Abbildung 7.3a ist abzulesen, dass die Selbstkonfiguration unter Verwendung der aggressiven Task-Decision-Strategie in Hormonzyklus 2 startete und am Ende von Zyklus 9 abgeschlossen war, was einer Dauer von 8 Hormonzyklen entspricht. Gemäß Satz 5.18 bzw. Korollar 5.19 sind für das gegebene Szenario mit $m = 10$ Tasks höchstens 10 Hormonzyklen nötig, die hier wegen der Zuteilung von je zwei Tasks in den Zyklen 4 und 6 sogar unterschritten werden.

Für die prioritätsbasierte Strategie ist in Abbildung 7.3b zu erkennen, dass die Selbstkonfiguration sich über die Hormonzyklen 2 bis einschließlich 20 erstreckte, was insgesamt 19 Hormonzyklen entspricht. Dies ist in den Wartezyklen begründet, in denen keine Taskzuteilung stattfinden kann und die immer einen Zyklus nach der Zuteilung der letzten Task eines Prioritätslevels auftreten. Diese Wartezyklen sind in der Abbildung in den Hormonzyklen 3, 5, 7, 9, 11, 13, 15, 17 und 19 gut erkennbar.[1]

Nach Satz 6.9 dauert eine Selbstkonfiguration bei Verwendung der prioritätsbasierten Task-Decision-Strategie für m Tasks mit $|P|$ verschiedenen Prioritäten höchstens $m + |P| - 1$ Hormonzyklen; für den hier vorliegenden Spezialfall, in dem alle Tasks verschiedene Prioritäten besitzen, ergeben sich wegen $|P| = m$ gerade die aus Satz 6.10 bekannten $2m - 1$ Hormonzyklen. Diese entsprechen für $m = 10$ jedoch gerade den hier beobachteten 19 Hormonzyklen: Die theoretisch bestimmte Worst-Case-Zeitschranke wird in diesem Evaluationsszenario exakt erreicht.

7.1.2.1. Betrachtung der Zuteilungsdauer

Die durch die Taskprioritäten definierte Zuteilungsreihenfolge ermöglicht für die prioritätsbasierte Task-Decision-Strategie eine Abschätzung der Zuteilungsdauer $\mathrm{D}(T)$ einer Task T, welche gemäß Definition 6.11 die Zeitspanne angibt, nach deren Ablauf garantiert werden kann, dass T einem PE zugeteilt ist. Da alle Prioritäten verschieden sind, entspricht $\mathrm{D}(T)$ jedoch gerade der Zeitspanne vom Beginn der Selbstkonfiguration bis zur tatsächlichen Zuteilung von T.[2] In Abbildung 7.3b kann $\mathrm{D}(T)$ daher analog zur Dauer der gesamten Selbstkonfiguration abgelesen werden. Die sich ergebenden Zuteilungsdauern sind in Tabelle 7.2 den Abschätzungen gemäß Satz 6.12 gegenübergestellt. Ebenso wie die Gesamtdauer der Selbstkonfiguration (die gerade der Zuteilungsdauer $\mathrm{D}(1)$ der niedrigstprioren Task 1 entspricht), welche in diesem Szenario exakt der

[1] Zyklus 21 folgt zwar ebenfalls auf die Zuteilung der letzten Task mit Priorität 1, ist jedoch nach Definition 5.14 kein Wartezyklus, da im System bereits alle Tasks zugeteilt sind.

[2] Würden mehrere Tasks pro Prioritätslevel existieren, entspräche $\mathrm{D}(T)$ der Zeitspanne bis zur Zuteilung der *letzten* Task des Prioritätslevels $\mathrm{prio}(T)$: Innerhalb eines Prioritätslevels ist die Zuteilungsreihenfolge der Tasks nicht definiert, weshalb T im schlimmsten Falls als letzte Task ihres Prioritätslevels zugeteilt wird.

Tabelle 7.2: Zuteilungsdauer D(T) der Tasks aus Evaluationsszenario 1 bei Verwendung der prioritätsbasierten Task-Decision-Strategie. Die beobachteten Werte sind aus Abbildung 7.3b entnommen, die Abschätzung stammt aus Satz 6.12.

	D(T)	
Task T	**beobachtet**	**abgeschätzt**
1	19	≤ 19
2	17	≤ 17
3	15	≤ 15
4	13	≤ 13
5	11	≤ 11
6	9	≤ 9
7	7	≤ 7
8	5	≤ 5
9	3	≤ 3
10	1	≤ 1

abgeschätzten Worst-Case-Zeitschranke entspricht, stimmen hier auch die beobachteten Zuteilungsdauern exakt mit dem jeweils abgeschätzten Worst Case überein, was durch die größtmögliche Anzahl auftretender Wartezyklen begründet ist.

7.1.2.2. Diskussion des Szenarios

Trotz seiner Einfachheit lieferte dieses erste Evaluationsszenario bereits mehrere Resultate:

Die Selbstkonfiguration dauert bei Verwendung der prioritätsbasierten Task-Decision-Strategie in diesem Szenario zwar länger, als dies beim Einsatz der aggressiven Strategie der Fall ist, dafür werden die Tasks jedoch nur mit ersterer Strategie in der korrekten, anhand ihrer Prioritäten definierten Reihenfolge zugeteilt. Beispielsweise wurde Task 10 im betrachteten Szenario nur von der prioritätsbasierten Strategie im ersten Hormonzyklus zugeteilt, während die aggressive Strategie diese – unter allen Tasks höchstpriore – Task erst im siebten Hormonzyklus zuteilte – und damit sogar *nach* Zuteilung der niedrigstprioren Task 1 im vierten Zyklus. In diesem qualitativen Aspekt der Zuteilungsreihenfolge ist die prioritätsbasierte Strategie daher dem klassischen AHS ohne Prioritätsberücksichtigung überlegen. Folglich wird die aggressive Strategie im Folgenden nicht mehr als Vergleichsnormal herangezogen; die einzige Ausnahme stellt Abschnitt 7.3 dar, in dem das Verhalten der prioritätsbasierten Strategie untersucht wird, falls alle Tasks im System dieselbe Priorität besitzen, da dieser Spezialfall einen Vergleich mit der aggressiven Strategie nahelegt.

Zudem ist festzuhalten, dass im gegebenen Sonderfall nach jedem vollständig zugeteilten Prioritätslevel (außer dem letzten) ein Wartezyklus auftrat, was gerade der maximal möglichen Anzahl von Wartezyklen entsprach. Dabei werden die zuvor theoretisch abge-

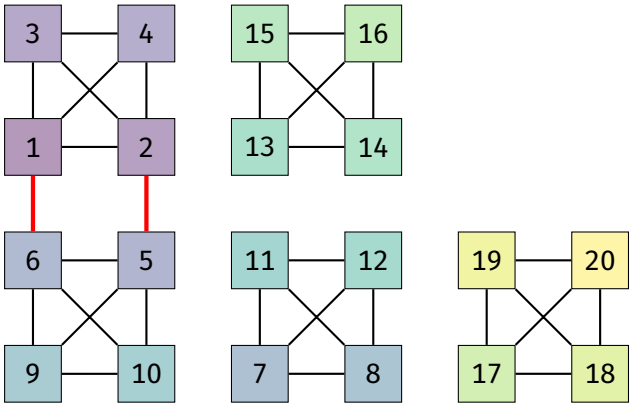


Abbildung 7.4: Tasks in Evaluationsszenario 2.1: Jede Task i hat die Priorität $\text{prio}(i) = i$, rote Verwandtschaftsbeziehungen weisen gegenüber den schwarzen einen doppelt so hohen Verwandtschaftsgrad auf.

schätzten Worst-Case-Zeitschranken der Selbstkonfiguration sowie der Zuteilungsdauern einzelner Tasks exakt erreicht, was ein erstes Indiz für deren Gültigkeit liefert und zudem vermuten lässt, dass diese Zeitschranken keineswegs zu pessimistisch abgeschätzt wurden.

7.2. Spezialfall: Alle Prioritäten verschieden ($|P| = m$)

Das erste Evaluationsszenario entsprach bereits einem Spezialfall der theoretisch hergeleiteten Worst-Case-Zeitschranke, in dem alle Tasks verschiedene Prioritäten besitzen und somit $|P| = m$ gilt. Diesen Spezialfall möchten wir im Folgenden hinsichtlich der Taskanzahl skalieren, um das Verhalten der prioritätsbasierten Task-Decision-Strategie in verschieden großen Systemen bewerten zu können.

7.2.1. 20 Tasks verschiedener Prioritäten

Hierzu betrachten wir zunächst ein Szenario mit $m = 20$ Tasks, deren Verwandtschaftsverhältnisse in Abbildung 7.4 dargestellt sind. Exemplarisch wurden je zwei Tasks durch besonders starke Verwandtschaftsverhältnisse miteinander verbunden, was in doppelt so großen Acceleratorhormonen für die betroffenen Tasks resultiert und so mit dem ansonsten größtenteils symmetrischen Aufbau des Evaluationsszenarios bricht.

7.2.1.1. Zwei PEs

Das beschriebene Szenario wurde zunächst auf zwei PEs evaluiert, die benachbart sind und jeweils maximal 10 Tasks übernehmen können.

Gleichartige PEs In Abbildung 7.5 ist der zeitliche Verlauf der Selbstkonfiguration nach PEs sowie Prioritäten gestapelt aufgetragen, falls beide PEs gleichartig sind,

also identische Hormonwerte für alle Tasks aufweisen. Es ist gut ersichtlich, dass die Tasks aufgrund ihrer Lastsuppressoren abwechselnd auf beiden PEs zugeteilt werden, wobei $PE_{(1,1)}$ aufgrund seines geringeren Index die erste Task übernimmt. Die Selbstkonfiguration beginnt auch in diesem Szenario in Zyklus 2 und endet in Zyklus 40, woraus sich eine Gesamtdauer von $39 = 2m - 1$ Hormonzyklen ergibt. Dies entspricht wiederum exakt der aus Satz 6.10 bekannten Abschätzung. Auch eine Überprüfung der Zuteilungsdauern aller Tasks analog zu Abschnitt 7.1.2.1 liefert dieselben Werte wie die Abschätzung aus Satz 6.12, weshalb hier auf deren Angabe verzichtet wird.

Verschiedenartige PEs In einer Variation des Evaluationsszenarios wurden die statischen Eignungswerte von $PE_{(1,1)}$ mit dem Faktor 20 skaliert, während $PE_{(2,1)}$ unverändert blieb. Der aus dieser Anpassung resultierende Verlauf der Selbstkonfiguration ist in Abbildung 7.6 dargestellt. Es ist hier gut erkennbar, dass die ersten zehn Tasks alle von $PE_{(1,1)}$ übernommen wurden, da dieses stets höhere modifizierte Eignungswerte als $PE_{(2,1)}$ senden konnte. Nach Übernahme der zehnten Task war dieses PE jedoch vollständig ausgelastet und hatte keine Kapazitäten mehr für weitere Tasks, weshalb $PE_{(2,1)}$ die verbleibenden zehn Tasks übernahm. Hierbei traten jedoch im Gegensatz zur Übernahme der ersten zehn Tasks keine Wartezyklen mehr auf, weil nur noch ein einziges PE für diese Tasks positive modifizierte Eignungswerte sendete – folglich wurde der *Almost-Winner*-Zustand für die betroffenen Tasks nicht mehr eingenommen.

Insgesamt dauerte die Selbstkonfiguration hier 30 Hormonzyklen (Beginn in Zyklus 2, Ende in Zyklus 31), womit die Worst-Case-Zeitschranke von $2m - 1 = 39$ Hormonzyklen aufgrund der teilweise entfallenen Wartezyklen unterschritten wurde.

Betrachtet man den Zeitpunkt der Zuteilung der verschiedenen Tasks, so werden die zehn höchstprioren Tasks wie zuvor jeweils bei Erreichen der oberen Schranke von $D(T)$ gemäß Satz 6.12 zugeteilt. Die zehn niedrigstprioren Tasks werden hier dagegen früher zugeteilt als von dieser Abschätzung vorhergesagt: Diese betrachtet lediglich den (allgemeinen) Worst Case und geht daher nicht auf die Eigenheiten der PEs ein. Wie aus diesem Szenario ersichtlich wird, kann die Selbstkonfiguration durch die PE-Konstellation beschleunigt werden, falls Wartezyklen entfallen, weil am Ende der Selbstkonfiguration nur noch ein einziges PE positive modifizierte Eignungswerte für Tasks sendet und so der *Almost-Winner*-Zustand nicht mehr eingenommen wird.

7.2.1.2. Ein einziges PE

Der soeben betrachtete Sonderfall kann noch weiter spezialisiert werden, indem nur ein einziges PE verwendet wird (das hierzu jedoch mindestens 20 Tasks übernehmen können muss). Der resultierende Verlauf der Selbstkonfiguration ist in Abbildung 7.7 dargestellt. Da hier über die gesamte Dauer der Selbstkonfiguration nur ein einziges PE positive modifizierte Eignungswerte senden kann, wird der *Almost-Winner*-Zustand nie eingenommen, weshalb keinerlei Wartezyklen auftreten. Als Resultat dauert die gesamte Selbstkonfiguration lediglich 20 Hormonzyklen (Start in Zyklus 2, Ende in Zyklus 21).

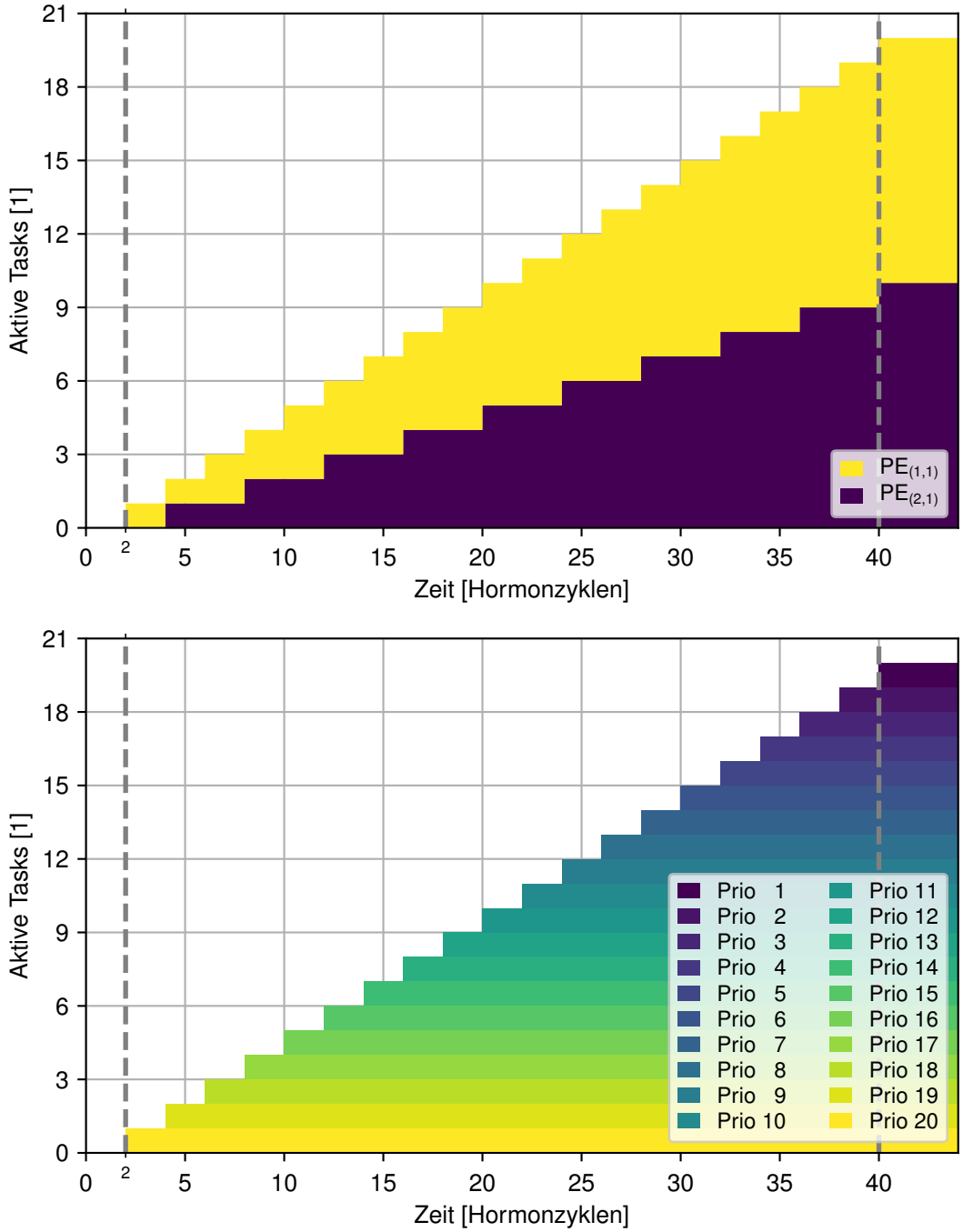


Abbildung 7.5: Zeitlicher Verlauf der Selbstkonfiguration für Evaluationsszenario 2.1 (Abbildung 7.4) auf zwei gleichartigen PEs

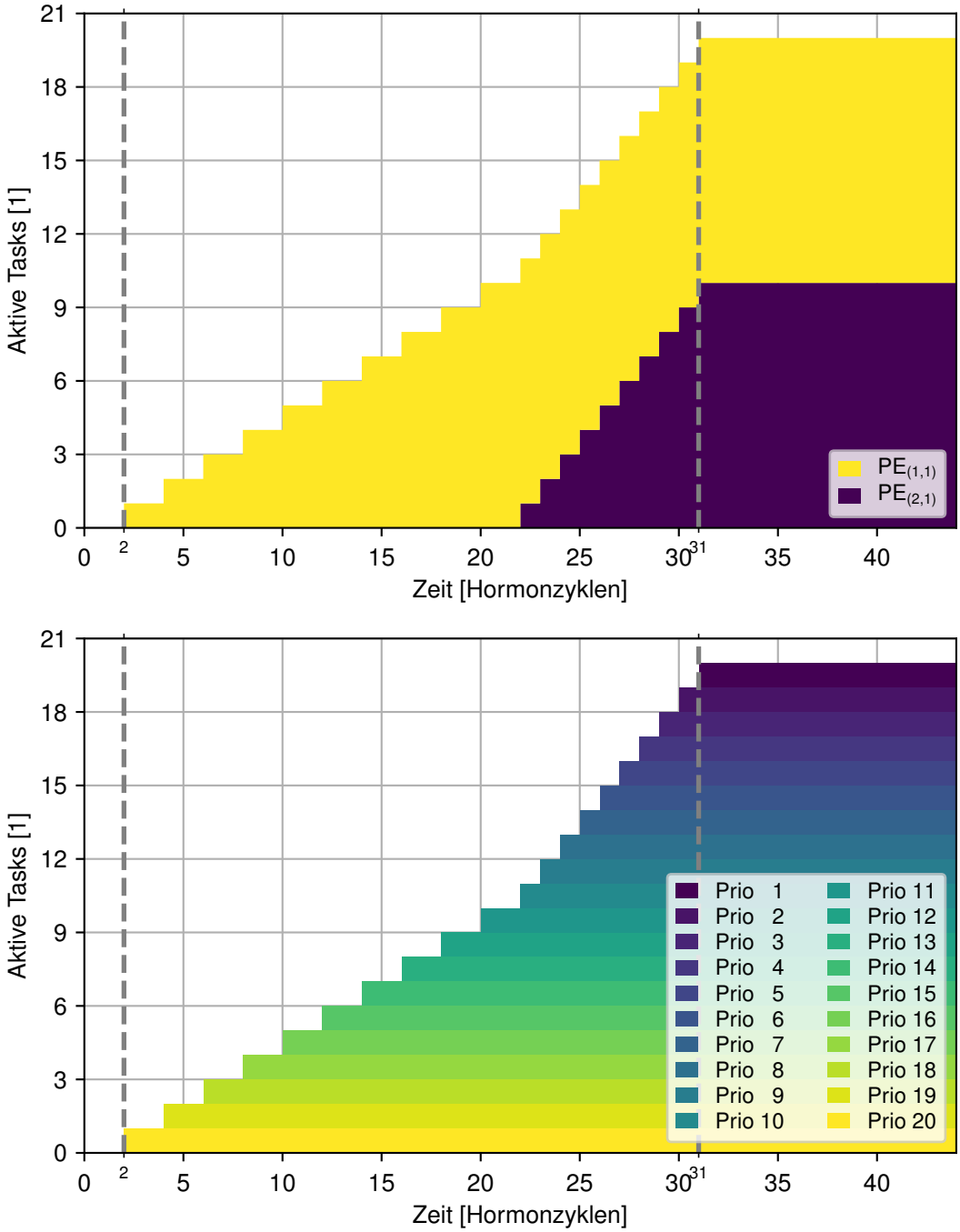


Abbildung 7.6: Zeitlicher Verlauf der Selbstkonfiguration für Evaluationsszenario 2.1 (Abbildung 7.4) auf zwei unterschiedlichen PEs

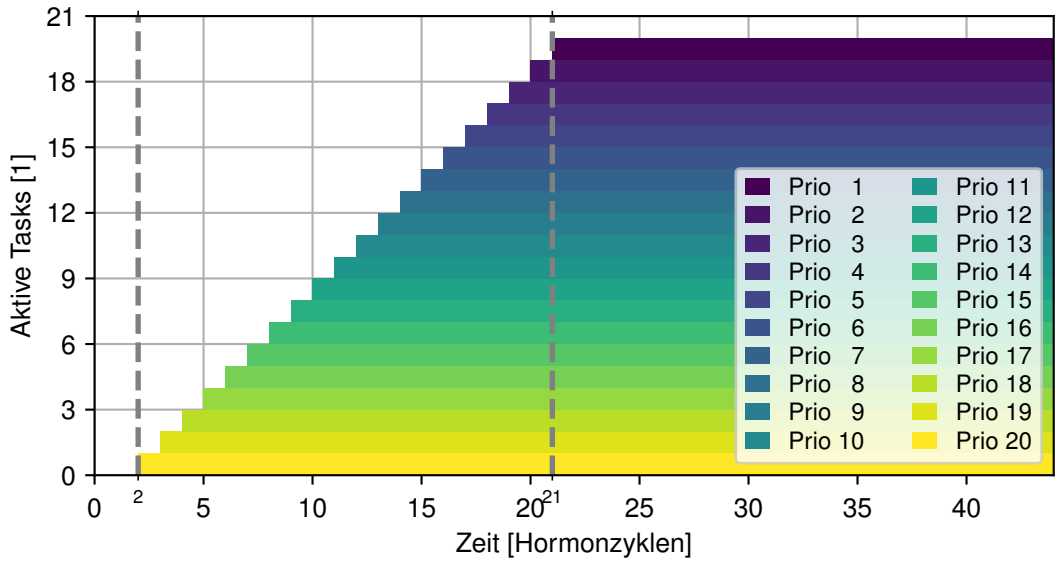


Abbildung 7.7: Zeitlicher Verlauf der Selbstkonfiguration für Evaluationsszenario 2.1 (Abbildung 7.4) auf einem einzigen PE, gestapelt nach den Prioritäten der Tasks

Dieses Verhalten wurde bereits von Bemerkung 6.8 vorhergesagt und ist insbesondere relevant für die Zeitschranke der Selbstheilung. Für die Selbstkonfiguration wird dagegen implizit angenommen, dass mehr als ein PE existiert (da ein einziges PE ohnehin einen Single-Point-of-Failure darstellen würde), weshalb die Dauer der Selbstkonfiguration durch Satz 6.10 hier weiterhin zu höchstens $2m - 1 = 39$ Hormonzyklen abgeschätzt wird.

Nichtsdestotrotz wird Bemerkung 6.8 durch diese beiden Versuche auf zwei verschiedenen PEs bzw. einem einzigen PE experimentell bestätigt. Im Folgenden werden wir uns jedoch wieder auf den Fall beschränken, dass mehr als ein PE existiert und alle PEs gleichartig sind, also identische Hormonwerte aufweisen.

7.2.1.3. Fünf PEs

Bei Verwendung von fünf gleichartigen PEs mit den in Abbildung 7.8 dargestellten Nachbarschaftsverhältnissen, die jeweils maximal fünf Tasks übernehmen können, ergab sich für das Evaluationsszenario im Wesentlichen dasselbe Verhalten wie im Fall von zwei gleichartigen PEs: Die Selbstkonfiguration benötigte wiederum insgesamt $2m - 1 = 39$ Hormonzyklen und auch die beobachteten Zuteilungsdauern waren identisch, lediglich die in Abbildung 7.9 dargestellte Verteilung der Tasks auf die PEs wies Unterschiede auf.

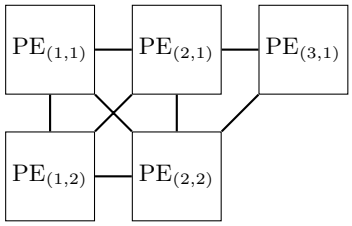


Abbildung 7.8: Konstellation aus fünf PEs

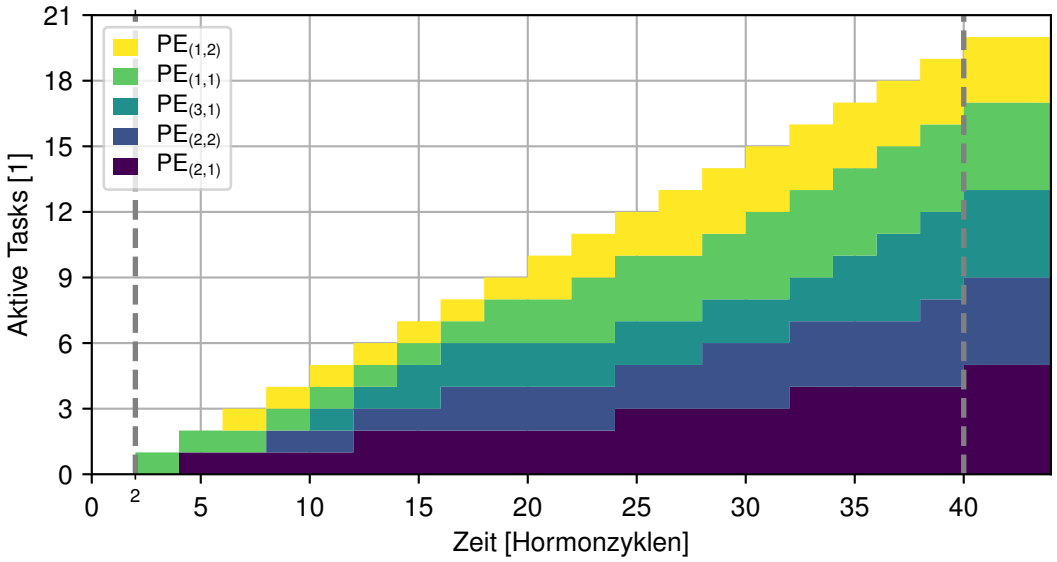
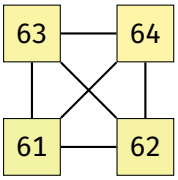


Abbildung 7.9: Zeitlicher Verlauf der Selbstkonfiguration für Evaluationsszenario 2.1 (Abbildung 7.4) auf fünf PEs, gestapelt nach der Anzahl Tasks pro PE. Nach Taskprioritäten gestapelt ergibt sich derselbe Graph wie in Abbildung 7.5.



zuzüglich drei Kopien des Szenarios mit 20 Tasks (Abbildung 7.4), deren Prioritäten auf 1–20, 21–40 und 41–60 angehoben wurden, um insgesamt 64 Prioritätslevel zu erhalten

Abbildung 7.10: Tasks in Evaluationsszenario 2.2: 64 Tasks verschiedener Prioritäten. Jede Task i hat die Priorität $\text{prio}(i) = i$.

7.2.2. 64 Tasks verschiedener Prioritäten auf 16 PEs

Für das nächste Evaluationsszenario wurden drei Kopien des soeben verwendeten Szenario mit 20 Tasks sowie angepassten Prioritäten um ein Cluster aus vier Tasks erweitert. Das so erhaltene und schematisch in Abbildung 7.10 dargestellte Szenario umfasste daher $m = 64$ Tasks verschiedener Prioritäten und wurde auf 16 PEs simuliert, die in einem 4×4-Gitter angeordnet waren und jeweils maximal fünf Tasks übernehmen konnten.

Hierbei wurden alle Tasks in der erwarteten Reihenfolge absteigender Priorität den PEs zugeteilt. Der nach den insgesamt 64 Prioritäten gestapelte zeitliche Verlauf der Selbstkonfiguration wird hier aus Gründen der Übersichtlichkeit nicht dargestellt, stattdessen zeigt Abbildung 7.11 lediglich den nach der Anzahl pro PE übernommener Tasks gestapelten Graphen.

Die gesamte Selbstkonfiguration umspannte in diesem Szenario die Hormonzyklen 2 bis 128, was einer Gesamtdauer von $127 = 2m - 1$ Hormonzyklen entsprach, wie abermals von Satz 6.10 vorhergesagt.

7.2.3. 256 Tasks verschiedener Prioritäten auf 64 PEs

Eine weitere Skalierungsstufe mit $m = 256$ Tasks wurde erhalten, indem Evaluationsszenario 2.1 zwölf mal mit jeweils angehobenen Prioritäten kopiert und um 16 zusätzliche Tasks erweitert wurde, was in dem in Abbildung 7.12 gezeigten Szenario resultierte, das auf 64 PEs untersucht wurde, die in einem 8×8-Gitter angeordnet waren.

Aus Gründen der Übersichtlichkeit wird hier auf die Abbildung der zugehörigen Graphen verzichtet; als Ergebnis dieses Evaluationsszenarios lässt sich jedoch festhalten, dass die prioritätsbasierte Task-Decision-Strategie auch hier alle Tasks in der erwarteten Reihenfolge zuteilte und hierfür exakt die durch Satz 6.10 abgeschätzten $2m - 1 = 511$ Hormonzyklen benötigte (Zyklen 2–512). Auch die Zuteilungsdauern der einzelnen Tasks stimmten mit den Worst-Case-Abschätzungen aus Satz 6.12 überein.

Diese Skalierung auf $m = 256$ Tasks stellt zudem das größte evaluierte Szenario dar: Zum einen unterstützt die AHS-Middleware in ihrer gegenwärtigen Implementierung höchstens 255 Tasks, weshalb das gegebene Szenario darin bereits nicht mehr realisierbar ist, sondern nur mit dem Hormonsimulator simuliert werden kann. Zum anderen wächst der Kommunikationsbedarf des AHS mit steigenden Anzahlen von Tasks und PEs. Für größere Szenarien ist das klassische AHS aufgrund der allein für Verwaltungsdaten

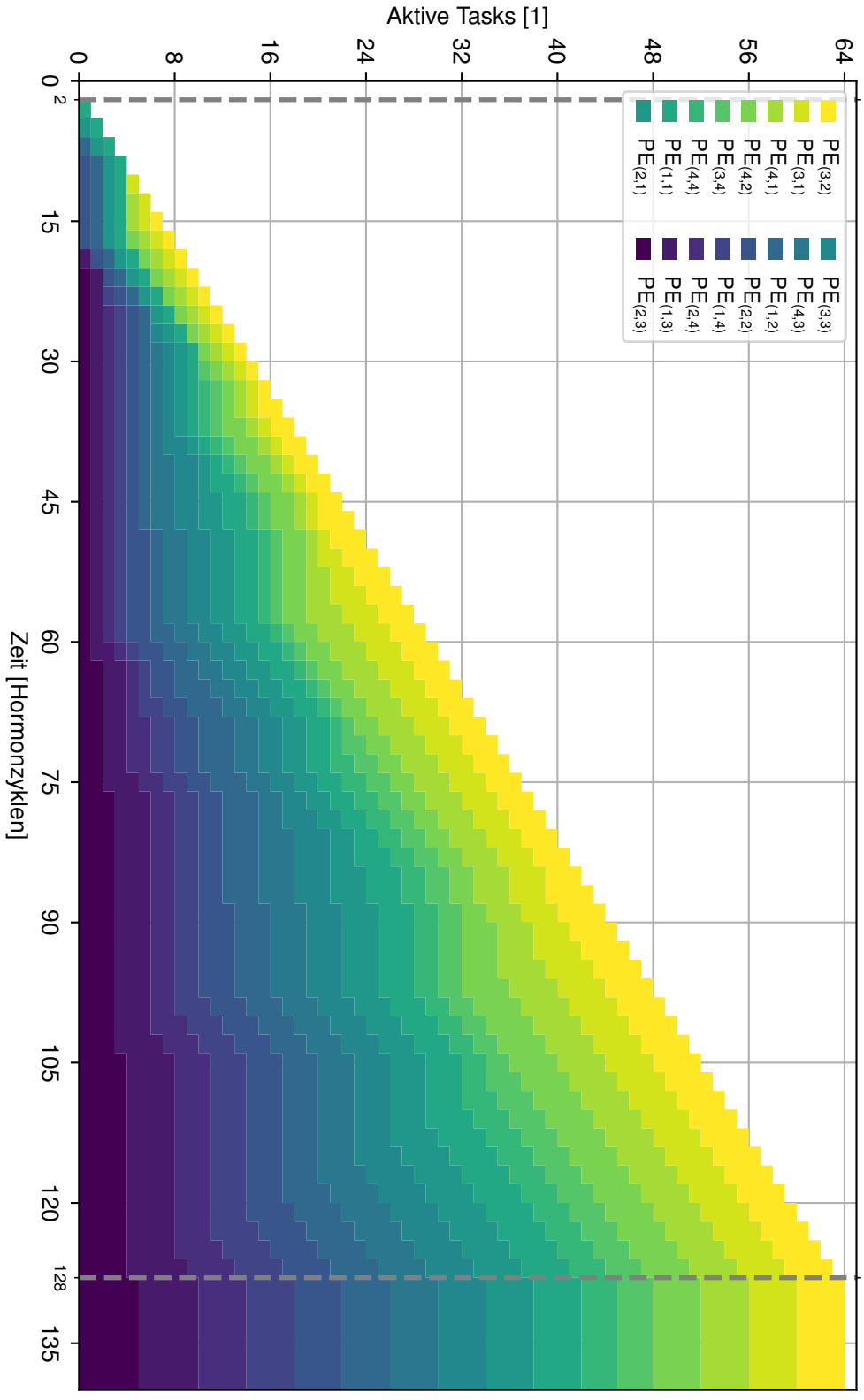
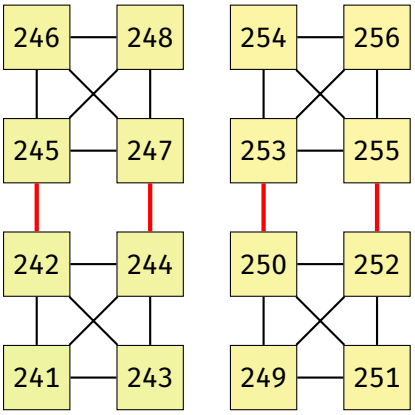


Abbildung 7.11: Zeitlicher Verlauf der Selbstkonfiguration für Evaluationsszenario 2.2 (Abbildung 7.10) auf 16 PEs, gestapelt nach der Anzahl Tasks pro PE.



zuzüglich zwölf Kopien des Szenarios mit 20 Tasks, deren Prioritäten jeweils angehoben wurden, um insgesamt 256 Prioritätslevel zu erhalten

Abbildung 7.12: Tasks in Evaluationsszenario 2.3: 256 Tasks verschiedener Prioritäten. Jede Task i hat die Priorität $\text{prio}(i) = i$.

(in Form von Hormonen) benötigten Bandbreite nicht mehr praktikabel, weshalb in [Lun19] verschiedene hierarchische Erweiterungen des AHS entwickelt wurden, die besser für solche Skalierungen geeignet sind. Daher werden in dieser Dissertation keine größeren Evaluationsszenarien als die bisher gezeigten betrachtet.

7.2.4. Diskussion der Szenarien mit $|P| = m$

Die bisher untersuchten Evaluationsszenarien zeigten jeweils, dass die prioritätsbasierte Task-Decision-Strategie im Rahmen der Selbstkonfiguration alle Tasks in der erwarteten, anhand ihrer Prioritäten definierten Reihenfolge den PEs zuteilte.

Hierbei wurde die in Satz 6.10 theoretisch hergeleitete Worst-Case-Zeitschranke von $2m - 1$ Hormonzyklen bei m Tasks mit $|P| = m$ verschiedenen Prioritäten niemals überschritten, sondern vielmehr zumeist exakt erreicht. Die einzige Ausnahme stellten Szenarien mit unterschiedlichen PEs oder nur einem einzigen PE dar: In diesen sendete nur ein einziges PE modifizierte Eignungswerte für bestimmte bzw. alle Tasks, wodurch keine Wartezyklen auftraten und die Selbstkonfiguration – wie von Bemerkung 6.8 vorhergesagt – schneller ablaufen konnte. Auch die obere Schranke der Zuteilungsdauer $\text{D}(T)$ aus Satz 6.12 wurde nie überschritten, sondern zumeist exakt erreicht.

7.3. Spezialfall: Nur ein Prioritätslevel ($|P| = 1$)

Bisher wurde nur der Spezialfall $|P| = m$ betrachtet, in dem alle Tasks verschiedene Prioritäten haben. In Hinblick auf reale, auf Basis des AHS entwickelte Systeme ist jedoch zu erwarten, dass zumeist weniger Prioritäten als Tasks existieren, also $|P| < m$ gilt. Bevor wir uns diesem allgemeineren Fall zuwenden, werden wir jedoch einen zweiten Spezialfall betrachten, in dem $|P| = 1$ gilt, also alle Tasks dieselbe Priorität besitzen.

In diesem Fall ist durch die Prioritäten keine Zuteilungsreihenfolge definiert, weshalb

die prioritätsbasierte Task-Decision-Strategie zur ihr zugrunde liegenden aggressiven Strategie degenerieren sollte. Insbesondere folgt aus Satz 6.9, dass die Selbstkonfiguration für $|P| = 1$ nach höchstens m Hormonzyklen abgeschlossen sein sollte, was genau mit der Zeitschranke der aggressiven Strategie aus Korollar 5.19 übereinstimmt.

Daher wird der Fokus im Folgenden nicht nur darauf gelegt, die Zeitschranke von m Hormonzyklen experimentell zu bestätigen, sondern zudem das Zuteilungsverhalten beider Strategien genauer miteinander zu vergleichen, um etwaige Unterschiede zu identifizieren. Hierzu wird eine Auswahl der bisherigen Szenarien verwendet, die jedoch insofern abgeändert wurden, als dass nur noch ein einziges Prioritätslevel zum Einsatz kam.

Da keine Zuteilungsreihenfolge der Tasks definiert ist, kann jedoch auch die Zuteilungsdauer $\mathrm{D}(T)$ nicht besser abgeschätzt werden als durch die *Gesamtdauer* der Selbstkonfiguration: Im Worst Case wird eine Task T als letzte Task des (einzigen) Prioritätslevels zugeteilt. Dies ist konsistent zu Satz 6.12, dessen Abschätzung im Fall $|P| = 1$ für alle $T \in \mathcal{T}$ zu $\mathrm{D}(T) \leq m$ Hormonzyklen degeneriert. Im Folgenden wird daher auf die Betrachtung der Zuteilungsdauer verzichtet.

7.3.1. 10 Tasks, 3 PEs

Das erste im Spezialfall $|P| = 1$ betrachtete Szenario entspricht Evaluationsszenario 1 (Abbildung 7.2), wobei für jede Task i jedoch $\mathrm{prio}(i) = 1$ gilt. Wie zuvor wurden drei PEs in einer 1×3-Gittertopologie verwendet, die jeweils maximal fünf Tasks übernehmen konnten.

Der zeitliche Verlauf der sich ergebenden Selbstkonfiguration ist in Abbildung 7.13 für die prioritätsbasierte Task-Decision-Strategie gestapelt nach der Anzahl Tasks pro PE aufgetragen. Die Selbstkonfiguration dauerte hier mit insgesamt 8 Hormonzyklen zwei Zyklen kürzer als dies durch die Worst-Case-Zeitschranke abgeschätzt wird, weil in den Zyklen 4 und 7 jeweils zwei Tasks zugeteilt wurden.

Vergleicht man den abgebildeten Kurvenverlauf mit Abbildung 7.3a, in der dasselbe Szenario (allerdings mit verschiedenen Prioritäten) bei Verwendung der (diese Prioritäten ignorierenden) aggressiven Strategie abgebildet ist, so fällt auf, dass der Kurvenverlauf der *Gesamtzahl* zugeteilter Tasks identisch ist. Bei einem genaueren Vergleich lässt sich sogar feststellen, dass die prioritätsbasierte Strategie die Tasks in der durch Tabelle 7.1 gegebenen Reihenfolge zuteilte (was aus dem abgebildeten Diagramm jedoch nicht abgelesen werden kann): Beide Task-Decision-Strategien zeigten für $|P| = 1$ in diesem Szenario also identisches Verhalten.

7.3.2. 20 Tasks, 5 PEs

In Abbildung 7.14 ist der zeitliche Verlauf des aus Abschnitt 7.2.1 bekannten Evaluationsszenarios auf fünf PEs abgebildet, bei dem jedoch wie im vorigen Szenario alle Prioritäten egalisiert wurden. Auch hier zeigten beide Strategien abermals identisches Verhalten bis hin zur Reihenfolge, in der die einzelnen Tasks zugeteilt wurden;

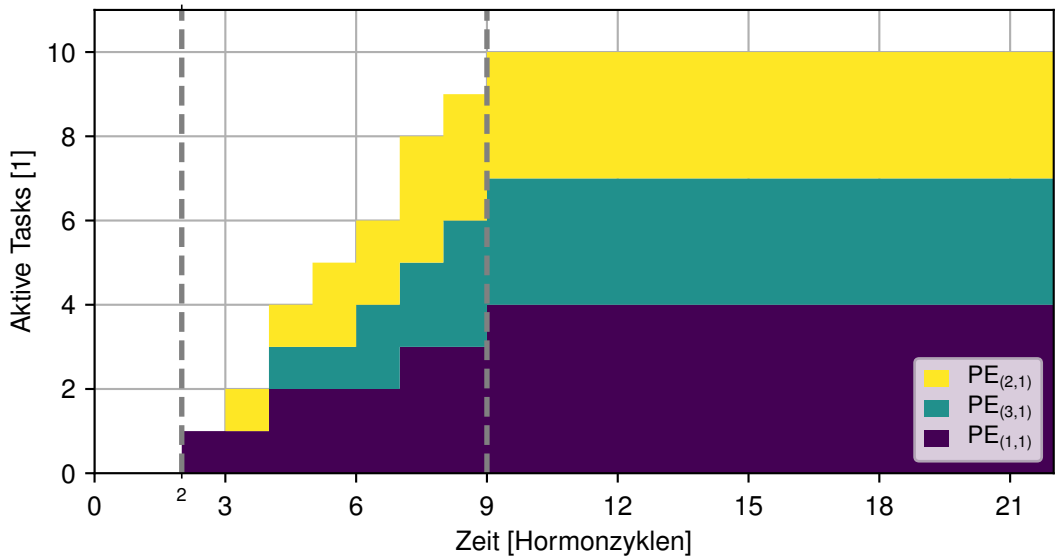


Abbildung 7.13: Zeitlicher Verlauf der Selbstkonfiguration für das aus 10 Tasks identischer Priorität bestehende Evaluationsszenario auf 3 PEs, gestapelt nach der Anzahl Tasks pro PE

die Selbstkonfiguration dauerte mit 14 Hormonzyklen abermals kürzer als durch die Worst-Case-Zeitschranke abgeschätzt.

Bei Verteilung der Tasks auf nur zwei PEs (nicht abgebildet) zeigten beide Strategien ebenfalls identisches Verhalten, die Selbstkonfiguration dauerte jedoch gerade die von den Zeitschranken beider Strategien abgeschätzten 20 Hormonzyklen.

7.3.3. 64 Tasks auf 16 PEs sowie 256 Tasks auf 64 PEs

Auch die Szenarien mit 64 Tasks gemäß Abbildung 7.10 auf 16 PEs im 4×4-Gitter sowie 256 Tasks gemäß Abbildung 7.12 auf 64 PEs im 16×16-Gitter wurden mit egalisierten Prioritäten erneut betrachtet. Aus Platzgründen wird auf die Abbildung der zugehörigen Graphen verzichtet; die Ergebnisse zeigten jedoch, dass auch hier beide Task-Decision-Strategien jeweils identisches Verhalten aufwiesen und die Selbstkonfigurationen mit 39 bzw. 136 Hormonzyklen zudem deutlich schneller beendet waren, als gemäß der Zeitschranken von höchstens 64 bzw. 256 Hormonzyklen zu erwarten war.

7.3.4. Diskussion der Szenarien mit $|P| = 1$

Als Ergebnis dieser Evaluationsszenarien lässt sich festhalten, dass die prioritätsbasierte Task-Decision-Strategie zur aggressiven Strategie degeneriert, wenn die Priorität aller Tasks identisch ist. Dieses Ergebnis ist jedoch keinesfalls überraschend, da die prioritätsbasierte Strategie als Erweiterung der aggressiven Strategie angesehen kann, deren Grundprinzipien sie im Wesentlichen übernimmt. Folglich kann die prioritätsbasierte

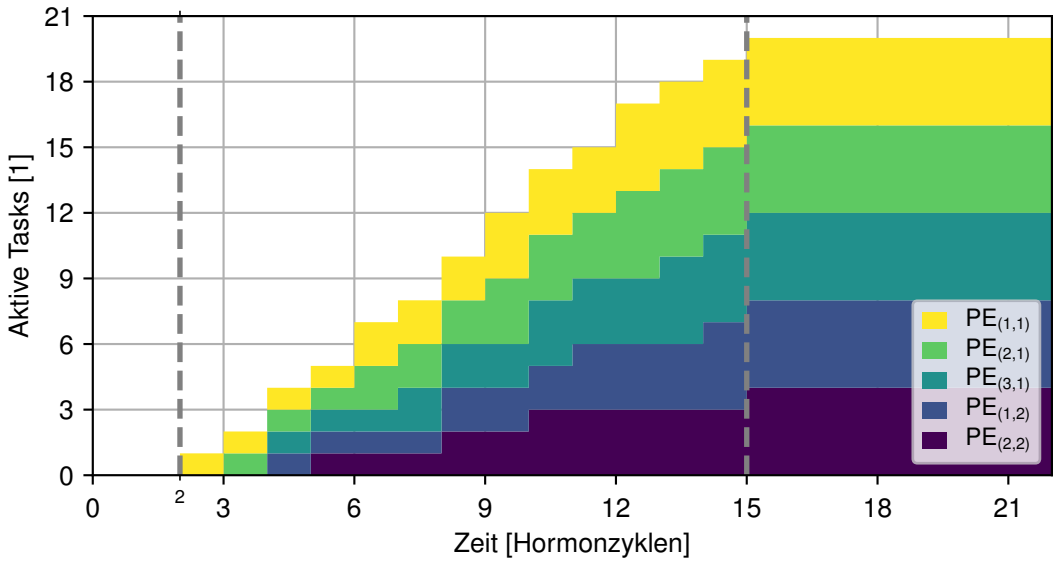


Abbildung 7.14: Zeitlicher Verlauf der Selbstkonfiguration für das aus 20 Tasks identischer Priorität bestehende Evaluationsszenario auf 5 PEs, gestapelt nach der Anzahl Tasks pro PE

Strategien in „klassischen“ prioritätslosen Anwendungsszenarien problemlos anstatt der aggressiven Strategie verwendet werden, ohne Nachteile befürchten zu müssen.

7.4. Weniger Prioritätslevel als Tasks ($1 < |P| < m$)

Bisher wurden nur die beiden Spezialfälle $|P| = m$ sowie $|P| = 1$ der allgemeinen, aus Satz 6.9 bekannten Zeitschranke von höchstens $m + |P| - 1$ Hormonzyklen betrachtet. Im Folgenden werden wir uns dagegen dem allgemeinen Fall $1 < |P| < m$ widmen, in dem also weniger Prioritätslevel als Tasks existieren, jedoch nicht alle Tasks dieselbe Priorität besitzen. Entsprechend ist die Zuteilungsreihenfolge der einzelnen Tasks innerhalb desselben Prioritätslevels nicht definiert, jedoch müssen alle Tasks eines Levels zugeteilt sein, bevor mit der Zuteilung von Tasks niedrigerer Priorität begonnen werden darf – folglich wird im Folgenden neben der Dauer der gesamten Selbstkonfiguration auch die Zuteilungsdauer $\mathrm{D}(T)$ betrachtet.

Zum Zwecke dieser Evaluationen wurden wieder die bereits bekannten Szenarien verwendet, deren Prioritäten jedoch abermals angepasst wurden: Konkret wurden für verschiedene Werte $1 < i < m$ je i Tasks demselben Prioritätslevel zugeordnet, wobei mit den i zuvor höchstprioren Tasks begonnen wurde, wie in Abbildung 7.15 schematisch dargestellt ist. Dies führt zu insgesamt $|P| = \lceil m/i \rceil$ Prioritätsleveln, von denen die höchstprioren je exakt i Tasks enthalten, während das niedrigstpriore mit

$$1 \leq m - \left(\left\lceil \frac{m}{i} \right\rceil - 1 \right) \cdot i \leq i \text{ Tasks}$$

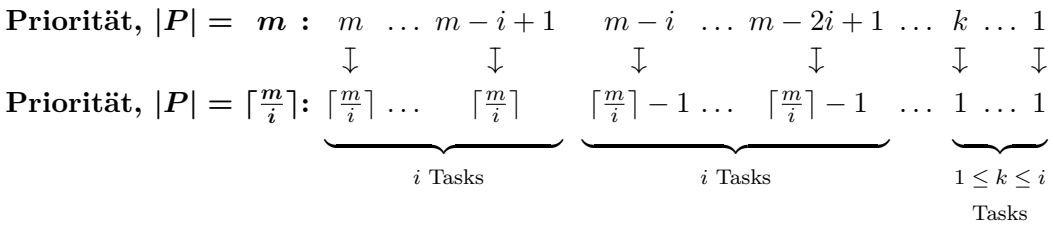


Abbildung 7.15: Abbildungsschema der bisherigen Prioritäten (bei $|P| = m$) auf $|P| = \lceil \frac{m}{i} \rceil$ Prioritätslevel. Das letzte Prioritätslevel umfasst mit $k = m - (\lceil \frac{m}{i} \rceil - 1) \cdot i$ Tasks unter Umständen weniger als i Tasks.

unter Umständen nur teilweise gefüllt ist.

7.4.1. Szenarien mit $i = 4$ Tasks pro Prioritätslevel

Zunächst wurden die bekannten Szenarien mit $i = 4$ Tasks pro Prioritätslevel versehen.

7.4.1.1. $m = 10$ Tasks auf drei PEs

Das aus Abbildung 7.2 bekannte Szenario aus $m = 10$ Tasks, die auf drei PEs verteilt werden, umfasst für $i = 4$ insgesamt $|P| = \lceil 10/4 \rceil = 3$ verschiedene Prioritätslevel zu 4, 4 und 2 Tasks.

Der resultierende Verlauf der Selbstkonfiguration ist in Abbildung 7.16 nach den Prioritätsleveln gestapelt aufgetragen. Es ist gut zu erkennen, dass Wartezyklen erst nach vollständiger Zuteilung eines Prioritätslevels auftreten, nicht jedoch *innerhalb* der Prioritätslevel. Die gesamte Selbstkonfiguration dauert hier 12 Hormonzyklen, was sich mit der theoretischen Zeitschranke von $m + |P| - 1 = 12$ Hormonzyklen exakt deckt.

Bezüglich der Betrachtung der Zuteilungsdauer beschränken wir uns darauf, die frühesten Zeitpunkte nach Beginn der Selbstkonfiguration zu bestimmen, zu denen jedes der drei Prioritätslevel vollständig zugeteilt ist: Eine separate Betrachtung aller Tasks ist nicht sinnvoll, da die Zuteilungsreihenfolge von Tasks desselben Prioritätslevels nicht definiert ist, entsprechend ist gemäß Bemerkung 6.13 auch die Abschätzung von $\mathrm{D}(T)$ aus Satz 6.12 für alle Tasks derselben Priorität identisch.

Konkret sind die drei Prioritätslevel am Ende des vierten, neunten bzw. zwölften Hormonzyklus nach Beginn der Selbstkonfiguration jeweils vollständig zugeteilt, wie aus dem Diagramm zu entnehmen ist. Dies entspricht jedoch stets genau den oberen Schranken für $\mathrm{D}(T)$, wie sie sich gemäß Satz 6.12 beziehungsweise Bemerkung 6.13 für je eine beliebige Task als Vertreter ihres Prioritätslevels berechnen lassen.

7.4.1.2. $m = 20$ Tasks auf fünf PEs

Abbildung 7.17 zeigt den Verlauf der Selbstkonfiguration für ein Szenario mit $m = 20$ Tasks auf fünf PEs, das für $i = 4$ Tasks pro Prioritätslevel aus dem aus Abbildung 7.4

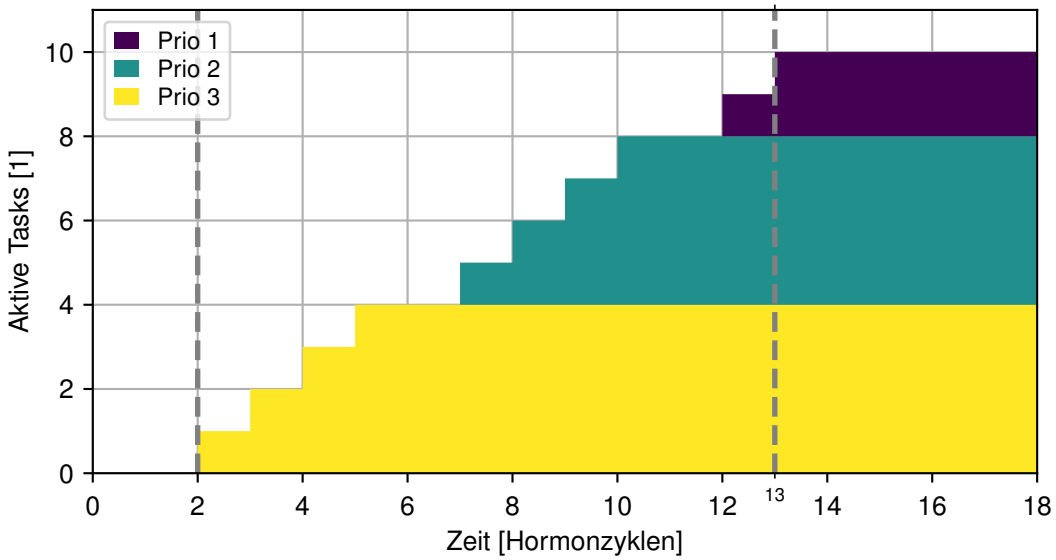


Abbildung 7.16: Zeitlicher Verlauf der Selbstkonfiguration für $m = 10$ Tasks und $i = 4$ Tasks pro Prioritätslevel auf 3 PEs

bekannten Szenario hervorgeht. Auch hier lässt sich deutlich erkennen, dass Wartezyklen nur nach der vollständigen Zuteilung eines jeden Prioritätslevels auftreten.

Die gesamte Selbstkonfiguration dauert in diesem Fall 20 Hormonzyklen und liegt damit unter der theoretischen Worst-Case-Zeitschranke von 24 Zyklen, da zu den Zeiten 17, 18, 20 und 21 je zwei Tasks gleichzeitig zugeteilt wurden. Die einzelnen Prioritätslevel sind hier 4, 9, 14, 17 und 20 Hormonzyklen nach dem Beginn der Selbstkonfiguration vollständig zugeteilt. Für die drei höchstprioren Prioritäten entspricht dies gerade den theoretischen oberen Schranken für $\mathrm{D}(T)$, für die Prioritätslevel 2 und 1 werden dagegen 19 und 24 Hormonzyklen abgeschätzt, wobei sich diese Unterschreitungen jedoch erneut durch die Zuteilungen von je zwei Tasks im jeweils selben Hormonzyklus erklären lassen.

7.4.1.3. $m = 64$ Tasks auf 16 PEs

Eine Adaption des aus Abbildung 7.10 bekannten Szenarios mit 64 Tasks, die auf 16 PEs zugeteilt wurden, liefert für $i = 4$ insgesamt 16 Prioritätslevel zu je 4 Tasks.

Der Verlauf der Selbstkonfiguration ist für dieses Szenario in Abbildung 7.18 aufgetragen. Bei genauerer Betrachtung ist auch hier gut zu erkennen, dass Wartezyklen nur nach vollständiger Zuteilung jedes Prioritätslevels auftreten. Die Selbstkonfiguration dauerte hier mit insgesamt 67 Hormonzyklen wiederum deutlich kürzer als der theoretische Worst Case von 79 Hormonzyklen, was abermals durch gelegentliche Zuteilungen von jeweils mehr als einer Task pro Hormonzyklus zu erklären ist.

Tabelle 7.3 listet die Anzahl der Hormonzyklen seit dem Beginn der Selbstkonfiguration auf, zu denen alle Tasks jedes Prioritätslevels zugeteilt sind und stellt diesen Werten

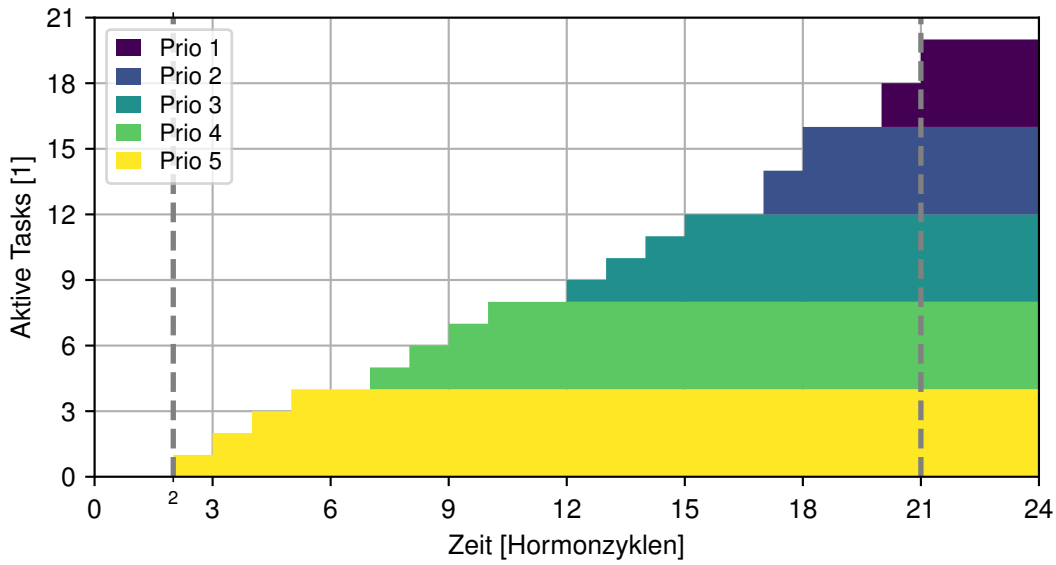


Abbildung 7.17: Zeitlicher Verlauf der Selbstkonfiguration für $m = 20$ Tasks und $i = 4$ Tasks pro Prioritätslevel auf 5 PEs

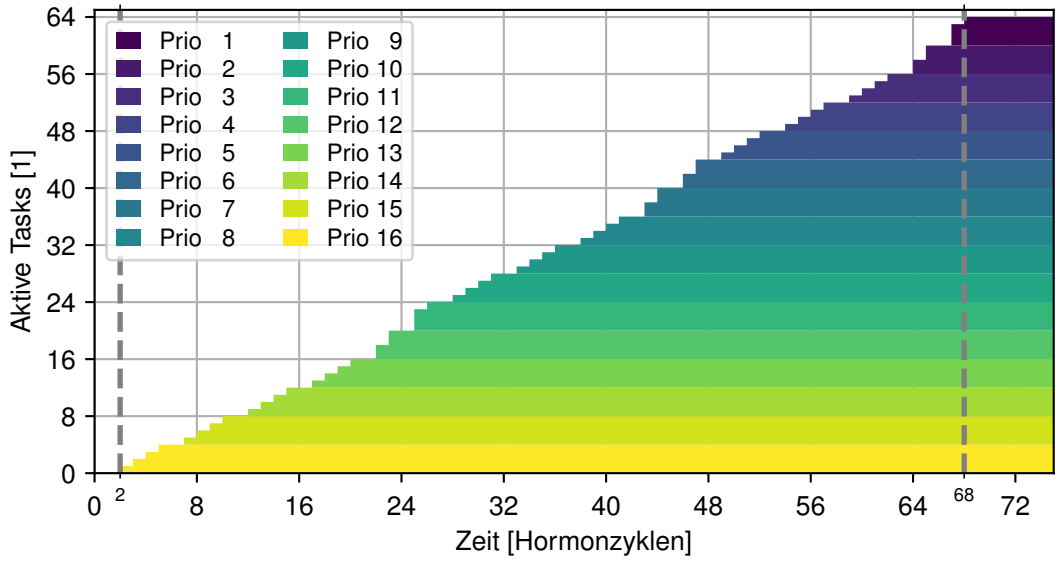


Abbildung 7.18: Zeitlicher Verlauf der Selbstkonfiguration für $m = 64$ Tasks und $i = 4$ Tasks pro Prioritätslevel auf 16 PEs

Tabelle 7.3: Dauer der Zuteilung der einzelnen Prioritätslevel für $m = 64$ Tasks und $i = 4$ Tasks pro Priorität gemäß Abbildung 7.18 sowie Abschätzung durch obere Schranke für $D(T)$ aus Satz 6.12

Prioritäts-level	**Vollständig zugeteilt nach [Hormonzyklen]**	**Abschätzung durch $D(T)$ [Hormonzyklen]**
16	4	$\leq$ 4
15	9	$\leq$ 9
14	14	$\leq$ 14
13	19	$\leq$ 19
12	22	$\leq$ 24
11	25	$\leq$ 29
10	30	$\leq$ 34
9	35	$\leq$ 39
8	40	$\leq$ 44
7	43	$\leq$ 49
6	46	$\leq$ 54
5	51	$\leq$ 59
4	56	$\leq$ 64
3	61	$\leq$ 69
2	64	$\leq$ 74
1	67	$\leq$ 79

die berechneten oberen Schranken gegenüber. Hier ist zu erkennen, dass Priorität 12 das erste Prioritätslevel ist, das schneller vollständig zugeteilt wird als durch $D(T)$ abgeschätzt; ein Vergleich mit dem zugehörigen Diagramm lässt erkennen, dass in diesem Prioritätslevel zum ersten Mal mehr als eine Task pro Hormonzyklus zugeteilt wurden, was diesen Effekt erklärt. Entsprechend weisen die Prioritätslevel 16–13 im betrachteten Fall gerade die theoretischen Worst-Case-Zuteilungsdauern auf, während alle übrigen Prioritäten ihrem jeweiligen Worst Case vorauseilen.

7.4.1.4. $m = 256$ Tasks auf 64 PEs

Mit $i = 4$ Tasks pro Prioritätslevel ergeben sich für das ursprünglich aus Abbildung 7.12 bekannte Szenario mit $m = 256$ Tasks auf 64 PEs insgesamt $|P| = 64$ Prioritätslevel zu je 4 Tasks.

Aus Gründen der Übersichtlichkeit wird hier auf die grafische Darstellung der Ergebnisse verzichtet; die gesamte Selbstkonfiguration war nach 271 Hormonzyklen jedoch erneut deutlich schneller abgeschlossen, als von der Worst-Case-Zeitschranke zu $m + |P| - 1 = 319$ Zyklen abgeschätzt. Dieser Trend setzt sich fort, wenn die Dauern der vollständigen Zuteilung der einzelnen Prioritätslevel betrachtet werden, die in Tabelle 7.4 aufgelistet sind: Lediglich für das höchstpriore Prioritätslevel 64 wird die durch $D(T)$ abgeschätzte Worst-Case-Schranke von 4 Hormonzyklen erreicht, während

alle niedrigeren Prioritäten ihre entsprechenden Schranken unterschreiten, da häufig mehrere Tasks pro Hormonzyklus im System zugeteilt werden können.

7.4.2. Szenarien mit $i = 8$ Tasks pro Prioritätslevel

Für die nächsten Evaluationen wurde die Anzahl der Prioritätslevel reduziert, indem nun $i = 8$ Tasks pro Prioritätslevel zum Einsatz kamen.

Hierbei wurde jedoch auf eine Diskussion des Szenarios mit $m = 10$ Tasks verzichtet, da hier – bis auf einen einzigen auftretenden Wartezyklus zwischen den beiden Prioritätsleveln (die 8 bzw. 2 Tasks umfassen), aufgrund dessen die Zeitschranke der Selbstkonfiguration um einen Hormonzyklus höher liegt – keine nennenswerten Abweichungen vom für $|P| = 1$ aus Abschnitt 7.3.1 bekannten Verhalten zu beobachten waren.

7.4.2.1. $m = 20$ Tasks auf fünf PEs

Für das bekannte Szenario mit $m = 20$ Tasks auf fünf PEs ergeben sich bei $i = 8$ entsprechend drei Prioritätslevel mit 8, 8 und 4 Tasks. Der zeitliche Verlauf der Selbstkonfiguration ist in Abbildung 7.19 aufgetragen, erneut sind die Wartezyklen nach vollständiger Zuteilung jedes Prioritätslevels gut zu erkennen.

Die gesamte Selbstkonfiguration dauert mit 20 Hormonzyklen zwei Zyklen kürzer als von der Worst-Case-Zeitschranke zu $m + |P| - 1 = 22$ Hormonzyklen abgeschätzt, dies ist abermals durch die Zyklen 4 und 19 zu erklären, in denen je zwei Tasks gleichzeitig im System zugeteilt werden. Entsprechend liegen auch die Zeitpunkte, zu denen die einzelnen Prioritätslevel vollständig zugeteilt sind, mit 7, 16 und 20 Hormonzyklen nach Beginn der Selbstkonfiguration unter den Schranken, die anhand von $\mathrm{D}(T)$ für Vertreter der einzelnen Prioritäten abgeschätzt werden können (8, 17 und 22 Hormonzyklen).

7.4.2.2. $m = 64$ Tasks auf 16 PEs

Für das Szenario aus $m = 64$ Tasks auf 16 PEs ergeben sich mit $i = 8$ insgesamt $|P| = 8$ Prioritätslevel zu je 8 Tasks. Der Verlauf der Selbstkonfiguration ist Abbildung 7.20 zu entnehmen.

Hier dauert die Selbstkonfiguration mit 50 Hormonzyklen abermals deutlich kürzer als die im Worst Case erwarteten 71 Zyklen. Tabelle 7.5 zeigt ferner, dass auch die Zuteilungsdauern der einzelnen Prioritätslevel stets unter den Worst-Case-Abschätzungen liegen, die durch $\mathrm{D}(T)$ erhalten werden können.

7.4.2.3. $m = 256$ Tasks auf 64 PEs

Für $m = 256$ Tasks ergeben sich für $i = 8$ nun $|P| = 32$ Prioritätslevel zu je 8 Tasks, die wie bisher auf 64 PEs zugeteilt wurden. Die Reduktion der Anzahl verschiedener Prioritäten erlaubt mit Abbildung 7.21 nun erstmals eine übersichtliche Darstellung dieser Skalierungsstufe.

Tabelle 7.4: Dauer der Zuteilung der einzelnen Prioritätslevel für $m = 256$ Tasks und $i = 4$ Tasks pro Priorität sowie Abschätzung durch obere Schranke für $\mathrm{D}(T)$ aus Satz 6.12. Alle Angaben in Hormonzyklen.

Priorität	Zugeteilt	Abschätzung	Priorität	Zugeteilt	Abschätzung
64	4	$\leq$ 4	**32**	140	$\leq$ 164
63	8	$\leq$ 9	**31**	143	$\leq$ 169
62	13	$\leq$ 14	**30**	148	$\leq$ 174
61	17	$\leq$ 19	**29**	153	$\leq$ 179
60	22	$\leq$ 24	**28**	158	$\leq$ 184
59	27	$\leq$ 29	**27**	161	$\leq$ 189
58	32	$\leq$ 34	**26**	164	$\leq$ 194
57	35	$\leq$ 39	**25**	169	$\leq$ 199
56	38	$\leq$ 44	**24**	174	$\leq$ 204
55	43	$\leq$ 49	**23**	179	$\leq$ 209
54	48	$\leq$ 54	**22**	182	$\leq$ 214
53	53	$\leq$ 59	**21**	185	$\leq$ 219
52	56	$\leq$ 64	**20**	190	$\leq$ 224
51	59	$\leq$ 69	**19**	195	$\leq$ 229
50	64	$\leq$ 74	**18**	200	$\leq$ 234
49	69	$\leq$ 79	**17**	204	$\leq$ 239
48	74	$\leq$ 84	**16**	207	$\leq$ 244
47	77	$\leq$ 89	**15**	212	$\leq$ 249
46	80	$\leq$ 94	**14**	217	$\leq$ 254
45	85	$\leq$ 99	**13**	222	$\leq$ 259
44	90	$\leq$ 104	**12**	225	$\leq$ 264
43	95	$\leq$ 109	**11**	228	$\leq$ 269
42	98	$\leq$ 114	**10**	233	$\leq$ 274
41	101	$\leq$ 119	**9**	238	$\leq$ 279
40	106	$\leq$ 124	**8**	243	$\leq$ 284
39	111	$\leq$ 129	**7**	246	$\leq$ 289
38	116	$\leq$ 134	**6**	249	$\leq$ 294
37	119	$\leq$ 139	**5**	254	$\leq$ 299
36	122	$\leq$ 144	**4**	259	$\leq$ 304
35	127	$\leq$ 149	**3**	264	$\leq$ 309
34	132	$\leq$ 154	**2**	268	$\leq$ 314
33	137	$\leq$ 159	**1**	271	$\leq$ 319

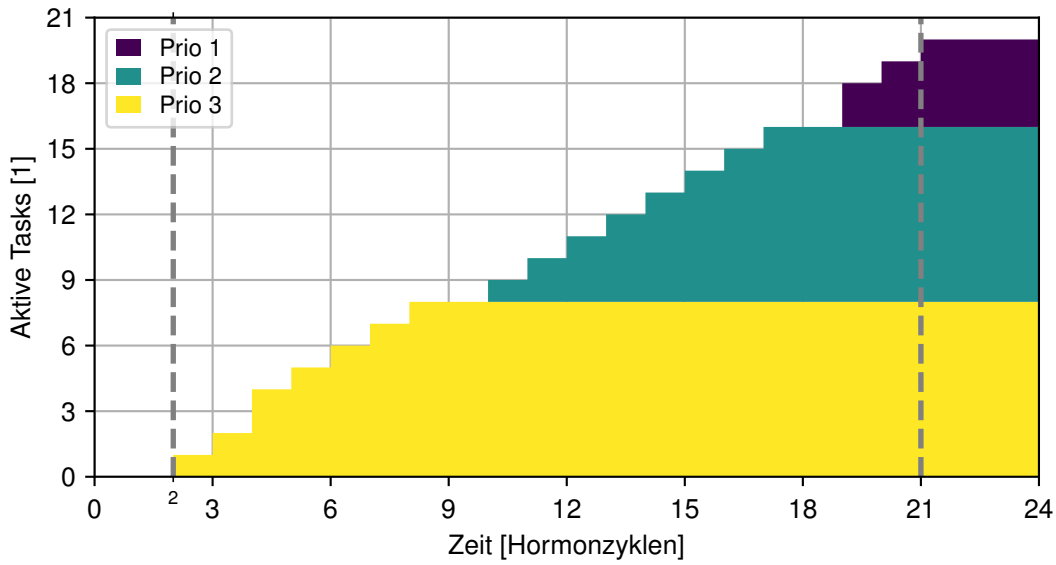


Abbildung 7.19: Zeitlicher Verlauf der Selbstkonfiguration für $m = 20$ Tasks und $i = 8$ Tasks pro Prioritätslevel auf 5 PEs

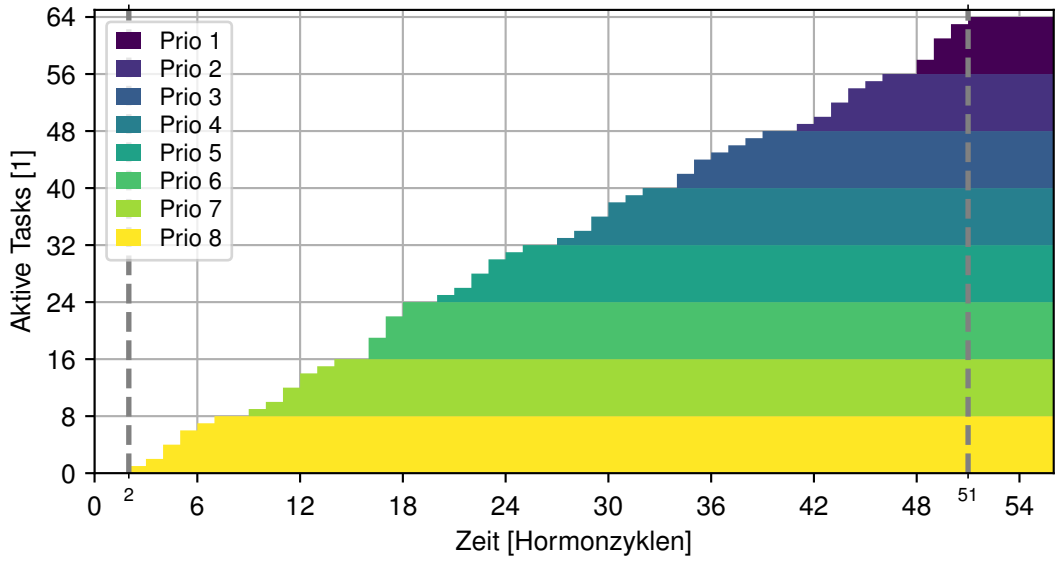


Abbildung 7.20: Zeitlicher Verlauf der Selbstkonfiguration für $m = 64$ Tasks und $i = 8$ Tasks pro Prioritätslevel auf 16 PEs

Tabelle 7.5: Dauer der Zuteilung der einzelnen Prioritätslevel für $m = 64$ Tasks und $i = 8$ Tasks pro Priorität gemäß Abbildung 7.20 sowie Abschätzung durch obere Schranke für $\mathrm{D}(T)$ aus Satz 6.12

Prioritäts-level	Vollständig zugeteilt nach [Hormonzyklen]	Abschätzung durch $\mathrm{D}(T)$ [Hormonzyklen]
8	6	$\leq$ 8
7	13	$\leq$ 17
6	17	$\leq$ 26
5	24	$\leq$ 35
4	31	$\leq$ 44
3	38	$\leq$ 53
2	45	$\leq$ 62
1	50	$\leq$ 71

Die Selbstkonfiguration dauert hier insgesamt 210 Hormonzyklen, was abermals deutlich unter den für den Worst Case erwarteten $m + |P| - 1 = 287$ Zyklen liegt. Tabelle 7.6 zeigt, dass dies erneut auch für die Zuteilungsdauern der einzelnen Prioritätslevel gilt, der Grund hierfür ist abermals in den gelegentlich auftretenden Zuteilungen mehrerer Tasks im gleichen Hormonzyklus zu finden.

7.4.3. Szenarien mit $i = 16$ Tasks pro Prioritätslevel

Weitere Evaluationsszenarien wurden erhalten, indem die Anzahl der Tasks pro Prioritätslevel abermals auf nun $i = 16$ verdoppelt wurde.

Hier degeneriert das Szenario mit $m = 10$ Tasks vollständig, da nun nur noch ein einziges Prioritätslevel existiert und das Szenario daher mit jenem aus Abschnitt 7.3.1 identisch ist. Für $m = 20$ Tasks ergaben sich dagegen noch zwei Prioritätslevel mit 16 bzw. 4 Tasks, was gegenüber dem aus Abschnitt 7.3.1 bekannten Szenario mit $|P| = 1$ jedoch nur marginale Unterschiede im Verhalten aufwies – hier ist vor allem ein Wartezyklus zu nennen, der erwartungsgemäß zwischen der vollständigen Zuteilung des ersten Prioritätslevels und dem Anbruch des zweiten Prioritätslevels auftrat. Aus diesen Gründen werden im Folgenden nur die beiden größten Skalierungsstufen mit $m = 64$ sowie $m = 256$ Tasks näher betrachtet.

7.4.3.1. $m = 64$ Tasks auf 16 PEs

Für das Szenario mit $m = 64$ Tasks ergeben sich bei $i = 16$ insgesamt $|P| = 4$ Prioritätslevel, die jeweils 16 Tasks enthalten. Im Worst Case sollte die Selbstkonfiguration höchstens $m + |P| - 1 = 67$ Hormonzyklen benötigen, im konkreten Fall ergeben sich 46 Zyklen, wie aus Abbildung 7.22 zu entnehmen ist.

Werden die einzelnen Prioritätslevel separat betrachtet, so dauert deren vollständige Zuteilung 12, 21, 34 bzw. 46 Hormonzyklen, was stets unter den mithilfe von $\mathrm{D}(T)$

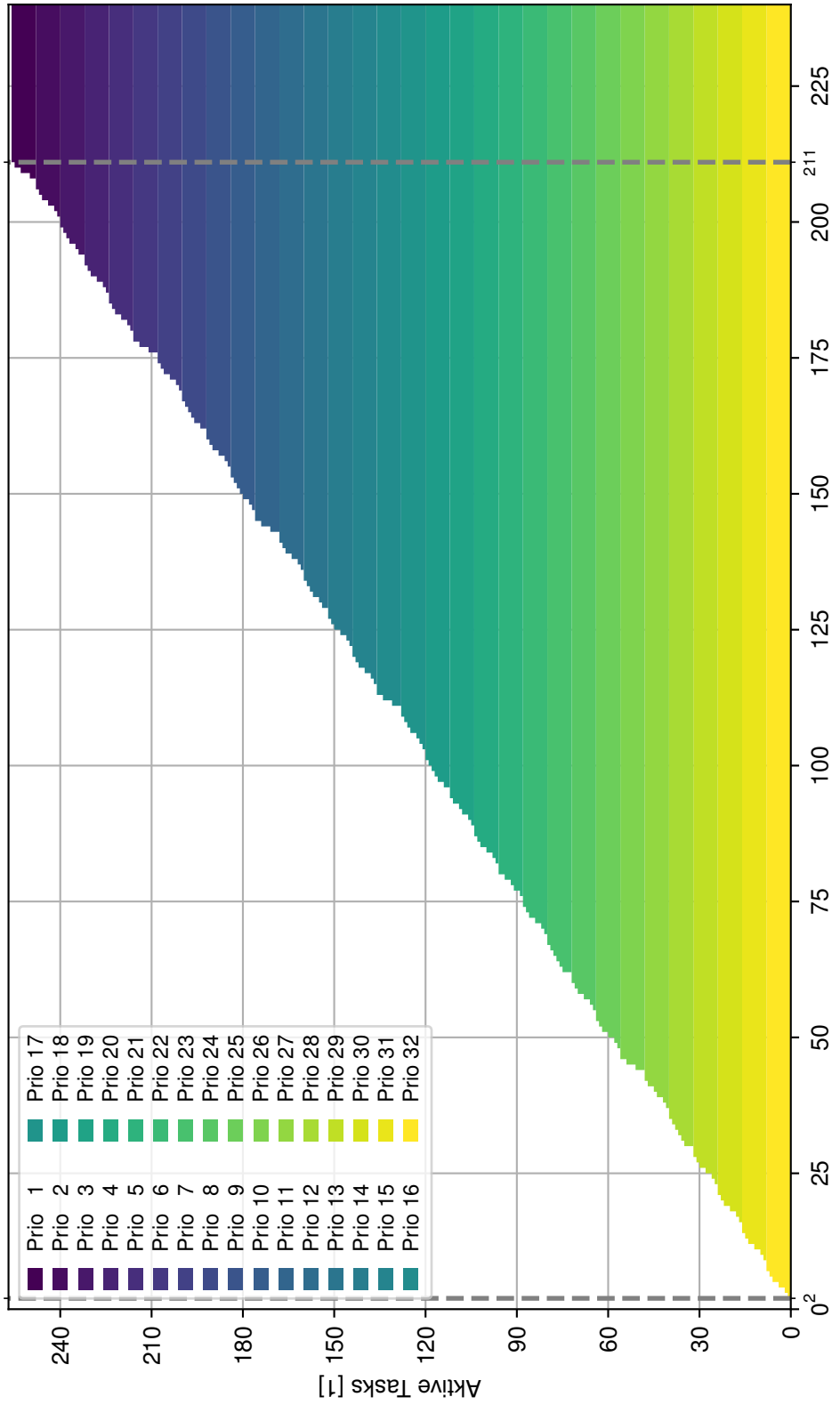


Abbildung 7.21: Zeitlicher Verlauf der Selbstkonfiguration für $m = 256$ Tasks und $i = 8$ Tasks pro Prioritätslevel auf 64 PEs

Tabelle 7.6: Dauer der Zuteilung der einzelnen Prioritätslevel für $m = 256$ Tasks und $i = 8$ Tasks pro Priorität gemäß Abbildung 7.21 und Abschätzung durch obere Schranke für $\mathrm{D}(T)$ aus Satz 6.12. Alle Angaben in Hormonzyklen.

Level	Zugeteilt	Abschätzung	Level	Zugeteilt	Abschätzung
32	6	$\leq$ 8	**16**	112	$\leq$ 152
31	13	$\leq$ 17	**15**	119	$\leq$ 161
30	20	$\leq$ 26	**14**	126	$\leq$ 170
29	27	$\leq$ 35	**13**	133	$\leq$ 179
28	34	$\leq$ 44	**12**	140	$\leq$ 188
27	41	$\leq$ 53	**11**	144	$\leq$ 197
26	45	$\leq$ 62	**10**	152	$\leq$ 206
25	52	$\leq$ 71	**9**	159	$\leq$ 215
24	59	$\leq$ 80	**8**	166	$\leq$ 224
23	66	$\leq$ 89	**7**	173	$\leq$ 233
22	73	$\leq$ 98	**6**	177	$\leq$ 242
21	79	$\leq$ 107	**5**	184	$\leq$ 251
20	86	$\leq$ 116	**4**	191	$\leq$ 260
19	93	$\leq$ 125	**3**	198	$\leq$ 269
18	100	$\leq$ 134	**2**	205	$\leq$ 278
17	108	$\leq$ 143	**1**	210	$\leq$ 287

abgeschätzten Höchstwerten von 16, 33, 50 und 67 Zyklen liegt und damit dem allgemein beobachtbaren Trend entspricht.

7.4.3.2. $m = 256$ Tasks auf 64 PEs

Für die größte betrachtete Skalierungsstufe ergeben sich $|P| = 16$ verschiedene Prioritäten. Die Selbstkonfiguration sollte hier im Worst Case höchstens $m + |P| - 1 = 271$ Hormonzyklen dauern, im konkreten Fall ist sie bereits nach 171 Hormonzyklen – und damit 100 Zyklen früher – abgeschlossen, wie Abbildung 7.23 zeigt.

Tabelle 7.7 stellt die beobachteten Zuteilungsdauern der Prioritätslevel den durch $\mathrm{D}(T)$ abgeschätzten in gewohnter Weise gegenüber; wie angesichts der großen Differenz von 100 Hormonzyklen für die Dauer der gesamten Selbstkonfiguration zu erwarten war, eilen die tatsächlichen Zuteilungsdauern den Abschätzungen stets voraus.

7.4.4. Szenarien mit $i = 16$ Tasks pro Prioritätslevel, ohne Acceleratoren

Zuletzt soll noch eine Variante der beiden letzten betrachteten Szenarien untersucht werden, für die das Senden von Acceleratoren abgeschaltet wurde. Folglich kann keine Organbildung verwandter Tasks auf benachbarten PEs erfolgen, das AHS führt daher lediglich eine Lastverteilung durch.

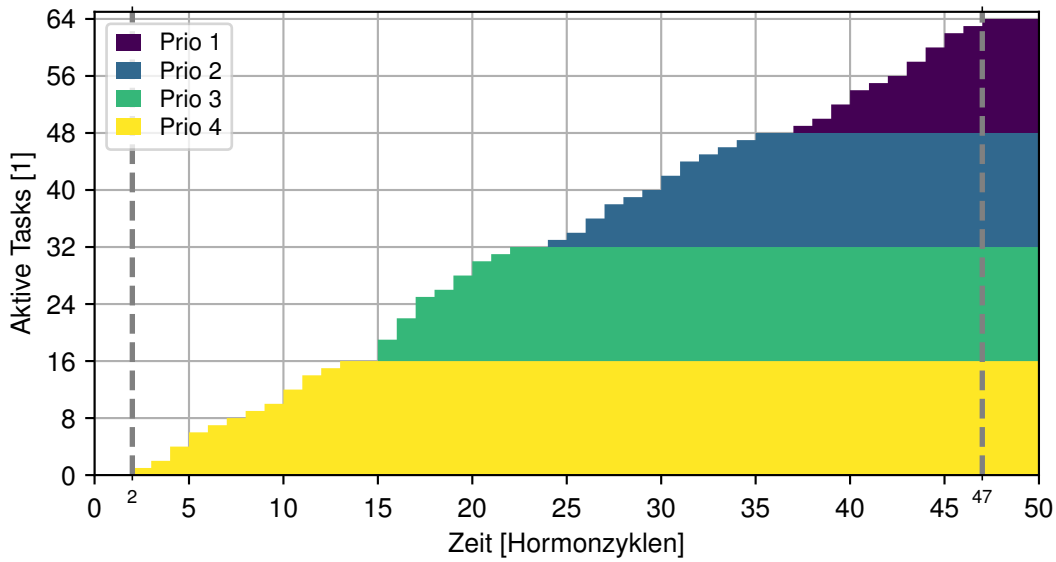


Abbildung 7.22: Zeitlicher Verlauf der Selbstkonfiguration für $m = 64$ Tasks und $i = 16$ Tasks pro Prioritätslevel auf 16 PEs

Tabelle 7.7: Dauer der Zuteilung der einzelnen Prioritätslevel für $m = 256$ Tasks und $i = 16$ Tasks pro Priorität gemäß Abbildung 7.23 und Abschätzung durch obere Schranke für $\mathrm{D}(T)$ aus Satz 6.12. Alle Angaben in Hormonzyklen.

Level	Zugeteilt	Abschätzung
16	10	$\leq$ 16
15	21	$\leq$ 33
14	32	$\leq$ 50
13	41	$\leq$ 67
12	52	$\leq$ 84
11	63	$\leq$ 101
10	74	$\leq$ 118
9	85	$\leq$ 135

Level	Zugeteilt	Abschätzung
8	94	$\leq$ 152
7	107	$\leq$ 169
6	117	$\leq$ 186
5	128	$\leq$ 203
4	139	$\leq$ 220
3	150	$\leq$ 237
2	161	$\leq$ 254
1	171	$\leq$ 271

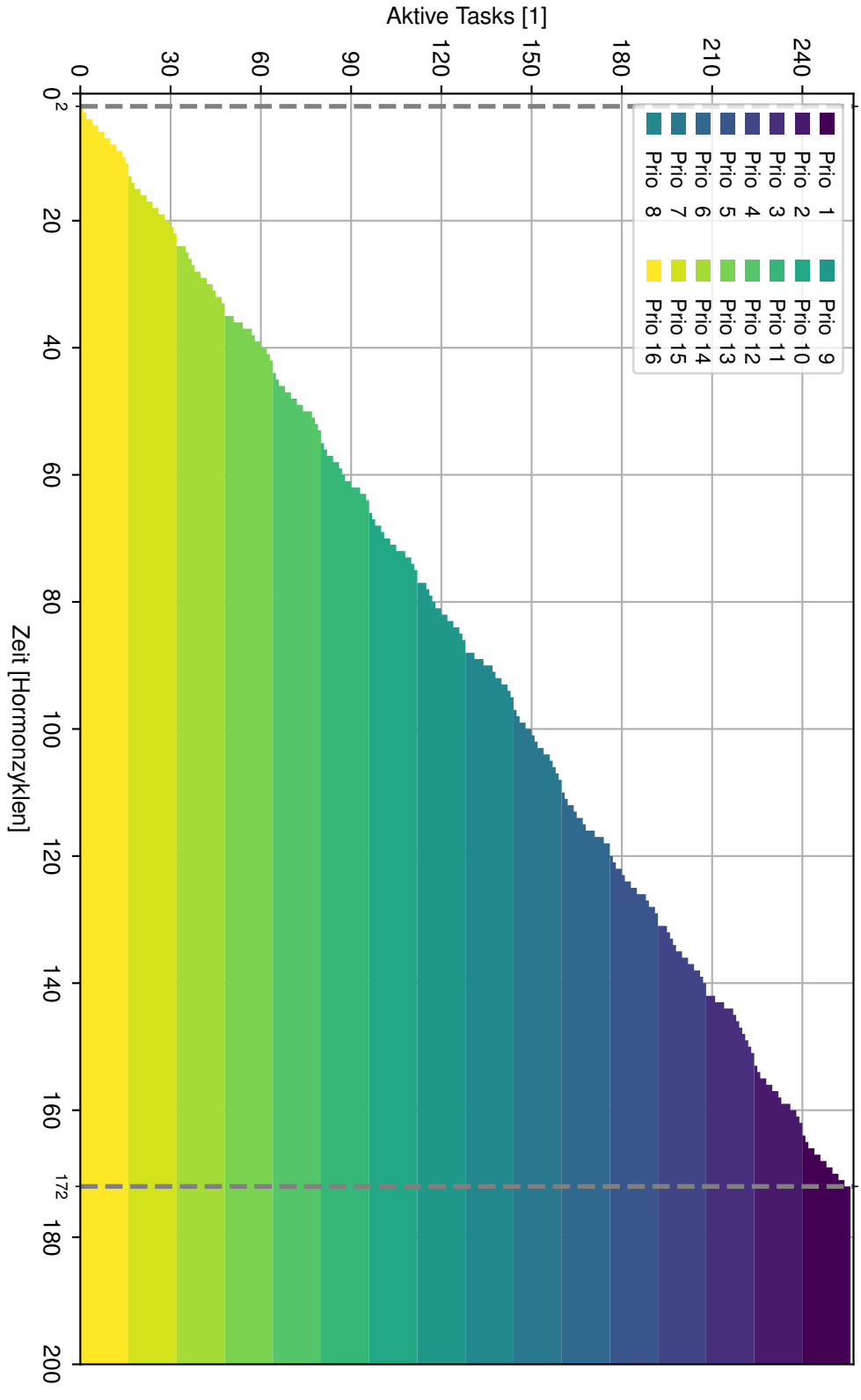


Abbildung 7.23: Zeitlicher Verlauf der Selbstkonfiguration für $m = 256$ Tasks und $i = 16$ Tasks pro Prioritätslevel auf 64 PEs

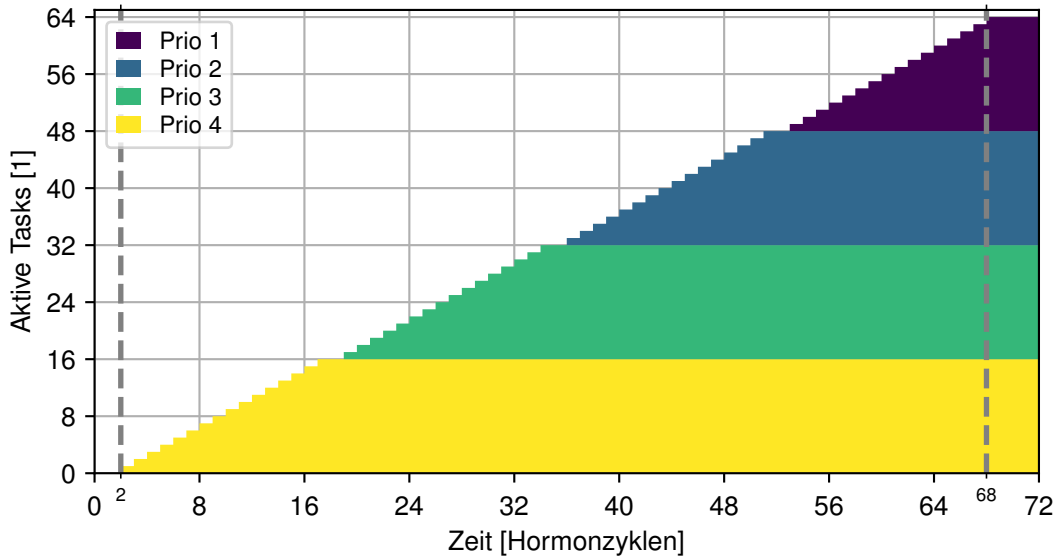


Abbildung 7.24: Zeitlicher Verlauf der Selbstkonfiguration für $m = 64$ Tasks und $i = 16$ Tasks pro Prioritätslevel auf 16 PEs bei abgeschalteten Acceleratoren

7.4.4.1. $m = 64$ Tasks auf 16 PEs

Für $m = 64$ Tasks ergab sich hierbei der in Abbildung 7.24 gezeigte Verlauf der Selbstkonfiguration. Darin ist deutlich zu erkennen, dass innerhalb jedes Prioritätslevels pro Hormonzyklus genau eine Task zugeteilt wird und zwischen zwei aufeinanderfolgenden Prioritätsleveln jeweils ein Wartezyklus auftritt; entsprechend wird die Worst-Case-Zeitschranke von $m+|P|-1 = 67$ Hormonzyklen hier exakt erreicht. Dies gilt ebenso für die Zuteilungsdauern der einzelnen Prioritätsleveln von 16, 33, 50 und 67 Hormonzyklen, die von $\mathrm{D}(T)$ als obere Schranke abgeschätzt werden.

7.4.4.2. $m = 256$ Tasks auf 64 PEs

Dieser Effekt lässt sich ebenfalls im $m = 256$ Tasks umfassenden Szenario beobachten, auf die Abbildung des zugehörigen Diagramms wurde jedoch aus Platzgründen verzichtet. Hier dauerte die Selbstkonfiguration exakt die abgeschätzten $m+|P|-1 = 271$ Hormonzyklen, auch die Zuteilungsdauern der einzelnen Prioritätslevel entsprachen genau den mithilfe von $\mathrm{D}(T)$ abgeschätzten Schranken, die bereits aus Tabelle 7.7 für das Szenario mit aktivierten Acceleratoren bekannt sind.

7.4.5. Diskussion der Szenarien mit weniger Prioritäten als Tasks

In diesem Abschnitt wurde der allgemeine Fall betrachtet, in dem $1 < |P| < m$ verschiedene Prioritäten existieren. Hier zeigte sich zum einen, dass die prioritätsbasierte Task-Decision-Strategie stets alle Tasks unter Beachtung ihrer Prioritäten zuteilte.

Zum anderen erinnerte der Verlauf der Selbstkonfiguration *innerhalb* jedes Prioritätslevels an das Verhalten der *aggressiven* Strategie, da häufig mehrere Tasks im selben Hormonzyklus zugeteilt wurden; lediglich zwischen der vollständigen Zuteilung eines Prioritätslevels und dem Anbruch des nächstniedrigeren Levels trat stets ein Wartezyklus auf. Dies entspricht jedoch gerade dem in Kapitel 6.4.1 theoretisch vorhergesagten Verhalten, anhand dessen insbesondere die Zeitschranke von $m + |P| - 1$ Hormonzyklen aus Satz 6.9 hergeleitet wurde.

Konkret wurde diese Zeitschranke niemals überschritten, sondern zumeist unterschritten. Selbiges galt für die Zuteilungsdauern der einzelnen Prioritätslevel, wie sie durch Satz 6.12 mithilfe von $\mathrm{D}(T)$ abgeschätzt wurden. Diese Unterschreitungen sind durch Hormonzyklen zu begründen, in denen im gesamten System mehr als eine Task zugeteilt wird. Entsprechend lagen die Zuteilungsdauern der meisten niedrigen Prioritätslevel unter den durch $\mathrm{D}(T)$ abgeschätzten Werten, zumeist erreichten lediglich einige wenige der höchsten Prioritätslevel die theoretische Schranke. Beim Vergleich der Werte für $m = 64$ Tasks aus Tabelle 7.3 (für $i = 4$ Tasks pro Priorität), Tabelle 7.5 ($i = 8$) sowie den in Abschnitt 7.4.3.1 für $i = 16$ aufgelisteten Werten zeigt sich, dass die Zuteilungsdauer der einzelnen Level den Schranken umso stärker vorauseilt, je mehr Tasks pro Prioritätslevel existieren; ab $i = 8$ wird die theoretische Schranke gar für jedes Prioritätslevel unterschritten. Dieser Trend konnte auch für das Szenario mit $m = 256$ Tasks beobachtet werden.

Eine Ausnahme stellten lediglich die Varianten mit abgeschalteten Acceleratoren dar: Hier wurden alle theoretischen Zeitschranken stets exakt erreicht, da pro Hormonzyklus im gesamten System nie mehr als eine Task zugeteilt wurde. Dieses Resultat ist jedoch keineswegs verwunderlich: Kommen keine Acceleratoren zum Einsatz, so bestimmt sich der Gewinner einer jeden Task nur aus dem lokalen statischen Eignungswert (der in den betrachteten Szenarien für jede Task auf jedem PE identisch war) sowie den lokal wirkenden Lastsuppressoren aufgrund zuvor übernommener Tasks. Entsprechend sendet in jedem Zyklus stets dasselbe PE für *jede* noch zu übernehmende Task den unter allen PEs höchsten modifizierten Eignungswert, weshalb dieses eine beliebige Task des höchsten zu vergebenen Prioritätslevels übernimmt (mit Ausnahme von Wartezyklen zwischen je zwei aufeinanderfolgenden Prioritätsleveln), eine gleichzeitige Übernahme mehrerer Tasks auf verschiedenen PEs ist daher nicht möglich.

Der Einsatz von Acceleratoren dagegen erhöht die modifizierten Eignungswerte, welche von PEs in der Nachbarschaft einer bereits zugeteilten Task T für mit T verwandte Tasks gesendet werden. Dies ermöglicht zum einen die Bildung virtueller Organe auf diesen PEs und erlaubt zum anderen, dass verschiedene PEs für verschiedene Tasks gleichzeitig den jeweils höchsten modifizierten Eignungswert senden können. Somit kann im Gesamtsystem in manchen Hormonzyklen mehr als eine Task gleichzeitig zugeteilt werden. Dass dies bei wenigen Tasks pro Prioritätslevel für die höchstprioren (und daher zuerst zugeteilten) Prioritäten meist nicht auftrat, ist damit zu begründen, dass zunächst eine bestimmte Anzahl Tasks zugeteilt worden sein muss, um genügend Acceleratoren auszuschütten, damit der beschriebene Beschleunigungseffekt auftreten kann.

Es sei jedoch darauf hingewiesen, dass das erst späte Eintreten dieses Effekts dennoch

keinesfalls als Nachteil anzusehen ist: In [BP12] wurde anhand eines in [BvRP08] publizierten Qualitätsmaßes gezeigt, dass die Qualität der Taskzuteilung leiden kann, wenn die Tasks *zu* schnell zugeteilt werden, da hierdurch die Bildung virtueller Organe durch Acceleratoren beeinträchtigt wird.

7.5. Zusammenfassung der Resultate

Zum Zwecke der Evaluation der Selbstkonfiguration unter Verwendung der prioritätsbasierten Task-Decision-Strategie wurden drei Fälle betrachtet:

Abschnitt 7.2 betrachtete den Sonderfall, in dem alle Tasks verschiedene Prioritäten besitzen, also $|P| = m$ gilt. Hier wurde die theoretisch hergeleitete Zeitschranke von höchstens $m + |P| - 1 = 2m - 1$ Hormonzyklen stets exakt erreicht, selbiges galt für die Zuteilungsdauer der einzelnen Tasks gemäß Satz 6.12. Für den Sonderfall, in dem nur ein PE existiert (oder nur ein einziges PE positive modifizierte Eignungswerte sendet), traten keine Wartezyklen auf, wodurch Bemerkung 6.8 experimentell bestätigt werden konnte.

Der Sonderfall mit $|P| = 1$ zeigte, dass die prioritätsbasierte Task-Decision-Strategie im Fall der Selbstkonfiguration vollständig zur ihr zugrunde liegenden aggressiven Strategie degeneriert, sofern nur eine einzige Priorität existiert. In diesem Fall ist es nicht möglich, die Zuteilungsdauer einzelner Tasks genauer abzuschätzen als durch die Dauer der *gesamten* Selbstkonfiguration, entsprechend liefert die Abschätzung aus Satz 6.12 hier ebenso wie Satz 6.9 die für die aggressive Strategie bekannten m Hormonzyklen. In den betrachteten Szenarien wurde diese Zeitschranke zudem nie exakt erreicht, sondern stets unterschritten; da jedes PE pro Hormonzyklus höchstens eine Task übernehmen kann, ist jedoch leicht einzusehen, dass ein aus nur einem *einzigen* PE bestehendes System diese Zeitschranke dagegen exakt erreichen muss.

Abschnitt 7.4 betrachtete schließlich den allgemeinen Fall mit $1 < |P| < m$ verschiedenen Prioritäten. Auch hier zeigte sich die Gültigkeit der theoretisch hergeleiteten Zeitschranken, die im Falle abgeschalteter Acceleratoren sogar stets exakt erreicht wurden. Ferner zeigte sich hier *innerhalb* jedes Prioritätslevels erneut das Verhalten der aggressiven Task-Decision-Strategie, während zwischen der Zuteilung aufeinanderfolgender Prioritätslevel jeweils stets ein Wartezyklus auftrat, in dem keine Task im System zugeteilt werden konnte.

Zusammenfassend lässt sich daher festhalten, dass die theoretisch hergeleiteten Zeitschranken experimentell bestätigt werden konnten. Ferner wurden diese Schranken in verschiedenen Szenarien jeweils exakt erreicht, woraus gefolgert werden kann, dass eine noch schärfere Abschätzung der Dauer der Selbstheilung im Worst-Case bzw. der Zuteilungsdauer bestimmter Prioritätslevel im Allgemeinen nicht möglich ist.

7.6. Vergleich mit der AHS-Middleware

Wie zuvor beschrieben, wurden alle bisherigen Evaluationen aufgrund der einfacheren Handhabung im Hormonsimulator durchgeführt. Um zu überprüfen, ob die so gewon-

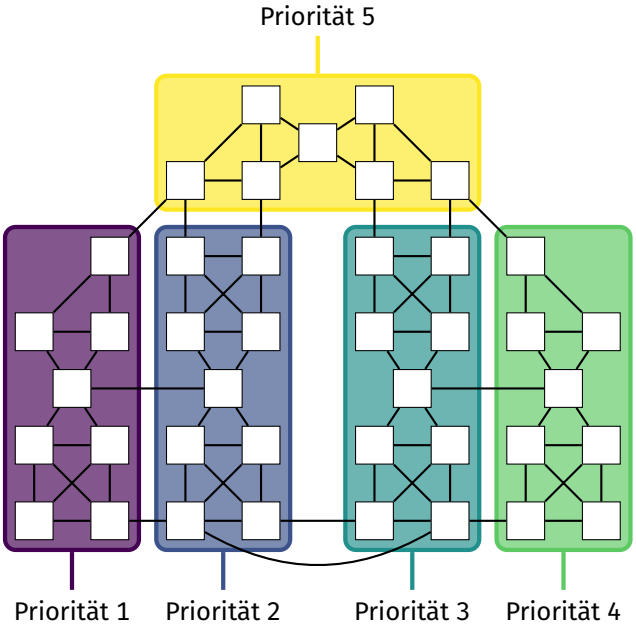


Abbildung 7.25: Tasks des Evaluationsszenarios für den Vergleich zwischen Hormonsimulator und AHS-Middleware

nenen Resultate sich auch auf die AHS-Middleware übertragen lassen, sollen beide Implementierungen der prioritätsbasierten Task-Decision-Strategie nun anhand eines exemplarischen Szenarios miteinander verglichen.

Gleichzeitig soll dieser Vergleich der beiden Implementierungen genutzt werden, ein weiteres Evaluationsszenario einzuführen, das sich von den bisherigen Szenarien unterscheidet: Diese sind zumeist aus mehreren Kopien kleinerer Szenarien aufgebaut und bestehen daher im Allgemeinen aus mehreren voneinander isolierten Teilmengen von Tasks, die höchstens mit Tasks *derselben* Teilmenge verwandt sind. Daher wurde für den Vergleich der Implementierungen ein aus $m = 41$ Tasks mit insgesamt $|P| = 5$ verschiedenen Prioritäten bestehendes Szenario konstruiert, das in Abbildung 7.25 schematisch abgebildet ist und insbesondere keine Aufteilung der Tasks in vollständig voneinander isolierte Teilmengen zulässt. Anschließend wurde dieses Szenario in einem Datenformat kodiert, das sowohl vom Hormonsimulator als auch von der Middleware unterstützt wird.

Da vonseiten des Hormonsimulators eine Anordnung der PEs in einer Gittertopologie erwartet wird, wurde für dieses Szenario ein 3×3-Gitter mit insgesamt neun PEs verwendet, die jeweils maximal 14 Tasks übernehmen konnten. Entsprechend wurden die Nachbarschaften zwischen den PEs in der AHS-Middleware, die beliebige Topologien unterstützt, manuell äquivalent konfiguriert.

Abbildung 7.26 zeigt den resultierenden Verlauf der Selbstkonfiguration, wie er in der AHS-Middleware beobachtet werden konnte, gestapelt nach der Anzahl Tasks pro PE sowie den einzelnen Prioritäten. Die vom Hormonsimulator durchgeführte

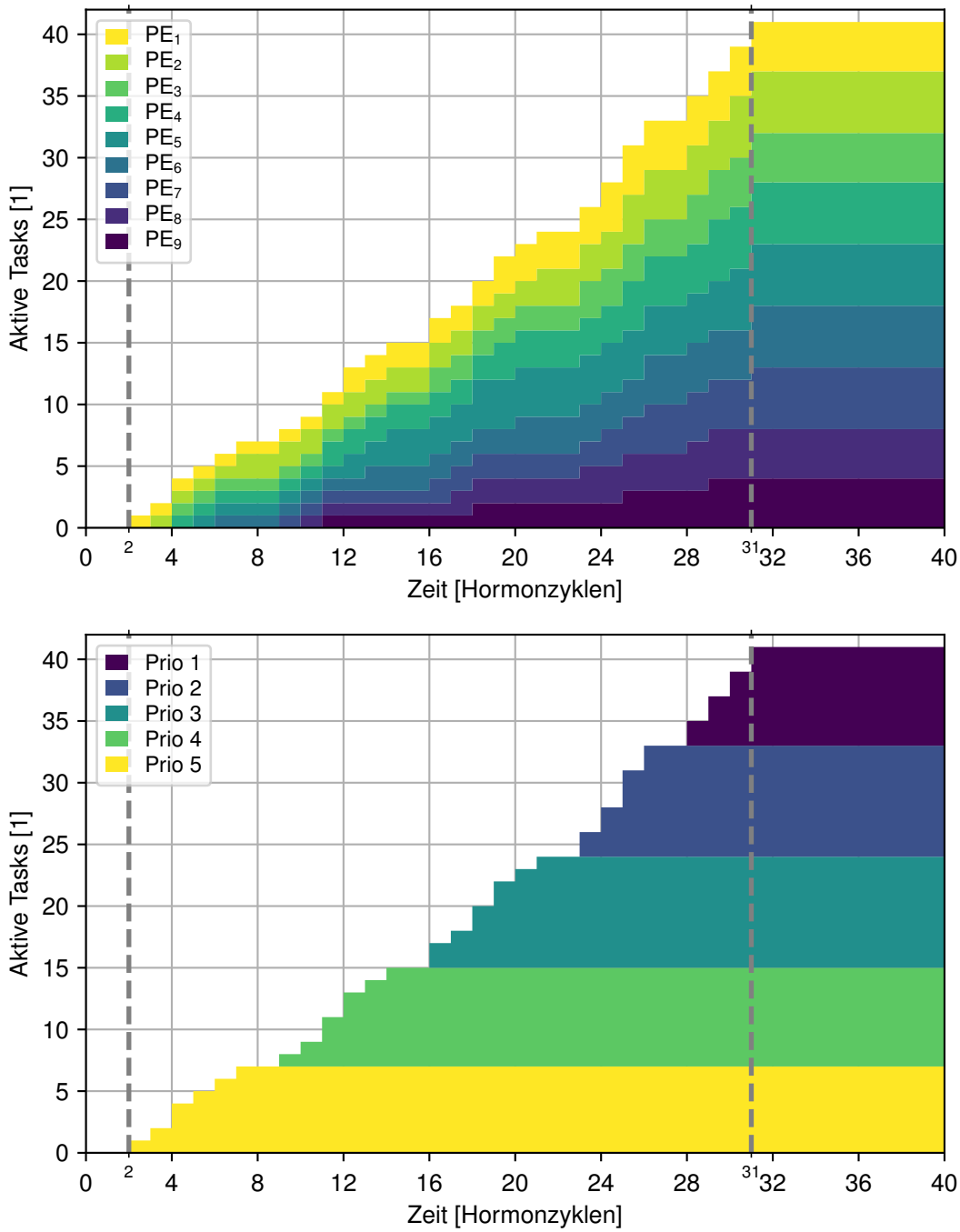


Abbildung 7.26: Verlauf der Selbstkonfiguration für das Evaluationsszenario aus Abbildung 7.25 auf neun PEs in der AHS-Middleware, gestapelt nach der Anzahl Tasks pro PE sowie den Prioritäten

Selbstkonfiguration ist nicht abgebildet, bis auf eine implementierungsbedingt andere Nummerierung der PEs ist diese jedoch zum Verhalten der Middleware identisch. Insbesondere wird jede Task von beiden Implementierungen zum gleichen Zeitpunkt jeweils vom selben PE übernommen.

Entsprechend überrascht es nicht, dass auch dieses Szenario die bereits häufig beobachteten Charakteristika wie beispielsweise die Wartezyklen nach vollständiger Zuteilung jedes Prioritätslevels aufweist. Nach Satz 6.9 dauert eine Selbstkonfiguration hier höchstens $m + |P| - 1 = 45$ Hormonzyklen. Diese Schranke wird mit nur 30 tatsächlich benötigten Hormonzyklen jedoch deutlich unterschritten, da in insgesamt dreizehn Zyklen jeweils zwei Tasks und in Zyklus 25 sogar drei Tasks gleichzeitig im System zugeteilt wurden.

Ebenso liegen die Zuteilungsdauern der einzelnen Prioritätslevel mit 6, 13, 20, 25 und 30 Hormonzyklen durchweg unter den mithilfe von $\mathrm{D}(T)$ bestimmten Schranken von höchstens 7, 16, 26, 36 und 45 Zyklen.

Das beobachtete Verhalten bestätigt daher, dass die beiden Implementierungen der prioritätsbasierten Task-Decision-Strategie in Middleware und Simulator sich für die Selbstkonfiguration identisch verhalten. Entsprechend ist kein abweichendes Verhalten zu erwarten, wenn die bisher ausschließlich im Hormonsimulator durchgeführten Evaluationen stattdessen mit der Middleware wiederholt würden.[1]

Nach diesen Evaluationen widmen wir uns im nächsten Kapitel dieser Dissertation der Analyse von Überlastsituationen, in denen nach dem Ausfall eines PEs die kombinierten Kapazitäten der verbleibenden PEs nicht mehr ausreichen, um *alle* Tasks zu übernehmen, weshalb keine vollständige Selbstheilung mehr durchgeführt werden kann.

[1] Mit Ausnahme der 256 Tasks umfassenden Szenarien, deren Taskanzahl zunächst reduziert werden müsste, um der implementierungsbedingten Beschränkung der Middleware auf 255 Tasks zu genügen.

8 Selbstheilung in Überlastsituationen

In diesem Kapitel soll nun die Selbstheilung des AHS bei Verwendung der prioritätsbasierten Task-Decision-Strategie analysiert werden, wenn eine *Überlastsituation* vorliegt, die Kapazitäten der funktionsfähigen PEs also nicht mehr ausreichen, um *alle* ausgefallenen Tasks zu übernehmen. Einige der im Folgenden vorgestellten Ergebnisse wurden bereits in [HB20] und [HB21b] veröffentlicht.

8.1. Szenario

Um eine Analyse der Selbstheilung in Überlastsituationen jedoch überhaupt erst zu ermöglichen, muss ein Modell für solche Situationen entwickelt werden.

Das System bestehe hierzu aus $v+1$ PEs, die mit $\mathrm{PE}_1 \ldots \mathrm{PE}_v$ sowie $\mathrm{PE}_\times$ bezeichnet seien. Nun falle $\mathrm{PE}_\times$ aus und alle v verbleibenden PEs seien voll ausgelastet (es sendet also jedes der v verbleibenden PEs aufgrund seiner eigenen Lastsuppressoren einen modifizierten Eignungwert $Em_T = 0$ für jede nicht im System zugeteilte Task T). Damit befindet sich das System gemäß Definition 6.2 in einer Überlastsituation.

Es gelte ferner $v \geq 1$, da ansonsten keine Selbstheilung möglich ist: Für $v = 0$ ist mit $\mathrm{PE}_\times$ gerade das letzte verbleibende PE ausgefallen.

Mithilfe zweier Definitionen wollen wir zudem festlegen, *wann* eine Selbstheilung in einer Überlastsituation beendet ist:

Ausfallzeitpunkt von $\mathrm{PE}_\times$, $t_\times$

Definition 8.1. Es bezeichne $t_\times$ den Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$.

Obwohl Hormonzyklen nicht atomar sind und PEs zu jedem Zeitpunkt während eines Hormonzyklus ausfallen können, nehmen wir zudem an, dass $t_\times \in \mathbb{N}$ gilt, wobei $t_\times = i$ bedeute, dass $\mathrm{PE}_\times$ zu einem (nicht näher bestimmten) Zeitpunkt während des i-ten Hormonzyklus ausfällt.

Dauer der Selbstheilung

Definition 8.2. Die *Dauer der Selbstheilung* in einer Überlastsituation sei die Länge der Zeitspanne von $t_\times$ bis

1. der Ausfall von $\mathrm{PE}_\times$ erkannt wurde und
2. wieder ein stabiler Systemzustand erreicht wurde.

Ein stabiler Systemzustand ist erreicht, wenn

1. alle v verbleibenden PEs voll ausgelastet sind und
2. es keine Task gibt, die keinem PE zugeteilt ist und eine höhere Priorität hat als mindestens eine der zugeteilten Tasks.

Bemerkung 8.3

Gemäß dieser Definition ist die Selbstheilung in einer Überlastsituation frühestens beendet, wenn der Ausfall von $\text{PE}_\times$ erkannt wurde.

Dies ist relevant, falls das System durch den Ausfall von $\text{PE}_\times$ zwar in eine Überlastsituation gerät, jedoch bereits ein stabiler Systemzustand vorliegt. In einem solchen Fall sind zwar nicht alle Tasks im System zugeteilt, aufgrund des Vorliegens der Überlastsituation werden jedoch keine Aktionen durchgeführt, da keine Tasks niedriger Priorität zugunsten von Tasks höherer Priorität gestoppt werden müssen.

Die Definition fordert daher, dass die Selbstheilung in einer Überlastsituation mindestens so lange dauert, bis der Ausfall von $\text{PE}_\times$ erkannt wurde, da erst zu diesem Zeitpunkt festgestellt werden kann, dass Tasks existieren, die im System nicht mehr zugeteilt sind.

Befindet sich das System zum Zeitpunkt des Ausfalls von $\text{PE}_\times$ dagegen *nicht* in einem stabilen Zustand, so können vor der Erkennung des Ausfalls ohnehin noch keine Aktionen durchgeführt werden, weshalb die gesamte Selbstheilung notwendigerweise länger dauern muss als die bloße Erkennung des Ausfalls von $\text{PE}_\times$.

m Es bezeichne ferner m die Anzahl der Tasks, die zu Beginn des Hormonzyklus $t_\times$ auf dem ausgefallenen $\text{PE}_\times$ zugeteilt waren und welche – je nach ihrer Priorität – auf den übrigen PEs neu zugeteilt werden müssen. Damit eine Task T dieser m Tasks überhaupt neu zugeteilt werden kann, muss jedoch mindestens ein verbleibendes PE_j existieren, dessen statischer Eignungswert $E_{T,j} > 0$ ist.

8.2. Annahmen

Zur Beherrschung der Komplexität der Analysen werden im Folgenden einige zusätzliche vereinfachende Annahmen getroffen:

- Es gibt im System keine mehrfach zuteilbaren Tasks, wie es in Kapitel 6.4.1.5 begründet wurde.
- Zum Zeitpunkt des Ausfalls von $\text{PE}_\times$ finden weder Selbstkonfiguration noch Selbstheilung oder Selbstoptimierung statt.
- Während der als Reaktion auf den Ausfall von $\text{PE}_\times$ gestarteten Selbstheilung fällt kein weiteres PE aus, ebenso wird währenddessen auch keine Selbstoptimierung gestartet.
- Monitoring-Suppressoren und -Acceleratoren bleiben für jedes PE während der Selbstheilung konstant, da ansonsten Anomalien auftreten können.[1]

[1] Insbesondere für die später erläuterte Eager-Task-Dropping-Strategie ist diese Annahme relevant: Diese stoppt in aufeinanderfolgenden Selbstheilungsphasen auf jedem PE jeweils genau so viele Tasks, dass das PE im nächsten Hormonzyklus ausreichende Kapazitäten für die Übernahme der höchstprioren nicht zugeteilten Task hat. Nehmen in der Zwischenzeit jedoch Monitoring-Suppressoren zu bzw.

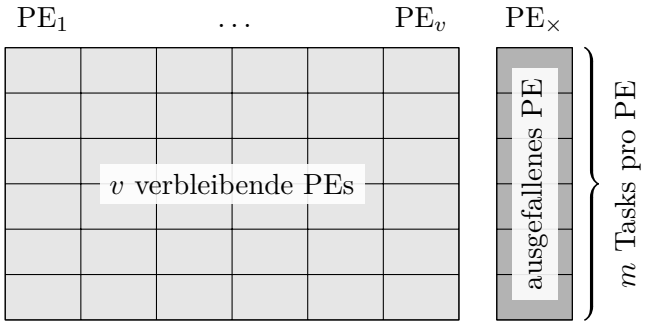


Abbildung 8.1: Für Überlastsituationen verwendetes Modell

- Jede Task kann prinzipiell auf jedem der verbleibenden PEs ausgeführt werden, es gibt also kein PE, das für eine Task einen statischen Eignungswert von Null besitzt.
- Der Lastsuppressor ist für alle Tasks auf allen PEs identisch und wird im Folgenden o.B.d.A. zu 1 angenommen.
- Jedes der v verbleibenden PEs kann höchstens m Tasks ausführen. Da eine Überlastsituation vorliegt, bedeutet dies, dass jedes dieser PEs zum Zeitpunkt des Ausfalls von $\text{PE}_\times$ *genau* m Tasks ausführt. Zusammen mit den m ausgefallenen Tasks, die auf $\text{PE}_\times$ zugeteilt waren, existieren im System also genau $|\mathcal{T}| = m{\cdot}(v{+}1)$ Tasks. Von diesen sind zum Zeitpunkt $t_\times$ gerade $m \cdot v$ Tasks auf $\text{PE}_1 \ldots \text{PE}_v$ zugeteilt.

Das aus diesen Annahmen resultierende Modell einer Überlastsituation ist in Abbildung 8.1 skizziert.

Des Weiteren nehmen wir im Folgenden zumeist an, dass $|P| = |\mathcal{T}| = m \cdot (v + 1)$ gilt und folglich alle Tasks verschiedene Prioritäten haben. Dieser Fall stellt gerade den Worst-Case der Selbstheilung dar, da er bei der Zuteilung von Tasks die maximale Anzahl Wartezyklen verursacht: Gemäß Satz 6.10 benötigt die Zuteilung von k Tasks dann höchstens $2k - 1$ Zyklen. Weil der Mechanismus der Selbstkonfiguration auch bei der Selbstheilung zum Einsatz kommt, liefert ein solcher Fall damit auch den Worst-Case der Selbstheilung.

Nachdem nun ein Modell für Überlastsituationen entworfen wurde, ist deren Analyse möglich. In Kapitel 6.3 wurde jedoch offen gelassen, *welche* Tasks ein PE stoppen soll, um in einer solchen Überlastsituation Kapazitäten für eine hochpriore, nicht zugeteilte Task freizumachen. Diese Entscheidung wird von einer *Task-Dropping-Strategie* getroffen. Zwei solcher Strategien werden in den folgenden Abschnitten vorgestellt und im Detail analysiert.

Task-Dropping-Strategie

In den nächsten beiden Abschnitten werden jedoch zunächst einige Vorbereitungen für die späteren Analysen getroffen, indem Eigenschaften der Selbstheilung in

Monitoring-Acceleratoren ab, gilt diese Voraussetzung unter Umständen nicht mehr. Daher wird im Rahmen der Analyse vorausgesetzt, dass ein solcher Fall nicht eintritt.

Überlastsituationen untersucht werden, die unabhängig von der konkret verwendeten Task-Dropping-Strategie gelten.

8.3. Erkennung von Überlastsituationen

Zunächst wollen wir betrachten, wie schnell die v verbleibenden PEs das Vorliegen einer Überlastsituation erkennen und darauf reagieren können. *Wie* eine solche Situation erkannt wird, wurde bereits in Kapitel 6.3.4 beschrieben: Werden für die höchstpriore nicht im System zugeteilte Task T keine positiven modifizierten Eignungswerte sowie mindestens *zwei* Zyklen lang keine Übernahmesuppressoren empfangen, so wird das Vorliegen einer Überlastsituation angenommen und es werden gemäß der verwendeten Task-Dropping-Strategie Tasks gestoppt, um im *darauffolgenden* Hormonzyklus einen positiven modifizierten Eignungswert für T senden zu können.

Der folgende Satz schätzt nun die Zeit ab, die zur Überlasterkennung nötig ist:

Satz 8.4. Das Vorliegen einer Überlastsituation kann frühestens in Zyklus $t_\times + a_{\min}$, spätestens jedoch in Zyklus $t_\times + a$ erkannt werden, was in einem Stopp niedrigpriorer Tasks resultiert. Damit benötigt die Erkennung der Überlastsituation mindestens $a_{\min} + 1$, höchstens jedoch $a + 1$ Zyklen.[1]

Die erste Taskübernahme ist daher frühestens in Zyklus $t_\times + a_{\min} + 1$ und spätestens in Zyklus $t_\times + a + 1$ möglich.

Beweis. Die Argumentation ist zum Teil analog zu jener aus dem Beweis von Satz 5.11; da jedoch eine Überlastsituation vorliegt, sendet jedes PE_j bereits aufgrund seiner Lastsuppressoren für jede Task T einen modifizierten Eignungswert $Em_{T,j} = 0$, selbst wenn keine Übernahmesuppressoren von $\mathrm{PE}_\times$ mehr empfangen werden. Erst der Stopp niedrigpriorer Tasks führt im folgenden Zyklus dazu, dass positive modifizierte Eignungswerte gesendet werden können.

Konkret sind zwei Fälle zu unterscheiden (hierbei gelte o.B.d.A. $t_\times = i$, $\mathrm{PE}_\times$ falle also im i-ten Hormonzyklus aus):

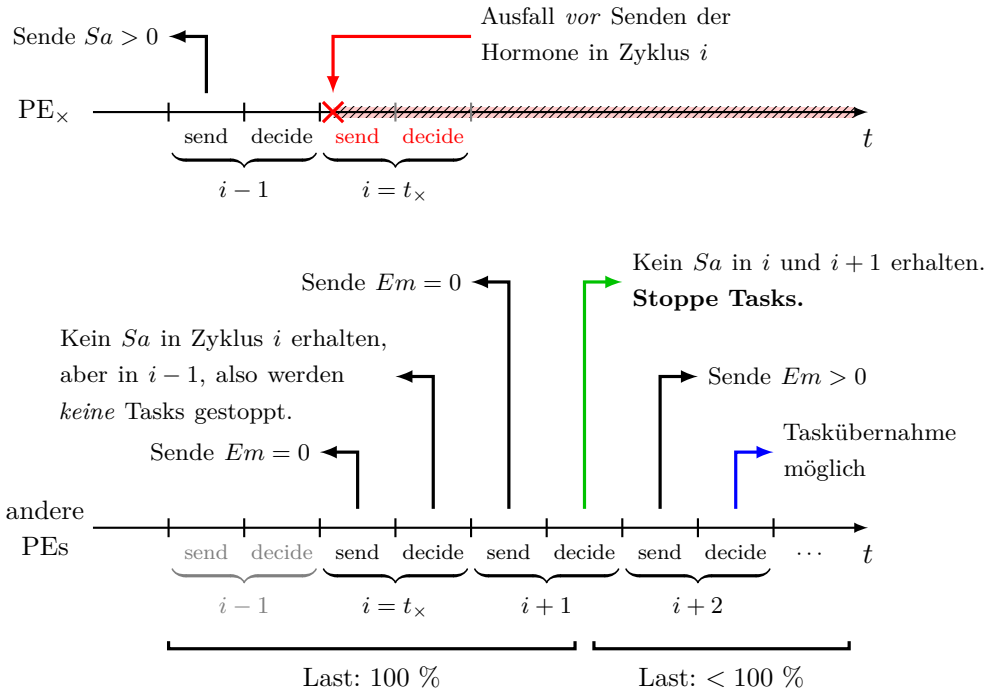
a) $\mathrm{PE}_\times$ fällt aus, *bevor* es seine Hormone in Zyklus i senden konnte.

   In Zyklus i werden von den verbleibenden PEs keine von $\mathrm{PE}_\times$ gesendeten Übernahmesuppressoren empfangen, aufgrund der Überlastsituation sendet in Zyklus $i + 1$ dennoch keines dieser PEs einen positiven modifizierten Eignungswert. In der Decide-Phase dieses Zyklus wird dann jedoch festgestellt, dass für die betroffenen Tasks keine positiven modifizierten Eignungswerte sowie zwei Zyklen lang keine Übernahmesuppressoren empfangen wurden, weshalb die Überlastsituation erkannt und entsprechend niedrigpriore Tasks gestoppt werden.

   Als Resultat sinkt die Last auf den PEs, weshalb diese in Zyklus $i + 2$ einen positiven modifizierten Eignungswert senden können, der zur Übernahme einer Task qualifiziert.

[1] Hierbei wird der Zyklus des Ausfalls von $\mathrm{PE}_\times$ mitgezählt, was konsistent mit der Selbstheilungsdauer in Unterlastsituationen ist.

Dies ist in folgender Grafik skizziert:



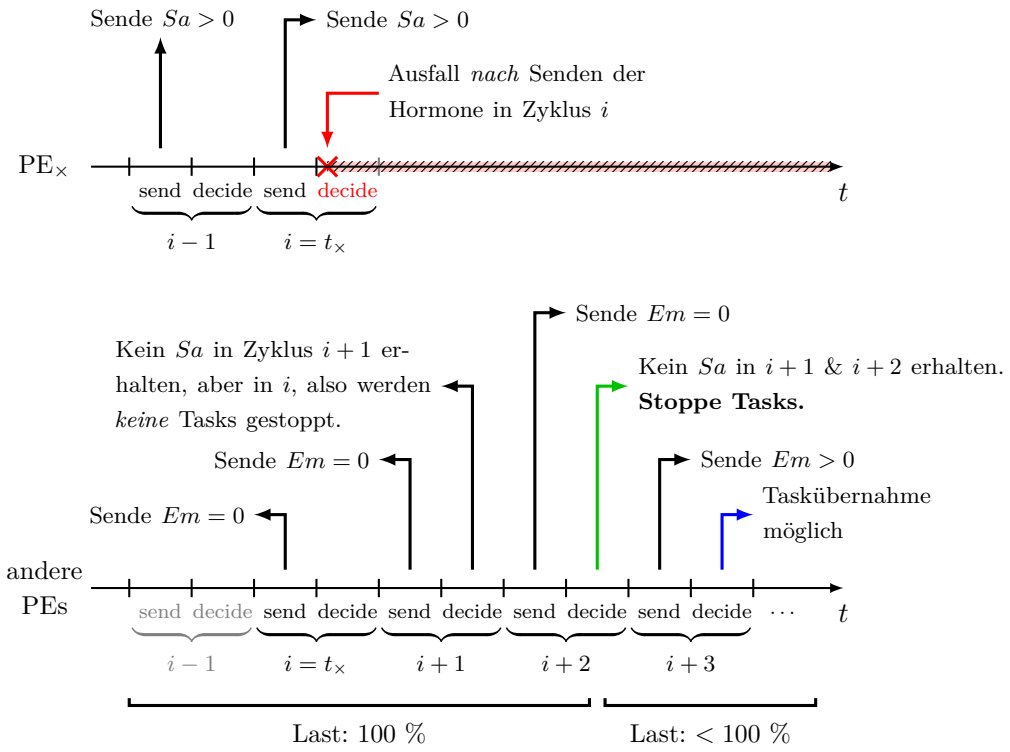
Wegen $a_{\min} = 1$ (Satz 5.11) ist die erste Taskübernahme damit gerade in Zyklus $t_\times + a_{\min} + 1$ möglich, womit der erste Teil der Aussage folgt.

b) $\text{PE}_\times$ fällt aus, *nachdem* es seine Hormone in Zyklus i gesendet hat.

In diesem Fall kann erst in Zyklus $i + 2$ ein modifizierter Eignungswert gesendet werden, der frei vom Einfluss der aus Zyklus i nachwirkenden Übernahmesuppressoren von $\text{PE}_\times$ ist; aufgrund der Überlastsituation ist dieser jedoch trotzdem Null. Da in der Decide-Phase von Zyklus $i + 2$ nun jedoch keine positiven modifizierten Eignungswerte sowie zwei Zyklen lang keine Übernahmesuppressoren empfangen wurden, wird die Überlastsituation erkannt, weshalb entsprechend niedrigpriore Tasks gestoppt werden.

In Folge sinkt die Last auf den verbleibenden PEs, weshalb diese in Zyklus $i + 3$ positive modifizierte Eignungswerte senden und Tasks übernehmen können.

Dies ist in folgender Grafik skizziert:



Wegen $a = 2$ (Satz 5.11) ist die erste Taskübernahme daher gerade in Zyklus $t_\times + a + 1$ möglich und die Aussage folgt. □

Im Vergleich zur Selbstheilung in Unterlastsituationen erfolgt die Übernahme der ersten neu zugeteilten Task folglich genau einen Zyklus später, da aufgrund der Überlastsituation in Zyklus $t_\times + a_{\min}$ bzw. $t_\times + a$ noch kein positiver modifizierter Eignungswert gesendet werden kann. Stattdessen werden in diesen Hormonzyklen Tasks gestoppt, sodass in den *darauffolgenden* Zyklen Taskübernahmen möglich sind. Es sei jedoch darauf hingewiesen, dass diese zum Stopp von Tasks genutzten Zyklen gemäß Definition 5.14 nicht als *Wartezyklen* zu klassifizieren sind, da keine ausreichenden Kapazitäten für die Übernahme der zuzuteilenden Tasks vorhanden sind.

Nach diesen Überlegungen sollen nun einige allgemeine Aussagen über die Selbstheilung in Überlastsituationen getroffen werden, die im Rahmen der späteren Analysen benötigt werden.

8.4. Allgemeine Lemmata

$T_{\max}$ **Definition 8.5.** Es bezeichne $T_{\max}$ die zum aktuellen Zeitpunkt höchstpriore nicht im System zugeteilte Task.[1]

[1] Später werden wir die Bedingung, dass alle Tasks verschiedene Prioritäten haben, in bestimmten Fällen aufheben. Falls in einem solchen Fall mehrere Tasks derselben Priorität existieren, welche die

Bemerkung 8.6. Zum Zeitpunkt $t_\times$ (und damit zu Beginn der Selbstheilung) ist $T_{\max}$ gerade die höchstpriore Task unter allen Tasks, die zuvor auf $\mathrm{PE}_\times$ zugeteilt waren.

$T_{\max}$ entspricht dabei gerade jener Task, die im Kontext von Abbildung 6.1a (S. 69) als missingTask bezeichnet wurde.

Lemma 8.7. Im Laufe der Selbstheilung nimmt die Priorität $\mathrm{prio}(T_{\max})$ der zum jeweiligen Zeitpunkt höchstprioren nicht im System zugeteilten Task $T_{\max}$ monoton ab.

Beweis. Die im System nicht zugeteilten Tasks werden von der prioritätsbasierten Task-Decision-Strategie in der Reihenfolge absteigender Prioritäten zugeteilt. Auf jedem PE können gemäß der Task-Decision-Strategie zudem stets nur Tasks gestoppt werden, die eine geringere Priorität als $T_{\max}$ haben. Würde $\mathrm{prio}(T_{\max})$ im Verlauf der Selbstheilung zunehmen, müssten jedoch Tasks gestoppt worden sein, deren Priorität größer als $\mathrm{prio}(T_{\max})$ ist.[1] □

Lemma 8.8. Eine Task T, die während der Selbstheilung auf einem PE_j zugeteilt wurde, wird im Laufe der Selbstheilung nicht mehr gestoppt.

Beweis. Da T zugeteilt wurde, war T zu diesem Zeitpunkt die höchstpriore nicht im System zugeteilte Task $T_{\max}$. Um T zu stoppen, müsste daher zu einem späteren Zeitpunkt eine nicht zugeteilte Task mit höherer Priorität als $\mathrm{prio}(T)$ existieren, was nach Lemma 8.7 nicht möglich ist. □

Es sei insbesondere explizit auf den folgenden Spezialfall des Lemmas hingewiesen:

Korollar 8.9. Eine Task T, die während der Selbstheilung von einem PE_j gestoppt wurde und auf einem PE_k erneut zugeteilt wurde (j und k müssen nicht notwendigerweise verschieden sein), wird im Laufe der Selbstheilung nicht noch einmal gestoppt.

Beweis. T wurde zugeteilt, siehe Lemma 8.8. □

Mit diesen Vorarbeiten wollen wir in den folgenden Abschnitten nun zwei konkrete Task-Dropping-Strategien vorstellen und hinsichtlich ihres Verhaltens analysieren.

8.5. Naives Task Dropping

Die erste im Rahmen dieser Dissertation betrachtete Task-Dropping-Strategie arbeitet wie folgt:

Strategie 8.10. Sobald eines der verbleibenden PEs den Ausfall von $\mathrm{PE}_\times$ bemerkt und die Situation als Überlastsituation erkannt hat, stoppt es *alle* seine Tasks, deren Priorität kleiner als $\mathrm{prio}(T_{\max})$ ist.

Naive Task-Dropping-Strategie

höchstprioren nicht im System zugeteilten Tasks darstellen, so bezeichne $T_{\max}$ eine beliebige dieser Tasks.

[1] Gemäß der Annahmen aus Abschnitt 8.2 fällt während der laufenden Selbstheilung kein weiteres PE aus.

Diese Strategie ist denkbar einfach, was ihren Namen erklärt. In der folgenden Analyse werden wir sehen, dass sie – bezogen auf die resultierenden Zeitschranken – im Allgemeinen sehr schlecht ist. Daher wird sie im Folgenden vor allem als Vergleichsnormal dienen, an dem sich das in Abschnitt 8.6 vorgestellte, komplexere *Eager-Task-Dropping* messen muss.

8.5.1. Worst-Case-Analyse

Wir möchten daher nun zunächst den Worst Case der Dauer einer Selbstheilung in Überlastsituationen bestimmen, falls die naive Task-Dropping-Strategie verwendet wird.

Satz 8.11. Die Selbstheilung in einer Überlastsituation unter Verwendung der naiven Task-Dropping-Strategie ist nach höchstens

$$mv + (mv - 1) \cdot [v > 1] + 1 + a \text{ Hormonzyklen}$$

beendet; für den Spezialfall $v > 1$ vereinfacht sich diese Schranke zu $2mv + a$ Hormonzyklen.

Beweis. Sei $T_{\max}$ die höchstpriore Task, die vor dem Ausfall von $\mathrm{PE}_\times$ auf diesem zugeteilt war.

Nach a Hormonzyklen erkennen die v verbleibenden PEs den Ausfall von $\mathrm{PE}_\times$, da insbesondere für $T_{\max}$ weder positive modifizierte Eignungswerte noch Übernahmesuppressoren empfangen werden.

Im darauffolgenden Zyklus wird die Überlastsituation erkannt, weshalb jedes PE gemäß der naiven Task-Dropping-Strategie alle seine Tasks stoppen wird, die eine geringere Priorität als $T_{\max}$ haben. Es bezeichne k die Anzahl der so gestoppten Tasks.

Beginnend mit dem nächsten Zyklus werden dann im Rahmen einer Selbstkonfiguration genau die k höchstprioren, nicht zugeteilten Tasks zugeteilt. Dies dauert nach Satz 6.10 höchstens $k + (k - 1) \cdot [v > 1]$ Hormonzyklen, da alle Taskprioritäten verschieden sind (gemäß Bemerkung 6.8 treten nur für $v > 1$ Wartezyklen auf). Am Ende dieser Zuteilung sind wieder alle PEs voll ausgelastet. Zudem wurden die k Tasks von der prioritätsbasierten Task-Decision-Strategie in der Reihenfolge absteigender Taskprioritäten zugeteilt, weshalb keine zugeteilte Task existieren kann, deren Priorität geringer als $\mathrm{prio}(T_{\max})$ ist. Folglich befindet sich das System nun in einem stabilen Zustand und die Selbstheilung ist beendet.

Im Worst Case ist $T_{\max}$ zum Zeitpunkt $t_\times$ jedoch gerade die unter allen Tasks des Systems höchstpriore Task. Daher gilt $k \leq mv$ und es folgt

$$\text{WCET} \leq mv + (mv - 1) \cdot [v > 1] + 1 + a \text{ Hormonzyklen.} \qquad \square$$

Im Worst Case muss also eine komplette Neukonfiguration des verbleibenden Teilsystems durchgeführt werden. Insbesondere kann es passieren, dass zunächst *alle* mv zugeteilten Tasks gestoppt werden und danach $T_{\max}$ sowie $mv - 1$ der soeben gestoppten Tasks erneut zugeteilt werden (und folglich diese $mv - 1$ Tasks unnötigerweise gestoppt

wurden). Ein solches Szenario ist für $v = 4$ in Abbildung 8.2 dargestellt. Hierbei steht jede Zahl i für eine Task T_i mit $\text{prio}(T_i) = i$.

8.5.1.1. Weniger Prioritäten als Tasks

Erlaubt man, dass mehrere Tasks dieselbe Priorität haben, so lässt sich die soeben hergeleitete Zeitschranke verallgemeinern:

Satz 8.12. Es bezeichne P die Menge der Taskprioritäten und a die Zeit, die höchstens benötigt wird, um den Ausfall von $\text{PE}_\times$ zu erkennen. Dann ist die Selbstheilung in einer Überlastsituation unter Verwendung der naiven Task-Dropping-Strategie nach höchstens

$$mv + (\min\{|P|, mv\} - 1) \cdot [v > 1] + 1 + a \text{ Hormonzyklen}$$

beendet; für den Spezialfall $v > 1$ ergibt sich diese Schranke zu

$$mv + \min\{|P|, mv\} + a \text{ Hormonzyklen.}$$

Beweis. Analog zu Satz 8.11, allerdings wird statt der Zeitschranke von $k+(k-1)\cdot[v > 1]$ Hormonzyklen für die Selbstkonfiguration von k Tasks die allgemeinere Zeitschranke von $k + (|P| - 1) \cdot [v > 1]$ Hormonzyklen verwendet, die sich unter Verwendung von Bemerkung 6.8 aus Satz 6.9 ergibt; an einer Selbstkonfiguration mit mv Tasks können jedoch maximal mv verschiedene Prioritäten beteiligt sein. □

In Abschnitt 8.6 wird die *Eager-Task-Dropping-Strategie* vorgestellt werden, die im Worst Case eine deutlich kürzere Selbstheilungsdauer garantieren kann; für den Vergleich der beiden Strategien ist jedoch vor allem die durch Satz 8.11 gegebene speziellere Zeitschranke von höchstens $mv + (mv - 1) \cdot [v > 1] + 1 + a$ Hormonzyklen relevant.

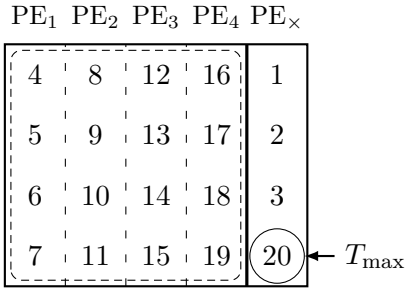
8.5.2. Average-Case-Analyse

In Bezug auf den Einsatz des AHS in Echtzeitsystemen ist eine Analyse des *Average Case* der Dauer einer Selbstheilung nicht zielführend, da im Kontext von Echtzeitanwendungen ausschließlich der Worst Case von Bedeutung ist.

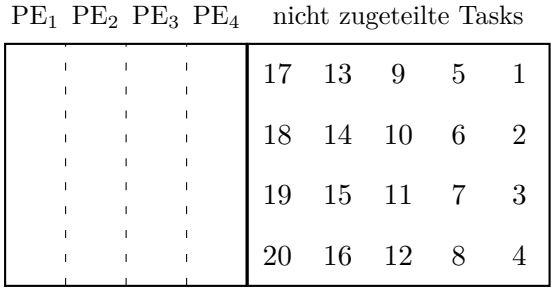
Nichtsdestotrotz wird eine solche in Anhang B durchgeführt. Dies hat zwei Gründe:

1. Eine Average-Case-Analyse erlaubt eine bessere Beurteilung des Verhaltens der naiven Task-Dropping-Strategie: Eine „gutmütige“ erwartete Selbstheilungsdauer könnte diese Strategie trotz ihrer schlechten Worst-Case-Laufzeit dennoch für den Einsatz in Szenarien qualifizieren, die keinen *harten* Zeitschranken unterliegen.

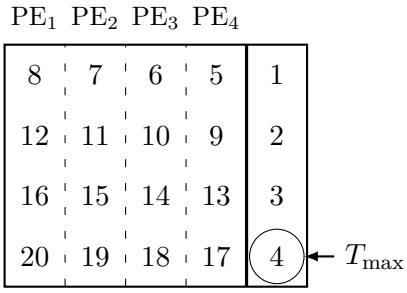
2. Ein Vergleich des *Average Case* der naiven Task-Dropping-Strategie mit dem *Worst Case* der bereits erwähnten Eager-Task-Dropping-Strategie, wie er in Abschnitt 8.7.2.1 durchgeführt werden wird, liefert interessante Resultate über die Güte der besagten Eager-Task-Dropping-Strategie.



(a) Zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$: Alle vom abgerundeten, gestrichelten Rechteck eingerahmten Tasks haben eine geringere Priorität als $T_{\max}$ und werden daher nach Erkennung der Überlastsituation gestoppt.



(b) Alle mv Tasks gestoppt nach $a + 1$ Hormonzyklen.



(c) Selbstheilung beendet nach insgesamt $2mv + a$ Hormonzyklen.

Abbildung 8.2: Worst-Case-Szenario der naiven Task-Dropping-Strategie (für $m = 4$ und $v = 4$): $T_{\max}$ sowie $mv - 1$ der insgesamt mv gestoppten Tasks werden (erneut) zugeteilt.

Um die Komplexität einer solchen Analyse beherrschbar zu halten, wird hierbei jedoch vorausgesetzt, dass alle Verteilungen der $m(v+1)$ Tasks auf die $(v+1)$ verschiedenen PEs gleich wahrscheinlich sind. Die Konsequenzen dieser fundamentalen Annahme werden in Anhang B.2 diskutiert.

8.5.2.1. Resultate

Im Folgenden wollen wir die wesentlichen Resultate dieser Average-Case-Analyse kurz wiedergeben, insofern sie für die Betrachtungen im weiteren Verlauf dieses Kapitels relevant sind. Für die Herleitungen und Beweise der folgenden Aussagen sei jeweils auf Anhang B verwiesen.

Erwartete Anzahl gestoppter Tasks Das erste Ergebnis der Average-Case-Analyse befasst sich mit der erwarteten Anzahl Tasks, die von der naiven Task-Dropping Strategie gestoppt werden:

Satz B.15. Sei X eine Zufallsvariable, welche die Anzahl der von der naiven Task-Dropping-Strategie gestoppten Tasks angibt.

Dann gilt für den Erwartungswert $\mathbf{E}[X]$ dieser Zufallsvariablen

$$\mathbf{E}[X] = \frac{m^2 v}{1+m}.$$

Betrachtet man den Quotienten

$$\begin{aligned} Q(m,v) &\coloneqq \frac{\text{Anzahl gestoppter Tasks im Worst Case}}{\text{Anzahl gestoppter Tasks im Average Case}} \\ &= \frac{mv}{m^2 v/(1+m)} = 1 + \frac{1}{m}, \end{aligned}$$

so gilt wegen $m \geq 1$ gerade $1 < Q(m,v) \leq 2$. Der Quotient ist also durch zwei Konstanten beschränkt, entsprechend haben Worst- und Average Case der Anzahl gestoppter Tasks asymptotisch die gleiche Komplexität. Mit wachsendem m strebt der Quotient gar gegen Eins, weshalb für große m in Worst- und Average Case ähnlich viele Tasks gestoppt werden.

Erwartete Dauer der Selbstheilung Ausgehend von der Anzahl der gestoppten Tasks lässt sich ebenso die erwartete Dauer der Selbstheilung in einer Überlastsituation bestimmen. Diese lässt sich im Allgemeinen jedoch nicht exakt, sondern nur innerhalb eines Korridors angeben:

Satz B.18. Es bezeichne $\mathrm{ACET}_{\mathrm{naiv}}$ den Average Case der naiven Task-Dropping-Strategie, welcher der erwarteten Dauer einer Selbstheilung in einer Überlastsituation bei Verwendung dieser Strategie entspreche.

Dann gilt

$$2\frac{m^2v}{1+m} - m + \frac{\binom{2m}{m+1} - 1}{\binom{mv+m}{m}} + 1 + a_{\min} \leq \mathrm{ACET}_{\mathrm{naiv}} \leq a + \begin{cases} \frac{m^2v}{1+m} + 1 - \frac{1}{\binom{mv+m}{m}}, & v = 1 \\ 2\frac{m^2v}{1+m}, & v > 1. \end{cases}$$

Wie im Fall der Anzahl gestoppter Tasks lässt sich auch für die Dauer der Selbstheilung zeigen, dass Worst- und Average Case asymptotisch in der gleichen Komplexitätsklasse liegen, da der Quotient

$$Q(m, v) \coloneqq \frac{\mathrm{WCET}_{\mathrm{naiv}}}{\mathrm{ACET}_{\mathrm{naiv}}}$$

durch zwei Konstanten beschränkt ist:

Satz B.21. Für den Quotienten $Q(m, v)$ aus der Dauer der Selbstheilung im Worst- und Average Case gilt $1 \leq Q(m, v) \leq 2$.

8.5.2.2. Diskussion

Aus diesen Ergebnissen, für deren Beweise erneut auf Kapitel B verwiesen sei, lässt sich schließen, dass sich Worst- und Average Case der naiven Task-Dropping-Strategie sehr ähnlich verhalten. Sowohl die Anzahl gestoppter Tasks als auch die daraus resultierende Selbstheilungsdauer liegen in Worst- und Average Case nur einen konstanten Faktor von höchstens 2 auseinander. Anhang B zeigt sogar, dass sich die Werte für große m und v jeweils einander annähern. Damit liegen Worst- und Average Case beider betrachteter Größen asymptotisch jeweils in der gleichen Komplexitätsklasse.

Diese Ergebnisse lassen sich auf zwei Arten interpretieren:

1. Pessimistisch betrachtet zerstören die Resultate die Hoffnung, dass die naive Task-Dropping-Strategie sich im Average Case deutlich gutmütiger als im Worst Case verhält. Die naive Strategie kann zwar die Einhaltung harter Zeitschranken garantieren, da die Worst-Case-Selbstheilungsdauer beschränkt ist. Da im Worst Case jedoch *alle* zugeteilten Tasks gestoppt werden, ist der praktische Nutzen dieser Garantie dennoch beschränkt. Selbst wenn keine *harten* Zeitschranken eingehalten werden müssen, sondern unter Umständen eine *best-efforts*-Selbstheilung ausreichend wäre, ist aufgrund der Resultate der Average-Case-Analyse kein signifikant besseres Verhalten zu erwarten.

2. Andererseits lässt sich das Resultat auch optimistischer betrachten: Die Worst-Case-Selbstheilungsdauer der naiven Strategie ist nur unwesentlich schlechter als ihr Average Case. Ist die erwartete Selbstheilungsdauer in einem nicht zeitkritischen Szenario akzeptabel, so sind beim sporadischen Auftreten des Worst Case keine allzu großen Ausreißer zu erwarten, was diese Strategie möglicherweise dennoch für den Einsatz in solchen Szenarien qualifizieren könnte.

Unabhängig vom konkret eingenommenen Standpunkt der Interpretation der Ergebnisse wird ein Vergleich mit der Worst-Case-Selbstheilungsdauer der im folgenden Abschnitt vorgestellten Eager-Task-Dropping-Strategie ein weiteres interessantes Resultat liefern.

8.6. Eager Task Dropping

Während gemäß der naiven Task-Dropping-Strategie 8.10 jedes PE beim Erkennen einer Überlastsituation *alle* seine Tasks stoppt, deren Priorität kleiner als jene der höchstprioren nicht im System zugeteilten Task $T_{\max}$ ist, stoppt die in diesem Abschnitt betrachtete Strategie weniger Tasks auf einmal, dafür werden unter Umständen jedoch mehrmals Tasks gestoppt:

Eager Task-Dropping-Strategie

Strategie 8.13. Sobald eines der verbleibenden PEs den Ausfall von $\mathrm{PE}_\times$ bemerkt und die Situation als Überlastsituation klassifiziert hat, stoppt es – beginnend mit der niedrigstprioren auf ihm zugeteilten Task – so viele Tasks mit geringerer Priorität als $T_{\max}$ wie nötig sind, um im nächsten Hormonzyklus einen positiven modifizierten Eignungswert für $T_{\max}$ senden zu können. Ist es jedoch nicht möglich, hinreichend viele solcher Tasks zu stoppen, so wird keine Aktion durchgeführt.[1]

Während der Name der naiven Task-Dropping-Strategie davon herrührte, dass sie eine sehr einfache Entscheidung trifft, um Überlastsituationen aufzulösen, wird die hier vorgestellte Strategie als *Eager* Task Dropping bezeichnet, um zu betonen, dass sie gerade so viele Tasks wie nötig stoppt, um im nächsten Zyklus einen positiven modifizierten Eignungswert für $T_{\max}$ senden zu können.

Im Rahmen des betrachteten Überlastmodells wurde jedoch in Abschnitt 8.2 insbesondere die Annahme getroffen, dass die Lastsuppressoren aller Tasks identisch sind. Unter dieser Voraussetzung lässt sich die Strategie wie folgt vereinfachen:

Strategie 8.14. Sobald eines der verbleibenden PEs den Ausfall von $\mathrm{PE}_\times$ bemerkt und die Situation als Überlastsituation klassifiziert hat, stoppt es die niedrigstpriore auf ihm zugeteilte Task, sofern diese eine geringere Priorität als $T_{\max}$ hat.

Im Folgenden ist der Begriff des Eager Task Dropping gleichbedeutend mit Strategie 8.14, beim Vorliegen einer Überlastsituation wird jedes PE also entweder die niedrigstpriore auf ihm zugeteilte Task stoppen oder aber keine Task.

Dies führt dazu, dass die Selbstheilung bei Verwendung der Eager-Task-Dropping-Strategie in mehreren *Selbstheilungsphasen* abläuft, die wie folgt definiert seien:

Selbstheilungsphase, Phase

Definition 8.15. Der Begriff *Selbstheilungsphase* oder *Phase* bezeichne einen Abschnitt einer Selbstheilung in einer Überlastsituation, der wie folgt charakterisiert ist:

1. Alle verbleibenden v PEs $\mathrm{PE}_1 \ldots \mathrm{PE}_v$ sind zunächst voll ausgelastet. Es werden also insbesondere für die höchstpriore nicht im System zugeteilte Task $T_{\max}$ von

[1] Dieser Fall tritt beispielsweise ein, wenn eine zu kleine Anzahl Tasks mit geringerer Priorität als $T_{\max}$ auf dem PE zugeteilt ist: In diesem Fall kann selbst der Stopp *aller* dieser Tasks den Lastsuppressor nicht in einem ausreichenden Maße senken.

keinem PE positive modifizierte Eignungswerte oder Übernahmesuppressoren gesendet. Folglich stoppt jedes verbleibende PE seine jeweils niedrigstpriore Task T_{min}, sofern $\text{prio}(T_{\text{min}}) < \text{prio}(T_{\text{max}})$. Dadurch werden insgesamt Kapazitäten für $k \leq v$ Tasks frei.

2. Im Anschluss werden insgesamt k Tasks neu zugeteilt, darunter insbesondere T_{max} (da für diese gerade k PEs jeweils eine Task gestoppt haben). Die übrigen $k-1$ zugeteilten Tasks stammen entweder aus der Menge der k gerade gestoppten Tasks oder aus der Menge der bereits zuvor nicht im System zugeteilten Tasks.

Nach einer Selbstheilungsphase liegt wieder eine Überlastsituation mit aktualisierten Werten für T_{max} und T_{min} vor. Falls die nun höchstpriore nicht zugeteilte Task T_{max} eine höhere Priorität hat als die niedrigstpriore zugeteilte Task T_{min} irgendeines PEs, so beginnt im Anschluss eine neue Phase, ansonsten ist die Selbstheilung beendet.

Abbildung 8.3 zeigt erneut das aus Abbildung 8.2 bekannte Worst-Case-Beispiel der naiven Task-Dropping-Strategie, dieses Mal kommt jedoch das Eager Task Dropping zum Einsatz. Wie zuvor steht jede Zahl i für eine Task T_i mit $\text{prio}(T_i) = i$. Die naive Strategie würde hier *alle* $mv = 16$ zugeteilten Tasks stoppen, während bei Verwendung von Eager Task Dropping nur vier Tasks gestoppt werden.

Ein weiteres Beispiel ist in Abbildung 8.4 dargestellt: Hier werden zwei Phasen benötigt, bis die Selbstheilung beendet ist und es werden insgesamt fünf Tasks gestoppt. Auch in diesem Fall hätte die naive Task-Dropping-Strategie *alle* zugeteilten Tasks gestoppt.

Die beiden Beispiele zeigen also, dass es Situationen gibt, in denen das Eager Task Dropping weniger Tasks stoppt als die naive Strategie. Bevor wir jedoch den Worst Case des Eager Task Dropping analysieren, möchten wir die Anzahl gestoppter Tasks mit der Dauer der Selbstheilung in Relation setzen.

Hierzu schätzt folgender Satz die Dauer einer einzelnen Selbstheilungsphase ab:

Satz 8.16. Es sei der Ausfall von $\text{PE}_\times$ bereits erkannt worden.

Eine Selbstheilungsphase, in der $0 \leq k \leq v$ Tasks gestoppt werden, dauert dann höchstens $2k$ Hormonzyklen.

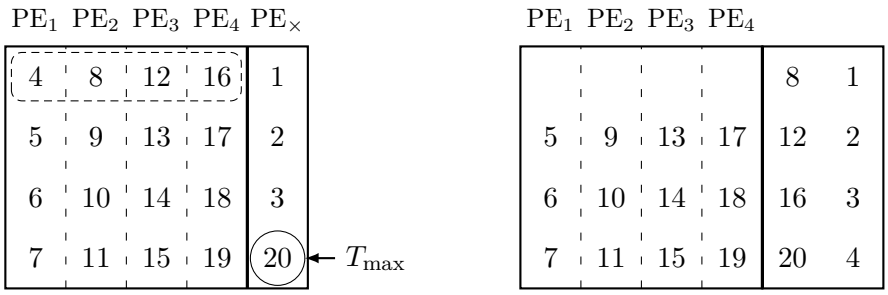
Beweis. Falls $k = 0$ Tasks gestoppt werden, ist keine Aktion nötig und die Behauptung folgt, da $2k = 0$ gilt. Daher betrachten wir im Folgenden nur den Fall $k \geq 1$.

Es sind nun zwei Fälle zu unterscheiden:

1. Die betrachtete Selbstheilungsphase ist die *erste* Phase der Selbstheilung.

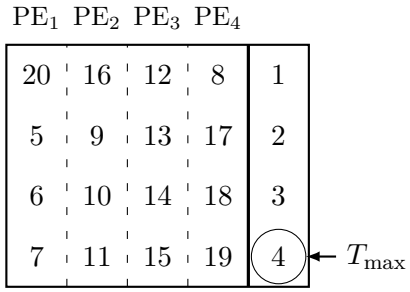
   In diesem Fall wurden für die höchstpriore nicht zugeteilte Task T_{max} bereits im letzten zur Erkennung des Ausfalls von $\text{PE}_\times$ benötigten Hormonzyklen weder Übernahmesuppressoren noch positive modifizierte Eignungswerte gesendet bzw. empfangen.

   Der folgende Hormonzyklus ist daher der zweite Hormonzyklus, in dem dies zutrifft, weshalb k PEs während der Decide-Phase jeweils eine Task (und damit



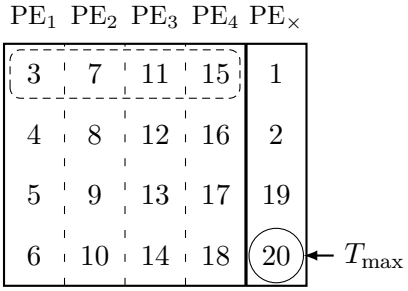
(a) Zum Zeitpunkt des Ausfalls von $PE_\times$: Die vom abgerundeten, gestrichelten Rechteck eingeschlossenen Tasks werden in Phase 1 gestoppt, um Kapazitäten für $T_{\max}$ zu schaffen.

(b) In Phase 1 werden vier Tasks gestoppt.

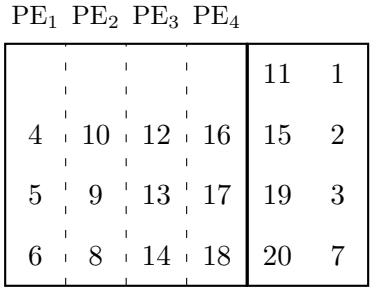


(c) Selbstheilung beendet nach Phase 1: $T_{\max}$ hat keine höhere Priorität als alle zugeteilten Tasks, daher folgt keine weitere Selbstheilungsphase.

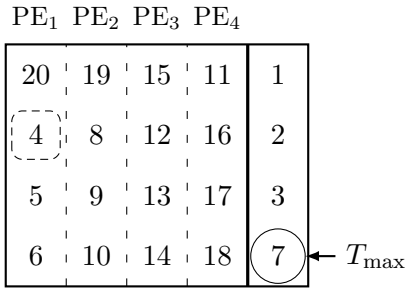
Abbildung 8.3: Beispiel einer Selbstheilung in einer Überlastsituation für das Szenario aus Abbildung 8.2, wenn stattdessen Eager Task Dropping verwendet wird.



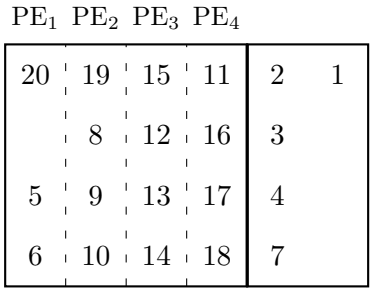
(a) Zum Zeitpunkt des Ausfalls von $PE_\times$: Die markierten Tasks werden in Phase 1 gestoppt.



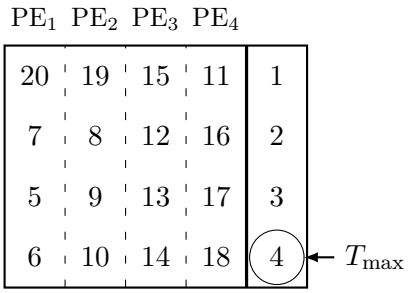
(b) In Phase 1 werden vier Tasks gestoppt.



(c) Phase 1 ist beendet, es liegt wieder eine Überlastsituation vor. In der folgenden Phase wird nur die markierte Task gestoppt, um Kapazitäten für $T_{\max}$ freizumachen, da die niedrigstprioren Tasks auf $PE_2 \ldots PE_4$ keine geringere Priorität als $T_{\max}$ aufweisen.



(d) In Phase 2 wird eine einzige Task gestoppt.



(e) Selbstheilung beendet nach Phase 2: $T_{\max}$ hat eine niedrigere Priorität als alle zugeteilten Tasks, daher folgt keine weitere Selbstheilungsphase.

Abbildung 8.4: Beispiel einer Selbstheilung in einer Überlastsituation mit zwei Phasen

insgesamt k Tasks) stoppen. Im darauffolgenden Hormonzyklus senden diese k PEs nun jeweils einen positiven modifizierten Eignungswert für $T_{\max}$ sowie alle übrigen aktuell keinem PE zugeteilten Tasks: Es folgt also eine Selbstkonfiguration der k höchstprioren Tasks, die aktuell keinem PE zugeteilt sind. Gemäß Satz 6.10 dauert eine solche Selbstkonfiguration höchstens $2k - 1$ Hormonzyklen.

In Summe ergeben sich daher $2k$ Hormonzyklen: Ein Hormonzyklus zum Stopp der k Tasks sowie $2k - 1$ Hormonzyklen zur Zuteilung.

Für die letzte so zugeteilte Task sendet insbesondere höchstens *ein* PE einen positiven modifizierten Eignungswert, da alle übrigen PEs wieder voll ausgelastet sind, weshalb kein Wartezyklus folgt.

2. Die betrachtete Selbstheilungsphase ist *nicht* die erste Phase der Selbstheilung. Dieser Fall ist ähnlich zu Fall 1:

   Wegen des Vorliegens einer Überlastsituation wurden für die höchstpriore nicht zugeteilte Task $T_{\max}$ im aktuellen Hormonzyklus keine positiven modifizierten Eignungswerte gesendet. Zudem wurden für $T_{\max}$ weder im vorherigen noch im aktuellen Hormonzyklus Übernahmesuppressoren empfangen:

   a) $T_{\max}$ war ursprünglich auf $\mathrm{PE}_\times$ zugeteilt, bevor dieses ausgefallen ist. Dann wurden seitdem keine Übernahmesuppressoren mehr für $T_{\max}$ empfangen, ansonsten hätte $T_{\max}$ zwischenzeitlich erneut zugeteilt und wieder gestoppt werden müssen, was nach Lemma 8.8 nicht möglich ist. Da die vorhergehende Selbstheilungsphase mindestens zwei Hormonzyklen umfasste (Stopp von Tasks sowie anschließende Zuteilung), kann im vorherigen Hormonzyklus kein Übernahmesuppressor für $T_{\max}$ empfangen worden sein.

   b) $T_{\max}$ war ursprünglich nicht auf $\mathrm{PE}_\times$ zugeteilt und wurde temporär gestoppt. Dann muss nach dem Stopp von $T_{\max}$ mindestens in einem weiteren Hormonzyklus eine Zuteilung einer anderen Task erfolgt sein. Folglich wurde im vorherigen Hormonzyklus ebenfalls kein Übernahmesuppressor für $T_{\max}$ empfangen.

   Folglich ist der aktuelle Hormonzyklus mindestens der zweite, in dem kein Übernahmesuppressor für $T_{\max}$ empfangen wurde, weshalb – analog zu Fall 1 – insgesamt k Tasks von k PEs gestoppt werden.

   Beginnend mit dem nächsten Hormonzyklus folgt erneut eine Selbstkonfiguration von k Tasks, die höchstens $2k - 1$ Hormonzyklen dauert. Ebenso wie in Fall 1 ist der darauffolgende Hormonzyklus kein Wartezyklus.

In Summe ergeben sich daher stets höchstens $2k$ Hormonzyklen: Ein Hormonzyklus zum Stopp von k Tasks sowie höchstens $2k - 1$ Hormonzyklen zur Zuteilung der k höchstprioren nicht zugeteilten Tasks. □

Es sei an dieser Stelle darauf hingewiesen, dass Satz 8.16 im Gegensatz zu den Zeitschranken der naiven Task-Dropping-Strategie *nicht* unterscheidet, ob $v = 1$ gilt: Für $v = 1$ gilt automatisch $0 \leq k \leq 1$, da jedes PE im betrachteten Modell höchstens

eine einzige Task pro Phase stoppt. Für $k = 0$ ergeben sich $0 = 2k$ Hormonzyklen, für $k = 1$ muss dagegen zunächst eine Task gestoppt werden, bevor im nächsten Hormonzyklus $T_{\max}$ zugeteilt werden kann, was ebenso in $2 = 2k$ Hormonzyklen resultiert.

Verallgemeinert lässt sich sogar zeigen, dass die *untere* Schranke der Dauer einer Selbstheilungsphase ebenfalls $2k$ Hormonzyklen beträgt, falls die Prioritäten aller in dieser Phase gestoppten Tasks verschieden sind:

Lemma 8.17. Eine Selbstheilungsphase, in der $0 \leq k \leq v$ Tasks gestoppt werden, dauert *mindestens* $2k$ Hormonzyklen, falls alle gestoppten Tasks verschiedene Prioritäten haben.

Beweis. Für $k = 0$ ergeben sich $0 = 2k$ Hormonzyklen, da keine Task gestoppt werden muss.

Für $k = 1$ wird genau eine Task gestoppt. Bei der anschließenden Zuteilung von $T_{\max}$ bietet zwar nur ein PE, weshalb kein Wartezyklus auftritt, der vorhergehende Zyklus des Taskstopps muss jedoch trotzdem berücksichtigt werden, weshalb insgesamt $2 = 2k$ Hormonzyklen benötigt werden.

Für $2 \leq k \leq v$ werden ebenfalls mindestens $2k$ Hormonzyklen benötigt: Im ersten Zyklus stoppen k PEs je eine Task. Die anschließende Selbstkonfiguration von k Tasks dauert jedoch mindestens $2k - 1$ Zyklen, da alle Prioritäten verschieden sind und folglich Wartezyklen nach *jeder* Taskzuteilung auftreten: Da jedes PE maximal eine Task stoppt, bieten bei jeder Taskzuteilung (außer der letzten) jeweils mindestens zwei PEs, was durch Auftreten des Almost-Winner-Zustands im darauffolgenden Zyklus jeweils zu einem Wartezyklus führt. □

Mithilfe von Satz 8.16 wollen wir nun die Dauer einer gesamten Selbstheilung abschätzen, wenn die Anzahl der gestoppten Tasks bekannt ist:

Satz 8.18. Sei k die Anzahl der Tasks, die im Rahmen einer Selbstheilung in einer Überlastsituation durch die Eager-Task-Dropping-Strategie *insgesamt* gestoppt werden.

Dann dauert die gesamte Selbstheilung höchstens $2k + a$ Hormonzyklen.

Beweis. Spätestens nach a Hormonzyklen wird der Ausfall von $\mathrm{PE}_\times$ erkannt. Für $k = 0$ folgt die Behauptung, da die Selbstheilung in einer Überlastsituation gemäß Definition 8.2 mit dem Erkennen des Ausfalls beendet ist, weshalb wir im Folgenden den Fall $k \geq 1$ betrachten.

Sei daher k_i die Anzahl Tasks, die in der i-ten Selbstheilungsphase gestoppt werden; entsprechend gilt dann $k = \sum_i k_i$.

Jede Selbstheilungsphase dauert nach Satz 8.16 höchstens $2k_i$ Hormonzyklen, weshalb die gesamte Selbstheilung insgesamt höchstens $a + \sum_i 2k_i = 2k + a$ Hormonzyklen dauert.[1] □

[1] Der Beweis von Satz 8.16 begründet zudem, warum zwischen den einzelnen Selbstheilungsphasen keine Wartezyklen betrachtet werden müssen.

Bemerkung 8.19

Haben alle gestoppten Tasks verschiedene Prioritäten, so dauert die gesamte Selbstheilung *nach* der Erkennung des Ausfalls von $\mathrm{PE}_\times$ sogar *mindestens* (und damit *genau*) $2k$ Hormonzyklen, da jede Phase, in der k_i Tasks gestoppt werden, nach Lemma 8.17 mindestens $2k_i$ Hormonzyklen dauert. Lediglich die zur Erkennung des Ausfalls benötigte Zeit kann hierbei nicht genauer abgeschätzt werden, da diese vom Ausfallzeitpunkt von $\mathrm{PE}_\times$ innerhalb des Hormonzyklus abhängt, wie in Kapitel 5.3 erläutert wurde.

Auch Satz 8.18 benötigt also, anders als die Abschätzungen der naiven Task-Dropping-Strategie, keinen gesonderte Betrachtung des Falls $v = 1$. Um die Worst-Case-Dauer der gesamten Selbstheilung zu bestimmen, muss daher lediglich abgeschätzt werden, wie viele Tasks vom Eager Task Dropping höchstens gestoppt werden können.

8.6.1. Definitionen

Bevor die Anzahl der gestoppten Tasks abgeschätzt werden kann, ist es jedoch zunächst nötig, einige Begriffe zu definieren.

Definition 8.20. Ein PE heiße

kalt, wenn es sich nicht mehr an der Selbstheilung beteiligt (bzw. beteiligen kann), weil die Priorität seiner niedrigstprioren Task größer oder gleich der Priorität der höchstprioren nicht zugeteilten Task $T_{\max}$ ist, **kalt**

heiß, wenn es sich noch an der Selbstheilung beteiligen kann, **heiß**

potenziell heiß, wenn nach den zum aktuellen Zeitpunkt vorliegenden Informationen unbekannt ist, ob es heiß oder kalt ist. **potentiell heiß**

Ferner werde der Übergang eines PEs von heiß nach kalt als *abkühlen* oder *erkalten* bezeichnet. **abkühlen, erkalten**

Beispiel 8.21

Zu Beginn des Szenarios aus Abbildung 8.4 sind alle PEs heiß. Nach Phase 1 kühlen $\mathrm{PE}_2 \ldots \mathrm{PE}_4$ ab, während PE_1 noch heiß ist und erst in Phase 2 abkühlt. Am Ende dieser Phase sind alle PEs kalt und die Selbstheilung ist daher beendet.

Definition 8.22. Wie zuvor bezeichne m die Anzahl der Tasks, die zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$ auf diesem zugeteilt waren; gemäß der Annahmen aus Abschnitt 8.2 stimmt m zudem mit der Anzahl Tasks überein, die jedes der v verbleibenden PEs höchstens ausführen kann. **global niedrigstprior**

Dann werden die m Tasks mit den kleinsten Prioritäten unter allen $m \cdot (v+1)$ Tasks des Systems als *global niedrigstpriore* Tasks bezeichnet.

Zudem heiße eine Task T ebenfalls *global niedrigstprior*, wenn sie zu den m global niedrigstprioren Tasks gehört.

Bemerkung 8.23. Zu beachten ist, dass eine Task, die global niedrigstprior ist, zum Zeitpunkt $t_\times$ auf $\mathrm{PE}_\times$ oder einem der verbleibenden v PEs $\mathrm{PE}_1 \ldots \mathrm{PE}_v$ zugeteilt gewesen sein kann. Die global niedrigstprioren Tasks sind also nicht auf ein PE beschränkt, sondern allein aufgrund ihrer Priorität als global niedrigstprior zu klassifizieren.

Beispiel 8.24

Im Szenario aus Abbildung 8.4 sind die Tasks T_1, T_2, T_3, T_4 global niedrigstprior. Die ersten beiden dieser Tasks waren zum Zeitpunkt $t_\times$ auf $\mathrm{PE}_\times$ zugeteilt, während T_3 und T_4 auf PE_1 zugeteilt waren.

$\mathcal{T}_{\mathrm{kalt}}$

Definition 8.25. Es bezeichne $\mathcal{T}_{\mathrm{kalt}}$ die Menge aller zum aktuellen Zeitpunkt nicht im System zugeteilten Tasks, die entweder auf mittlerweile erkalteten PEs oder ursprünglich auf $\mathrm{PE}_\times$ zugeteilt waren (und somit zum Beginn der Selbstheilung ausgefallen sind; $\mathrm{PE}_\times$ kann in diesem Sinne also als kaltes PE angesehen werden).

Beispiel 8.26

Wir betrachten erneut das aus Abbildung 8.4 bekannte Szenario:

- Zum Zeitpunkt $t_\times$ (Abbildung 8.4a) gilt $\mathcal{T}_{\mathrm{kalt}} = \{T_1, T_2, T_{19}, T_{20}\}$, da die angegebenen Tasks aktuell nicht zugeteilt sind, zuvor jedoch auf $\mathrm{PE}_\times$ zugeteilt waren.
- Nach dem Stopp von Tasks in Phase 1 (Abbildung 8.4b) gilt weiterhin $\mathcal{T}_{\mathrm{kalt}} = \{T_1, T_2, T_{19}, T_{20}\}$: Die PEs, auf denen die Tasks T_3, T_7, T_{11} und T_{15} ursprünglich zugeteilt waren, sind noch heiß.
- Nach Phase 1 (Abbildung 8.4c) gilt $\mathcal{T}_{\mathrm{kalt}} = \{T_1, T_2, T_7\}$: Alle diese Tasks sind aktuell nicht zugeteilt. T_1 und T_2 waren ursprünglich auf $\mathrm{PE}_\times$ zugeteilt, während T_7 auf dem mittlerweile erkalteten PE_2 zugeteilt war. Die derzeit ebenfalls keinem PE zugeteilte Task T_3 gehört dagegen nicht zu $\mathcal{T}_{\mathrm{kalt}}$, da sie ursprünglich auf PE_1 zugeteilt war, das aktuell noch heiß ist.
- Nach dem Stopp von Tasks in Phase 2 (Abbildung 8.4d) gilt weiterhin $\mathcal{T}_{\mathrm{kalt}} = \{T_1, T_2, T_7\}$: T_3 und T_4 waren beide ursprünglich auf dem noch heißen PE_1 zugeteilt und gehören daher nicht zu $\mathcal{T}_{\mathrm{kalt}}$.
- Nach Phase 2 (Abbildung 8.4e) gilt $\mathcal{T}_{\mathrm{kalt}} = \{T_1, T_2, T_3, T_4\}$: Da PE_1 mittlerweile erkaltet ist, gelten nun $T_3 \in \mathcal{T}_{\mathrm{kalt}}$ und $T_4 \in \mathcal{T}_{\mathrm{kalt}}$.

8.6.2. Lemmata

In Vorbereitung auf die Worst-Case-Analyse des Eager Task Dropping werden nun einige Lemmata betrachtet:

Lemma 8.27. In einer Überlastsituation gilt:

1. Die Selbstheilung ist beendet, wenn alle m global niedrigstprioren Tasks gestoppt und alle PEs voll ausgelastet sind.

2. In jeder Selbstheilungsphase wird mindestens eine der m global niedrigstprioren Tasks gestoppt.

3. Eine nicht zugeteilte Task T, die global niedrigstprior ist, kann nicht auf einem PE zugeteilt werden.

4. Es gibt höchstens m Selbstheilungsphasen.

Beweis.

1. Wenn alle m global niedrigstprioren Tasks gestoppt und alle PEs voll ausgelastet sind, kann keine weitere Task zugeteilt werden. Zudem ist die höchstpriore nicht zugeteilte Task $T_{\max}$ dann gerade die höchstpriore der global niedrigstprioren Tasks. Dann müssen aber die Prioritäten aller zugeteilten Tasks größer sein als die Priorität von $T_{\max}$, weshalb keine Task mehr gestoppt werden kann.

2. Zu Beginn einer Selbstheilungsphase stoppt jedes PE seine niedrigstpriore Task, sofern diese eine geringere Priorität hat als die höchstpriore nicht zugeteilte Task $T_{\max}$. Hierbei sind zwei Fälle zu unterscheiden:

   a) $T_{\max}$ ist global niedrigstprior. Dies bedeutet, dass alle m global niedrigstprioren Tasks gestoppt sind;[1] in diesem Fall ist die Selbstheilung gemäß Aussage 1 beendet und die vorangegangene Selbstheilungsphase war die letzte Phase.

   b) $T_{\max}$ ist nicht global niedrigstprior, also sind *nicht* alle m global niedrigstprioren Tasks gestoppt. Folglich muss mindestens eine Task T dieser global niedrigstprioren Tasks auf einem PE_j zugeteilt sein und auf diesem die niedrigstpriore Task darstellen (da T global niedrigstprior ist). Folglich wird T gestoppt.

3. Beweis durch Widerspruch: Angenommen, es liegt eine Überlastsituation vor und T wird dennoch zugeteilt. Dann muss T zum Zeitpunkt der Zuteilung die höchstpriore nicht zugeteilte Task gewesen sein und es müssen auf mindestens einem PE_j Kapazitäten für eine weitere Task verfügbar gewesen sein. Da jedoch eine Überlastsituation vorliegt, müssen dann zu diesem Zeitpunkt mindestens $m+1$ Tasks nicht zugeteilt gewesen sein (nämlich die generell m nicht zugeteilten Tasks sowie mindestens eine weitere Task, die gestoppt wurde, um Kapazitäten für die Zuteilung von T zu schaffen), weshalb T als eine der m global niedrigstprioren Tasks nicht die höchstpriore nicht zugeteilte Task gewesen sein kann.

[1] Ansonsten müsste es eine andere Task T geben, die eine höhere Priorität hat und ebenfalls keinem PE zugeteilt ist, weshalb $T_{\max}$ nicht global niedrigstprior sein könnte.

4. Folgt direkt aus den Aussagen 1 und 2.

   Insbesondere gilt: Sind zu Beginn der Selbstheilung alle m global niedrigstprioren Tasks auf demselben $\mathrm{PE}_j \neq \mathrm{PE}_\times$ zugeteilt, so dauert die Selbstheilung genau m Phasen, da jedes PE pro Selbstheilungsphase höchstens eine Task stoppen kann und alle m global niedrigstprioren Tasks gestoppt werden müssen. □

Der Beweis der Worst-Case-Zeitschranke wird später insbesondere die Anzahl heißer PEs betrachten. Daher beschäftigen sich die beiden folgenden Lemmata mit der Frage, wann PEs zwingend erkalten müssen:

Lemma 8.28. Im Laufe der Selbstheilung kühlt jeweils mindestens ein PE ab, wenn

a) Eine Task T, die von einem aktuell noch heißen PE_j gestoppt wurde, auf einem PE_k erneut zugeteilt wird; j und k müssen hierbei nicht notwendigerweise verschieden sein.

b) In einer Phase alle Tasks aus $\mathcal{T}_{\text{kalt}}$ zugeteilt wurden, also $\mathcal{T}_{\text{kalt}} = \emptyset$ erreicht wird.

Beweis.

a) Da T gestoppt wurde, war T zum Zeitpunkt ihres Stopps die niedrigstpriore Task auf PE_j. Da sie erneut zugeteilt wurde, existierte zu diesem Zeitpunkt keine nicht zugeteilte Task T' mit $\mathrm{prio}(T') > \mathrm{prio}(T)$. Folglich kann T gemäß Lemma 8.8 nicht erneut gestoppt werden. Da alle übrigen Tasks auf PE_j jedoch eine Priorität $p \geq \mathrm{prio}(T)$ haben müssen, kann von PE_j in der nächsten Phase keine Task mehr gestoppt werden. Somit kann sich PE_j an keiner der folgenden Phasen (falls es solche gibt) mehr beteiligen, weshalb es durch die erneute Zuteilung von T erkaltet ist.[1]

b) Nach Erreichen von $\mathcal{T}_{\text{kalt}} = \emptyset$ muss die höchstpriore nicht zugeteilte Task $T_{\max}$ von einem PE_j stammen, das zuvor noch heiß war (ansonsten würde $\mathcal{T}_{\text{kalt}} \neq \emptyset$ gelten). Da $T_{\max}$ zum Zeitpunkt ihres Stopps jedoch die niedrigstpriore Task auf PE_j war, haben alle übrigen auf PE_j zugeteilten Tasks eine mindestens genauso hohe Priorität wie $T_{\max}$ und können daher nicht gestoppt werden: PE_j wird sich folglich an der folgenden Phase (sowie allen weiteren Phasen) nicht beteiligen und erkaltet daher. □

Bemerkung 8.29. Es sei darauf hingewiesen, dass der Zustand $\mathcal{T}_{\text{kalt}} = \emptyset$ stets nur übergangsweise eingenommen wird: Durch Zuteilung aller Tasks aus $\mathcal{T}_{\text{kalt}}$ kühlt nach Lemma 8.28 ein PE_j ab, weshalb $\mathcal{T}_{\text{kalt}}$ um alle Tasks erweitert wird, die zuvor auf PE_j zugeteilt waren.

Lemma 8.30. Sind während der Selbstheilung noch n PEs heiß, so kühlt nach höchstens $\lceil |\mathcal{T}_{\text{kalt}}|/n \rceil$ Phasen eines der heißen PEs ab.

[1] Hat die anstelle von T auf PE_j zugeteilte Task T'' eine geringere Priorität als alle übrigen Tasks auf PE_j, so gilt die Aussage dennoch: Gemäß Lemma 8.8 kann auch T'' nicht mehr gestoppt werden.

Beweis. Die Zuteilung von Tasks aus $\mathcal{T}_{\text{kalt}}$ kühlt im Worst Case kein PE ab,[1] solange nach der Zuteilung $\mathcal{T}_{\text{kalt}} \neq \emptyset$ gilt: Erst nach Zuteilung der *letzten* Task aus $\mathcal{T}_{\text{kalt}}$ kühlt zwangsläufig ein PE ab. Im Worst Case werden daher immer nur Tasks aus $\mathcal{T}_{\text{kalt}}$ zugeteilt (und *keine* Tasks, die von aktuell noch heißen PEs gestoppt wurden und deshalb nicht zu $\mathcal{T}_{\text{kalt}}$ gehören). Hierdurch werden $\mathcal{T}_{\text{kalt}}$ jedoch so lange keine zusätzlichen Tasks hinzugefügt, bis eines der n heißen PEs abkühlt und alle seine bisher gestoppten und noch nicht erneut zugeteilten Tasks zu $\mathcal{T}_{\text{kalt}}$ hinzugefügt werden.

Da alle der n heißen PEs bis zu ihrem Erkalten an der Selbstheilung beteiligt sind und dabei pro Phase jeweils eine Task stoppen, um eine höherpriore Task übernehmen zu können, dauert es folglich höchstens $\lceil |\mathcal{T}_{\text{kalt}}|/n \rceil$ Phasen, bis $\mathcal{T}_{\text{kalt}} = \emptyset$ gilt und gemäß Lemma 8.28 ein PE abkühlt. □

8.6.3. Worst-Case-Analyse

Bevor mit der Analyse des Eager Task Dropping begonnen wird, soll an dieser Stelle noch das Verständnis von $\mathcal{T}_{\text{kalt}}$ vertieft werden.

Zu Beginn der Selbstheilung besteht $\mathcal{T}_{\text{kalt}}$ gerade aus allen m Tasks, die auf $\text{PE}_\times$ zugeteilt waren. Die Zuweisung von Tasks aus dieser Menge kühlt per se nicht notwendigerweise ein PE ab. Vielmehr erkaltet gemäß Lemma 8.28 mindestens ein PE, wenn

a) eine Task T, die irgendwann zuvor von einem aktuell noch heißen PE_j gestoppt wurde, erneut zugeteilt wird oder

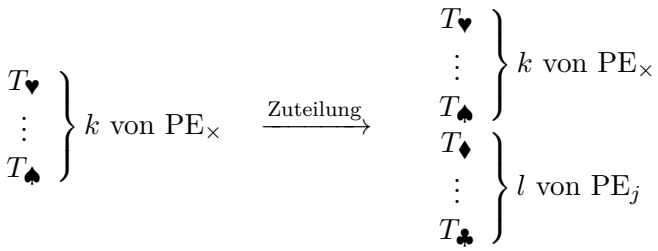
b) *alle* gegenwärtig in $\mathcal{T}_{\text{kalt}}$ enthaltenen Tasks zugeteilt werden, am Ende der Phase also $\mathcal{T}_{\text{kalt}} = \emptyset$ gilt.

Solange nur Tasks aus $\mathcal{T}_{\text{kalt}}$ zugewiesen werden, nimmt $|\mathcal{T}_{\text{kalt}}|$ mit jeder Zuteilung ab. $\mathcal{T}_{\text{kalt}}$ werden jedoch mindestens dann neue Tasks hinzugefügt, wenn

- Gemäß Fall a) eine Task T eines heißen PE_j erneut zugeteilt wird, welches dadurch abkühlt: In diesem Fall werden $\mathcal{T}_{\text{kalt}}$ *alle* bisher von PE_j gestoppten und noch nicht wieder zugeteilten Tasks hinzugefügt (das sind aber, da PE_j heiß war, genau alle bisher von PE_j gestoppten Tasks *außer* T), wie folgendes Beispiel illustriert. Um hierbei nicht fälschlicherweise den Eindruck einer impliziten Ordnungsrelation über den dargestellten Tasks zu erwecken, verwenden dieses sowie das folgende Beispiel keine Zahlen oder Buchstaben, sondern verschiedene Symbole als Taskindizes:

  Beispiel 8.31

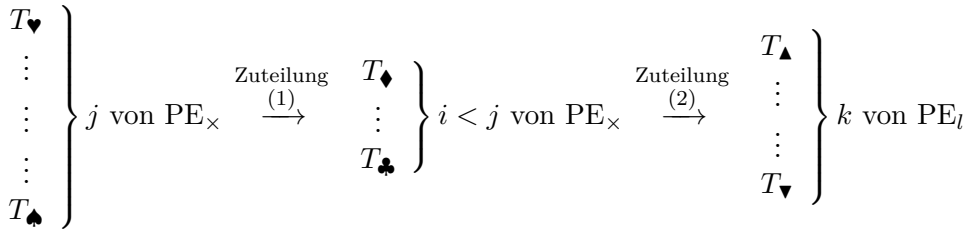
[1] Ein PE kann natürlich abkühlen, wenn nach Zuteilung einer Task aus $\mathcal{T}_{\text{kalt}}$ seine niedrigstpriore Task eine mindestens so hohe Priorität hat wie die höchstpriore nicht zugeteilte Task. Für die Abschätzung des Worst Case muss jedoch angenommen werden, dass dieser Fall *nicht* eintritt.



In der Zuteilung, die zwischen den beiden dargestellten Momentaufnahmen von $\mathcal{T}_{\text{kalt}}$ stattfindet, wird keine Task aus $\mathcal{T}_{\text{kalt}}$ zugeteilt, jedoch mindestens eine zuvor vom heißen PE_j gestoppte Task $T_\bigstar$, wodurch PE_j erkaltet und alle l zuvor von diesem gestoppten und noch nicht wieder zugeteilten Tasks (also alle bisher auf PE_j gestoppten Tasks *außer* $T_\bigstar$) zu $\mathcal{T}_{\text{kalt}}$ hinzugefügt werden.

- Gemäß Fall b) *alle* Tasks aus $\mathcal{T}_{\text{kalt}}$ zugeteilt wurden: In diesem Fall kühlt ein PE_l ab; $\mathcal{T}_{\text{kalt}}$ werden dann *alle* bisher von PE_l gestoppten und noch nicht erneut zugeteilten Tasks hinzugefügt, wie in folgendem Beispiel gezeigt:

Beispiel 8.32



In Zuteilung (1) werden $j - i$ Tasks aus $\mathcal{T}_{\text{kalt}}$ zugeteilt, jedoch keine anderen Tasks, weshalb $|\mathcal{T}_{\text{kalt}}|$ um ebenso viele Tasks abnimmt. In Zuteilung (2) werden die verbleibenden i Tasks aus $\mathcal{T}_{\text{kalt}}$ zugeteilt (jedoch keine Tasks heißer PEs), weshalb (nur) PE_l abkühlt und alle seine k gestoppten, aber noch nicht erneut zugeteilten Tasks zu $\mathcal{T}_{\text{kalt}}$ hinzugefügt werden.

Mit diesen Überlegungen kann nun der folgende zentrale Satz bewiesen werden:

Satz 8.33. Es bezeichne PE_i mit $1 \leq i \leq v$ das PE, das als i-tes abkühlt. Dann ist PE_i an höchstens $\lceil m/(v-i+1) \rceil$ Selbstheilungsphasen beteiligt und stoppt daher im Laufe der Selbstheilung höchstens $\lceil m/(v-i+1) \rceil$ Tasks.

Beweis. Gemäß Lemma 8.27 umfasst eine Selbstheilung höchstens m Phasen. Für $v = 1$ ist daher leicht einzusehen, dass PE_1 als einziges PE an höchstens $m = \lceil m/v \rceil$ Phasen beteiligt ist.

Für $v > 1$ wird die Behauptung durch Induktion über $i \leq v$ für ein beliebiges, aber festes $m \geq 1$ gezeigt. Hierbei bezeichne k_i die Anzahl der Selbstheilungsphasen, an denen PE_i bis zu seinem Abkühlen beteiligt ist.

Induktionsanfang für $i = 1$. Zu zeigen ist, dass PE_1 nach höchstens

$$\lceil m/(v-1+1) \rceil = \lceil m/v \rceil \text{ Phasen abkühlt.}$$

Beweis. Zu Beginn der Selbstheilung umfasst $\mathcal{T}_{\text{kalt}}$ gerade jene m Tasks, die zuvor auf $\mathrm{PE}_\times$ zugeteilt waren, es gilt daher $|\mathcal{T}_{\text{kalt}}| = m$. Gemäß Lemma 8.30 kühlt PE_1 daher spätestens nach $k_1 \leq \lceil |\mathcal{T}_{\text{kalt}}|/v \rceil = \lceil m/v \rceil$ Phasen ab, da alle v PEs zu Beginn heiß sind.

Da PE_1 per definitionem das erste abkühlende PE ist, sind bei dessen Abkühlen *alle* PEs an k_1 Phasen beteiligt gewesen, das sind jedoch höchstens die besagten $\lceil m/v \rceil$ Phasen. Die übrigen $\mathrm{PE}_2 \ldots \mathrm{PE}_v$ sind nach dem Abkühlen von PE_1 potenziell noch heiß, da zu diesem Zeitpunkt noch keine Aussage über sie getroffen werden kann. ◊

Induktionsschritt $i \to i+1$.

Induktionsannahme. Für ein i mit $1 \leq i < v$ gelte:

1. PE_j mit $j \in \{1, \ldots, i\}$ war an höchstens $\lceil m/(v-j+1) \rceil$ Phasen beteiligt und ist kalt.
2. PE_j mit $j \in \{i+1, \ldots, v\}$ war bisher an $k_i \leq \lceil m/(v-i+1) \rceil$ Phasen beteiligt und ist potenziell noch heiß.

Induktionsbehauptung. Wenn die Induktionsannahme gilt, so gilt auch:

1. PE_j mit $j \in \{1, \ldots, i+1\}$ war an höchstens $\lceil m/(v-j+1) \rceil$ Phasen beteiligt und ist kalt.
2. PE_j mit $j \in \{i+2, \ldots, v\}$ war bisher an $k_{i+1} \leq \lceil m/(v-(i+1)+1) \rceil = \lceil m/(v-i) \rceil$ Phasen beteiligt und ist potenziell noch heiß.

Beweis. Es sind zwei Fälle zu unterscheiden:

a) $i + 1 = v$ bzw. $i = v - 1$, PE_v ist also das letzte potenziell heiße PE. Dieses war gemäß Induktionsannahme bisher an $k_i = k_{v-1} \leq \lceil m/(v-(v-1)+1) \rceil = \lceil m/2 \rceil$ Phasen beteiligt.

Folglich sind $l \geq \lceil m/2 \rceil$ der global niedrigstprioren Tasks gestoppt. Diese l Tasks können nicht mehr zugeteilt werden, weshalb von den m nicht zugeteilten Tasks höchstens noch $m - l$ Tasks zugeteilt werden können. Da nur noch höchstens ein PE potenziell heiß ist, dauert dies höchstens $m - l$ Phasen. Folglich ist PE_v an

$$k_v \leq k_{v-1} + m - l \leq \lceil m/2 \rceil + (m - \lceil m/2 \rceil) = m = \lceil m/(v-i) \rceil$$

Phasen beteiligt, bevor es erkaltet und die Induktionsbehauptung folgt.

b) $i+1 < v$, es sind also noch $v - i \geq 2$ PEs potenziell heiß; jedes dieser PEs war gemäß der Induktionsannahme bisher an $k_i \leq \lceil m/(v-i+1) \rceil$ Phasen beteiligt und hat folglich k_i Tasks gestoppt.

Am Ende der Phase, in der PE_i erkaltete, sind für das gemäß Induktionsannahme potenziell noch heiße PE_{i+1} nun zwei Fälle zu unterscheiden:

i) PE_{i+1} ist auch tatsächlich noch heiß (also sind alle $v - i$ potenziell heißen PEs tatsächlich noch heiß: PE_{i+1} ist per definitionem das als nächstes erkaltende PE).

Jedes der heißen PEs war gemäß Induktionsannahme an k_i Phasen beteiligt und hat ebenso viele Tasks gestoppt, von denen jedoch noch keine Task erneut zugeteilt wurde (ansonsten wäre das PE gemäß Lemma 8.28 bereits erkaltet). Dies erlaubt es nun, Rückschlüsse auf die Kardinalität von $\mathcal{T}_{\text{kalt}}$ zu ziehen: Es existieren $v - i$ heiße PEs, die jeweils k_i Tasks gestoppt haben. Von diesen Tasks wurde jedoch noch keine Task erneut zugeteilt (wiederum wäre ansonsten das zugehörige PE gemäß Lemma 8.28 bereits erkaltet). Folglich existieren mindestens $(v-i) \cdot k_i$ nicht zugeteilte Tasks, die *nicht* zu $\mathcal{T}_{\text{kalt}}$ gehören. Da im betrachteten Modell zudem zwischen je zwei Phasen jeweils genau m Tasks keinem PE zugeteilt sind, gilt $|\mathcal{T}_{\text{kalt}}| \leq m - (v-i) \cdot k_i$. Die obere Schranke dieses Terms ist im folgenden Schema veranschaulicht, das alle m aktuell keinem PE zugeteilten Tasks zeigt:

T_1 } $|\mathcal{T}_{\text{kalt}}| = m - (v-i) \cdot k_i$ Tasks von kalten PEs
$\vdots$ }
$\vdots$ } k_i Tasks
$\vdots$ } $(v-i) \cdot k_i$ Tasks von heißen PEs
$\vdots$ } k_i Tasks
T_m }

PE_{i+1} kühlt *spätestens* dann ab, wenn eine seiner gestoppten Tasks erneut zugewiesen wird oder aber alle Tasks aus $\mathcal{T}_{\text{kalt}}$ vergeben wurden. Es sind noch $(v-i)$ PEs heiß, also kühlt PE_{i+1} gemäß Lemma 8.30 nach höchstens $\lceil |\mathcal{T}_{\text{kalt}}|/(v-i) \rceil$ weiteren Phasen ab.

Da PE_{i+1} gemäß Induktionsannahme bisher an $k_i \leq \lceil m/(v-i+1) \rceil$ Phasen beteiligt war, folgt

$$k_{i+1} \leq k_i + \left\lceil \frac{|\mathcal{T}_{\text{kalt}}|}{v-i} \right\rceil$$

$$
\begin{aligned}
&\leq k_i + \left\lceil \frac{m - (v-i) \cdot k_i}{v-i} \right\rceil \\
&= \left\lceil \frac{k_i \cdot (v-i)}{v-i} + \frac{m - (v-i) \cdot k_i}{v-i} \right\rceil \\
&= \left\lceil \frac{m}{v-i} \right\rceil,
\end{aligned}
$$

PE_{i+1} kühlt also nach insgesamt $k_{i+1} \leq \lceil m/(v-i) \rceil$ Phasen ab. Zusammen mit der Induktionsannahme folgt, dass PE_j für $j \in \{1, \ldots, i+1\}$ an höchstens $\lceil m/(v-j+1) \rceil$ Phasen beteiligt war und kalt ist, was dem ersten Teil der Induktionsbehauptung entspricht.

Da PE_{i+1} per definitionem das erste nach PE_i abkühlende PE ist, war PE_j für $j \in \{i+2, \ldots, v\}$ bisher an $k_{i+1} \leq \lceil m/(v-i) \rceil$ Phasen beteiligt und ist potenziell noch heiß, was dem zweiten Teil der Induktionsbehauptung entspricht.

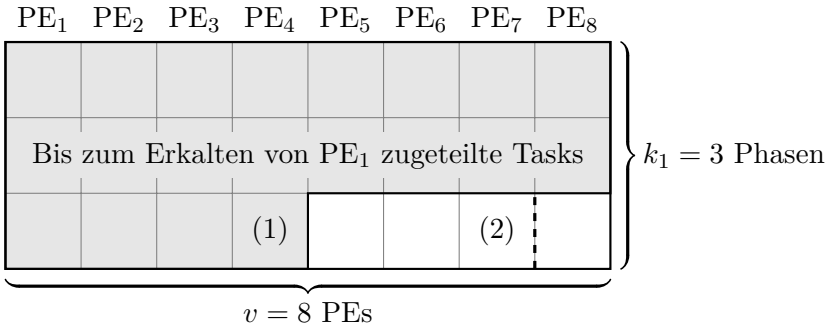
ii) Das potenziell heiße PE_{i+1} ist tatsächlich bereits erkaltet.

Dass dieser Fall tatsächlich eintreten kann, wird vom folgenden Beispiel verdeutlicht:

Beispiel 8.34

Es gelten $m = 20$ und $v = 8$. Somit ist PE_1 an höchstens $k_1 = \lceil m/8 \rceil = 3$ Phasen beteiligt, bis es abkühlt. Hierzu nehmen wir an, dass von den acht verbleibenden PEs in den bisherigen $k_1 = 3$ Phasen die maximale Zahl von $k_1 \cdot v = 24$ Tasks gestoppt wurde, weil von jedem dieser PEs in jeder Phase je eine Task gestoppt wurde.

Zu Beginn der Selbstheilung umfasst $\mathcal{T}_{\text{kalt}}$ gerade alle $m = 20$ Tasks, die vor dem Ausfall von $\mathrm{PE}_\times$ auf diesem zugeteilt waren. Wir nehmen nun an, dass diese $|\mathcal{T}_{\text{kalt}}| = m = 20$ Tasks in jeder Phase jeweils gleichmäßig von links nach rechts den verbleibenden PEs zugeteilt werden:



Die mit (1) markierte Task ist entsprechend gerade die letzte Task aus $\mathcal{T}_{\text{kalt}}$, die zugeteilt wird, entsprechend führt ihre Zuteilung nach Lem-

ma 8.28 zum Erkalten von PE_1. Folglich werden $\mathcal{T}_{\mathrm{kalt}}$ im Anschluss an diese Zuteilung alle bisher von PE_1 gestoppten und noch nicht erneut zugeteilten Tasks hinzugefügt. Da bisher jedoch nur Tasks aus $\mathcal{T}_{\mathrm{kalt}}$ zugeteilt wurden (und somit keine von PE_1 gestoppten Tasks), sind dies gerade *alle* $k_i = 3$ bisher von PE_1 gestoppten Tasks, daher gilt nun $|\mathcal{T}_{\mathrm{kalt}}| = k_1 = 3$.

Da wir annahmen, dass alle $v = 8$ PEs in jeder der betrachteten Phasen jeweils eine Task stoppten, müssen $\mathrm{PE}_5 \ldots \mathrm{PE}_8$ im Folgenden je eine weitere Task übernehmen. Werden hierzu auf $\mathrm{PE}_5 \ldots \mathrm{PE}_7$ gerade die drei in $\mathcal{T}_{\mathrm{kalt}}$ enthaltenen Tasks zugeteilt, so führt die letzte, mit (2) markierte Zuteilung einer solchen Task zum Erkalten von PE_2, da erneut $\mathcal{T}_{\mathrm{kalt}} = \emptyset$ erreicht wird. Entsprechend werden $\mathcal{T}_{\mathrm{kalt}}$ im Anschluss die $k_2 = k_1 = 3$ bisher von PE_2 gestoppten Tasks hinzugefügt.

Folglich kühlen im betrachteten Beispiel zwei PEs in derselben Phase ab, während stets nur Tasks aus $\mathcal{T}_{\mathrm{kalt}}$ zugeteilt wurden. Es sei darauf hingewiesen, dass bis zum Ende der dritten Phase auch PE_8 eine weitere Task übernehmen muss. Aufgrund dieser Zuteilung kühlt jedoch nicht notwendigerweise ein weiteres PE ab: Beispielsweise könnte erneut eine in $\mathcal{T}_{\mathrm{kalt}}$ enthaltene (und damit von PE_2 gestoppte) Task zugeteilt werden, wodurch nicht zwingend ein weiteres PE erkaltet.

Bemerkung 8.35

Ein PE_i kann ebenfalls erkalten, wenn die aktuell höchstpriore nicht zugeteilte Task $T_{\max}$ *keine* höhere Priorität als die niedrigstpriore auf dem PE zugeteilte Task besitzt.

In diesem Fall gelten die folgenden Überlegungen ebenso, da PE_i gemäß Induktionsannahme an $k_i \leq \lceil m/(v-i+1) \rceil$ Phasen beteiligt war und sich nun an keinen weiteren Phasen mehr beteiligen wird.

PE_{i+1} ist per definitionem das erste nach PE_i abkühlende PE, also müssen alle übrigen (potenziell noch heißen) $\mathrm{PE}_{i+2}, \ldots, \mathrm{PE}_v$ bisher an ebenso vielen Phasen beteiligt gewesen sein wie PE_{i+1}. Unter Verwendung der Induktionsannahme folgt somit der zweite Teil der Induktionsbehauptung, wenn gezeigt werden kann, dass PE_{i+1} an höchstens $k_{i+1} \leq \lceil m/(v-i) \rceil$ Phasen beteiligt war.

Um den ersten Teil der Induktionsbehauptung zu zeigen, muss dagegen gezeigt werden, dass PE_j mit $j \in \{1, \ldots, i+1\}$ an höchstens $\lceil m/(v-j+1) \rceil$ Phasen beteiligt war und kalt ist. Für $\mathrm{PE}_1 \ldots \mathrm{PE}_i$ gilt diese Aussage jedoch gemäß der Induktionsannahme; es muss die Behauptung also nur noch für PE_{i+1} gezeigt werden.

Da PE_{i+1} bereits erkaltet ist, muss es in derselben Phase wie PE_i erkaltet sein, das jedoch gemäß Induktionsannahme an höchstens $\lceil m/(v-i+1)\rceil$ Phasen beteiligt war. Es gilt daher $k_{i+1} = k_i$ und beide Teile der Induktionsbehauptung folgen, wenn $k_{i+1} \leq \lceil m/(v-i)\rceil$ gezeigt werden kann.

Es gilt also

$$k_{i+1} = k_i \leq \left\lceil \frac{m}{v-i+1} \right\rceil,$$

wegen der Monotonie von $f(i) \coloneqq m/i$ und der Monotonie der Aufrundungsfunktion gilt somit auch

$$k_{i+1} \leq \left\lceil \frac{m}{v-i+1} \right\rceil \leq \left\lceil \frac{m}{v-i} \right\rceil. \tag{8.1}$$

Somit ist PE_{i+1} nach $k_{i+1} \leq \left\lceil \frac{m}{v-i} \right\rceil$ Phasen erkaltet und die potenziell noch heißen $\text{PE}_{i+2} \dots \text{PE}_v$ waren bisher alle ebenfalls an $k_{i+1} \leq \lceil m/(v-i)\rceil$ Phasen beteiligt, woraus beide Teile der Induktionsbehauptung folgen. ♢

Bemerkung 8.36

Die Abschätzung $\left\lceil \frac{m}{v-i+1} \right\rceil \leq \left\lceil \frac{m}{v-i} \right\rceil$ aus Gleichung 8.1 ist nicht notwendigerweise eine *Überabschätzung*:

In Fall i) wurde $|\mathcal{T}_{\text{kalt}}| \leq m - (v-i) \cdot k_i$ abgeschätzt. Gilt jedoch $m-(v-i)\cdot k_i \leq 0$ für $k_i = \lceil m/(v-i+1)\rceil$, so lässt sich $z \coloneqq v-i+1$ setzen (wegen $i+1 < v$ gilt dann $z \geq 3$) und Lemma C.1 liefert

$$\left\lceil \frac{m}{z} \right\rceil = \left\lceil \frac{m}{z-1} \right\rceil \quad \text{und damit} \quad \left\lceil \frac{m}{v-i+1} \right\rceil = \left\lceil \frac{m}{v-i} \right\rceil.$$

Die Bedingung $m - (v-i) \cdot k_i \leq 0$ besagt hierbei, dass alle $v-i$ potenziell noch heißen PEs (von denen PE_{i+1} tatsächlich bereits erkaltet ist) gemeinsam genau m Tasks gestoppt haben („$= 0$“) bzw. mehr als m Tasks gestoppt haben („< 0“). Folglich könnte der Grund für das Erkalten von PE_{i+1} beispielsweise darin liegen, dass

a) Wegen $|\mathcal{T}_{\text{kalt}}| = 0$ (Fall „$= 0$“) die höchstpriore nicht zugeteilte Task $T_{\max}$ ursprünglich von PE_{i+1} gestoppt wurde, weshalb dieses nach Lemma 8.28, Fall b) abgekühlt ist.

b) Mindestens eine der von PE_{i+1} gestoppten Tasks erneut zugeteilt wurde (Fall „> 0“): Am Ende einer jeden Phase gibt es genau m Tasks, die keinem PE zugeteilt sind. Da die PEs gemeinsam mehr als m Tasks gestoppt haben, muss also mindestens eine

> der von diesen gestoppten Tasks erneut zugeteilt worden sein, weshalb PE_{i+1} somit gemäß Lemma 8.28, Fall a) abgekühlt ist.

Da die Induktion für $1 \leq i \leq v$ durchgeführt wurde, folgt die Behauptung für alle v PEs. □

Mithilfe von Satz 8.33 lässt sich nun die Dauer der Selbstheilung in einer Überlastsituation bei Verwendung des Eager Task Dropping abschätzen:

Satz 8.37. Die Selbstheilung in Überlastsituationen dauert bei Verwendung von Eager Task Dropping höchstens

$$2 \cdot \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + a \text{ Hormonzyklen.} \tag{8.2}$$

Beweis. Gemäß Satz 8.33 ist PE_i für $1 \leq i \leq v$ nach dem Bemerken des Ausfalls an $k_i \leq \lceil m/(v-i+1) \rceil$ Selbstheilungsphasen beteiligt und stoppt ebenso viele Tasks.

Folglich werden insgesamt

$$k := \sum_{i=1}^{v} k_i \leq \sum_{i=1}^{v} \left\lceil \frac{m}{v-i+1} \right\rceil = \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil$$

Tasks gestoppt. Mit Satz 8.18 folgt daher, dass die gesamte Selbstheilung höchstens

$$2k + a = 2 \cdot \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + a \text{ Hormonzyklen}$$

dauert, was zu zeigen war. □

8.6.3.1. Ausfall eines weiteren PEs

Bisher wurde angenommen, dass im gesamten System genau $m \cdot (v+1)$ Tasks existieren und ein $\text{PE}_\times$ ausfällt. Die bisherigen Überlegungen gelten jedoch uneingeschränkt weiter, wenn *nach* dem Abschluss der Selbstheilung ein weiteres PE ausfällt: Es existieren zwar weiterhin $m \cdot (v+1)$ Tasks. Da die vorherige Selbstheilung vor dem nächsten PE-Ausfall jedoch abgeschlossen war, sind zu diesem Zeitpunkt die m global niedrigstprioren Tasks gestoppt gewesen. Diese können jedoch nach Lemma 8.27 aufgrund der vorliegenden Überlastsituation nicht mehr zugeteilt werden, weshalb sie für die weitere Analyse ausgeblendet werden können. Folglich kann die neue Situation wie ein Szenario mit $v-1$ verbleibenden PEs und $m \cdot v = m \cdot ((v-1)+1)$ Tasks analysiert werden.

Die Selbstheilung der durch den zweiten PE-Ausfall verschärften Überlastsituation dauert daher nach Satz 8.37

$$\text{WCET}_{\text{zweite Selbstheilung}} \leq 2 \cdot \sum_{i=1}^{v-1} \left\lceil \frac{m}{i} \right\rceil + a \text{ Hormonzyklen,}$$

sofern $v - 1 \geq 1$ gilt (ansonsten ist keine Selbstheilung mehr möglich, da das letzte PE ausgefallen ist).

Verallgemeinert verbleiben nach dem n-ten PE-Ausfall[1] für $1 \leq n \leq v$ noch $v-(n-1)$ PEs und die zugehörige Selbstheilung dauert

$$\text{WCET}_{n\text{-te Selbstheilung}} \leq 2 \cdot \sum_{i=1}^{v-n+1} \left\lceil \frac{m}{i} \right\rceil + a \text{ Hormonzyklen.} \tag{8.3}$$

Für $n = v+1$ ist dagegen gerade das letzte PE ausgefallen, weshalb keine Selbstheilung mehr möglich ist.

Am Ende der n-ten Selbstheilung ist garantiert, dass nur die $m \cdot (v - (n-1))$ höchstprioren Tasks zugeteilt sind. Zu beachten ist jedoch, dass die $(n+1)$-te Selbstheilung gemäß Gleichung (8.3) zwar schneller beendet ist als die n-te Selbstheilung, es sind danach jedoch nur noch die $m \cdot (v - n)$ höchstprioren Tasks zugeteilt und somit gerade m Tasks weniger als nach der n-ten Selbstheilung.

8.6.3.2. Grafische Betrachtung der oberen Schranke

Abbildung 8.5 stellt die WCET der Selbstheilung in Überlastsituationen bei Verwendung des Eager Task Dropping aus Satz 8.37 für $a = 2$ sowie $1 \leq m \leq 50$ und $1 \leq v \leq 50$ grafisch dar. Die Abbildungen 8.6 und 8.7 zeigen dagegen Schnitte durch diesen Graphen, die bei Fixierung von exemplarisch gewählten Werten von m bzw. v entstehen, um den Kurvenverlauf besser erkennbar zu machen.

Fixierte Anzahl Tasks pro PE Für Praxisszenarien sind insbesondere die in Abbildung 8.6 dargestellten Schnitte relevant: Hier ist m jeweils fixiert, während v variiert wird. Dies ermöglicht es, Szenarien zu betrachten, in denen sukzessive PEs ausfallen: Beispielsweise ist die erste Selbstheilung für $m = 10$ und $v = 50$ nach höchstens $2 \cdot \sum_{i=1}^{50} \lceil 10/i \rceil + 2 = 148$ Hormonzyklen abgeschlossen, wie aus Abbildung 8.6a ersichtlich ist. Mit jedem zusätzlich ausfallenden PE sinken sowohl v als auch die zur Selbstheilung maximal benötigte Zeit gemäß Gleichung (8.3). Hierbei lassen sich für $m = 10$ beispielsweise mehrere abschnittsweise lineare Kurvenverläufe erkennen:

- Für $9 \leq v \leq 50$ haben je zwei aufeinanderfolgende WCET-Werte einen Abstand von zwei Hormonzyklen,
- für $4 \leq v \leq 9$ beträgt dieser Abstand je vier Hormonzyklen,
- zwischen den WCET-Werten für $1 \leq v \leq 4$ liegen dagegen 10, 8 und 6 Hormonzyklen; der Wert für $v = 1$ beträgt 22 Hormonzyklen.

Der Grund hierfür wird offensichtlich, wenn die Summanden von $\sum_{i=1}^{v} \lceil 10/i \rceil$ betrachtet werden:

[1] Wobei $n = 1$ für den ersten PE-Ausfall stehe, der das System von einer Unter- in eine Überlastsituation führt.

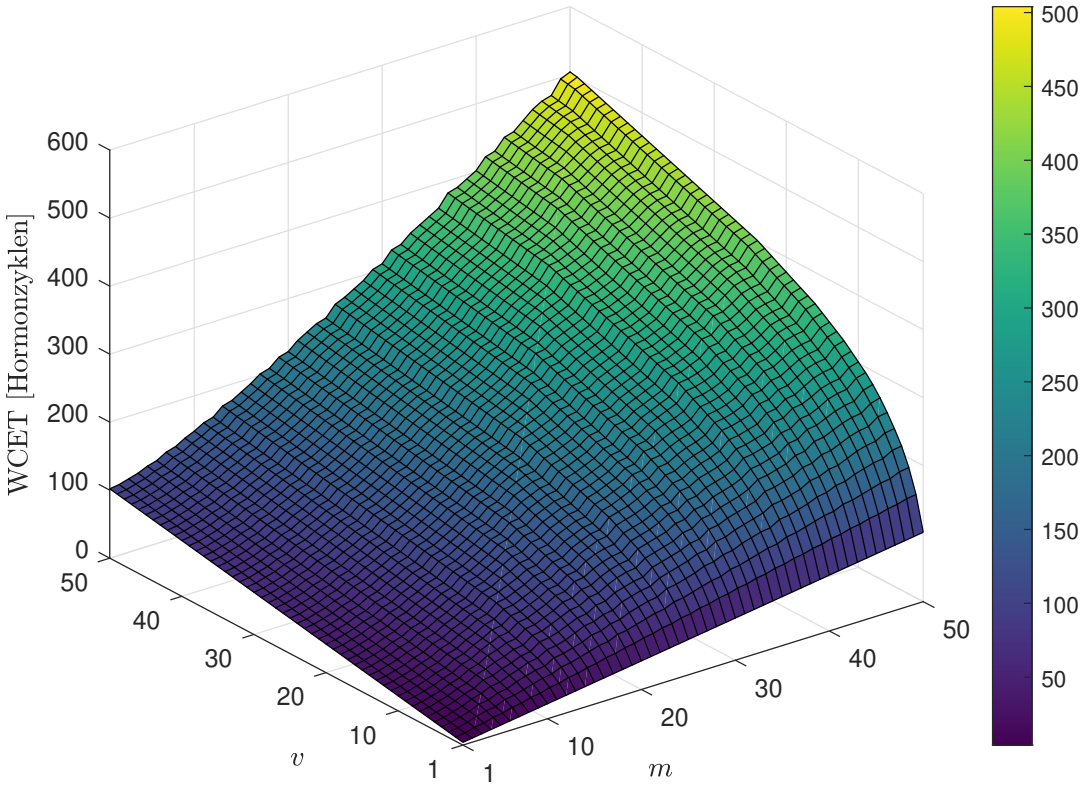
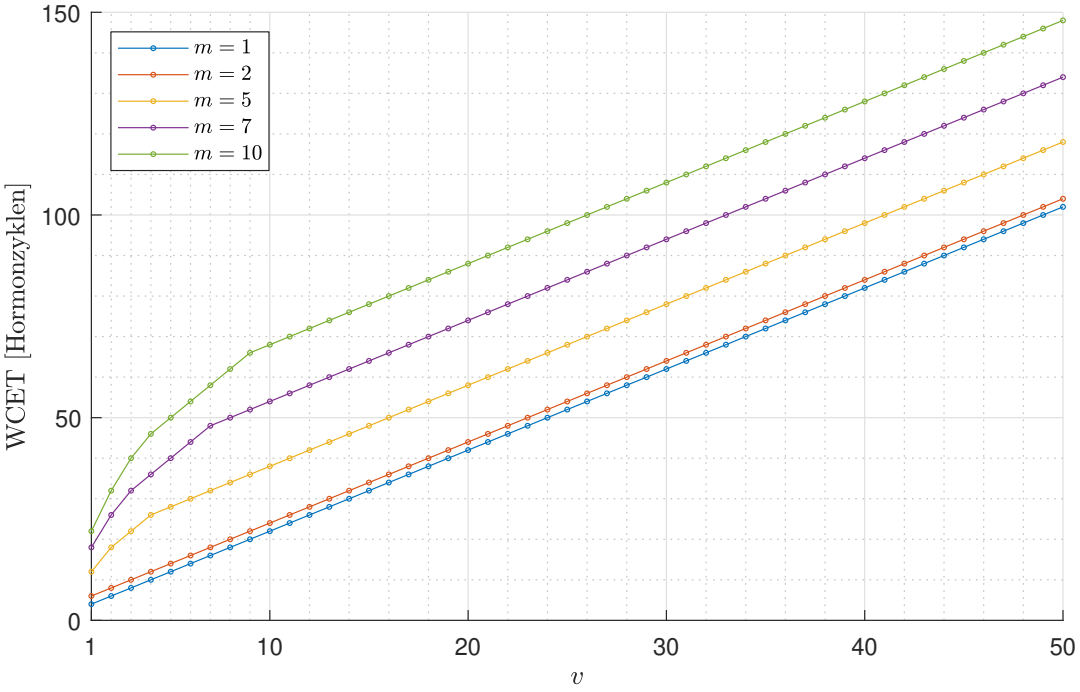
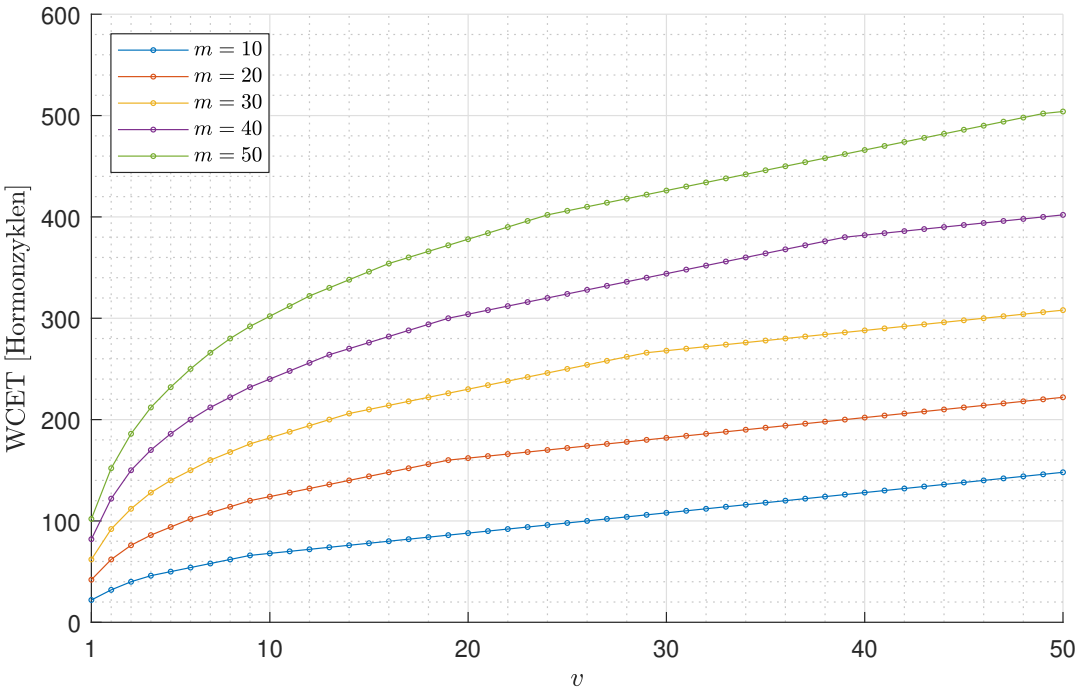


Abbildung 8.5: WCET der Eager-Task-Dropping-Strategie

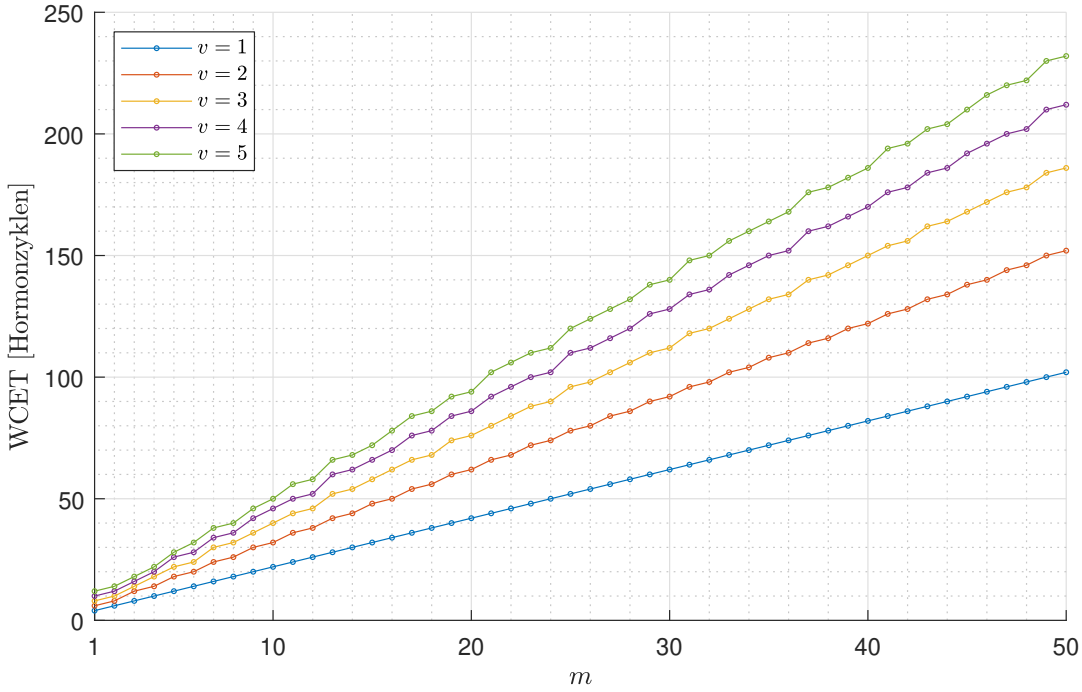


(a) $m \in \{1, 5, 7, 10\}$

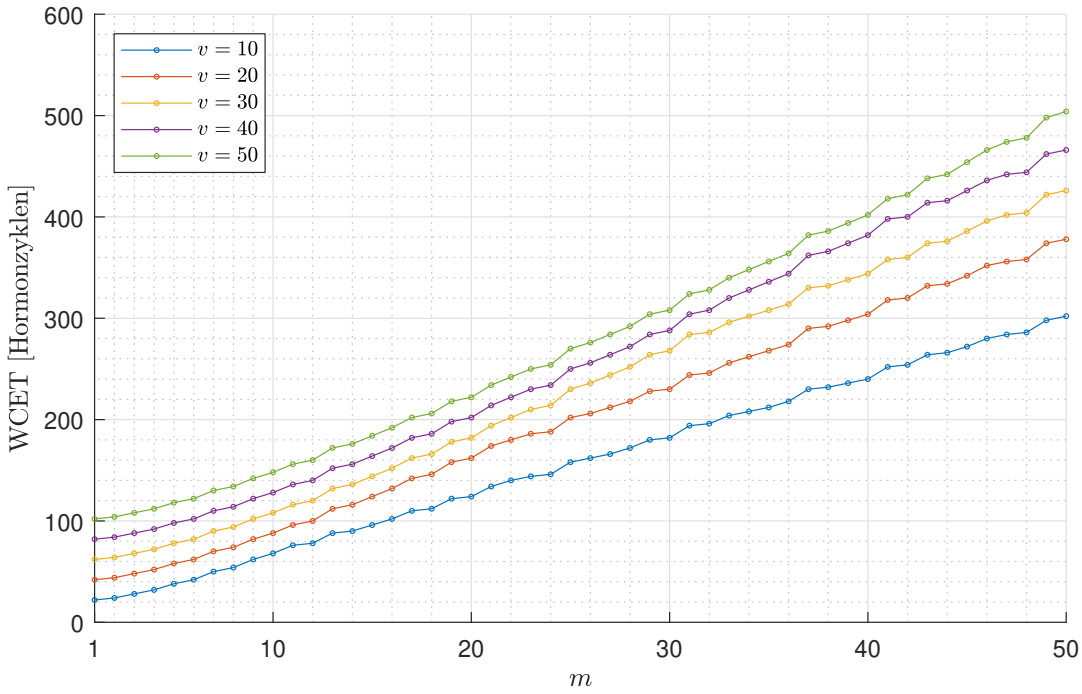


(b) $m \in \{10, 20, 30, 40, 50\}$

Abbildung 8.6: WCET der Eager-Task-Dropping-Strategie für fixiertes m



(a) $v \in \{1, 2, 3, 4, 5\}$



(b) $v \in \{10, 20, 30, 40, 50\}$

Abbildung 8.7: WCET der Eager-Task-Dropping-Strategie für fixiertes v

$\boldsymbol{i}$	1	2	3	4	5	6	7	8	9	≥ 10
$\lceil \mathbf{10}/\boldsymbol{i} \rceil$	10	5	4	3	2	2	2	2	2	1

Jeder Summand $\lceil 10/i \rceil$ entspricht für $i \geq 2$ gerade der *Hälfte* der Differenz der WCET-Werte, die sich für $v = i$ und $v = i - 1$ bei identischem m ergeben.

Auch in Abbildung 8.6b lassen sich zum Teil abschnittsweise lineare Kurvenverläufe erkennen. Insbesondere beträgt die Steigung der Graphen für $v \geq m - 1$ nur noch 2, da die Summanden $\lceil m/i \rceil$ für $i \geq m$ jeweils den Wert 1 haben: Sofern $v \geq m$ gilt, dauert die Selbstheilung für v verbleibende PEs also höchstens zwei Zyklen länger als eine Selbstheilung für $(v - 1)$ PEs.

Es sei an dieser Stelle jedoch noch einmal darauf hingewiesen, dass die zur Selbstheilung benötigte Zeit zwar mit jedem weiteren PE-Ausfall abnimmt, gleichzeitig jedoch auch die Anzahl der am Ende der Selbstheilung zugeteilten Tasks um je m abnimmt, wie in Abschnitt 8.6.3.1 beschrieben: Die Selbstheilung für $m = 10$ und $v = 50$ dauert höchstens 148 Hormonzyklen. Nach dieser Zeit ist garantiert, dass nur die $mv = 500$ höchstprioren Tasks im System zugeteilt sind. Die Selbstheilung für $m = 10$ und $v = 10$ hingegen dauert zwar nicht länger als 68 Hormonzyklen, danach sind jedoch nur noch $mv = 100$ Tasks zugeteilt.

Fixierte Anzahl verbleibender PEs Im Rahmen dieser Dissertation wird angenommen, dass die Hormonwerte für alle Tasks konstant sind, weshalb auch die Anzahl m der pro PE maximal ausführbaren Tasks für ein System während seiner Laufzeit unveränderlich ist. Daher ist die praktische Relevanz der in Abbildung 8.7 gezeigten Variationen von m bei jeweils fixiertem v gering. Nichtsdestotrotz liefern die dargestellten Kurven einen guten Einblick in den Verlauf der dreidimensionalen Darstellung aus Abbildung 8.5:

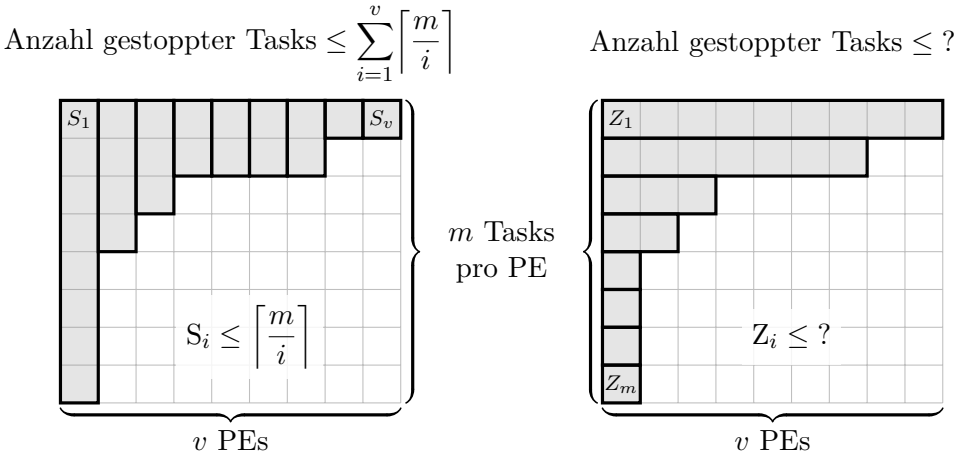
In Abbildung 8.7a sind die entsprechenden Kurven für $1 \leq v \leq 5$ dargestellt: Die zu $v = 1$ gehörende Kurve entspricht einer Geraden mit WCET $= 2m + a$. Die Kurvenverläufe für $2 \leq v \leq 5$ scheinen auf den ersten Blick ebenfalls angenähert lineares Verhalten zu zeigen, in den Teilbereichen mit $1 \leq m \leq 5$ lässt sich jedoch erahnen, dass dies im Allgemeinen nicht gilt.

Abbildung 8.7b zeigt die entsprechenden Kurven für größere Werte von v. Insbesondere für $v = 50$ bestätigt sich hier der Eindruck der Nichtlinearität. In Abschnitt 8.6.3.5 werden wir sehen, wie sich der Kurvenverlauf allgemein asymptotisch beschreiben lässt sowie unter welchen Voraussetzungen dennoch ein angenähert linearer Verlauf angenommen werden darf.

Nach diesen grafischen Betrachtungen der Zeitschranke wollen wir uns in den nächsten Abschnitten jedoch zunächst weiteren Analysen zuwenden.

8.6.3.3. Phasenorientierte Betrachtung

Satz 8.33 gibt eine obere Schranke für die Anzahl Tasks an, die ein PE im Laufe der Selbstheilung stoppt und liefert damit gewissermaßen eine *PE-orientierte* Sichtweise auf die Selbstheilung. Zur späteren Beantwortung anderer Fragestellungen wäre jedoch



(a) PE-orientierte (spaltenorientierte) Sichtweise: gestoppte Tasks pro PE.

(b) Phasenorientierte (zeilenorientierte) Sichtweise: gestoppte Tasks pro Phase.

Abbildung 8.8: Motivation der PE- und phasenorientierten Sichtweise auf die Selbstheilung in Überlastsituationen, hier für $m = 8$ und $v = 9$ dargestellt.

eine *phasenorientierte* Sichtweise auf die Selbstheilung erstrebenswert, wie sie in Abbildung 8.8 der bisherigen, PE-orientierten Sichtweise gegenübergestellt ist: Konkret werden wir in Abschnitt 8.6.3.4 zunächst anhand einer solchen phasenorientierten Sichtweise zeigen, dass die besagte Zeitschranke sogar eine *kleinste* obere Schranke darstellt. In Abschnitt 8.6.4 werden wir die phasenorientierte Sichtweise nutzen, um die Zeitschranke für die *Dauer* der Selbstheilung auf solche Szenarien zu verallgemeinern, in denen mehrere Tasks dieselbe Priorität besitzen.

In Vorbereitung dessen werden wir daher nun eine solche phasenorientierte Sichtweise entwickeln:

Satz 8.38. Für $1 \leq i \leq m$ bezeichne Z_i die Anzahl Tasks, die in Phase i gestoppt werden. Dann gilt

$$Z_i \leq \begin{cases} v, & i = 1 \\ \min\left\{v, \left\lfloor \frac{m-1}{i-1} \right\rfloor\right\}, & \text{sonst.} \end{cases}$$

Beweis. Um Z_i zu bestimmen, zählen wir die Anzahl der PEs, die in Phase i eine Task stoppen können. Gemäß Satz 8.33 stoppt PE_j höchstens $\lceil m/(v-j+1) \rceil$ Tasks. Der Ausdruck $[\lceil m/(v-j+1) \rceil \geq i]$ ist daher genau dann 1, wenn PE_j an Phase i beteiligt sein *kann*. Das Summieren über alle v PEs liefert daher

$$Z_i \leq \sum_{j=1}^{v} \left[\left\lceil \frac{m}{v-j+1} \right\rceil \geq i \right] = \sum_{j=1}^{v} \left[\left\lceil \frac{m}{j} \right\rceil \geq i \right]$$

Daher folgt die Behauptung, wenn

$$\sum_{j=1}^{v} \left\lfloor \left\lceil \frac{m}{j} \right\rceil \geq i \right\rfloor = \begin{cases} v, & i = 1, \\ \min\left\{v, \left\lfloor \frac{m-1}{i-1} \right\rfloor\right\}, & i > 1 \end{cases}$$

gezeigt werden kann. Hierzu müssen zwei Fälle unterschieden werden:

a) $i = 1$: Wegen $\lceil m/j \rceil \geq 1$ für alle $m \geq 1$ und $j \geq 1$ folgt unmittelbar die Behauptung.

b) $i > 1$. Es bezeichne $j' \in \mathbb{N}^+$ die kleinste positive natürliche Zahl, für die $\lceil m/j' \rceil < i$ gilt:

$$j' \coloneqq \min\left\{ j \in \mathbb{N}^+ \mid \lceil m/j \rceil < i \right\}$$

Wegen $i > 1$ muss ein solches j' existieren: Für $j \geq m$ gilt in jedem Fall $\lceil m/j \rceil = 1 < i$.

Da die Funktion $f(j) \coloneqq \lceil m/j \rceil$ monoton fallend ist, folgt

$$\sum_{j=1}^{\infty} \left\lfloor \left\lceil \frac{m}{j} \right\rceil \geq i \right\rfloor = j' - 1.$$

Wir benötigen jedoch nur die Teilsumme von $j = 1$ bis v; diese Summe kann aber nicht größer als v werden und es folgt

$$\sum_{j=1}^{v} \left\lfloor \left\lceil \frac{m}{j} \right\rceil \geq i \right\rfloor = \min\{v, j' - 1\}.$$

Dieser Term entspricht schon fast der Behauptung, es muss lediglich gezeigt werden, dass

$$j' - 1 = \left\lfloor \frac{m-1}{i-1} \right\rfloor \quad \text{für } i > 1,\ m \geq 1 \text{ gilt.}$$

Äquivalenzumformungen liefern schließlich

$$j' = \left\lceil \frac{m}{i-1} \right\rceil.$$

Da $j' \coloneqq \min\{j \in \mathbb{N}^+ \mid \lceil m/j \rceil < i\}$ gerade als die kleinste positive Zahl definiert ist, für die $\lceil m/j \rceil < i$ gilt, folgt die Behauptung, wenn die Aussagen

(1) $\left\lceil \frac{m}{\left\lceil \frac{m}{i-1} \right\rceil} \right\rceil < i$ für $i > 1$, $m \geq 1$ sowie

(2) $\left\lceil \frac{m}{\left\lceil \frac{m}{i-1} \right\rceil - 1} \right\rceil \geq i$ für $i > 1$, $m \geq 1$ und $\left\lceil \frac{m}{i-1} \right\rceil > 1$

gezeigt werden können: Während Teilaussage (1) ausdrückt, dass $\lceil m/j' \rceil < i$ gilt, stellt Teilaussage (2) sicher, dass es keine *kleinere* positive natürliche Zahl j'' gibt, für die dies gilt: Aufgrund der Monotonie von $f(j) = \lceil m/j \rceil$ genügt es, $f(j'-1) \geq i$ zu zeigen, um $f(k) \geq i$ für alle $k \in \mathbb{N}^+$ mit $k < j'$ zu beweisen.

Wenn dagegen $\lceil m/(i-1) \rceil = 1$ gilt, so muss Teilaussage (2) nicht gezeigt werden, da in diesem Fall keine kleinere positive natürliche Zahl als $\lceil m/(i-1) \rceil$ existiert.

Wir beweisen nun die beiden Behauptungen (1) und (2):

(1) *Beweis.* Für $m \geq 1$ und $i > 1$ gilt

$$\frac{m-1}{i-1} < \frac{m}{i-1},$$

also gilt auch

$$\frac{m-1}{i-1} < \left\lceil \frac{m}{i-1} \right\rceil$$

und damit

$$\frac{m-1}{\left\lceil \frac{m}{i-1} \right\rceil} < i-1.$$

Dann gilt jedoch auch

$$\left\lfloor \frac{m-1}{\left\lceil \frac{m}{i-1} \right\rceil} \right\rfloor < i-1, \text{ also}$$

$$\left\lfloor \frac{m-1}{\left\lceil \frac{m}{i-1} \right\rceil} \right\rfloor + 1 < i.$$

Eine Umwandlung in die Aufrundungsfunktion liefert schließlich die Behauptung

$$\left\lceil \frac{m}{\left\lceil \frac{m}{i-1} \right\rceil} \right\rceil < i. \qquad \lozenge$$

(2) *Beweis.* Wir beginnen mit der Aussage

$$\frac{m-1}{i-1} \geq \left\lfloor \frac{m-1}{i-1} \right\rfloor,$$

die für $i > 1$, $m \geq 1$ und $\left\lceil \frac{m}{i-1} \right\rceil > 1$ offensichtlich wahr ist. Eine Reihe von Äquivalenzumformungen liefert

$$\begin{aligned} \frac{m-1}{i-1} &\geq \left\lfloor \frac{m-1}{i-1} \right\rfloor + 1 - 1 \\ \frac{m-1}{i-1} &\geq \left\lceil \frac{m}{i-1} \right\rceil - 1 \\ \frac{m-1}{\left\lceil \frac{m}{i-1} \right\rceil - 1} &\geq i - 1. \end{aligned}$$

Da $i - 1$ ganzzahlig ist, gilt auch

$$\left\lfloor \frac{m-1}{\left\lceil \frac{m}{i-1} \right\rceil - 1} \right\rfloor \geq i - 1,$$

also

$$\left\lfloor \frac{m-1}{\left\lceil \frac{m}{i-1} \right\rceil - 1} \right\rfloor + 1 \geq i$$

und schließlich

$$\left\lceil \frac{m}{\left\lceil \frac{m}{i-1} \right\rceil - 1} \right\rceil \geq i,$$

was zu zeigen war. ◊

Damit folgt die Behauptung für $i > 1$.

Aus den Fällen a) und b) folgt nun die Behauptung. □

Lemma 8.39. Es gilt

$$\sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil = v + \sum_{i=2}^{m} \min \left\{ v, \left\lfloor \frac{m-1}{i-1} \right\rfloor \right\}.$$

Beweis. Folgt bei Summation von Z_i über alle m möglichen Selbstheilungphasen aufgrund der Konstruktion der Summanden Z_i gemäß Satz 8.38. □

Mithilfe von Satz 8.38 lässt sich nun eine alternative Formulierung der Zeitschranke zeigen:

Satz 8.40. Die Selbstheilung in Überlastsituationen benötigt höchstens

$$2 \cdot \left(v + \sum_{i=2}^{m} \min \left\{ v, \left\lfloor \frac{m-1}{i-1} \right\rfloor \right\} \right) + a \text{ Hormonzyklen.} \tag{8.4}$$

Beweis. Die Aussage folgt durch Transformation der Zeitschranke aus Satz 8.37 gemäß Lemma 8.39. Zur Motivation der folgenden Überlegungen wollen wir diese Zeitschranke jedoch analog zum Beweis von Satz 8.37 begründen:

Der Ausfall von $\mathrm{PE}_\times$ wird nach spätestens a Hormonzyklen erkannt. Eine Selbstheilungsphase, in der k Tasks gestoppt werden, dauert nach Satz 8.16 höchstens $2k$ Hormonzyklen. Summiert man über alle m möglichen Selbstheilungsphasen, folgt unter Verwendung von Satz 8.38

$$\begin{aligned}\mathrm{WCET} &\leq 2 \cdot \sum_{i=1}^{m} Z_i + a \\ &\leq 2 \cdot \left(v + \sum_{i=2}^{m} \min \left\{ v, \left\lfloor \frac{m-1}{i-1} \right\rfloor \right\} \right) + a. \end{aligned}$$ □

Aus dem Beweis von Satz 8.40 folgt nun unmittelbar, dass die zugehörige Gleichung (8.4) eine phasenorientierte Sichtweise auf die Selbstheilung in einer Überlastsituation liefert. Konkret lassen sich diese Abschätzung sowie jene aus Satz 8.37 bzw. Gleichung (8.2) wie folgt interpretieren:

a) $\mathrm{WCET} \leq 2 \cdot \sum_{i=1}^{v} \lceil \frac{m}{i} \rceil + a$ Hormonzyklen (Gleichung (8.2)):

   PE_i ist – bei entsprechender Nummerierung der PEs – an höchstens $\lceil \frac{m}{i} \rceil$ Selbstheilungsphasen beteiligt und stoppt somit höchstens $\lceil \frac{m}{i} \rceil$ Tasks.

b) $\mathrm{WCET} \leq 2 \cdot \left(v + \sum_{i=2}^{m} \min \left\{ v, \left\lfloor \frac{m-1}{i-1} \right\rfloor \right\} \right) + a$ Hormonzyklen (Gleichung (8.4)):

   - In der ersten Selbstheilungsphase werden höchstens v Tasks gestoppt.
   - In Selbstheilungsphase i mit $2 \leq i \leq m$ werden höchstens $\min \left\{ v, \left\lfloor \frac{m-1}{i-1} \right\rfloor \right\}$ Tasks gestoppt.

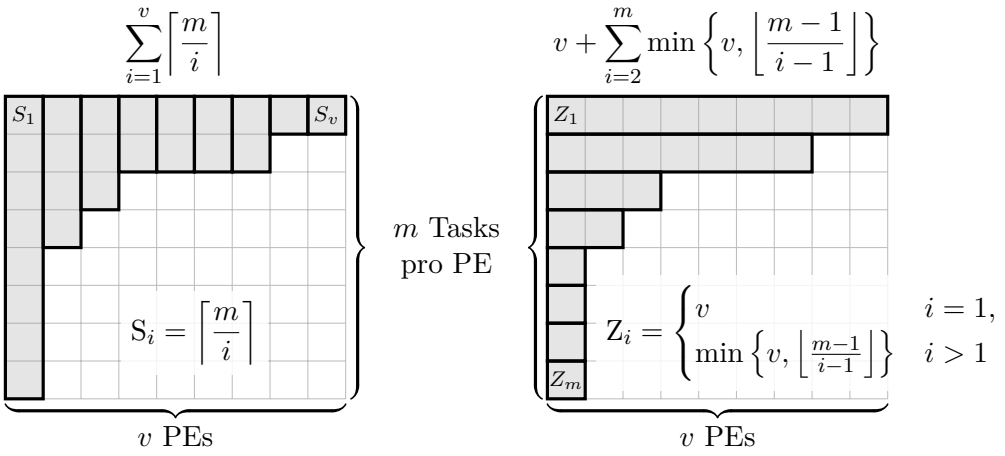
Diese beiden Interpretationen sind in Abbildung 8.9 gegenübergestellt.

Bemerkung 8.41. Für $v \geq m - 1$ vereinfacht sich die Abschätzung aus Satz 8.40 zu

$$\mathrm{WCET} \leq 2 \cdot \left(v + \sum_{i=2}^{m} \left\lfloor \frac{m-1}{i-1} \right\rfloor \right) + a,$$

da in diesem Fall $\lfloor (m-1)/(i-1) \rfloor \leq v$ gilt und daher die Bestimmung der Minima in den Summanden entfallen kann.

Diese Abschätzung lässt sich jedoch auch durch algebraische Umformungen begrün-



(a) PE-orientierte (spaltenorientierte) Sichtweise: gestoppte Tasks pro PE.

(b) Phasenorientierte (zeilenorientierte) Sichtweise: gestoppte Tasks pro Phase.

Abbildung 8.9: PE- und phasenorientierte Sichtweise auf die Selbstheilung in Überlastsituationen für $m = 8$ und $v = 9$.

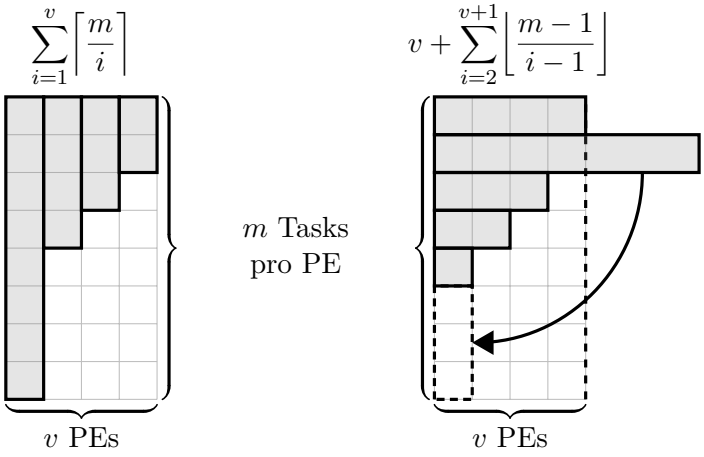
den: Nach Satz 8.37 gilt

$$\begin{aligned} \text{WCET} &\leq 2 \cdot \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + a \\ &= 2 \cdot \sum_{i=1}^{v} \left(\left\lfloor \frac{m-1}{i} \right\rfloor + 1 \right) + a \\ &= 2 \cdot \left(v + \sum_{i=1}^{v} \left\lfloor \frac{m-1}{i} \right\rfloor \right) + a \\ &= 2 \cdot \left(v + \sum_{i=2}^{v+1} \left\lfloor \frac{m-1}{i-1} \right\rfloor \right) + a. \end{aligned} \tag{8.5}$$

Für $i-1 > m-1$ gilt jedoch $\lfloor (m-1)/(i-1) \rfloor = 0$. Da zudem gemäß der Voraussetzung $v \geq m-1$ und damit $v+1 \geq m$ gilt, kann die obere Summationsgrenze auf m beschränkt werden:

$$= 2 \cdot \left(v + \sum_{i=2}^{m} \left\lfloor \frac{m-1}{i-1} \right\rfloor \right) + a.$$

Im Gegensatz zur vorherigen Herleitung liefert diese Umformung jedoch keinen Beweis, dass es sich hierbei tatsächlich um eine phasenorientierte Sichtweise auf den Ablauf der Selbstheilung handelt. Insbesondere ist obige Umformung nur für $v \geq m-1$ korrekt. Bis einschließlich Gleichung (8.5) gelten die Überlegungen zwar auch für $v < m-1$, diese



(a) PE-orientierte (spaltenorientierte) Sichtweise: gestoppte Tasks pro PE.

(b) Phasenorientierte (zeilenorientierte) Sichtweise: gestoppte Tasks pro Phase.

Abbildung 8.10: Falsche Resultate bei phasenorientierter Sichtweise für $v < m - 1$ gemäß Gleichung (8.5), hier exemplarisch für $m = 8$ und $v = 4$ dargestellt.

Abschätzung liefert für diesen Fall jedoch keine korrekte phasenorientierte Sichtweise, da $\lfloor (m-1)/i \rfloor > v$ gelten kann, wie Abbildung 8.10 verdeutlicht.[1]

Im Gegensatz dazu liefert Gleichung (8.4) aus Satz 8.40 auch im Fall $v < m - 1$ korrekte Resultate, wie Abbildung 8.11 illustriert.

8.6.3.4. Abschätzung nach unten

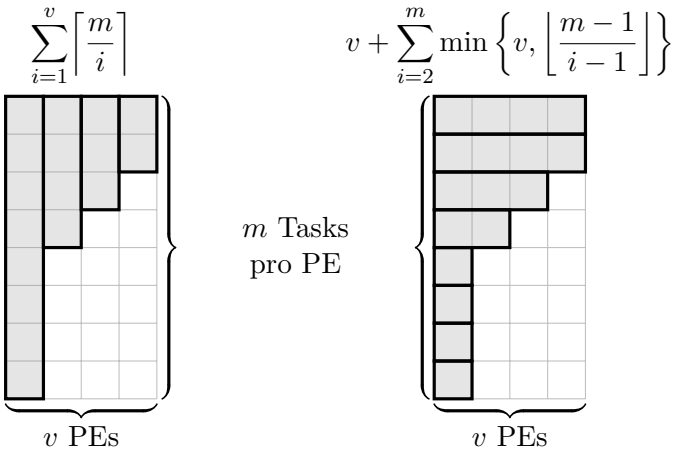
Nachdem mit Satz 8.37 (beziehungsweise dem daraus abgeleiteten Satz 8.40) eine obere Schranke für die Dauer der Selbstheilung in einer Überlastsituation bei Verwendung des Eager Task Dropping gefunden werden konnte, wollen wir in diesem Abschnitt nachweisen, dass die gefundene Schranke sogar die *kleinste* obere Schranke darstellt:

Satz 8.42. Die Zeitschranke von $2 \cdot \sum_{i=1}^{v} \lceil \frac{m}{i} \rceil + a$ Hormonzyklen aus Satz 8.37 ist die kleinste obere Schranke für die Selbstheilung in Überlastsituationen bei Verwendung von Eager Task Dropping, falls $v \geq m$ gilt.

Beweis. Konkret ist zu zeigen, dass für alle $v \geq m$ Szenarien existieren, in denen die Selbstheilung in Überlastsituationen *genau* $2 \cdot \sum_{i=1}^{v} \lceil \frac{m}{i} \rceil + a$ Hormonzyklen benötigt.

Sei $v \geq m$. Wir zeigen die Behauptung nun anhand der für diesen Fall äquivalenten Zeitschranke von $2 \cdot \left(v + \sum_{p=2}^{m} \left\lfloor \frac{m-1}{p-1} \right\rfloor \right) + a$ Hormonzyklen aus Bemerkung 8.41.

[1] Die *Summe* der einzelnen Terme ist jedoch identisch zur PE-orientierten Sichtweise und daher korrekt, lediglich zur phasenorientierten Sichtweise auf die Selbstheilung ist die Abschätzung für $v < m - 1$ nicht geeignet.



(a) PE-orientierte (spaltenorientierte) Sichtweise: gestoppte Tasks pro PE.

(b) Phasenorientierte (zeilenorientierte) Sichtweise: gestoppte Tasks pro Phase.

Abbildung 8.11: Korrekte Resultate bei phasenorientierter Sichtweise für $v < m - 1$ gemäß Gleichung (8.4), hier exemplarisch für $m = 8$ und $v = 4$ dargestellt.

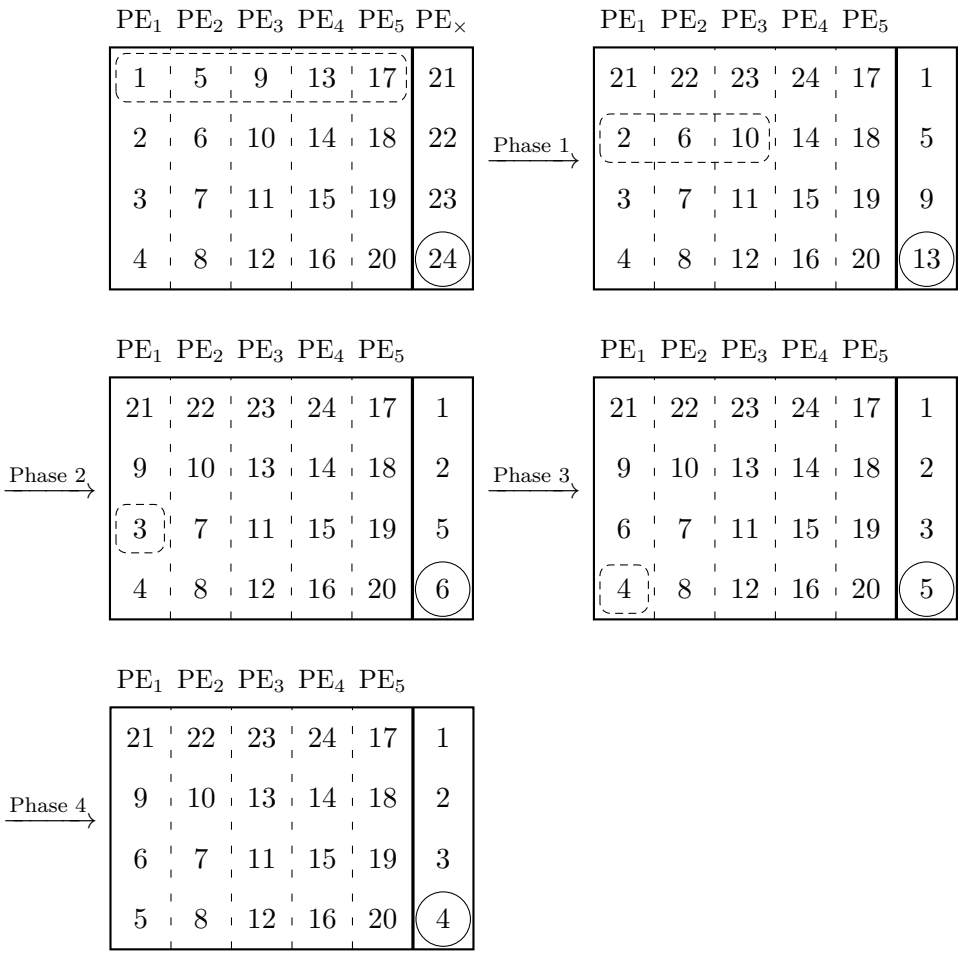
Im Worst Case dauert die Erkennung des Ausfalls von $\mathrm{PE}_\times$ gerade a Hormonzyklen. Aufgrund von Lemma 8.17 sowie Satz 8.18 in Verbindung mit Bemerkung 8.19 folgt die Behauptung daher, wenn wir eine Taskverteilung von $m(v+1)$ Tasks paarweise verschiedener Prioritäten finden können, sodass

1. In Phase 1 genau v PEs jeweils eine Task stoppen.
2. In den Phasen $2 \leq p \leq m$ jeweils genau $\left\lfloor \frac{m-1}{p-1} \right\rfloor$ PEs jeweils eine Task stoppen.

Beispiel 8.43

Die Idee ist es, ein Szenario zu konstruieren, in dem alle Tasks auf PE_j eine höhere Priorität als alle Tasks auf PE_k mit $j > k$ haben; zudem haben alle Tasks auf $\mathrm{PE}_\times$ eine höhere Priorität als alle übrigen Tasks. Dies impliziert insbesondere, dass die m global niedrigstprioren Tasks alle auf PE_1 zugeteilt sind und die Selbstheilung folglich genau m Phasen benötigt.

Für $m = 4$ und $v = 5$ ist der Ablauf einer solchen Selbstheilung im Folgenden exemplarisch illustriert. Wie in den vorherigen Beispielen werden in der letzten Spalte jeweils die m nicht zugeteilten Tasks dargestellt (zu Beginn sind das gerade jene Tasks, die zum Zeitpunkt des Ausfalls auf $\mathrm{PE}_\times$ zugeteilt waren). Die höchstpriore nicht zugeteilte Task $T_{\max}$ ist jeweils markiert, ebenso wie die aufgrund von $T_{\max}$ gestoppten Tasks.



Nach $m = 4$ Phasen sind alle m global niedrigstprioren Tasks gestoppt, weshalb die Selbstheilung beendet ist.

Insgesamt werden hier also 10 Tasks gestoppt und neu zugeteilt, was $20 + a$ Hormonzyklen dauert. Dies stimmt exakt mit dem durch die Zeitschranke vorhergesagten Wert überein:

$$\begin{aligned} 2 \cdot \left(v + \sum_{p=2}^{m} \left\lfloor \frac{m-1}{p-1} \right\rfloor \right) + a &= 2 \cdot \left(5 + \sum_{p=2}^{4} \left\lfloor \frac{3}{p-1} \right\rfloor \right) + a \\ &= 2 \cdot (5 + \lfloor 3/1 \rfloor + \lfloor 3/2 \rfloor + \lfloor 3/3 \rfloor) + a \\ &= 2 \cdot (5 + 3 + 1 + 1) + a \\ &= 20 + a \text{ Hormonzyklen.} \end{aligned}$$

Für $m = 4$ und $v = 5$ ist diese Zeitschranke folglich auch die *kleinste* obere Schranke. Um dies allgemein zu beweisen, werden wir die Taskverteilung im Folgenden formalisieren.

Sei $\mathcal{T} = \left\{T_1, \ldots, T_{(v+1)m}\right\}$ die Menge aller Tasks des Systems. Es gelte $\text{prio}(T_i) = i$ für alle $T_i \in \mathcal{T}$. Im Folgenden wird der Einfachheit halber folgende Konvention verwendet:

Konvention 8.44. Statt eine Task T_i explizit mit T_i zu bezeichnen, kann sie auch durch bloße Angabe ihrer Priorität i bezeichnet werden.

Da alle Prioritäten verschieden sind, bestimmt die Angabe einer Priorität $1 \leq i \leq (v+1)m$ eindeutig die zugehörige Task T_i.

Wir betrachten nun die folgende Verteilung der $|\mathcal{T}| = (v+1)m$ Tasks auf $\text{PE}_\times$ sowie die v verbleibenden PEs zum Zeitpunkt des Ausfalls von $\text{PE}_\times$:

PE_1	PE_2	$\ldots$	PE_i	$\ldots$	PE_v	$\text{PE}_\times$
1	$m+1$		$(i-1)m+1$		$(v-1)m+1$	$vm+1$
2	$m+2$		$(i-1)m+2$		$(v-1)m+2$	$vm+2$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
j	$m+j$		$(i-1)m+j$		$(v-1)m+j$	$vm+j$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
m	$2m$		im		vm	$(v+1)m$

Insbesondere sind also alle m global niedrigstprioren Tasks auf PE_1 zugeteilt, weshalb nach Lemma 8.27 genau m Selbstheilungsphasen nötig sind.

In der ersten Phase ist die höchstpriore nicht zugeteilte Task $T_{\max}$ gerade Task $(v+1)m$. Wegen $(v+1)m > (i-1)m+1$ für alle $1 \leq i \leq v$ werden alle v verbleibenden PEs ihre jeweils niedrigstpriore Task $(i-1)m+1$ stoppen, es werden also, wie in Bedingung 1 gefordert, genau v Tasks in Phase 1 gestoppt.

Nun werden die v höchstprioren nicht zugeteilten Tasks zugeteilt. Dies sind zunächst alle m Tasks von $\text{PE}_\times$, die o.B.d.A. auf $\text{PE}_m \ldots \text{PE}_1$ zugeteilt werden. Im Anschluss daran werden noch die $(v-m)$ höchstprioren der gerade gestoppten Tasks wieder zugeteilt; dies sind gerade die Tasks $(v-1)m+1, \ldots, ((m+1)-1)m+1$, die o.B.d.A. auf ihren ursprünglichen PEs erneut zugewiesen werden, weshalb diese gemäß Lemma 8.28 erkalten. Folglich sind $\text{PE}_{m+1} \ldots \text{PE}_v$ nun kalt, während $\text{PE}_1 \ldots \text{PE}_m$ potenziell noch heiß sind.

Für $m = 1$ ist die Selbstheilung damit abgeschlossen und die Behauptung somit bewiesen, da in Phase 1 exakt v Tasks gestoppt wurden. Für $m > 1$ folgen dagegen weitere Selbstheilungsphasen, wobei gezeigt werden muss, dass in Phase $2 \leq p \leq m$ genau $\lfloor (m-1)/(p-1) \rfloor$ Tasks gestoppt werden. Dies geschieht durch Induktion über die Anzahl der Selbstheilungsphasen bei einem beliebigen, aber festen $m > 1$:

Induktionsanfang. Für $m > 1$ und Phase $p = 2$ gilt:

1. Genau $\lfloor (m-1)/1 \rfloor = m-1$ PEs sind an dieser Phase beteiligt und stoppen jeweils genau eine Task.

2. Am Ende der Phase existiert ein $l \in \{1, 2\}$, sodass die m nicht zugeteilten Tasks nach folgendem Schema gegeben sind (für $l = 1$ entfällt die erste Zeile):

$$\left.\begin{array}{r} \left\lfloor \frac{m-1}{2} \right\rfloor m + l \\ \left\lfloor \frac{m-1}{2} \right\rfloor m + 1 \\ \\ \left(\left\lfloor \frac{m-1}{2} \right\rfloor - 1 \right) m + 2 \\ \left(\left\lfloor \frac{m-1}{2} \right\rfloor - 1 \right) m + 1 \\ \\ \vdots \\ \\ 0m + 2 \\ 0m + 1 \end{array}\right\} m \text{ Tasks}$$

Entsprechend ist die höchstpriore nicht zugeteilte Task gerade $\left\lceil \frac{m-1}{2} \right\rceil m + l$.

Beweis. Am Ende von Phase 1 sind die Tasks wie folgt auf die PEs verteilt (die kalten $\mathrm{PE}_{m+1} \ldots \mathrm{PE}_v$ werden im Folgenden nicht mehr dargestellt; auf jedem PE sind die Tasks nach aufsteigender Priorität sortiert abgebildet):

PE_1	PE_2	$\ldots$	PE_i	$\ldots$	PE_m	Nicht zugeteilte Tasks
2	$m+2$		$(i-1)m+2$		$(m-1)m+2$	1
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
j	$m+j$		$(i-1)m+j$		$(m-1)m+j$	$(j-1)m+1$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
m	$2m$		im		m^2	$(m-2)m+1$
$vm+1$	$vm+2$		$vm+i$		$(v+1)m$	$(m-1)m+1$

Die Prioritäten der nicht zugeteilten Tasks sind also von der Form $(j-1)m+1$ für $1 \leq j \leq m$, die zugehörigen Tasks sind gerade die m in Phase 1 gestoppten Tasks, die nicht erneut zugeteilt wurden. Die höchstpriore nicht zugeteilte Task

hat also die Priorität $(m-1)m+1$; die niedrigstprioren Tasks der potenziell noch heißen PEs haben dagegen die Prioritäten $(i-1)m+2$ für $1 \leq i \leq m$.[1]

Zu zeigen ist nun, dass genau $\lfloor (m-1)/1 \rfloor = m-1$ PEs jeweils ihre niedrigstpriore Task stoppen. Ist dies der Fall, müssen dies jedoch gerade $\mathrm{PE}_1 \ldots \mathrm{PE}_{m-1}$ sein, da diese unter allen potenziell noch heißen PEs die Tasks mit den kleinsten Prioritäten aufweisen. Zu zeigen ist also:

i) PE_i mit $1 \leq i \leq m-1$ stoppt seine niedrigstpriore Task, da ihre Priorität geringer als $\mathrm{prio}(T_{\max}) = (m-1)m+1$ ist.

   Es muss hierzu lediglich die niedrigstpriore auf PE_{m-1} zugeteilte Task $((m-1)-1)m+2$ untersucht werden, da die übrigen infrage kommenden Tasks der anderen PEs noch geringere Prioritäten haben und daher ebenfalls gestoppt werden, wenn Task $((m-1)-1)m+2$ gestoppt wird. Ein Vergleich mit $\mathrm{prio}(T_{\max}) = (m-1)m+1$ liefert:

$$\begin{gathered} ((m-1)-1)m+2 < (m-1)m+1 \\ \Updownarrow \\ 1 < m \end{gathered}$$

   Diese Aussage ist für alle $m > 1$ wahr (für $m = 1$ ist die Selbstheilung ohnehin bereits beendet; die Task $(i-1)m+2$ ist in diesem Fall auch gar nicht auf PE_i zugeteilt).

ii) PE_i mit $i \geq m$ stoppt seine niedrigstpriore Task *nicht*, da ihre Priorität mindestens so hoch wie $\mathrm{prio}(T_{\max}) = (m-1)m+1$ ist.

   Hierbei muss lediglich die niedrigstpriore auf PE_m zugeteilte Task $(m-1)m+2$ betrachtet werden, da die übrigen infrage kommenden Tasks der anderen PEs noch höhere Prioritäten haben und ebenfalls nicht gestoppt werden, wenn Task $(m-1)m+2$ nicht gestoppt wird. Ein Vergleich mit $\mathrm{prio}(T_{\max}) = (m-1)m+1$ liefert:

$$\begin{gathered} (m-1)m+2 \geq (m-1)m+1 \\ \Updownarrow \\ 2 \geq 1 \end{gathered}$$

   Diese Aussage ist ebenfalls wahr.

Folglich stoppen genau $\mathrm{PE}_1 \ldots \mathrm{PE}_{m-1}$ jeweils eine Task in Phase 2. Dies sind jedoch gerade $m-1 = \lfloor (m-1)/1 \rfloor$ PEs, womit die erste Teilaussage des Induktionsanfangs gezeigt ist.

[1] Die niedrigstprioren Tasks auf $\mathrm{PE}_{m+1} \ldots \mathrm{PE}_v$ haben dagegen die Prioritäten $(i-1)m+1$ für $m+1 \leq i \leq v$. Da die von diesen PEs gestoppten Tasks in Phase 1 aber erneut zugeteilt wurden, sind die PEs bereits erkaltet, können daher keine Tasks mehr stoppen und werden im Folgenden nicht mehr betrachtet.

Um auch den zweiten Teil zu zeigen, betrachten wir die insgesamt $m + m - 1 = 2m - 1$ zum aktuellen Zeitpunkt nicht im System zugeteilten Tasks, die nach folgendem Schema gegeben sind:

In Phase 2 gestoppt ($m-1$ Tasks)	Bereits zuvor gestoppt (m Tasks)
	$(m-1)m+1$
$(m-2)m+2$	
Differenz: $\Delta = 1$	$(m-2)m+1$
$(m-3)m+2$	
Differenz: $\Delta = 1$	$(m-3)m+1$
$\vdots$	$\vdots$
$0m+2$	
Differenz: $\Delta = 1$	$0m+1$

Da in dieser Phase insgesamt $m-1$ Tasks gestoppt wurden, werden nun die $m-1$ höchstprioren aller nicht zugeteilten Tasks zugeteilt. Im Anschluss wird eine Task der Form $km + l$ mit $l \in \{1, 2\}$ die höchstpriore nicht zugeteilte Task sein. Dies ist in obigem Schema jedoch gerade die $(m - 1 + 1) = m$-te Task von oben bzw. die m-te Task von unten. Von unten beginnend startet k bei 0 und wird nach je zwei Tasks inkrementiert, folglich ergibt sich k für die höchstpriore nicht zugeteilte Task gerade zu $\lfloor (m-1)/2 \rfloor$.

Die in der nächsten Phase höchstpriore nicht zugeteilte Task hat somit die Priorität $km + l = \left\lfloor \frac{m-1}{2} \right\rfloor m + l$ für ein $l \in \{1, 2\}$; die Prioritäten der am Ende von Phase 2 nicht zugeteilten Tasks sind daher nach folgendem Schema gegeben (für $l = 1$ entfällt die erste Zeile):

$$\left.\begin{array}{r}
\left\lfloor \frac{m-1}{2} \right\rfloor m + l \\
\left\lfloor \frac{m-1}{2} \right\rfloor m + 1 \\
\\
\left(\left\lfloor \frac{m-1}{2} \right\rfloor - 1 \right) m + 2 \\
\left(\left\lfloor \frac{m-1}{2} \right\rfloor - 1 \right) m + 1 \\
\\
\vdots \\
\\
0m + 2 \\
0m + 1
\end{array}\right\} m \text{ Tasks}$$

Damit sind beide Teilaussagen des Induktionsanfangs gezeigt. ◊

Bemerkung 8.45

Im Folgenden wird die Verteilung der Tasks auf die PEs nicht mehr dargestellt, da sie mit jeder weiteren Phase komplexer zu formalisieren wird. Stattdessen sei auf das zuvor betrachtete Beispiel 8.43 verwiesen.

Nichtsdestotrotz sei darauf hingewiesen, dass wir im Folgenden zu Beginn einer jeden Phase p überprüfen werden, ob PE_i seine niedrigstpriore Task stoppt, was genau dann der Fall ist, wenn ihre Priorität geringer als $\mathrm{prio}(T_{\max})$ ist. Die niedrigstpriore Task auf PE_i ist in Phase p jedoch entweder eine Task T, die auf PE_i in einer Phase $p' < p$ (erneut) zugeteilt wurde, oder aber eine Task T' mit $\mathrm{prio}(T') = (i-1)m + p$: Zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$ ist T' gerade die p-höchstpriore Task auf PE_i, weshalb diese frühestens in Phase p gestoppt werden kann, da PE_i pro Phase höchstens eine Task stoppt.

Entsprechend reicht es aus, zu überprüfen, ob $T_{\max}$ eine höhere Priorität als $\mathrm{prio}(T') = (i-1)m + p$ besitzt, um festzustellen, ob PE_i in Phase p eine Task stoppt: Falls die niedrigstpriore Task von PE_i nicht T' ist, sondern stattdessen besagte erneut zugeteilte Task T, so kann diese gemäß Lemma 8.8 nicht mehr gestoppt werden, da sie im Laufe der Selbstheilung zugeteilt wurde. Dann kann jedoch auch die *höherpriore* Task T' erst recht nicht mehr gestoppt werden.

Induktionsschritt $p \to p + 1$.

Induktionsannahme. Für die Phasen $2 \leq p < m$ gelten die folgenden Aussagen:

1. An Phase p waren genau $\left\lfloor \frac{m-1}{p-1} \right\rfloor$ PEs beteiligt, es wurden also ebenso viele Tasks gestoppt.

2. Am Ende von Phase p sind die m nicht zugeteilten Tasks für ein $l \in \{1, \ldots, p\}$ nach folgendem Schema gegeben:

$$\left.\begin{array}{r}
\left\lfloor \frac{m-1}{p} \right\rfloor m + l \\
\vdots \\
\left\lfloor \frac{m-1}{p} \right\rfloor m + 1 \\
\left(\left\lfloor \frac{m-1}{p} \right\rfloor - 1 \right) m + p \\
\vdots \\
\left(\left\lfloor \frac{m-1}{p} \right\rfloor - 1 \right) m + 1 \\
\\
\vdots \\
\\
0m + p \\
\vdots \\
0m + 1
\end{array}\right\} m \text{ Tasks}$$

Folglich ist die höchstpriore nicht zugeteilte Task $T_{\max}$ gerade $\left\lfloor \frac{m-1}{p} \right\rfloor m + l$.

Induktionsbehauptung. Wenn die Induktionsannahme gilt, so gelten auch die folgenden Aussagen:

1. An Phase $p+1$ sind genau $\left\lfloor \frac{m-1}{p} \right\rfloor$ PEs beteiligt, es werden also ebenso viele Tasks gestoppt.
2. Am Ende von Phase $p+1$ sind die m nicht zugeteilten Tasks für ein $l' \in \{1, \ldots, p+1\}$ nach folgendem Schema gegeben:

$$\left.\begin{array}{r}
\left\lfloor \frac{m-1}{p+1} \right\rfloor m + \quad l' \\
\vdots \\
\left\lfloor \frac{m-1}{p+1} \right\rfloor m + \quad 1 \\
\left(\left\lfloor \frac{m-1}{p+1} \right\rfloor - 1 \right) m + p + 1 \\
\vdots \\
\left(\left\lfloor \frac{m-1}{p+1} \right\rfloor - 1 \right) m + \quad 1 \\
\\
\vdots \\
\\
0m + p + 1 \\
\vdots \\
0m + \quad 1
\end{array}\right\} m \text{ Tasks}$$

Folglich ist die Priorität der höchstprioren nicht zugeteilten Task $T_{\max}$ gerade $\left\lfloor \frac{m-1}{p+1} \right\rfloor m + l'$.

Beweis. Gemäß Induktionsannahme ist die Priorität der höchstprioren nicht zugeteilten Task $T_{\max}$ am Ende von Phase p (und somit auch am Anfang von Phase $p+1$) gerade $\left\lfloor \frac{m-1}{p} \right\rfloor m + l$ für ein $l \in \{1, \ldots, p\}$.

Zu zeigen ist nun zunächst, dass in Phase $p+1$ genau $\left\lfloor \frac{m-1}{p} \right\rfloor$ Tasks gestoppt werden, nämlich je eine Task von $\mathrm{PE}_1 \ldots \mathrm{PE}_{\lfloor (m-1)/p \rfloor}$. Es sind also zwei Aussagen zu zeigen:

i) PE_i mit $1 \leq i \leq \lfloor (m-1)/p \rfloor$ stoppt seine niedrigstpriore Task, da diese eine geringere Priorität als $T_{\max}$ hat. Dies gilt gemäß Bemerkung 8.45, wenn folgende Ungleichung erfüllt ist:

$$(i-1)\,m + (p+1) < \left\lfloor \frac{m-1}{p} \right\rfloor m + l$$

Die linke Seite wird maximal für $i = \lfloor (m-1)/p \rfloor$, die rechte Seite minimal für $l = 1$:

$$\begin{aligned}
&\Updownarrow \\
\left(\left\lfloor \frac{m-1}{p} \right\rfloor - 1 \right) m + (p+1) &< \left\lfloor \frac{m-1}{p} \right\rfloor m + 1 \\
&\Updownarrow \\
p + 1 - 1 &< \left(\left\lfloor \frac{m-1}{p} \right\rfloor - \left\lfloor \frac{m-1}{p} \right\rfloor + 1 \right) m \\
&\Updownarrow \\
p &< m
\end{aligned}$$

Diese Aussage ist also für alle $p+1 \leq m$ wahr (im Fall $p+1 > m$ kann ohnehin keine weitere Phase gestartet werden, da die Selbstheilung nach spätestens m Phasen beendet sein muss).

ii) PE_i mit $i > \lfloor (m-1)/p \rfloor$ (also $i \geq \lfloor (m-1)/p \rfloor + 1$) stoppt seine niedrigstpriore Task *nicht*, da deren Priorität mindestens so hoch wie $\mathrm{prio}(T_{\max})$ ist. Dies gilt gemäß Bemerkung 8.45, wenn folgende Ungleichung erfüllt ist:

$$(i-1)m + (p+1) \geq \left\lfloor \frac{m-1}{p} \right\rfloor m + l$$

Die linke Seite wird minimal für $i = \lfloor (m-1)/p \rfloor + 1$, die rechte Seite maximal für $l = p$:

$$\begin{aligned}
&\Updownarrow \\
\left(\left\lfloor \frac{m-1}{p} \right\rfloor + 1 - 1 \right) m + (p+1) &\geq \left\lfloor \frac{m-1}{p} \right\rfloor m + p
\end{aligned}$$

$$\Updownarrow$$
$$1 \geq 0$$

Diese Aussage ist wahr und somit stoppt kein PE_i mit $i > \lfloor (m-1)/p \rfloor$ in Phase $p+1$ eine Task.

Folglich sind an Phase $p+1 \leq m$ genau $\lfloor (m-1)/p \rfloor$ PEs beteiligt, es werden also ebenso viele Tasks gestoppt, womit der erste Teil der Induktionsbehauptung bewiesen ist.

Damit muss nun nur noch gezeigt werden, dass am Ende von Phase $p+1$ die nicht zugeteilten Tasks dem Schema des zweiten Teils der Induktionsbehauptung entsprechen. Sortiert man die gerade gestoppten Tasks in das Schema der am Ende von Phase p nicht zugeteilten Tasks ein, wie es durch die Induktionsannahme gegeben ist, so ergibt sich für ein $l \in \{1, \ldots, p\}$ folgendes Schema:

In Phase $p+1$ gestoppt ($\lfloor (m-1)/p \rfloor$ Tasks)	Bereits zuvor gestoppt (m Tasks)	
Differenz zwischen je zwei aufeinanderfolgenden Prioritäten: $\Delta = 1$	$\left\lfloor \frac{m-1}{p} \right\rfloor m + l$ $\vdots$ $\left\lfloor \frac{m-1}{p} \right\rfloor m + 1$	l Tasks
$\left(\left\lfloor \frac{m-1}{p} \right\rfloor - 1\right) m + (p+1)$ Differenz zwischen je zwei aufeinanderfolgenden Prioritäten: $\Delta = 1$	$\left(\left\lfloor \frac{m-1}{p} \right\rfloor - 1\right) m + p$ $\vdots$ $\left(\left\lfloor \frac{m-1}{p} \right\rfloor - 1\right) m + 1$	$p+1$ Tasks
$\vdots$	$\vdots$	
$0m + (p+1)$ Differenz zwischen je zwei aufeinanderfolgenden Prioritäten: $\Delta = 1$	$0m + p$ $\vdots$ $0m + 1$	$p+1$ Tasks

Da in dieser Phase $\lfloor (m-1)/p \rfloor$ Tasks gestoppt wurden, werden nun die $\lfloor (m-1)/p \rfloor$ höchstprioren aller $m + \lfloor (m-1)/p \rfloor$ nicht zugeteilten Tasks zugeteilt. Im Anschluss wird eine Task der Form $km + l'$ mit $l' \in \{1, \ldots, p+1\}$ die höchstpriore nicht zugeteilte Task sein. Dies ist im obigen Schema jedoch gerade die $(\lfloor (m-1)/p \rfloor + 1)$-te Task von oben bzw. die m-te Task von unten. Von unten beginnend startet k bei 0 und wird nach je $p+1$ Tasks inkrementiert, folglich ergibt sich k für die höchstpriore nicht zugeteilte Task gerade zu $\lfloor (m-1)/(p+1) \rfloor$.

Die am Ende von Phase $p+1$ (und somit am Anfang von Phase $p+2$) höchstpriore nicht zugeteilte Task $T_{\max}$ hat daher die Priorität $\left\lfloor \frac{m-1}{p+1} \right\rfloor m + l'$ für ein $l' \in \{1, \ldots, p+1\}$, die Prioritäten der am Ende von Phase $p+1$ nicht zugeteilten Tasks sind also nach folgendem Schema gegeben:

$$\left.\begin{array}{r} \left\lfloor \frac{m-1}{p+1} \right\rfloor m + \quad l' \\ \vdots \\ \left\lfloor \frac{m-1}{p+1} \right\rfloor m + \quad 1 \\ \left(\left\lfloor \frac{m-1}{p+1} \right\rfloor - 1 \right) m + p + 1 \\ \vdots \\ \left(\left\lfloor \frac{m-1}{p+1} \right\rfloor - 1 \right) m + \quad 1 \\ \\ \vdots \\ \\ 0m + p + 1 \\ \vdots \\ 0m + \quad 1 \end{array}\right\} m \text{ Tasks}$$

Dieses Schema ist jedoch gerade das von der Induktionsbehauptung postulierte Schema, womit nun beide Teile der Induktionsbehauptung gezeigt sind. ◊

Folglich gilt für alle $2 \leq p \leq m$, dass an Phase p genau $\left\lfloor \frac{m-1}{p-1} \right\rfloor$ PEs beteiligt sind und je eine Task stoppen. Zusammen mit der zuvor bewiesenen Tatsache, dass an Phase 1 genau v PEs beteiligt sind, folgt daher, dass im betrachteten Szenario genau

$$v + \sum_{i=2}^{m} \left\lfloor \frac{m-1}{p-1} \right\rfloor$$

Tasks gestoppt werden und daher gemäß Lemma 8.17 sowie Satz 8.18 in Verbindung mit Bemerkung 8.19 genau

$$2 \cdot \left(v + \sum_{i=2}^{m} \left\lfloor \frac{m-1}{p-1} \right\rfloor \right) + a$$

Hormonzyklen für die Selbstheilung benötigt werden.

Somit ist die Zeitschranke von $2 \cdot \sum_{i=1}^{v} \lceil m/i \rceil + a$ Hormonzyklen die kleinste obere Schranke für die Selbstheilung in Überlastsituationen, falls das Eager Task Dropping verwendet wird und $v \geq m$ gilt. □

Für den Fall $v < m$ trifft Satz 8.42 keine Aussage, weshalb unklar ist, ob die Zeitschranke auch in diesem Fall die *kleinste* obere Schranke darstellt. Diese Vermutung liegt jedoch nahe:

Satz 8.46. Die Zeitschranke von $2 \cdot \sum_{i=1}^{v} \lceil \frac{m}{i} \rceil + a$ Hormonzyklen aus Satz 8.37 ist die kleinste obere Schranke für die Selbstheilung in Überlastsituationen, falls $1 \leq m \leq 1000$ und $1 \leq v \leq 1000$.

Beweis. Diese Aussage wurde für die angegebenen Werte von m und v durch Simulationen validiert, indem für jedes Paar (m, v) die Anzahl der gestoppten Tasks für dieselbe initiale Taskverteilung wie im Beweis von Satz 8.42 bestimmt wurde. Es ergab sich, dass für jedes dieser Paare exakt $\sum_{i=1}^{v} \lceil m/i \rceil$ Tasks gestoppt werden. Mit Lemma 8.17 und Satz 8.18 folgt daher die Behauptung. □

Bemerkung 8.47. Laut Satz 8.42 ist die betrachtete Zeitschranke für $v \geq m$ also die kleinste obere Schranke, laut Satz 8.46 gilt dies zumindest für bestimmte Werte von m und v auch, falls $v < m$.

Auch wenn diese Ergebnisse darauf hindeuten, dass die Behauptung für *alle* positiven Werte von v und m zu gelten scheint, erscheint ein allgemeiner Beweis der Behauptung für $v < m$ dennoch komplexer als für den Fall $v \geq m$, da sich die Formalisierung der auf jedem PE zugeteilten Tasks hier schwieriger gestaltet.

Daher wird im Rahmen dieser Dissertation auf eine weitere Betrachtung dieses Falls verzichtet. Die praktische Relevanz einer solchen Analyse wäre jedoch ohnehin gering: Wie bereits in Kapitel 7.2.3 beschrieben, unterstützt die AHS-Middleware in ihrer aktuellen Implementierung höchstens 255 Tasks, lediglich der Hormonsimulator unterstützt größere Skalierungen. Zudem gibt [Lun19] für ein Szenario aus 250 PEs, die *insgesamt* 250 Tasks verteilen, ein Datenaufkommen von 3.9 MByte/s an – in diesem Wert sind jedoch nur Hormone und keinerlei Nutzdaten enthalten. Mit fast 32 Mbit/s, die allein für den Hormonaustausch benötigt werden, ist dieses Szenario daher beispielsweise schon nicht mehr für ein 10-Mbit/s-Ethernetnetzwerk geeignet.

In einer Überlastsituation mit $m = 250$ Tasks pro PE sowie insgesamt $v+1 = 250$ PEs (inklusive dem ausfallenden $\text{PE}_\times$) existieren dagegen sogar $m \cdot (v+1) = 62500$ Tasks. Ein solches Szenario ist erst recht nicht mehr praxistauglich, zumal die Kommunikationslast mit dem *Produkt* aus den Anzahlen von PEs und Tasks skaliert [Lun19].

Folglich sind mit den simulierten Werten von $1 \leq m \leq 1000$ und $1 \leq v \leq 1000$ nicht nur bereits alle praxistauglichen Skalierungen, sondern auch eine noch viel größere Zahl praktisch irrelevanter Szenarien abgedeckt.

8.6.3.5. Asymptotische Abschätzungen

Nachdem nun auch eine phasenweise Sichtweise auf die Dauer der Selbstheilung entwickelt wurde, die insbesondere dazu genutzt werden konnte, zu zeigen, dass die hergeleitete Zeitschranke unter bestimmten Voraussetzungen auch eine kleinste obere Schranke darstellt, wollen wir uns nun einigen asymptotischen Abschätzungen zuwenden. In Abschnitt 8.6.3.2 wurde für variables m bei fixiertem v bereits ein näherungsweise

linearer Zusammenhang vermutet, die folgenden asymptotischen Betrachtungen werden diese Vermutung bestätigen.

Hierzu betrachten wir zunächst eine asymptotisch *obere* Schranke:

Satz 8.48. Die Dauer der Selbstheilung in Überlastsituationen lässt sich bei Verwendung von Eager Task Dropping durch

$$\text{WCET} \leq a + 2 \cdot \begin{cases} v + m - 1, & v = 1 \vee m \in \{1, 2\} \\ v + m - 1 + (m-1) \cdot \ln\left(\min\{v,\, m-1\}\right), & \text{sonst} \end{cases}$$

Hormonzyklen abschätzen.

Beweis. Nach Satz 8.37 gilt

$$\frac{1}{2} \cdot (\text{WCET} - a) \leq \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil,$$

die rechte Seite ist gemäß Gleichung (8.5) äquivalent zu

$$\begin{aligned} &= v + \sum_{i=2}^{v+1} \left\lfloor \frac{m-1}{i-1} \right\rfloor \\ &= v + \sum_{i=1}^{v} \left\lfloor \frac{m-1}{i} \right\rfloor. \end{aligned}$$

Für $i > m-1$ gilt jedoch $\lfloor (m-1)/i \rfloor = 0$, die obere Summationsgrenze kann daher beschränkt werden:

$$= v + \sum_{i=1}^{\min\{v, m-1\}} \left\lfloor \frac{m-1}{i} \right\rfloor \tag{8.6}$$

$$\leq v + \sum_{i=1}^{\min\{v, m-1\}} \frac{m-1}{i}$$

$$= v + (m-1) \cdot \sum_{i=1}^{\min\{v, m-1\}} i^{-1}. \tag{8.7}$$

Es bezeichne

Harmonische Zahl, H_k

$$H_k \coloneqq \sum_{i=1}^{k} i^{-1}$$

die k-te *harmonische Zahl*. Dann kann Ungleichung (8.7) wie folgt umgeschrieben werden:

$$\frac{1}{2} \cdot (\text{WCET} - a) \leq v + (m-1) \cdot H_{\min\{v, m-1\}}.$$

Für $k > 1$ existiert die Abschätzung $H_k < \ln k + 1$ [GKP94, 6.60]; für $\min\{v, m-1\} \geq 2$ folgt daher

$$\begin{aligned} &< v + (m-1) \cdot (\ln(\min\{v,\, m-1\}) + 1) \\ &= v + m - 1 + (m-1) \cdot \ln(\min\{v,\, m-1\})\,. \end{aligned} \tag{8.8}$$

Gilt dagegen $\min\{v, m-1\} \in \{0, 1\}$, so kann die Abschätzung von H_k durch den Logarithmus nicht angewendet werden, in diesem Fall lässt sich jedoch die Summe aus Ungleichung (8.7) auflösen:

$$\frac{1}{2} \cdot (\mathrm{WCET} - a) \leq \begin{cases} v, & \min\{v,\, m-1\} = 0 \\ v + m - 1, & \min\{v,\, m-1\} = 1 \end{cases}$$

Da für $v = 0$ keine Selbstheilung möglich ist, kann wie bisher $v \geq 1$ gefordert werden. Damit lässt sich die Fallunterscheidung umschreiben zu

$$\begin{aligned} \frac{1}{2} \cdot (\mathrm{WCET} - a) &\leq \begin{cases} v, & m = 1 \\ v + m - 1, & m = 2 \vee (m > 1 \wedge v = 1) \end{cases} \\ &= v + m - 1, \text{ falls } v = 1 \vee m \in \{1, 2\}. \end{aligned}$$

Mit Ungleichung (8.8) für den Fall $\min\{v, m-1\} \geq 2$ folgt schließlich

$$\mathrm{WCET} \leq a + 2 \cdot \begin{cases} v + m - 1, & v = 1 \vee m \in \{1, 2\} \\ v + m - 1 + (m-1) \cdot \ln(\min\{v,\, m-1\})\,, & \text{sonst,} \end{cases}$$

was zu zeigen war. □

Im Folgenden ist jedoch auch eine asymptotisch *untere* Schranke der *maximalen* Selbstheilungsdauer interessant:

Satz 8.49. Unter der Annahme, dass die Worst-Case-Selbstheilungsdauer in einer Überlastsituation bei Verwendung des Eager Task Dropping für alle $m \geq 1$ und $v \geq 1$ *genau* $2 \cdot \sum_{i=1}^{v} \lceil m/i \rceil + a$ Hormonzyklen beträgt, gilt

$$\mathrm{WCET} \geq a + 2 \cdot \begin{cases} v + m - 1, & v = 1 \vee m \in \{1, 2\} \\ v - \min\{v, m-1\} + m \cdot \ln(\min\{v, m-1\}), & \text{sonst.} \end{cases}$$

Beweis. Nach Gleichung (8.6) gilt

$$\begin{aligned} \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil &= v + \sum_{i=1}^{\min\{v, m-1\}} \left\lfloor \frac{m-1}{i} \right\rfloor \\ &= v + \sum_{i=1}^{\min\{v, m-1\}} \left(\left\lceil \frac{m}{i} \right\rceil - 1 \right) \end{aligned}$$

$$
\begin{aligned}
&= v - \min\{v, m-1\} + \sum_{i=1}^{\min\{v,m-1\}} \left\lceil \frac{m}{i} \right\rceil \\
&\geq v - \min\{v, m-1\} + \sum_{i=1}^{\min\{v,m-1\}} \frac{m}{i} \\
&= v - \min\{v, m-1\} + m \cdot H_{\min\{v,m-1\}}. \qquad (8.9)
\end{aligned}
$$

Für $k > 1$ existiert die Abschätzung $\ln k < H_k$ [GKP94, 6.60], für $\min\{v, m-1\} \geq 2$ folgt daher

$$
\sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil > v - \min\{v, m-1\} + m \cdot \ln(\min\{v, m-1\}).
$$

Für $\min\{v, m-1\} \in \{0, 1\}$ kann dagegen Ungleichung (8.9) vereinfacht werden:

$$
\sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil \geq \begin{cases} v, & \min\{v, m-1\} = 0 \\ v + m - 1, & \min\{v, m-1\} = 1. \end{cases}
$$

Wie bisher kann $v \geq 1$ angenommen werden und es ergibt sich

$$
= v + m - 1 \text{ für } v = 1 \vee m \in \{1, 2\}.
$$

Werden beide Fälle kombiniert, so ergibt sich

$$
\sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil \geq \begin{cases} v + m - 1, & v = 1 \vee m \in \{1, 2\} \\ v - \min\{v, m-1\} + m \cdot \ln(\min\{v, m-1\}), & \text{sonst} \end{cases}
$$

und daraus folgt die Behauptung. □

Bemerkung 8.50. Satz 8.49 setzt voraus, dass die WCET exakt $2 \cdot \sum_{i=1}^{v} \lceil m/i \rceil + a$ beträgt, dieser Term also insbesondere auch eine untere Schranke der WCET ist. Satz 8.42 besagt lediglich für $v \geq m$, dass dies der Fall ist, während Satz 8.46 diese Bedingung zudem für alle $1 \leq m \leq 1000$ sowie $1 \leq v \leq 1000$ bestätigt.

Für alle übrigen Werte von m und v ist jedoch unklar, ob der gegebene Term die WCET nach unten hin beschränkt, auch wenn diese Vermutung nahe liegt.

Da die in dieser Dissertation betrachteten Werte von m und v jedoch alle von den genannten Sätzen abgedeckt sind, wird im Folgenden dennoch davon ausgegangen, dass Satz 8.49 stets gilt.

Grafische Betrachtung Die asymptotischen Abschätzungen erlauben es nun, den bereits aus den Abbildungen 8.6 und 8.7 bekannten Kurvenverlauf genauer zu klassifizieren.

Fixierte Anzahl Tasks pro PE Hierzu zeigt Abbildung 8.12 die WCET für variables v bei $m = 10$ sowie $m = 40$, zusätzlich sind jedoch die soeben hergeleiteten asymptotischen Schranken eingezeichnet. Für $v = 1$ reduzieren beide Schranken auf den Sonderfall $m + v - 1$ und stimmen mit dem exakten Wert überein, für alle übrigen Werte von v liefern sie eine Unter- bzw. Überabschätzung.

In Abschnitt 8.6.3.2 wurde bereits begründet, warum die Graphen der WCET für $v \geq m - 1$ einer Geraden mit Steigung 2 entsprechen. Dieser Zusammenhang wird auch durch die asymptotischen Schranken bestätigt:

Für $m \in \{1, 2\}$ ergibt sich wegen

$$a + 2 \cdot (v + m - 1) \leq \text{WCET} \leq a + 2 \cdot (v + m - 1)$$

lineares Verhalten in v bei fixiertem m, unabhängig davon, ob $v \geq m - 1$ gilt.

Für ein fixiertes $m \geq 3$ bei variablem $v \geq m - 1$ gilt dagegen $\min\{v, m - 1\} = m - 1$, also kann insbesondere $\ln(\min\{v, m - 1\})$ als Konstante c betrachtet werden. Damit ergibt sich[1]

- für die obere Schranke:

$$\begin{aligned} \text{WCET} &< a + 2 \cdot (v + m - 1 + (m - 1) \cdot \ln(\min\{v, m - 1\})) \\ &= 2v + 2(c + 1) \cdot m - 2(c + 1) + a. \end{aligned}$$

- für die untere Schranke:

$$\begin{aligned} \text{WCET} &> a + 2 \cdot (v - \min\{v, m - 1\} + m \cdot \ln(\min\{v, m - 1\})) \\ &= 2v + 2(c - 1) \cdot m + 2 + a. \end{aligned}$$

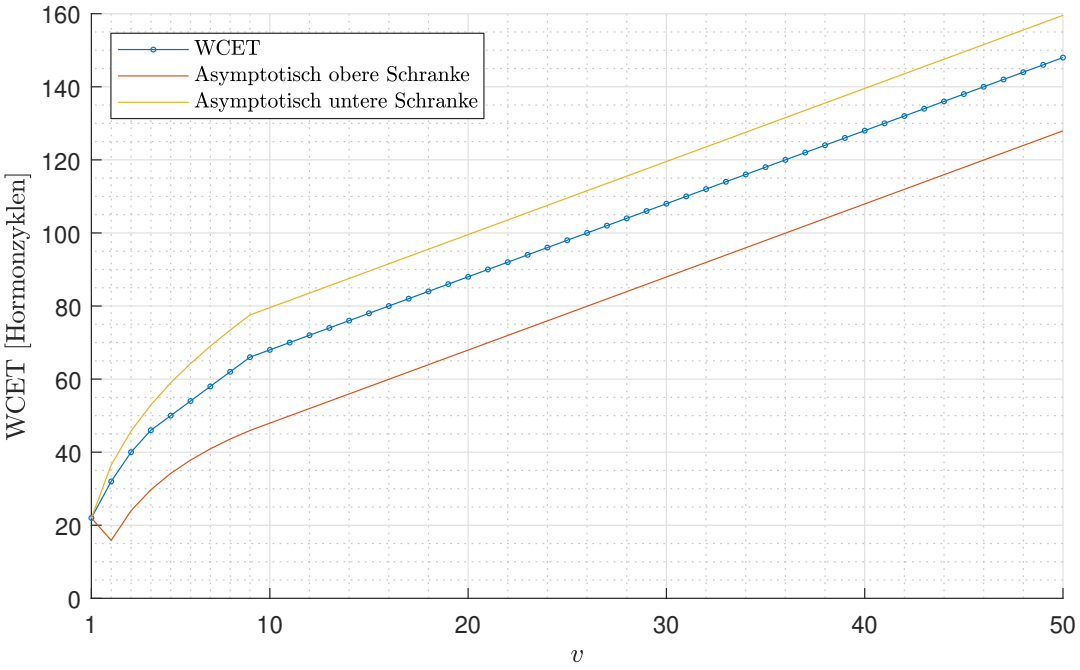
Unter den obigen Voraussetzungen ist die WCET also nach oben und unten hin jeweils durch eine in v lineare Funktion beschränkt; der Übergang zum für $v \geq m - 1$ linearen Kurvenverlauf ist in den Abbildungen 8.12a und 8.12b insbesondere für die asymptotisch oberen Schranken gut erkennbar.

Fixierte Anzahl verbleibender PEs Ähnliche Betrachtungen lassen sich für fixiertes v durchführen. Hierzu zeigt Abbildung 8.13 die WCET für variables m bei $v = 5$ sowie $v = 50$ samt der zugehörigen asymptotischen Schranken. Beide Schranken stimmen für die Sonderfälle $m = 1$ und $m = 2$ jeweils mit dem exakten Wert überein, für größere m liefern sie wie zuvor eine Unter- bzw. Überabschätzung.

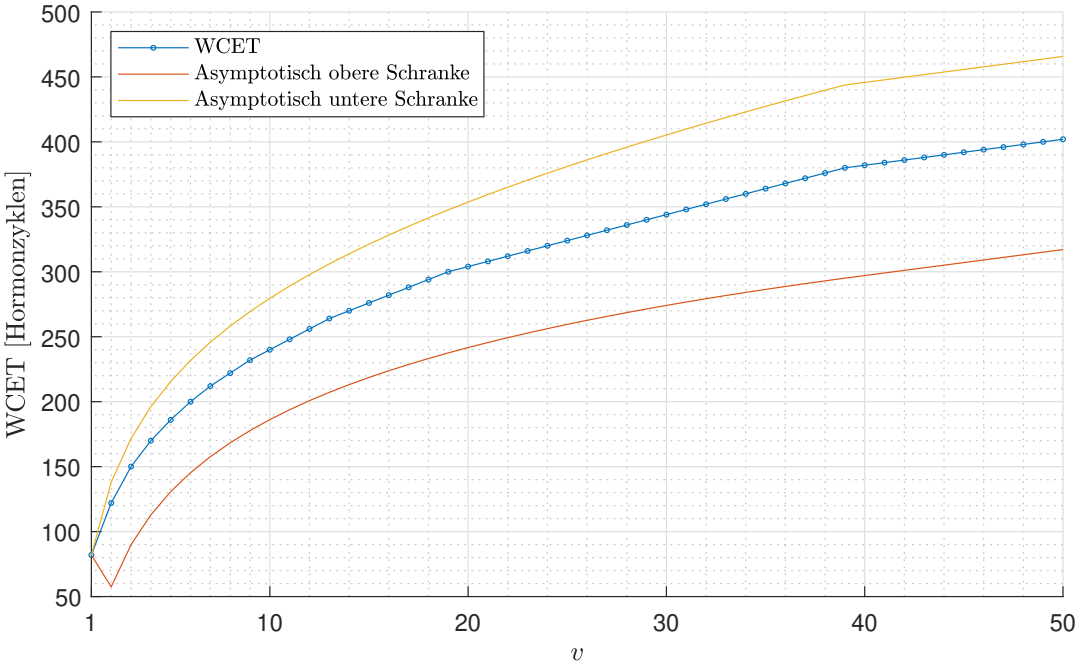
Die beiden Schranken zeigen nun, warum der aus Abbildung 8.7a bekannte Kurvenverlauf für kleine v einen (angenähert) linearen Zusammenhang vermuten ließ: Zumindest für $v = 1$ ergibt sich gemäß der Sonderfälle der asymptotischen Schranken

$$a + 2 \cdot (v + m - 1) \leq \text{WCET} \leq a + 2 \cdot (v + m - 1)$$

[1] Aus den Beweisen der asymptotischen Schranken wird deutlich, dass die Relationen „<“ bzw. „>“ gelten, wenn der Sonderfall $v = 1 \vee m \in \{1, 2\}$ *nicht* zutrifft.

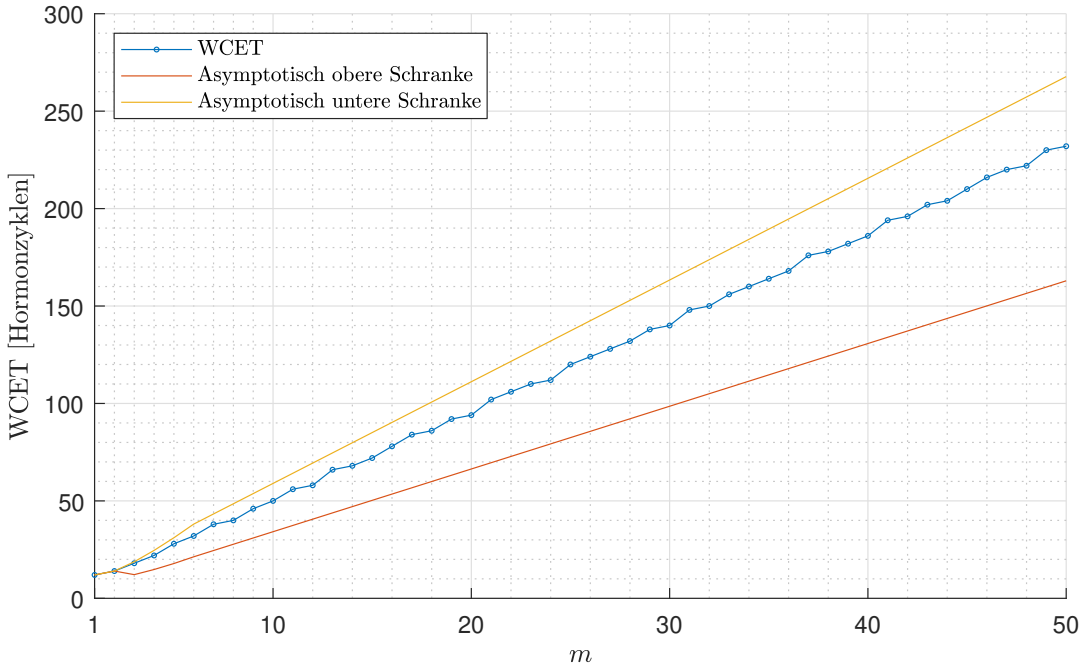


(a) WCET für $m = 10$

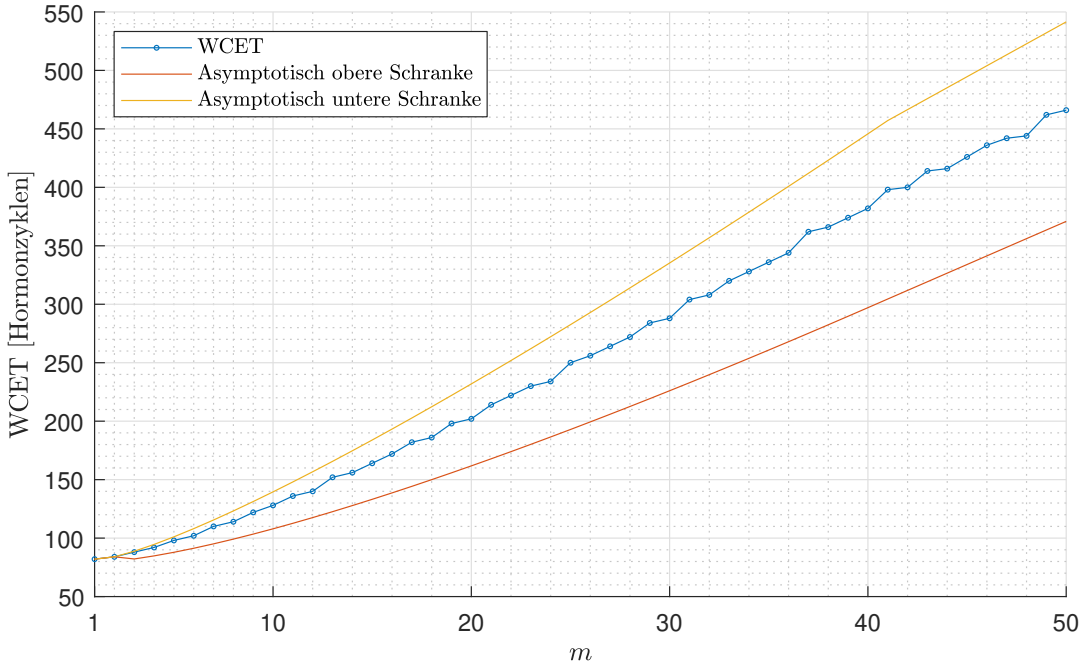


(b) WCET für $m = 40$

Abbildung 8.12: WCET der Eager-Task-Dropping-Strategie für fixiertes m inklusive asymptotischer Schranken



(a) WCET für $v = 5$



(b) WCET für $v = 40$

Abbildung 8.13: WCET der Eager-Task-Dropping-Strategie für fixiertes v inklusive asymptotischer Schranken

und damit lineares Verhalten in m bei fixiertem v.

Für ein fixiertes $v > 1$ bei variablem $m \geq v + 1$ gilt dagegen $\min\{v, m-1\} = v$. In diesem Fall kann insbesondere $\ln(\min\{v, m-1\})$ als Konstante c betrachtet werden. Damit ergibt sich[1]

- für die obere Schranke:

$$\begin{aligned}\text{WCET} &< a + 2 \cdot (v + m - 1 + (m-1) \cdot \ln(\min\{v, m-1\})) \\ &= 2(c+1) \cdot m + 2 \cdot (v - c - 1) + a.\end{aligned}$$

- für die untere Schranke:

$$\begin{aligned}\text{WCET} &> a + 2 \cdot (v - \min\{v, m-1\} + m \cdot \ln(\min\{v, m-1\})) \\ &= 2c \cdot m + a.\end{aligned}$$

Die WCET ist unter diesen Voraussetzungen also sowohl nach oben als auch nach unten hin jeweils durch eine in m lineare Funktion beschränkt; der Übergang zu einem für $m > v$ linearen Verlauf der beiden asymptotischen Schranken ist in den Abbildungen 8.13a sowie 8.13b bei $m = 6$ bzw. $m = 41$ vor allem für die asymptotisch oberen Schranken gut zu erkennen.

Zum Abschluss der Worst-Case-Analyse des Eager Task Dropping soll im nächsten Abschnitt noch eine Verallgemeinerung der Zeitschranke betrachtet werden.

8.6.4. Verallgemeinerung der Zeitschranke

Für viele Anwendungsszenarien wird die Annahme, dass *alle* Tasks verschiedene Prioritäten haben, zu pessimistisch sein: In praktischen Szenarien ist zu erwarten, dass mehrere Tasks dieselbe Priorität aufweisen, insgesamt also nur wenige verschiedene Prioritätslevel existieren. Daher soll nun eine Verallgemeinerung der Zeitschranke hergeleitet werden, falls die Anzahl der Prioritätslevel klein ist:

Satz 8.51. Es bezeichne $|P|$ die Anzahl der verschiedenen Prioritäten. Dann benötigt die Selbstheilung in Überlastsituationen bei Verwendung des Eager Task Dropping höchstens

$$\sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \min\{v, |P|\} + \sum_{i=2}^{m} \min\left\{ v, \left\lfloor \frac{m-1}{i-1} \right\rfloor, |P| \right\} + a \text{ Hormonzyklen.}$$

Beweis. Die bisherigen Beweise für Zeitschranken nahmen an, dass alle Tasks verschiedene Prioritäten haben. In diesem Fall dauert eine Selbstheilungsphase, in der k Tasks gestoppt werden, genau $2k$ Hormonzyklen: Ein Hormonzyklus zum Stopp von k Tasks sowie $2k - 1$ Hormonzyklen zur Zuteilung von k Tasks.

[1] Erneut gelten hier die Relationen „<“ bzw. „>“, da der Sonderfall $v = 1 \vee m \in \{1, 2\}$ *nicht* zutrifft.

Die Zuteilung von Z_i Tasks in Phase i lässt sich jedoch gemäß Satz 6.9 auch durch $Z_i + |P_i| - 1$ Hormonzyklen abschätzen, wenn $|P_i|$ die Anzahl der Prioritäten dieser Z_i Tasks bezeichnet.

Mit dieser Verfeinerung dauert die i-te Selbstheilungsphase, in der Z_i Tasks gestoppt werden, höchstens $Z_i + |P_i|$ Hormonzyklen. Für Selbstheilungsphase i ist $|P_i|$ im Allgemeinen zwar nicht abschätzbar, in jedem Fall gilt jedoch $|P_i| \leq \min\{Z_i, |P|\}$: Z_i Tasks können höchstens Z_i verschiedene Prioritäten haben und die Gesamtzahl der Prioritäten ist durch $|P|$ beschränkt.

Die Summation über die Dauer aller Selbstheilungsphasen liefert dann

$$\begin{aligned}\text{WCET} &\leq \sum_{i=1}^{m} (Z_i + |P_i|) + a \\ &= \underbrace{\sum_{i=1}^{m} Z_i}_{} + \sum_{i=1}^{m} |P_i| + a.\end{aligned}$$

Der umklammerte Term gibt gerade die maximale Anzahl während der Selbstheilung gestoppter Tasks an. Gemäß Satz 8.37 beträgt diese jedoch gerade $\sum_{i=1}^{v} \lceil m/i \rceil$ und es folgt

$$\begin{aligned}&= \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \sum_{i=1}^{m} |P_i| + a \\ &= \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \underbrace{|P_1|}_{\text{Phase 1}} + \underbrace{\sum_{i=2}^{m} |P_i|}_{\text{Phasen 2 bis } m} + a \\ &\leq \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \min\{Z_1, |P|\} + \sum_{i=2}^{m} \min\{Z_i, |P|\} + a.\end{aligned}$$

Mit der aus Satz 8.38 bekannten Abschätzung für Z_i ergibt sich nun

$$\begin{aligned}&\leq \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \min\{v, |P|\} + \sum_{i=2}^{m} \min\left\{\min\left\{v, \left\lfloor \frac{m-1}{i-1} \right\rfloor\right\}, |P|\right\} + a \\ &= \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \min\{v, |P|\} + \sum_{i=2}^{m} \min\left\{v, \left\lfloor \frac{m-1}{i-1} \right\rfloor, |P|\right\} + a.\end{aligned}$$

□

Bemerkung 8.52. Für $|P| \geq v$ degeneriert die Abschätzung aus Satz 8.51 zur aus Satz 8.37 bekannten Zeitschranke von höchstens

$$2 \cdot \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + a \text{ Hormonzyklen.}$$

Beweis. Für $|P| \geq v$ kann die Abschätzung aus Satz 8.51 umgeschrieben werden:

$$\begin{aligned}
&\sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \min\{v, |P|\} + \sum_{i=2}^{m} \min\left\{v, \left\lfloor \frac{m-1}{i-1} \right\rfloor, |P|\right\} + a \\
&= \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \underbrace{v + \sum_{i=2}^{m} \min\left\{v, \left\lfloor \frac{m-1}{i-1} \right\rfloor\right\}} + a
\end{aligned}$$

Die Anwendung von Lemma 8.39 auf den umklammerten Term liefert schließlich

$$= \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + a = 2 \cdot \sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + a. \qquad \square$$

Folglich liefert Satz 8.51 nur dann eine bessere Abschätzung als Satz 8.37, wenn es weniger verschiedene Prioritäten als verbleibende PEs gibt.

8.7. Vergleich der Task-Dropping-Strategien

In den vorherigen Abschnitten wurden sowohl die naive Task-Dropping-Strategie als auch das Eager Task Dropping in Bezug auf ihre Selbstheilungsdauer im Worst-Case analysiert. Für die naive Strategie wurde zudem eine Average-Case-Analyse durchgeführt (in Anhang B), die zeigte, dass Worst- und Average Case dieser Strategie asymptotisch in der gleichen Komplexitätsklasse liegen.

Im Folgenden sollen daher beide Strategien miteinander verglichen werden. Hierbei werden jedoch nur Szenarien gemäß der Annahmen aus Abschnitt 8.2 betrachtet: Insbesondere haben alle Tasks verschiedene Prioritäten, weshalb auch keine Verallgemeinerungen zum Einsatz kommen, wie beispielsweise die Zeitschranken aus Satz 8.12 (naive Strategie) sowie Satz 8.51 (Eager Task Dropping).

8.7.1. Vergleich der Anzahl gestoppter Tasks

Zunächst bietet sich ein Vergleich der Anzahl gestoppter Tasks an. Im Folgenden bezeichnen s_{eager} und s_{naiv} die Anzahl der gestoppten Tasks für das Eager Task Dropping bzw. die naive Strategie.

Zunächst ist eine allgemeine Aussage hervorzuheben:

Satz 8.53. Für identische Parameter m und v wird Eager Task Dropping niemals mehr Tasks stoppen als die naive Task-Dropping-Strategie.

Beweis. Durch Widerspruch.

In einer Überlastsituation stoppt die naive Task-Dropping-Strategie gerade *alle* Tasks, die eine geringere Priorität haben als jene der höchstprioren nicht zugeteilten Task $T_{\max}$.

Angenommen, das Eager Task Dropping würde in einem solchen Szenario $s_{\text{eager}} > s_{\text{naiv}}$ Tasks stoppen. Da $s_{\text{eager}} > s_{\text{naiv}}$, muss dann entweder

a) mindestens eine Task T mit $\text{prio}(T) \geq \text{prio}(T_{\max})$ gestoppt worden sein oder

b) eine Task mehrfach gestoppt worden sein.

Fall a) widerspricht aber der Funktionsweise des Eager Task Dropping (Strategie 8.14). Für Fall b) dagegen müsste eine Task mindestens einmal zugeteilt worden sein, um ein zweites Mal gestoppt werden zu können. Gemäß Lemma 8.8 kann eine im Rahmen der Selbstheilung zugeteilte Task aber während derselben Selbstheilung nicht noch einmal gestoppt werden. □

Bezogen auf die Anzahl gestoppter Tasks ist das Eager Task Dropping also niemals eine schlechtere Wahl als die naive Strategie.

Es ist zudem leicht einzusehen, dass das Eager Task Dropping bezogen auf die Anzahl gestoppter Tasks in zwei Fällen degeneriert und mit der naiven Strategie identisch ist:

- Für $v = 1$ stoppen beide Strategien gerade *alle* $k \leq mv = m$ Tasks, deren Priorität geringer als jene von $T_{\max}$ ist.

- Für $m = 1$ kann dagegen maximal eine einzige Selbstheilungsphase stattfinden. Diese hat jedoch gerade den Effekt, dass jedes PE *alle* seine Tasks stoppt, deren Priorität geringer als jene von $T_{\max}$ ist. Das entspricht aber gerade der naiven Task-Dropping-Strategie, folglich verhalten sich beide Strategien für $m = 1$ vollkommen identisch.

Vergleich des jeweiligen Worst Case Abbildung 8.14a stellt nun den Quotienten

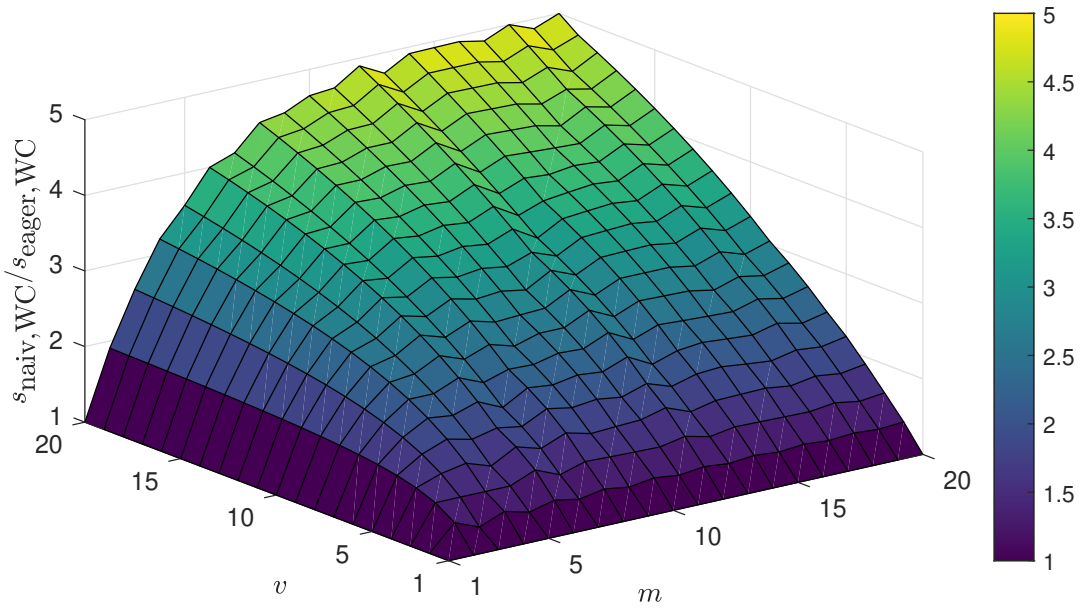
$$\frac{s_{\text{naiv,WC}}}{s_{\text{eager,WC}}} = \frac{mv}{\sum_{i=1}^{v} \lceil m/i \rceil}$$

der im Worst Case der beiden Strategien gestoppten Tasks für $1 \leq m \leq 20$ und $1 \leq v \leq 20$ grafisch dar. Da Eager Task Dropping gemäß Satz 8.53 in dieser Hinsicht nie schlechter als die naive Strategie sein kann, überrascht es nicht, dass dieser Quotient stets größer oder gleich Eins ist; die Gleichheit gilt zudem nur für die gerade betrachteten Sonderfälle $v = 1$ sowie $m = 1$, in denen sich beide Strategien – bezogen auf die Anzahl gestoppter Tasks – identisch verhalten. Dies wird deutlich, wenn der Quotient zu

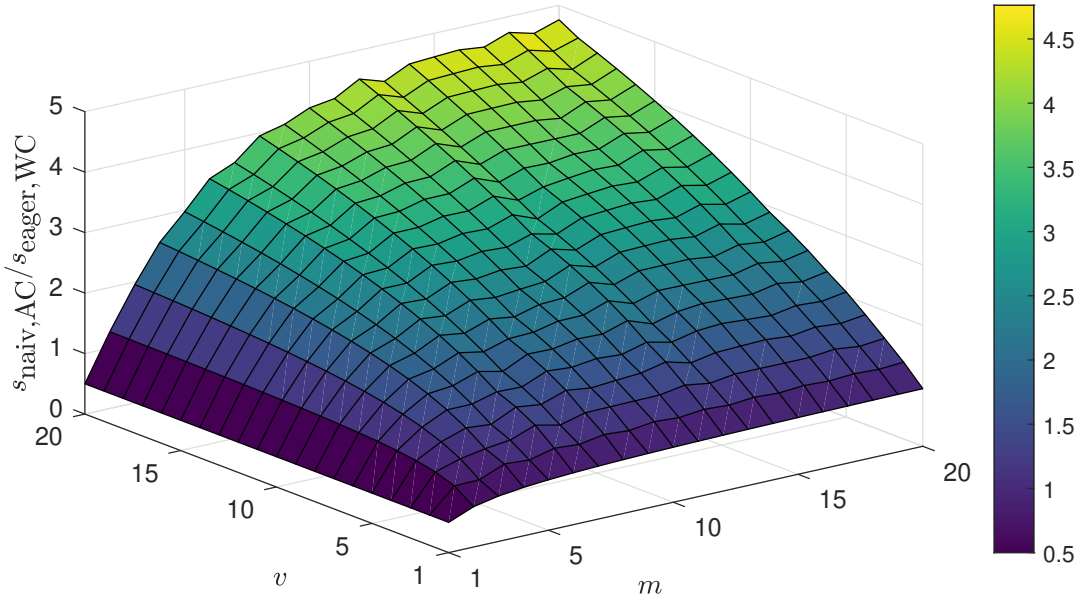
$$\frac{\sum_{i=1}^{v} m}{\sum_{i=1}^{v} \lceil m/i \rceil}$$

umgeformt wird: Sind sowohl m als auch v größer als Eins, so ist mindestens der Summand $\lceil m/v \rceil$ im Nenner *kleiner* als m, während alle Summanden des Zählers exakt m betragen, der resultierende Quotient ist also größer als Eins.

In der Abbildung ist zudem deutlich die Überlegenheit des Eager Task Dropping zu erkennen: Je größer m und v sind, desto weniger Tasks stoppt Eager Task Dropping im Vergleich zur naiven Strategie.



(a) Worst Case der naiven Strategie sowie Worst Case des Eager Task Dropping



(b) Average Case der naiven Strategie sowie Worst Case des Eager Task Dropping

Abbildung 8.14: Quotient der gestoppten Tasks für die naive Task-Dropping-Strategie sowie das Eager Task Dropping

Vergleich von Worst- und Average Case Angesichts der Tatsache, dass der Average Case der naiven Strategie asymptotisch in der gleichen Komplexitätsklasse wie ihr Worst Case liegt, überrascht es nicht, dass auch der Quotient

$$\frac{s_{\text{naiv,AC}}}{s_{\text{eager,WC}}} = \frac{m^2 v}{(1+m) \cdot \sum_{i=1}^{v} \lceil m/i \rceil}$$

aus Average Case der naiven Strategie und Worst Case des Eager Task Dropping, wie er in Abbildung 8.14b dargestellt ist, einen ähnlichen Verlauf aufweist. Lediglich für sehr kleine Werte von m und v liegt der Quotient unter Eins.[1]

In Hinblick auf Echtzeitanwendungen, für die lediglich der Worst Case von Relevanz ist, legen diese Untersuchungen nahe, dass das Eager Task Dropping der naiven Strategie stets vorzuziehen ist, wenn die *Anzahl* der gestoppten Tasks betrachtet wird. Insbesondere ist der *Worst* Case des Eager Task Dropping in den meisten Fällen sogar dem *Average* Case der naiven Strategie überlegen.

Im nächsten Abschnitt werden wir die resultierende Dauer der Selbstheilung betrachten. Dabei wird sich herausstellen, dass ein Sonderfall existiert, in dem die naive Strategie dem Eager Task Dropping dennoch überlegen ist.

8.7.2. Vergleich der Selbstheilungsdauer

Im Folgenden werden die Worst-Case-Selbstheilungdauern der beiden Strategien mit $\text{WCET}_{\text{eager}}$ sowie $\text{WCET}_{\text{naiv}}$ bezeichnet. Für den Average Case der naiven Strategie konnten nur untere und obere Schranken hergeleitet werden, die mit $\text{ACET}_{\text{naiv}}^{\text{min}}$ und $\text{ACET}_{\text{naiv}}^{\text{max}}$ bezeichnet werden.

Abbildung 8.15a stellt nun den Quotienten

$$\frac{\text{WCET}_{\text{naiv}}}{\text{WCET}_{\text{eager}}} = \frac{mv + (mv-1) \cdot [v > 1] + 1 + a}{2 \cdot \sum_{i=1}^{v} \lceil m/i \rceil + a}$$

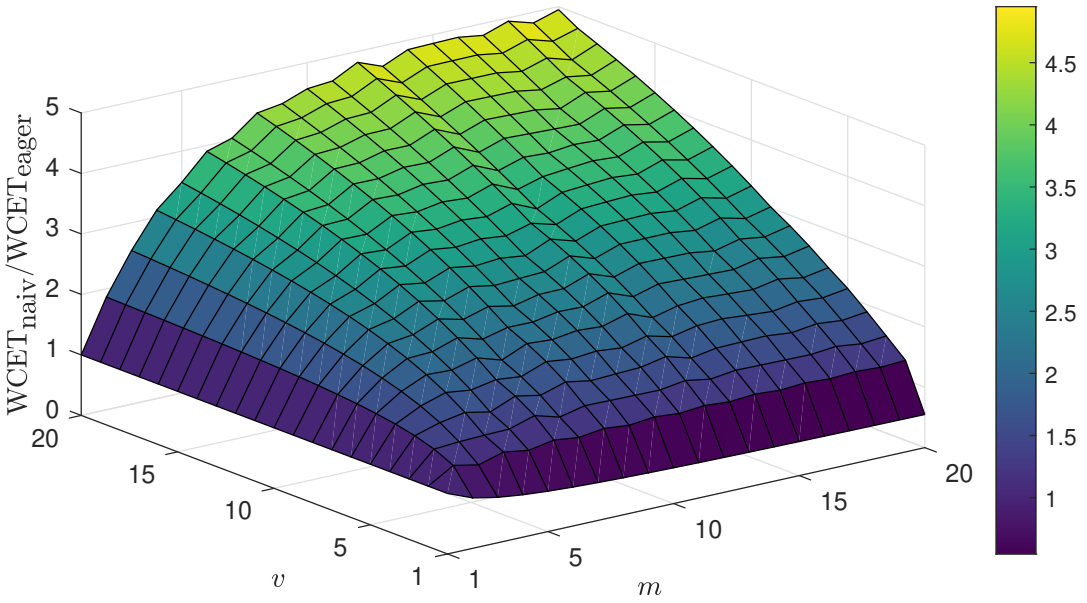
dar. Es lässt sich erahnen, dass der Wert dieses Quotienten für $v > 1$ mindestens Eins zu betragen scheint. Dies lässt sich durch eine genauere Betrachtung des Quotienten bestätigen, der sich für den Fall $v > 1$ zu

$$\frac{2mv + a}{2 \cdot \sum_{i=1}^{v} \lceil m/i \rceil + a}$$

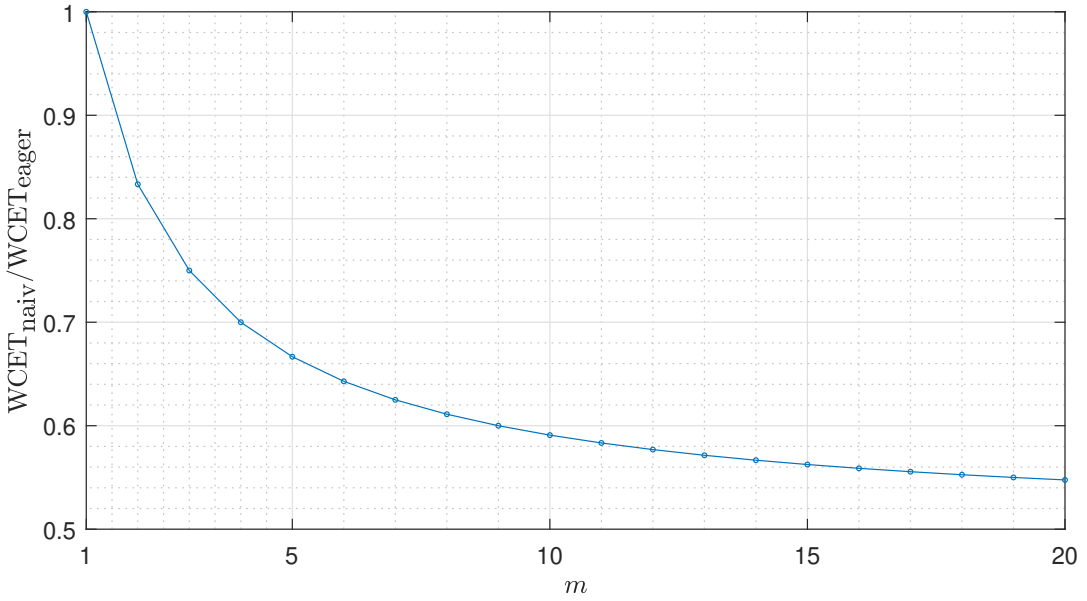
vereinfacht. Für $m = 1$ sind Zähler und Nenner identisch und der Quotient daher gleich Eins.

Für $m \geq 2$ und $v \geq 2$ kann leicht eingesehen werden, dass der Quotient größer als Eins ist: Wie bereits zuvor beim Vergleich der Anzahl gestoppter Tasks demonstriert

[1] Konkret beträgt der Quotient für $m = 1$ stets genau $\frac{1}{2}$. Ebenso liegt der Quotient für $v = 1$ sowie die beiden Paare (m, v) mit $(2, 2)$ sowie $(3, 2)$ unter Eins. Für $(2, 3)$ beträgt er genau Eins und in allen übrigen Fällen ist er größer als Eins.



(a) $1 \leq v \leq 20$ sowie $1 \leq m \leq 20$



(b) $v = 1$ sowie $1 \leq m \leq 20$

Abbildung 8.15: Quotient $\mathrm{WCET}_{\mathrm{naiv}}/\mathrm{WCET}_{\mathrm{eager}}$ der Selbstheilungsdauern im Worst Case

wurde, kann der Zähler des Quotienten umgeformt werden:

$$\frac{2 \cdot \sum_{i=1}^{v} m + a}{2 \cdot \sum_{i=1}^{v} \lceil m/i \rceil + a}$$

Wie bereits zuvor argumentiert wurde, ist für $m \geq 2$ und $v \geq 2$ mindestens der Summand $\lceil m/v \rceil$ im Nenner kleiner als der korrespondierende Summand m im Zähler und der Quotient daher insgesamt größer als Eins: In diesem Fall ist das Eager Task Dropping der naiven Strategie also überlegen.

Für $v = 1$ ergibt sich dagegen ein Sonderfall, in dem sich der Quotient zu

$$\frac{m + 1 + a}{2m + a}$$

berechnet. Die Werte des Quotienten sind für $v = 1$ in Abbildung 8.15b separat aufgetragen. Es wird deutlich, dass die naive Strategie hier für $m \geq 2$ dem Eager Task Dropping überlegen ist, obwohl beide Strategien dieselbe Anzahl Tasks stoppen.

Der Grund hierfür ist, dass beide Strategien sich für $v = 1$ nur bezogen auf die Anzahl der gestoppten Tasks identisch verhalten: Nach Erkennen des Ausfalls von $\mathrm{PE}_{\times}$ stoppt die naive Task-Dropping-Strategie im Worst Case in einem Zyklus m Tasks, in den folgenden m Zyklen werden dann die m höchstprioren nicht zugeteilten Tasks zugeteilt (wegen $v = 1$ treten keine Wartezyklen auf). Folglich dauert die Selbstheilung höchstens $m + 1 + a$ Hormonzyklen.

Im Gegensatz dazu stoppt das Eager Task Dropping die Tasks phasenweise. Konkret werden m Phasen benötigt, die aus jeweils einem Zyklus zum Stopp einer Task und einem Zyklus zum Zuteilen von $T_{\max}$ bestehen, was in einer Gesamtdauer von $2m + a$ Hormonzyklen resultiert.

Lediglich für $m = 1$ gilt $m + 1 + a = 2m + a$, für $m \geq 2$ ist der Quotient daher kleiner als Eins und die naive Task-Dropping-Strategie somit überlegen. Dennoch benötigt das Eager Task Dropping hier höchstens doppelt so viele Hormonzyklen zur Selbstheilung: Eine Grenzwertbetrachtung liefert

$$\lim_{m \to \infty} \frac{m + 1 + a}{2m + a} = \frac{1}{2},$$

da $a = 2$ eine Konstante ist.

8.7.2.1. Vergleich mit dem Average Case der naiven Strategie

Da der Average Case der Selbstheilungsdauer unter Verwendung der naiven Task-Dropping-Strategie gemäß Satz B.18 nur innerhalb eines Korridors zwischen unterer Schranke $\mathrm{ACET}_{\mathrm{naiv}}^{\min}$ und oberer Schranke $\mathrm{ACET}_{\mathrm{naiv}}^{\max}$ angegeben werden kann, betrachten wir im Folgenden nur den Quotienten

$$\frac{\mathrm{ACET}_{\mathrm{naiv}}^{\min}}{\mathrm{WCET}_{\mathrm{eager}}}$$

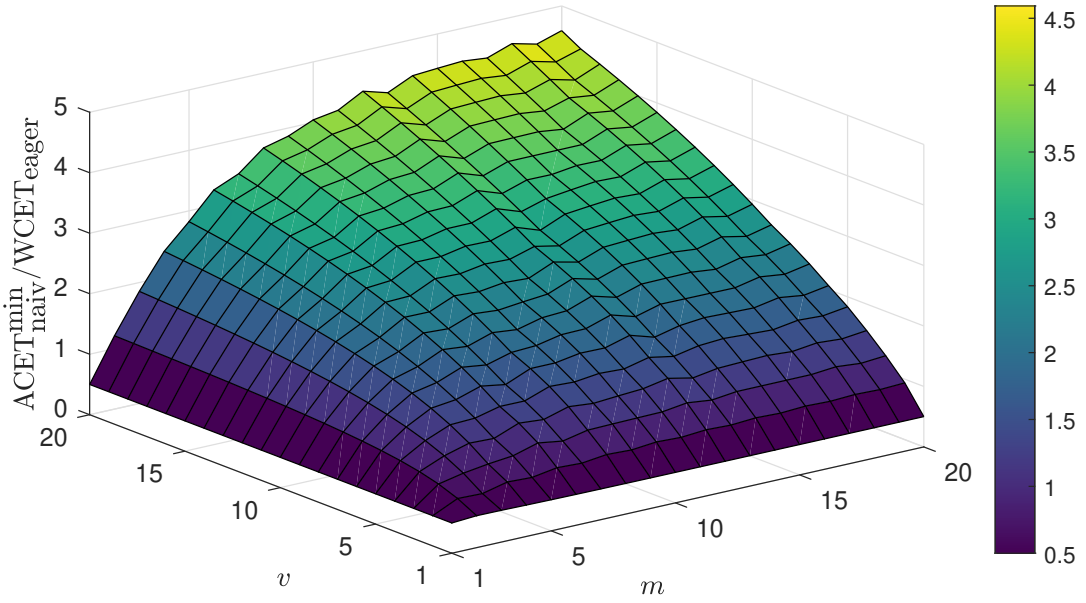


Abbildung 8.16: Quotient $\mathrm{ACET}^{\min}_{\mathrm{naiv}}/\mathrm{WCET}_{\mathrm{eager}}$ der Selbstheilungsdauern im Average Case (naive Strategie) bzw. Worst Case (Eager Task Dropping)

Tabelle 8.1: Sonderfälle mit $\mathrm{ACET}^{\min}_{\mathrm{naiv}}/\mathrm{WCET}_{\mathrm{eager}} < 1$

m	v
1	*beliebig*
2	3, 4, 5
3	3
4	3
beliebig	1, 2

als *Minimum* des möglichen Quotienten aus ACET der naiven Strategie und WCET des Eager Task Dropping. Dieser ist in Abbildung 8.16 dargestellt. Qualitativ – und größtenteils auch quantitativ – unterscheidet sich der Kurvenverlauf des nicht abgebildeten Quotienten $\mathrm{ACET}^{\max}_{\mathrm{naiv}}/\mathrm{WCET}_{\mathrm{eager}}$ jedoch nur unwesentlich.

Es ist zu erkennen, dass der Quotient in den meisten Fällen über Eins liegt. Wie zuvor existieren lediglich einige Sonderfälle für kleine Werte von m und v, in denen der Average Case der naiven Strategie besser abschneidet als der Worst Case des Eager Task Dropping. Konkret sind dies im dargestellten Bereich $1 \leq m \leq 20$ sowie $1 \leq v \leq 20$ lediglich die in Tabelle 8.1 aufgelisteten Fälle. Dabei beträgt der Wert des Quotienten für $m = 1$ genau $\frac{1}{2}$, in allen übrigen Fällen liegt er zwischen $\frac{1}{2}$ und 1.

Da $\mathrm{ACET}_{\mathrm{naiv}}$ gemäß Satz B.21 asymptotisch in der gleichen Komplexitätsklasse wie $\mathrm{WCET}_{\mathrm{naiv}}$ liegt, sind für größere Werte von m und v neben den oben aufgelisteten keine weiteren Sonderfälle zu erwarten. Als Resultat ist der *Worst* Case des Eager

Task Dropping selbst bezogen auf die Selbstheilungsdauer in den meisten Fällen dem *Average* Case der naiven Strategie überlegen, auch wenn es hier im Gegensatz zur Anzahl gestoppter Task mehr Sonderfälle zu betrachten gibt.

8.8. Zusammenfassung

In diesem Kapitel wurde die Selbstheilung in Überlastsituationen analysiert. Das bisherige AHS unterstützt keine Zuteilungsprioritäten, weshalb in damit realisierten Systemen nicht definiert werden kann, *welche* Tasks in Überlastsituationen ausgeführt werden sollen, in denen nicht mehr *alle* Tasks des Systems zugeteilt werden können. Die prioritätsbasierte Task-Decision-Strategie dagegen ermöglicht es, in Überlastsituationen selektiv Tasks niedriger Priorität zu stoppen, um Kapazitäten für die Zuteilung hochpriorer Tasks zu schaffen. Die Entscheidung, *wie viele* und *welche* Tasks gestoppt werden, um dieses Ziel zu erreichen, wird dabei von einer Task-Dropping-Strategie getroffen und basiert auf der fundamentalen Annahme 6.5. Diese fordert, dass eine Task jederzeit (temporär) gestoppt werden darf, um Kapazitäten für Tasks *höherer* Priorität zu schaffen.

Im Rahmen des in Abschnitt 8.2 vorgestellten Überlastmodells, in dem $m \cdot (v+1)$ einfach zuteilbare Tasks mit paarweise verschiedenen Prioritäten existieren und jedes der insgesamt $v+1$ PEs exakt m (beliebige) Tasks ausführen kann, wurden zwei verschiedene Task-Dropping-Strategien vorgestellt und im Detail analysiert:

Die naive Task-Dropping-Strategie stoppt bei Bemerken einer Überlastsituation auf allen PEs jeweils *alle* Tasks, deren Priorität geringer ist als jene der aktuell höchstprioren nicht zugeteilten Task $T_{\max}$.

Das Eager Task Dropping geht phasenweise vor: Es wird auf jedem PE jeweils höchstens *eine* Task gestoppt, um sicherzustellen, dass für $T_{\max}$ im nächsten Hormonzyklus ein positiver modifizierter Eignungswert gesendet werden kann. Als Resultat stoppen $k \leq v$ PEs jeweils eine Task. Im Anschluss werden die k höchstprioren Tasks nicht zugeteilten Tasks (wieder) einem PE zugeteilt. Am Ende einer solchen Phase liegt erneut eine Überlastsituation vor. Daher folgt jeweils eine weitere Phase, falls ein PE existiert, dessen niedrigstpriore Task eine geringere Priorität hat als jene der nun höchstprioren nicht zugeteilten Task $T_{\max}$.

Im Worst Case stoppt die naive Strategie gerade *alle* mv im System zugeteilten Tasks, im Average Case $m^2v/(1+m)$ Tasks. Der Average Case dieser Strategie liegt asymptotisch in der gleichen Größenordnung wie ihr Worst Case, der insbesondere linear in der Anzahl mv im System überhaupt noch zuteilbarer Tasks ist.

Für das Eager Task Dropping konnte dagegen gezeigt werden, dass höchstens $\sum_{i=1}^{v} \lceil m/i \rceil$ Tasks gestoppt werden. Für $m = 1$ oder $v = 1$ ist dieser Term jeweils mit mv identisch, in allen übrigen Fällen jedoch stets kleiner als mv. Folglich degeneriert das Eager Task Dropping bezogen auf die Anzahl gestoppter Tasks für diese beiden Sonderfälle zur naiven Task-Dropping-Strategie, der es in allen übrigen Fällen dagegen überlegen ist.

Ebenso stoppt das Eager Task Dropping im *Worst Case* in den meisten Fällen weniger Tasks als die naive Strategie im *Average Case*, nur in wenigen Fällen muss es sich dem Average Case der naiven Strategie geschlagen geben.

Ein Vergleich der *Dauer* der Selbstheilung lieferte ähnliche Ergebnisse: Bezogen auf diese Größe ist das Eager Task Dropping in den meisten Fällen deutlich besser als die navie Strategie. Lediglich im Sonderfall $v = 1$ ist die naive Strategie besser als das Eager Task Dropping, da hier keine Wartezyklen nach der Zuteilung von Tasks auftreten, während beim Eager Task Dropping in jeder Phase ein Hormonzyklus zum Stopp von Tasks benötigt wird.

Ein Vergleich der Strategien ergab zudem, dass eine Selbstheilung bei Verwendung der naiven Task-Dropping-Strategie in den meisten Fällen sogar im *Average Case* länger dauert als der *Worst Case* des Eager Task Dropping. Lediglich in einigen wenigen Fällen (darunter der bereits genannte Fall $v = 1$) wird das System vom Eager Task Dropping langsamer geheilt als von der naiven Strategie im Average Case.

Zumeist kann man ferner annehmen, dass ein System im voll funktionsfähigen Zustand deutlich mehr als zwei PEs zur vollständigen Realisierung seiner Funktionalität benötigt. Entsprechend ist daher fraglich, ob im Extremfall $v = 1$ vom letzten verbleibenden PE überhaupt noch signifikante Teilfunktionalitäten des Systems realisiert werden können. Folglich ist das Eager Task Dropping der naiven Task-Dropping-Strategie im Regelfall vorzuziehen, da dieser Sonderfall für die meisten Systeme von geringer praktischer Relevanz ist.

Nach diesen Analysen widmen wir uns im nächsten Kapitel einer Evaluation der hergeleiteten Zeitschranken.

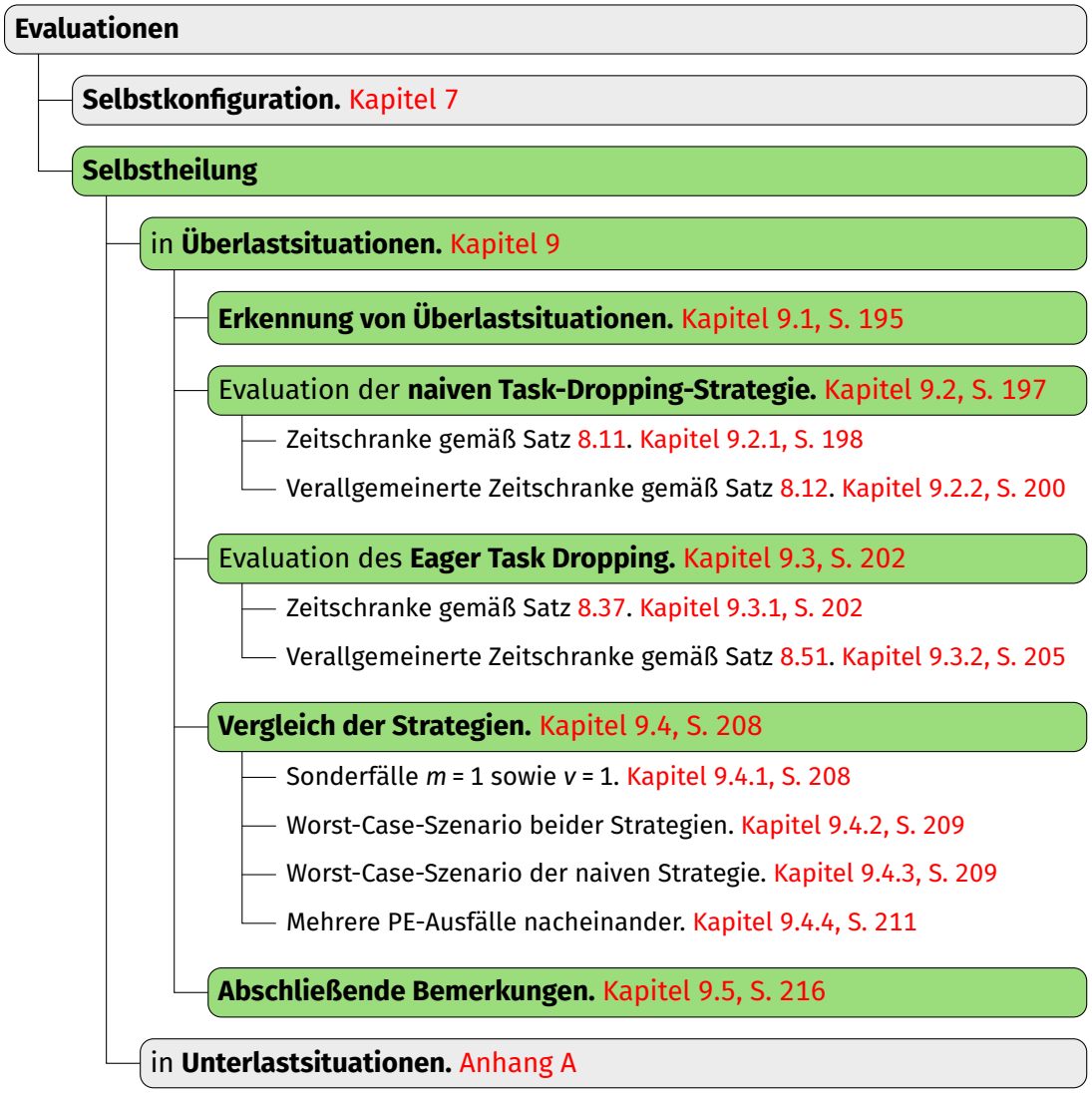


Abbildung 9.1: Überblick über die Evaluationen der Selbstheilung in Überlastsituationen

9 Evaluation der Selbstheilung in Überlastsituationen

In diesem Kapitel sollen nun die Zeitschranken der Selbstheilung in Überlastsituationen evaluiert werden. Der Kürze halber beschränken wir uns jedoch darauf, diese zu bestätigen, indem vor allem die ihnen zugrunde liegenden Abstraktionen überprüft werden. Um die resultierenden Graphen übersichtlich zu halten, kommen zudem vorrangig vergleichsweise kleine Skalierungen des AHS zum Einsatz und auf die Abbildung der nach der Anzahl pro PE zugeteilter Tasks gestapelten Graphen wird zumeist verzichtet.

Außerdem verzichten wir im Folgenden auf die Verwendung von Acceleratoren: Wie in Kapitel 7.4.5 begründet, können diese die Taskverteilung beschleunigen. Der Verzicht auf Acceleratoren ermöglicht es daher im Umkehrschluss, das Auftreten des Worst Case zu erzwingen. Zum anderen sieht das aus Kapitel 8.2 bekannte Modell der Überlastsituationen vor, dass alle Tasks verschiedene Prioritäten haben, weshalb die Verwendung Acceleratoren ohnehin zu keiner Beschleunigung der Taskverteilung führen würden, da in diesem Fall nach jeder Taskzuteilung ein Wartezyklus auftritt.

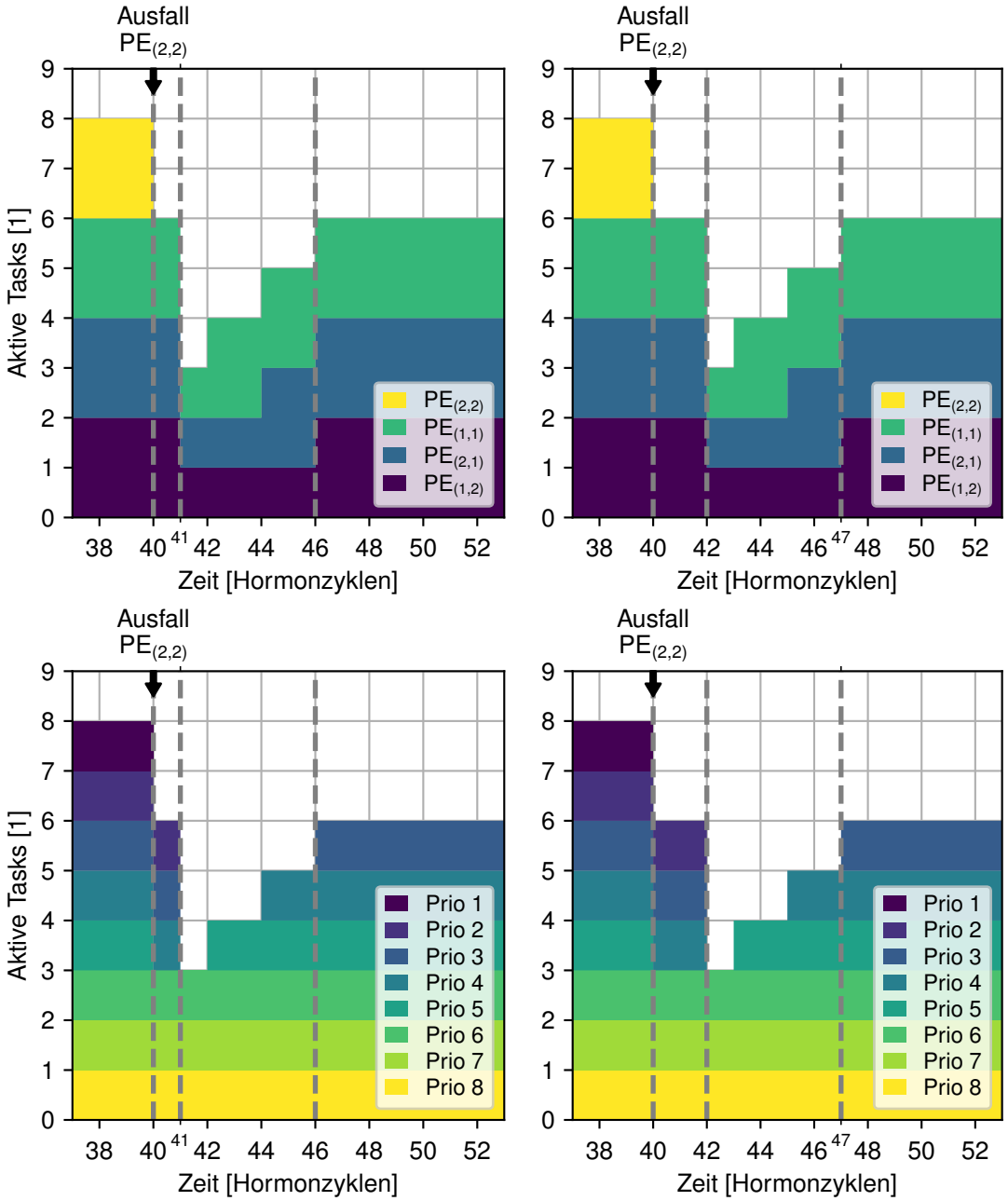
Abbildung 9.1 liefert einen grafischen Überblick über die Struktur der folgenden Evaluationen.

9.1. Erkennung von Überlastsituationen

Zunächst wollen wir die zur Erkennung des Vorliegens einer Überlastsituation benötigte Zeit betrachten. Laut Satz 8.4 wird eine durch ein zum Zeitpunkt $t_\times$ ausfallendes $\mathrm{PE}_\times$ hervorgerufene Überlastsituation frühestens in Hormonzyklus $t_\times + a_{\min}$ und spätestens in Hormonzyklus $t_\times + a$ erkannt, weshalb die Erkennung mindestens $a_{\min}+1$ und höchstens $a+1$ Hormonzyklen benötigt. Die erste Taskübernahme ist jeweils im darauffolgenden Hormonzyklus möglich. Ferner betragen die Werte dieser symbolischen Konstanten für die aktuelle Implementierung laut Satz 5.11 $a_{\min} = 1$ bzw. $a = 2$ Hormonzyklen.

Um diese Schranken der Überlasterkennung zu validieren, betrachten wir im Folgenden ein Szenario, wie es in ähnlicher Form bereits im Beweis von Satz 8.4 genutzt wurde, um die Problematik der Erkennung einer Überlastsituation zu demonstrieren. Konkret verwenden wir $v + 1 = 4$ PEs, die insgesamt acht Tasks verschiedener Prioritäten ausführen, wobei jedes PE jedoch höchstens $m = 2$ Tasks übernehmen kann.

Betrachtung von $a_{\min}$ Abbildung 9.2a zeigt das Verhalten des AHS, wenn – nach einer nicht abgebildeten initialen Selbstkonfiguration – $\mathrm{PE}_\times = \mathrm{PE}_{(2,2)}$ in Hormonzyklus $t_\times = 40$ ausfällt, *bevor* es seine Hormone senden konnte. Dieser Ausfall führt zu einer Überlastsituation mit $v = 3$ verbleibenden PEs und $m = 2$ Tasks pro PE. Wie aus dem unteren Graphen hervorgeht, der die im System zugeteilten Tasks nach ihren Prioritäten gestapelt darstellt, gilt in dieser Überlastsituation $T_{\max} = 5$: Die



(a) Best Case: Ausfall *vor* Senden der Hormone in Zyklus 40, die Überlastsituation wird in Zyklus 41 erkannt.

(b) Worst Case: Ausfall *nach* Senden der Hormone in Zyklus 40, die Überlastsituation wird in Zyklus 42 erkannt.

Abbildung 9.2: Erkennung einer Überlastsituation bei PE-Ausfall zu verschiedenen Zeitpunkten während eines Hormonzyklus und anschließende Selbstheilung des Systems unter Verwendung der naiven Task-Dropping-Strategie

höchstpriore Task, die auf $\mathrm{PE}_\times$ lief und mit diesem PE ausfällt, hat die Priorität 5, entsprechend ist diese Task nun die höchstpriore nicht zugeteilte Task. In Zyklus 41 wird die vorliegende Überlastsituation erkannt und jedes der verbleibenden PEs stoppt gemäß der hier verwendeten naiven Task-Dropping-Strategie alle Tasks mit niedrigerer Priorität als 5, um Kapazitäten für $T_{\max}$ zu schaffen – aus dem oberen Graphen geht hervor, dass jedes PE im betrachteten Szenario genau eine Task stoppt. Im darauffolgenden Hormonzyklus 42 erfolgt nun die Zuteilung von $T_{\max}$; der restliche Verlauf der Selbstheilung entspricht gerade jenem einer Selbstkonfiguration, bis das System nach Zuteilung der letzten Task 4 in Zyklus 46 wieder einen stabilen Zustand erreicht hat.

Folglich lässt sich festhalten, dass die Überlastsituation in Hormonzyklus $41 = 40 + 1 = t_\times + a_{\min}$ erkannt wurde, woraufhin in Zyklus $42 = 40 + 1 + 1 = t_\times + a_{\min} + 1$ die erste Taskübernahme stattfinden konnte. Dies entspricht jedoch gerade der von Satz 8.4 für den Best Case getroffenen Aussage.

Betrachtung von *a* Abbildung 9.2b zeigt ein Szenario, das sich vom vorherigen nur darin unterscheidet, dass $\mathrm{PE}_\times$ in Hormonzyklus $t_\times = 40$ erst *nach* dem Senden seiner Hormone ausfällt. Ein Vergleich mit Abbildung 9.2a zeigt, dass die Überlastsituation nun genau einen Hormonzyklus später erkannt wird, weshalb die erste Taskübernahme ebenfalls einen Hormonzyklus verzögert stattfindet. In allen übrigen Aspekten zeigen beide Fälle identisches Verhalten, weshalb darauf nicht weiter eingegangen werden soll.

Dieses Verhalten bestätigt jedoch gerade den von Satz 8.4 vorhergesagten Worst Case, da die Überlastsituation hier in Hormonzyklus $42 = 40 + 2 = t_\times + a$ erkannt wurde, gefolgt von der ersten Taskübernahme in Zyklus $43 = 40 + 2 + 1 = t_\times + a + 1$.

Da die Erkennung einer Überlastsituation auch in allen übrigen Szenarien nach demselben Mechanismus funktioniert und insbesondere keine verzögernden Faktoren zu tragen kommen, kann Satz 8.4 daher neben dem formalen Beweis auch als experimentell bestätigt betrachtet werden. Im Folgenden wird jedoch stets angenommen, dass $\mathrm{PE}_\times$ immer *nach* dem Senden seiner Hormone ausfällt, um den Worst Case zu betrachten.

9.2. Evaluation der naiven Task-Dropping-Strategie

Nachdem soeben gezeigt wurde, dass Überlastsituationen korrekt erkannt werden, soll nun zunächst die in Kapitel 8.5 vorgestellte naive Task-Dropping-Strategie evaluiert werden. Gemäß Strategie 8.10 stoppt diese bei Erkennung einer Überlastsituation auf jedem verbleibenden PE *alle* Tasks, deren Priorität *kleiner* als $\mathrm{prio}(T_{\max})$ ist, woran sich eine Selbstkonfiguration anschließt, bis das System wieder einen stabilen Zustand erreicht hat. Der Mechanismus einer solchen Selbstkonfiguration wurde bereits in Kapitel 7 umfassend evaluiert, weshalb wir uns im Folgenden auf wenige exemplarisch gewählte Spezialfälle beschränken.

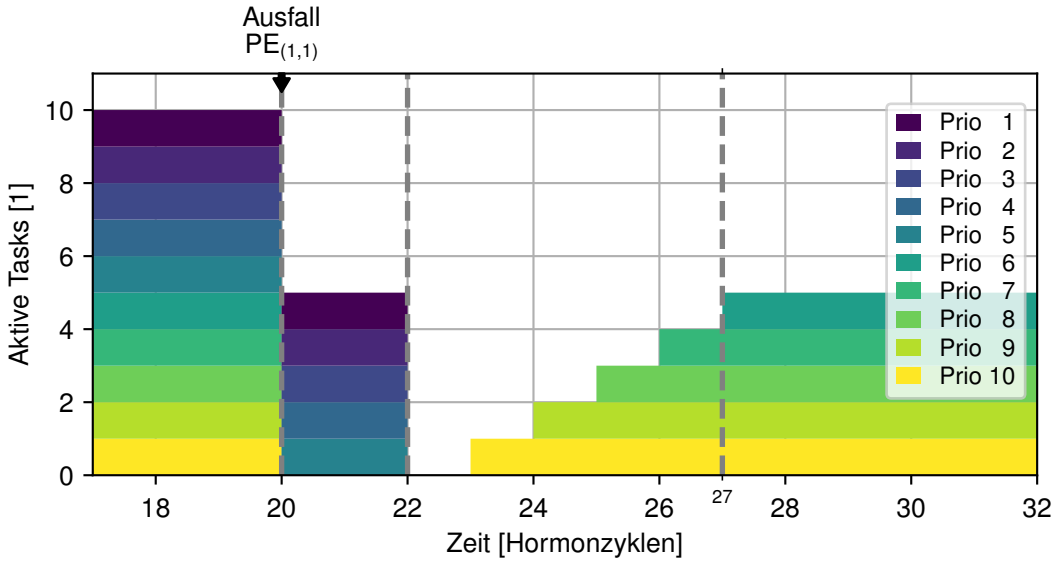


Abbildung 9.3: Selbstheilung in einer Überlastsituation für $m = 5$ und $v = 1$ bei Verwendung der naiven Task-Dropping-Strategie

9.2.1. Evaluation der Zeitschranke gemäß Satz 8.11 (alle Prioritäten verschieden)

Gemäß Satz 8.11 dauert die Selbstheilung in einer Überlastsituation bei Verwendung der naiven Task-Dropping-Strategie höchstens $mv + (mv - 1) \cdot [v > 1] + 1 + a$ Hormonzyklen. Im Gegensatz zur Zeitschranke der Selbstkonfiguration, welche die Existenz von mindestens zwei PEs annimmt, werden hier also die Fälle $v = 1$ und $v > 1$ unterschieden.

9.2.1.1. Betrachtung des Falls $v = 1$

Liegt eine Überlastsituation mit nur einem einzigen verbleibenden PE vor, so dauert die Selbstheilung gemäß Satz 8.11 höchstens $mv+1+a$ Hormonzyklen, da keine Wartezyklen auftreten. Da dieses Verhalten für die Selbstkonfiguration bereits in Kapitel 7.2.1.2 experimentell bestätigt wurde, überrascht es nicht, dass es auch in Abbildung 9.3 zu beobachten ist: Besagte Abbildung zeigt das im Worst Case beobachtete Verhalten des Systems in einer Überlastsituation mit $m = 5$ Tasks pro PE bei einem einzigen verbleibenden PE.

Auf dem in Zyklus 20 ausgefallenen $\text{PE}_\times = \text{PE}_{(1,1)}$ war zu Beginn die höchstpriore Task 10 zugeteilt. Bei Erkennung der Überlastsituation in Zyklus 22 stoppt das verbleibende PE daher *alle* seine zugeteilten Tasks, gefolgt von einer Selbstkonfiguration der $mv = 5$ höchstprioren Tasks, die in Zyklus 23 startet und in Zyklus 27 beendet wird. Insbesondere treten dabei keine Wartezyklen auf, da nur ein einziges PE an dieser Selbstkonfiguration beteiligt ist. Damit dauert die gesamte Selbstheilung

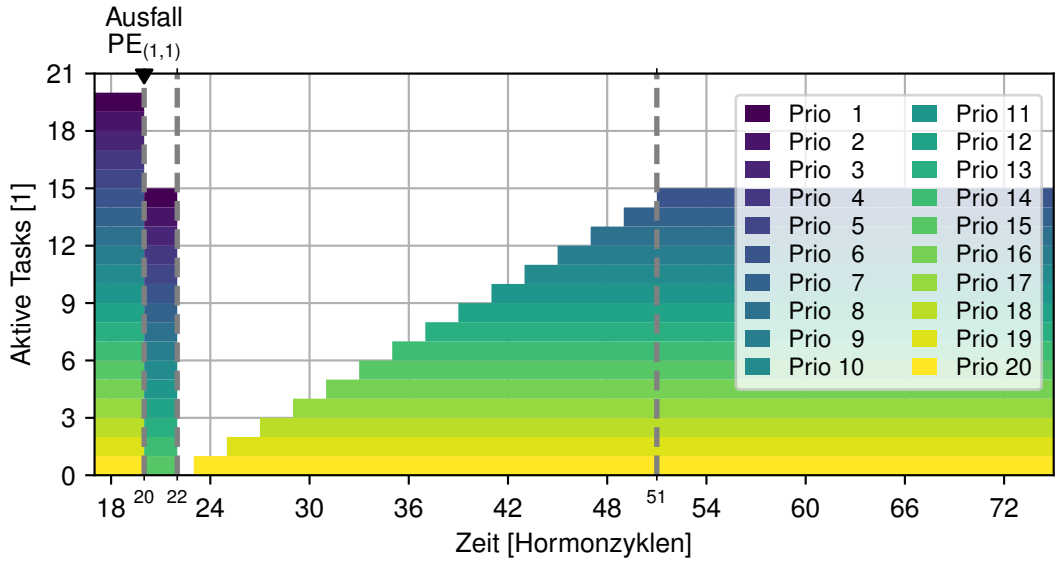


Abbildung 9.4: Selbstheilung in einer Überlastsituation für $m = 5$ und $v = 3$ bei Verwendung der naiven Task-Dropping-Strategie

gerade $8 = 5 \cdot 1 + 1 + 2 = mv + 1 + a$ Hormonzyklen, was exakt der durch Satz 8.11 vorhergesagten Anzahl entspricht.

9.2.1.2. Betrachtung des Falls $v > 1$

Verbleibt dagegen mehr als ein funktionsfähiges PE, schätzt Satz 8.11 die Dauer der Selbstheilung in einer Überlastsituation zu höchstens $2mv + a$ Hormonzyklen ab.

Abbildung 9.4 zeigt das Systemverhalten bei Vorliegen einer Überlastsituation in einem solchen Fall. Konkret existieren nach dem Ausfall von $\text{PE}_\times = \text{PE}_{(1,1)}$ in Hormonzyklus 20 noch $v = 3$ PEs, die jeweils $m = 5$ Tasks übernehmen können. Wie zuvor war die höchstpriore Task (in diesem Fall Task 20) auf $\text{PE}_\times$ zugeteilt, weshalb der Worst Case eintritt und die verbleibenden PEs bei Erkennung der Überlastsituation in Zyklus 22 *alle* auf ihnen zugeteilten Tasks stoppen. Im darauffolgenden Hormonzyklus 23 findet die erste Taskübernahme statt; der weitere Verlauf der Selbstheilung entspricht jenem einer Selbstkonfiguration von $mv = 15$ Tasks, die mit Übernahme der letzten Task in Zyklus 51 abgeschlossen wird. Im Gegensatz zum zuvor betrachteten Fall $v = 1$ tritt hier nach jeder Taskzuteilung (außer der letzten) ein Wartezyklus auf.

Insgesamt dauert die Selbstkonfiguration somit $32 = 2 \cdot 5 \cdot 3 + 2 = 2mv + a$ Hormonzyklen, was sich wiederum exakt mit der Vorhersage aus Satz 8.11 deckt.

Dass die abgeschätzte Anzahl Hormonzyklen in beiden betrachteten Fällen auch tatsächlich exakt erreicht wurde, ist jedoch nicht überraschend, da diese Szenarien jeweils exakt dem Worst Case entsprachen: Die Ausfälle von $\text{PE}_\times$ fanden jeweils *nach* dem Senden der Hormone statt, auf dem ausfallenden $\text{PE}_\times$ war die jeweils höchstpriore Task zugeteilt, weshalb *alle* verbleibenden Tasks gestoppt wurden, und alle Tasks hatten

verschiedene Prioritäten, weshalb im Fall $v > 1$ die maximal mögliche Anzahl von Wartezyklen auftrat. Als Ergebnis ist daher festzuhalten, dass die in Satz 8.11 getroffene Abschätzung keinesfalls zu pessimistisch ist, sondern den Worst Case vielmehr exakt charakterisiert.

Nichtsdestotrotz möchten wir uns im folgenden Abschnitt davon überzeugen, dass die naive Task-Dropping-Strategie sich auch in Szenarien korrekt verhält, die *nicht* dem Worst Case entsprechen.

9.2.1.3. Betrachtung eines Nicht-Worst-Case-Szenarios

Die zuvor betrachtete Abbildung 9.2 zeigte zwei Szenarien, in denen jeweils eine Selbstheilung in einer Überlastsituation unter Verwendung der naiven Task-Dropping-Strategie durchgeführt wurde. Diese beiden – bis auf den Ausfallzeitpunkt identischen – Szenarien wollen wir an dieser Stelle daher als exemplarische Vertreter von Nicht-Worst-Case-Szenarien genauer betrachten.

Konkret handelte es sich jeweils um Überlastsituationen mit $m = 2$ Tasks pro PE bei $v = 3$ verbleibenden PEs. Da die höchstpriore auf $\mathrm{PE}_\times = \mathrm{PE}_{(2,2)}$ zugeteilte Task $T_{\max} = 5$ war, wurden, wie zuvor gesehen, nur Tasks mit einer geringeren Priorität gestoppt, was genau dem gemäß Strategie 8.10 erwarteten Verhalten entspricht: Die Implementierung der naiven Task-Dropping-Strategie stoppt also in einer solchen Nicht-Worst-Case-Situation keinesfalls zu viele Tasks.

In beiden Szenarien folgte im auf die Erkennung der Überlastsituation folgenden Hormonzyklus eine Selbstkonfiguration der drei höchstprioren nicht zugeteilten Tasks, bei der wegen $v > 1$ nach jeder Taskübernahme ein Wartezyklus auftrat. Insgesamt dauerte die gesamte Selbstheilung

a) $7 = 6 + 1 = 2 \cdot 3 + a_{\min}$ Hormonzyklen, falls $\mathrm{PE}_\times$ zum Best-Case-Zeitpunkt ausfiel (Abbildung 9.2a),

b) $8 = 6 + 2 = 2 \cdot 3 + a$ Hormonzyklen, falls $\mathrm{PE}_\times$ zum Worst-Case-Zeitpunkt ausfiel (Abbildung 9.2b).

Auch im zweiten Szenario, das in Bezug auf den Ausfallzeitpunkt dem Worst Case entsprach, war die Selbstheilung also schneller beendet, als von Satz 8.11 für $v > 1$ zu höchstens $2mv + a = 14$ Hormonzyklen abgeschätzt würde. Dies überrascht jedoch ebenfalls nicht, da die zugrunde liegende Selbstkonfiguration (die für beide Szenarien identisch ablief) hier nur drei statt der im Worst Case nötigen $mv = 6$ Tasks zuteilen musste.

Zusammenfassend lässt sich daher festhalten, dass die naive Task-Dropping-Strategie auch in Nicht-Worst-Case-Szenarien korrekt funktioniert, Satz 8.11 in einem solchen Fall jedoch eine Überabschätzung der Dauer der Selbstheilung liefert.

9.2.2. Evaluation der verallgemeinerten Zeitschranke gemäß Satz 8.12

Wie die bisherigen Szenarien verdeutlichten, entspricht die naive Task-Dropping-Strategie nach der Erkennung einer Überlastsituation sowie den damit verbundenen

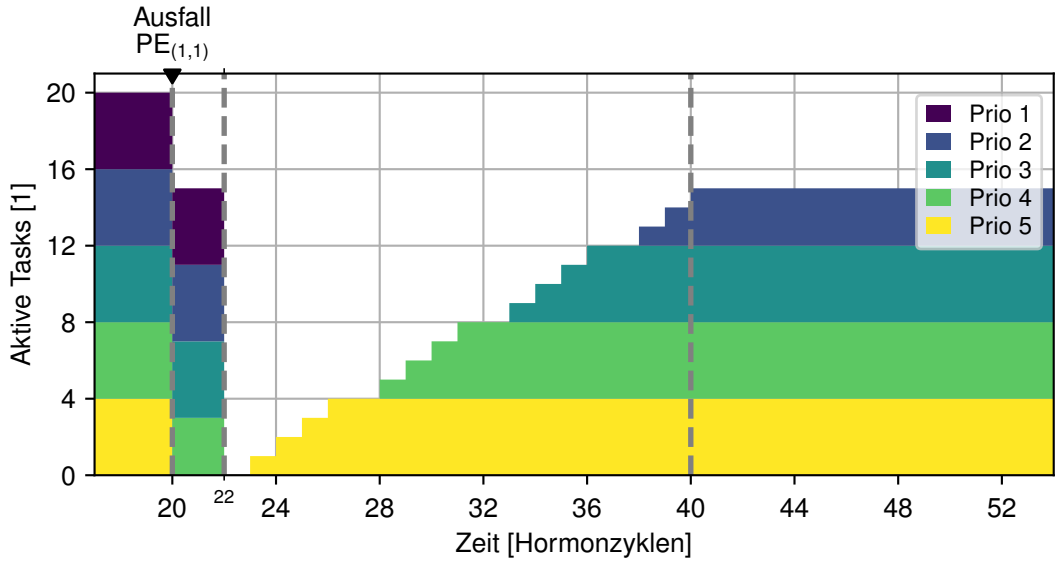


Abbildung 9.5: Selbstheilung in einer Überlastsituation für $m = 5$, $v = 3$ und $|P| = 5$ bei Verwendung der naiven Task-Dropping-Strategie

Taskstopps im Wesentlichen einer Selbstkonfiguration der höchstprioren nicht zugeteilten Tasks. Entsprechend lässt sich die Zeitschranke leicht verallgemeinern, um Szenarien, in denen mehrere Tasks dieselben Prioritäten besitzen, besser abschätzen zu können. Dies leistet Satz 8.12, der die Dauer einer solchen Selbstheilung bei insgesamt $|P|$ verschiedenen Prioritäten zu höchstens $mv+(\min\{|P|, mv\}-1)\cdot[v>1]+1+a$ Hormonzyklen abschätzt.

Da diese Abschätzung in den Grenzfällen $v = 1$ und $|P| \geq mv$ jeweils zur aus Satz 8.11 bekannten Form degeneriert, betrachten wir im Folgenden nur den Spezialfall von höchstens $mv + |P| + a$ Hormonzyklen, der sich für $v > 1$ und $|P| < mv$ ergibt.

Abbildung 9.5 stellt eine Abwandlung des aus Abbildung 9.4 bekannten Szenarios mit $m = 5$ Tasks pro PE und $v = 3$ verbleibenden PEs dar. Gegenüber dem vorherigen Szenario wurden die Prioritäten der insgesamt $m\cdot(v+1) = 20$ Tasks jedoch analog zum aus Abbildung 7.15 (S. 105) bekannten Schema angepasst. Konkret besitzen nun jeweils $i = 4$ Tasks dieselbe Priorität, was in $|P| = \lceil(m\cdot(v+1))/i\rceil = 5$ verschiedenen Prioritäten resultiert. Wie nun aus der Abbildung hervorgeht, fällt in Zyklus 20 $\mathrm{PE}_\times = \mathrm{PE}_{(1,1)}$ aus, auf dem alle Tasks der höchsten Priorität 5 sowie eine Task der Priorität 4 liefen. In Zyklus 22 wird dieser Ausfall erkannt, woraufhin *alle* zugeteilten Tasks gestoppt werden. Im darauffolgenden Zyklus beginnt dann eine Selbstkonfiguration der $mv = 15$ höchstprioren nicht zugeteilten Tasks. Hierbei ist gut zu erkennen, dass nur insgesamt drei Wartezyklen nach Vervollständigung der drei höchsten Prioritätslevel auftreten, jedoch nicht innerhalb der einzelnen Level.

Entsprechend dauert die Selbstheilung insgesamt 21 Hormonzyklen, während Satz 8.12 hierfür höchstens $mv+|P|+a = 22$ Hormonzyklen abschätzt. Diese Diskrepanz von einem Hormonzyklus erklärt sich dadurch, dass aufgrund der Überlastsituation keine

Task mit Priorität 1 mehr im System zugeteilt werden kann, weshalb kein Wartezyklus für Priorität 2 berücksichtigt werden muss.

Bei einer anderen Verteilung der Tasks auf die verschiedenen Prioritäten könnte es jedoch ebenso möglich sein, dass trotz der Überlastsituation noch Tasks *aller* Prioritätslevel zugeteilt werden können, weshalb die allgemeine Abschätzung nach Satz 8.12 nicht schärfer formuliert werden kann. In einem solchen Fall wäre jedoch nicht definiert, welche konkreten Tasks der niedrigsten Priorität zugeteilt würden. Diese Tatsache ist auch im hier betrachteten Szenario relevant, da aufgrund der Überlastsituation nur noch drei der vier Tasks mit Priorität 2 zugeteilt werden können: Ähnlich wie die aggressive Task-Decision-Strategie in Überlastsituation nicht definiert, welche konkrete Untermenge *aller* Tasks zugeteilt wird, definiert die prioritätsbasierte Strategie hier nicht, welche konkrete Untermenge der kleinsten, nur noch *teilweise zuteilbaren* Priorität zugeteilt wird.

Zusammenfassend lässt sich festhalten, dass die naive Task-Dropping-Strategie sich wie von den theoretischen Betrachtungen vorhergesagt verhält und die Abschätzungen der Sätze 8.11 und 8.12 als gültig betrachtet werden können.

9.3. Evaluation des Eager Task Dropping

Als Nächstes widmen wir uns dem Eager Task Dropping. Hierzu betrachten wir zunächst die Zeitschranke aus Satz 8.37, die annimmt, dass alle Tasks verschiedene Prioritäten haben, bevor wir später die Verallgemeinerung auf wenige Prioritätslevel betrachten.

9.3.1. Evaluation der Zeitschranke gemäß Satz 8.37 (alle Prioritäten verschieden)

Gemäß Satz 8.37 dauert eine Selbstheilung in einer Überlastsituation unter Verwendung des Eager Task Dropping (Strategie 8.14) höchstens $2\sum_{i=1}^{v}\lceil\frac{m}{i}\rceil + a$ Hormonzyklen.

Laut Satz 8.42 ist dies die kleinste obere Schranke, falls $v \geq m$ gilt, laut Satz 8.46 gilt dies sogar im Fall $v < m$, sofern $v \leq 1000$ und $m \leq 1000$. Daher verwenden wir im Folgenden das (verallgemeinerte) Worst-Case-Szenario aus Beispiel 8.43.

9.3.1.1. Betrachtung des Falls $m = 4$ und $v = 5$

Abbildung 9.6 zeigt das Verhalten des Eager Task Dropping für $m = 4$ Tasks pro PE sowie $v = 5$ verbleibende PEs. Die Ausgangssituation vor dem Ausfall von $\mathrm{PE}_{\times} = \mathrm{PE}_{(1,1)}$ in Hormonzyklus 20 entspricht hierbei exakt dem aus Beispiel 8.43 bekannten Worst-Case-Szenario.

Mit der Erkennung das Vorliegens einer Überlastsituation in Hormonzyklus 22 beginnt die erste Selbstheilungsphase, an der alle $v = 5$ verbleibenden PEs beteiligt sind und jeweils ihre niedrigstpriore Task stoppen. Im Anschluss werden die fünf höchstprioren nicht zugeteilten Tasks im Rahmen einer Selbstkonfiguration zugeteilt, wobei Wartezyklen auftreten. Lediglich die letzte Zuteilung in Zyklus 31 ruft keinen Wartezyklus hervor, da nur noch ein einziges PE positive modifizierte Eignungswerte

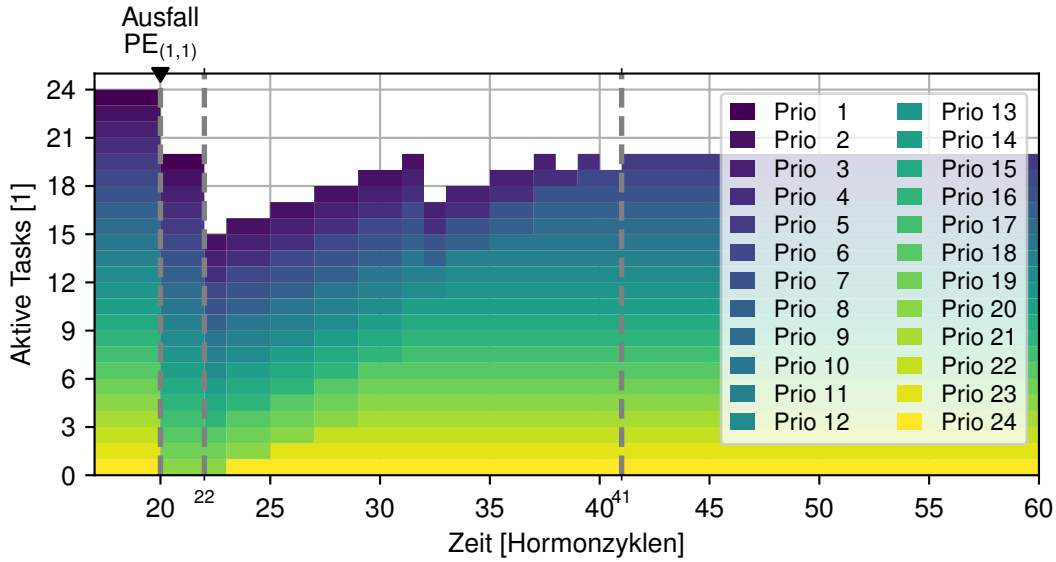


Abbildung 9.6: Selbstheilung in einer Überlastsituation für $m = 4$ und $v = 5$ bei Verwendung des Eager Task Dropping

für die entsprechende Task sendet. Diese letzte Zuteilung führt das System jedoch erneut in eine Überlastsituation, weshalb im darauffolgenden Zyklus 32 wiederum Tasks gestoppt werden, was den Start der zweiten Selbstheilungsphase markiert. An dieser sind jedoch nur noch drei PEs beteiligt, weshalb diese Phase auch bereits in Zyklus 37 beendet wird. An den folgenden beiden Phasen, die in den Zyklen 38 und 40 starten, ist dagegen jeweils nur ein einziges PE beteiligt. Für eine genauere Betrachtung der Verteilung der Tasks auf die PEs nach jeder einzelnen Phase sei erneut auf Beispiel 8.43 verwiesen.

Insgesamt wurde in diesem Szenario also die maximal mögliche Anzahl von $m = 4$ Selbstheilungsphasen benötigt. Da die Selbstheilung mit dem Ausfall von $\mathrm{PE}_{(1,1)}$ in Zyklus 20 startete und mit der Zuteilung von Task 5 in Zyklus 41 endete, wurden hierzu insgesamt 22 Hormonzyklen benötigt, was sich exakt mit der durch Satz 8.37 vorhergesagten Zahl von höchstens $2\sum_{i=1}^{v}\lceil\frac{m}{i}\rceil + a = 2\sum_{i=1}^{5}\left\lceil\frac{4}{i}\right\rceil + 2 = 22$ Hormonzyklen deckt und damit experimentell bestätigt, dass das betrachtete Szenario tatsächlich dem Worst Case entspricht.

9.3.1.2. Betrachtung des Falls $m = 3$ und $v = 2$

Eine kleinere Skalierung eines Überlastszenarios ist in Abbildung 9.7 dargestellt. Hier fällt $\mathrm{PE}_{\times} = \mathrm{PE}_{(1,1)}$ wie zuvor in Hormonzyklus 20 aus, es verbleiben jedoch nur $v = 2$ PEs, die jeweils $m = 3$ Tasks ausführen können. Hierzu wurde das zuvor genutzte Worst-Case-Szenario entsprechend skaliert. Der Verlauf der Selbstheilung ist dabei ähnlich zum zuvor betrachteten Szenario: Auch hier tritt die maximale Zahl von $m = 3$ Selbstheilungsphasen auf, wobei an den ersten beiden Phasen je zwei PEs

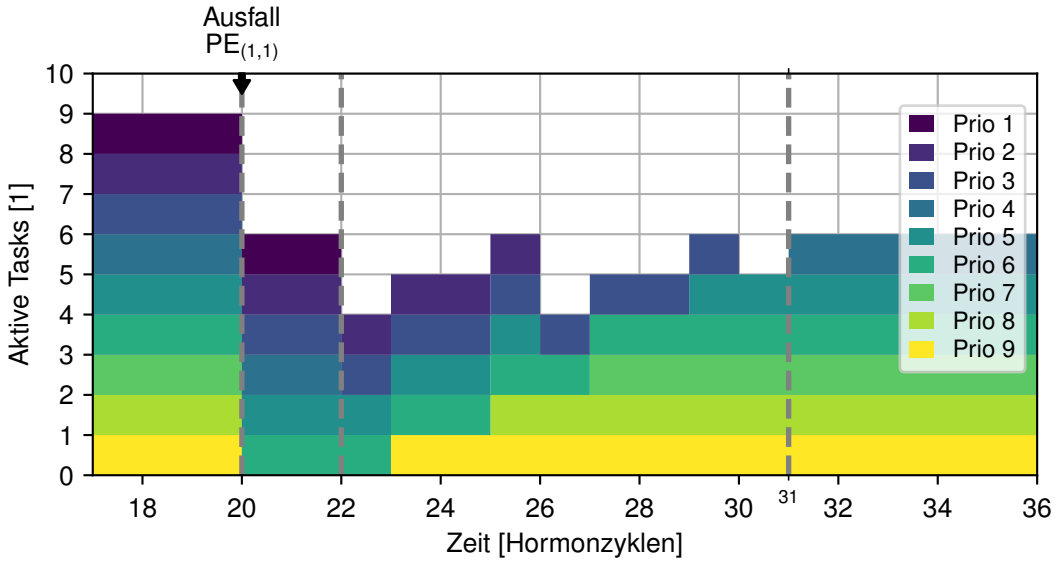


Abbildung 9.7: Selbstheilung in einer Überlastsituation für $m = 3$ und $v = 2$ bei Verwendung des Eager Task Dropping

beteiligt sind, während sich an der letzten Phase dagegen nur noch ein einziges PE beteiligt. Insgesamt dauert die Selbstheilung hier 12 Hormonzyklen, was wiederum exakt mit der von Satz 8.37 getroffenen Vorhersage von höchstens $2\sum_{i=1}^{v}\lceil\frac{m}{i}\rceil + a = 2\sum_{i=1}^{2}\left\lceil\frac{3}{i}\right\rceil + 2 = 12$ Hormonzyklen übereinstimmt.

9.3.1.3. Betrachtung des Falls $m = 5$ und $v = 1$

Im Fall der naiven Task-Dropping-Strategie ergab sich für den Sonderfall eines einzigen verbleibenden PEs eine Beschleunigung der Selbstheilung, da keine Wartezyklen auftreten. Eine ähnliche Beschleunigung tritt für das Eager Task Dropping jedoch nicht ein, wie Abbildung 9.8 zeigt: Es werden hier erneut die maximal mögliche Zahl von $m = 5$ Selbstheilungsphasen benötigt, was in einer Selbstheilungsdauer von 12 Hormonzyklen resultiert und sich erneut exakt mit der von Satz 8.37 getroffenen Abschätzung deckt. Es treten zwar keine Wartezyklen auf, dafür fällt jedoch umso mehr ins Gewicht, dass zu Beginn jeder Phase ein Hormonzyklus aufgewendet werden muss, um eine Task zu stoppen. Konkret stoppt das verbleibende PE sukzessive *alle* ursprünglich auf ihm zugeteilten Tasks, um sie durch höherpriore Tasks zu ersetzen.

Das gezeigte Szenario wurde in Abschnitt 9.2.1.1 bzw. Abbildung 9.3 bereits für die naive Strategie betrachtet. Diese stoppte jedoch alle Tasks *gleichzeitig* gefolgt von einer (ohne Wartezyklen ablaufenden) Selbstkonfiguration und konnte das System daher bereits nach 8 Hormonzyklen in einen stabilen Zustand überführen.

Aufgrund dieser Besonderheit wäre die naive Strategie dem Eager Task Dropping im Sonderfall $v = 1$ daher vorzuziehen, wie in Kapitel 8.7.2 bereits festgestellt wurde. Da jedoch erwartet werden kann, dass reale Systeme aus vergleichsweise vielen PEs

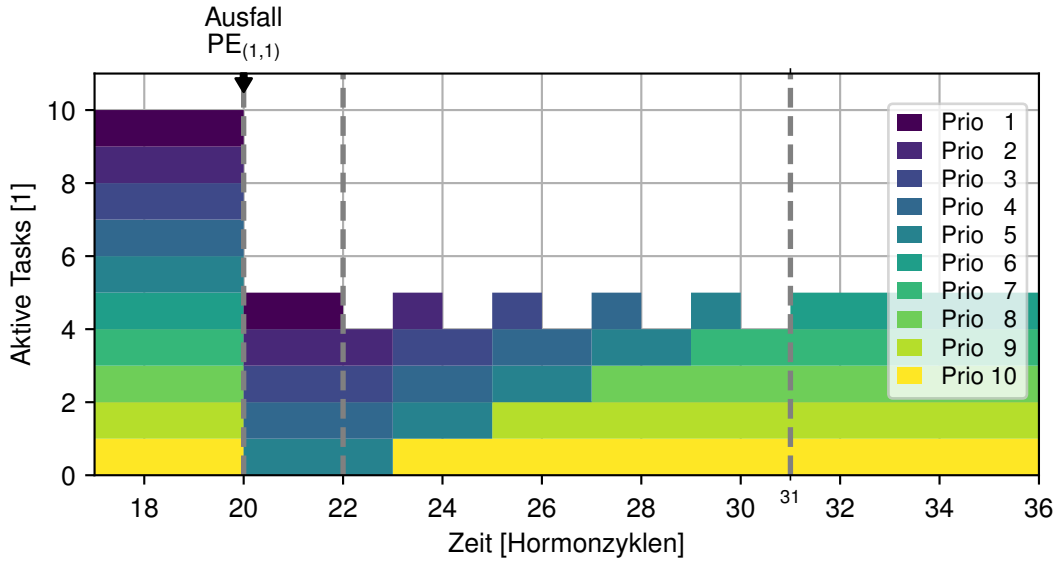


Abbildung 9.8: Selbstheilung in einer Überlastsituation für $m = 5$ und $v = 1$ bei Verwendung des Eager Task Dropping

bestehen, ist jedoch zumindest fraglich, ob von sehr wenigen verbleibenden PEs überhaupt noch eine hinreichend große Teilfunktionalität des Systems aufrecht erhalten werden kann oder das Gesamtsystem bereits zuvor effektiv als vollständig ausgefallen angesehen werden muss. Folglich ist fraglich, ob die theoretische Überlegenheit der naiven Strategie im Fall $v = 1$ überhaupt von praktischer Relevanz ist. Daher wollen wir uns hier nicht länger mit diesem Sonderfall beschäftigen, sondern im Folgenden das Verhalten des Eager Task Dropping untersuchen, wenn mehrere Tasks dieselbe Priorität besitzen.

9.3.2. Evaluation der verallgemeinerten Zeitschranke gemäß Satz 8.51

Lässt man zu, dass mehrere Tasks dieselbe Priorität besitzen, so besagt Satz 8.51, dass eine Selbstheilung in einer Überlastsituation bei Verwendung des Eager Task Dropping höchstens

$$\sum_{i=1}^{v} \left\lceil \frac{m}{i} \right\rceil + \min\{v, |P|\} + \sum_{i=2}^{m} \min\left\{ v, \left\lfloor \frac{m-1}{i-1} \right\rfloor, |P| \right\} + a \text{ Hormonzyklen}$$

dauert, wobei $|P|$ die Anzahl der verschiedenen Prioritäten bezeichnet. Im Folgenden wollen wir diese verallgemeinerte Zeitschranke anhand zweier exemplarischer Überlastszenarien evaluieren.

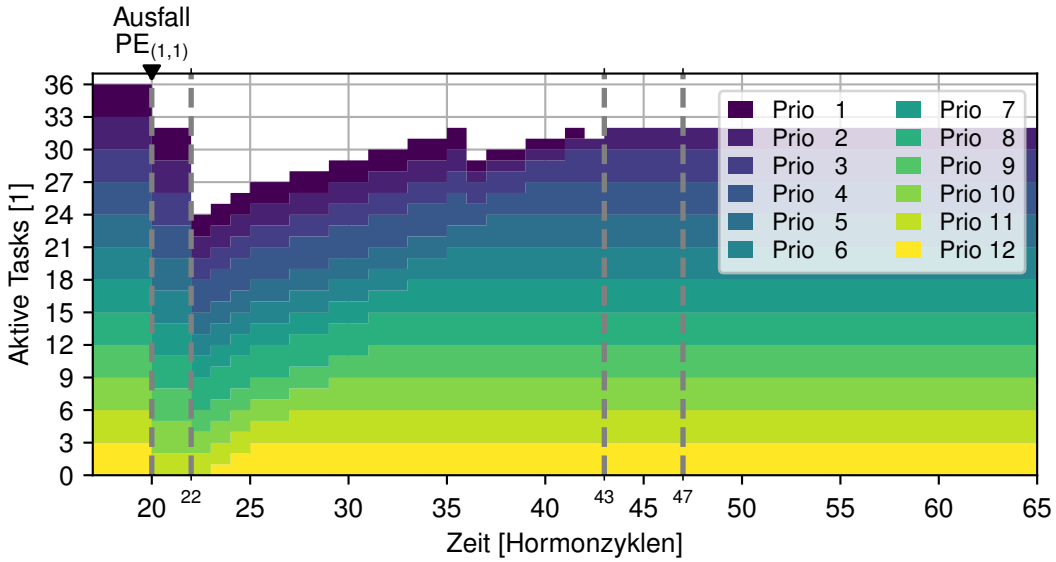


Abbildung 9.9: Selbstheilung in einer Überlastsituation für $m = 4$, $v = 8$ und $|P| = 12$ bei Verwendung des Eager Task Dropping

9.3.2.1. Betrachtung des Falls $m = 4$, $v = 8$ und $|P| = 12$

Abbildung 9.9 zeigt ein Selbstheilungsszenario mit $m = 4$ Tasks pro PE und $v = 8$ verbleibenden PEs, bei denen jeweils drei der insgesamt 36 Tasks dieselbe Priorität besitzen, was in $|P| = 36/3 = 12$ verschiedenen Prioritäten resultiert. Die initiale Verteilung der Tasks auf die PEs entspricht wiederum dem aus den vorherigen Evaluationen bekannten Worst-Case-Schema.

Im Graphen ist gut zu erkennen, dass die Selbstheilung in drei Phasen abläuft. Zu Beginn der ersten Phase treten zudem keine Wartezyklen auf, da zunächst nur Tasks der Priorität 12 zugeteilt werden. Mit der Zuteilung einer Task der Priorität 2 in Zyklus 43 wird die Selbstheilung beendet, die somit insgesamt 24 Hormonzyklen dauerte. Satz 8.51 schätzt die Dauer dagegen zu höchstens 28 Hormonzyklen ab, wonach die Selbstheilung spätestens in Zyklus 47 beendet würde. Diese Abschätzung widerspricht der Beobachtung also nicht, liefert im Gegensatz zu den zuvor betrachteten Szenarien, in denen alle Tasks verschiedene Prioritäten besaßen, jedoch eine Überabschätzung.

Zudem liefert Satz 8.51 hier dieselbe Abschätzung wie Satz 8.37, da beide Abschätzungen im hier vorliegenden Fall $|P| \geq v$ identisch sind, wie bereits in Bemerkung 8.52 festgestellt wurde.

9.3.2.2. Betrachtung des Falls $m = 4$, $v = 8$ und $|P| = 4$

Aus diesem Grund verringern wir die Anzahl der verschiedenen Prioritäten, sodass – bei unveränderten Werten von $m = 4$ und $v = 8$ – nun je neun Tasks dieselbe Priorität besitzen, woraus $|P| = 4$ folgt. Wegen $|P| < v$ liefert Satz 8.51 mit höchstens

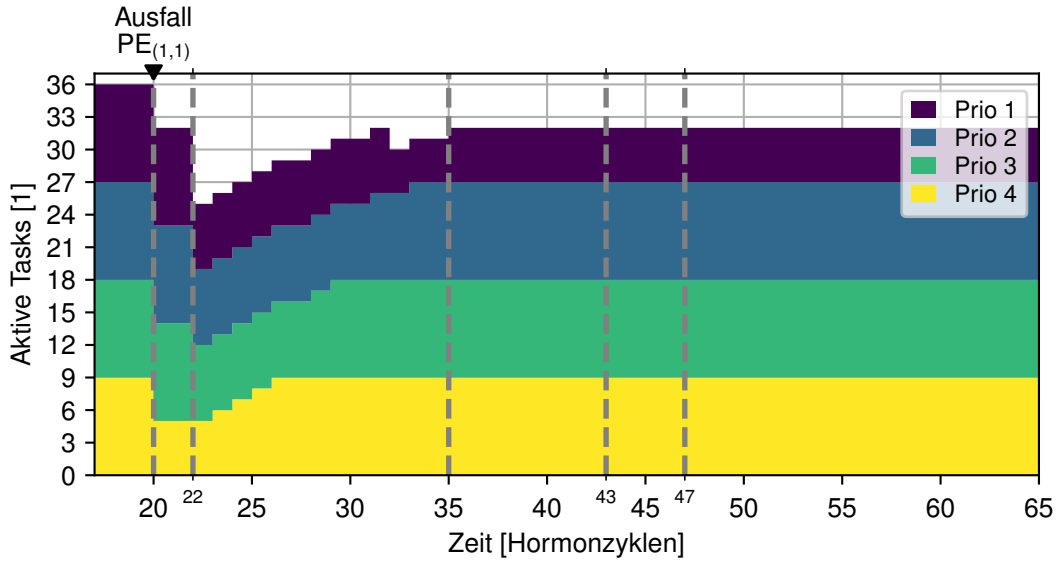


Abbildung 9.10: Selbstheilung in einer Überlastsituation für $m = 4$, $v = 8$ und $|P| = 4$ bei Verwendung des Eager Task Dropping

24 Hormonzyklen nun eine schärfere Abschätzung als Satz 8.37, der wie zuvor höchstens 28 Hormonzyklen abschätzt.

Der resultierende Verlauf der Selbstheilung ist in Abbildung 9.10 aufgetragen. Es sind in der ersten Selbstheilungsphase gut die innerhalb desselben Prioritätslevels jeweils entfallenden Wartezyklen zu erkennen. Im Vergleich zum vorherigen Szenario dauert die Selbstheilung hier nur noch zwei Phasen und ist insgesamt schneller beendet: Es werden lediglich 16 Hormonzyklen benötigt, bis das System am Ende von Zyklus 35 wieder einen stabilen Zustand erreicht, während die Abschätzungen hier acht bzw. zwölf zusätzliche Zyklen veranschlagt hätten, womit die Selbstheilung erst in den Zyklen 43 bzw. 47 beendet gewesen wäre.

Im hier betrachteten Szenario liefert die verallgemeinerte Zeitschranke also eine bessere Abschätzung als die ursprüngliche Abschätzung es vermochte. Zwar ist die Selbstheilung in der Praxis deutlich schneller beendet als abgeschätzt, doch liefert Satz 8.51 hier dennoch eine Verbesserung gegenüber der spezielleren Abschätzung aus Satz 8.37. Nichtsdestotrotz wären im Rahmen zukünftiger Forschung weitere Untersuchungen des allgemeinen Falls mit weniger Prioritäten als Tasks sinnvoll, um hier noch bessere Abschätzungen zu finden: Die hier betrachtete Zeitschranke entstand durch eine Verallgemeinerung der Analyse, die ursprünglich für den Fall paarweise verschiedener Taskprioritäten durchgeführt wurde. Würde in den Analysen von vornherein berücksichtigt, dass nur wenige verschiedene Prioritäten existieren, ließen sich möglicherweise deutlich schärfere Zeitschranken herleiten.

9.4. Vergleich der Strategien

Im Folgenden wollen wir die naive Task-Dropping-Strategie in einigen ausgewählten Szenarien mit dem Eager Task Dropping vergleichen. Hierzu nehmen wir erneut an, dass alle Tasks verschiedene Prioritäten besitzen, wie dies auch in Kapitel 8.7 im Rahmen des Vergleichs der Strategien vorausgesetzt wurde.

9.4.1. Sonderfälle $m = 1$ sowie $v = 1$

Kapitel 8.7.2 zeigte, dass das Eager Task Dropping der naiven Strategie in Bezug auf die Dauer der Selbstheilung im Allgemeinen überlegen ist. Lediglich in zwei Sonderfällen, die im Folgenden betrachtet werden, zeigt sich im Vergleich ein anderes Bild.

9.4.1.1. Sonderfall $m = 1$: Identisches Verhalten beider Strategien

Im Sonderfall $m = 1$ verhalten sich das Eager Task Dropping und die naive Task-Dropping-Strategie identisch, da hier jedes PE bei Erkennung der Überlastsituation je nach Priorität von $T_{\max}$ gerade *alle* seine (höchstens $m = 1$) Tasks stoppt, worauf eine Selbstkonfiguration der $mv = v$ höchstprioren nicht zugeteilten Tasks folgt. Im Falle des Eager Task Dropping kann sich somit auch insbesondere keine zweite Selbstheilungsphase anschließen. Dies wurde bereits in Kapitel 8.7.1 begründet.

Entsprechend überrascht es nicht, dass auch die Implementierungen der beiden Strategien ein exakt identisches Verhalten aufweisen: Abbildung 9.11 zeigt zur Illustration eine Selbstheilung für $m = 1$ und $v = 6$. Der dargestellte Verlauf kann dabei entweder als Verhalten der naiven Strategie interpretiert werden (bei Erkennung der Überlastsituation werden alle Tasks mit geringerer Priorität als 7 gestoppt) oder aber als Verhalten des Eager Task Dropping (bei Erkennung der Überlastsituation stoppt jedes PE seine niedrigstpriore Task, wenn deren Priorität geringer als 7 ist).

Die Selbstheilung dauert für beide Strategien somit 14 Hormonzyklen, was auch von Satz 8.11 für die naive Strategie bzw. Satz 8.37 für das Eager Task Dropping abgeschätzt wird: Wie in Kapitel 8.7.2 begründet wurde, liefern beide Zeitschranken für $m = 1$ dieselbe Abschätzung.

9.4.1.2. Sonderfall $v = 1$ mit $m > 1$: Überlegenheit der naiven Strategie

Einen weiteren Sonderfall stellt der Fall $v = 1$ dar, sofern nicht der gerade behandelte Fall $m = 1$ vorliegt. Ein solcher Fall wurde für $m = 5$ bereits in Abschnitt 9.2.1.1 bzw. Abbildung 9.3 für die naive Strategie sowie in Abschnitt 9.3.1.3 bzw. Abbildung 9.8 für das Eager Task Dropping evaluiert und diskutiert. Hier stoppten beide Strategien zwar dieselbe Anzahl an Tasks, die naive Strategie konnte die Selbstheilung aber dennoch schneller abschließen als das Eager Task Dropping.

Da dieses Resultat bereits in Abschnitt 9.3.1.3 diskutiert wurde und zudem die theoretischen Betrachtungen aus den Kapiteln 8.7.1 und 8.7.2 bestätigt, sei an dieser Stelle auf die weitere Betrachtung des Sonderfalls $v = 1$ mit $m > 1$ verzichtet. Stattdessen konzentrieren wir uns im Folgenden auf den allgemeinen Fall, in dem v und m jeweils

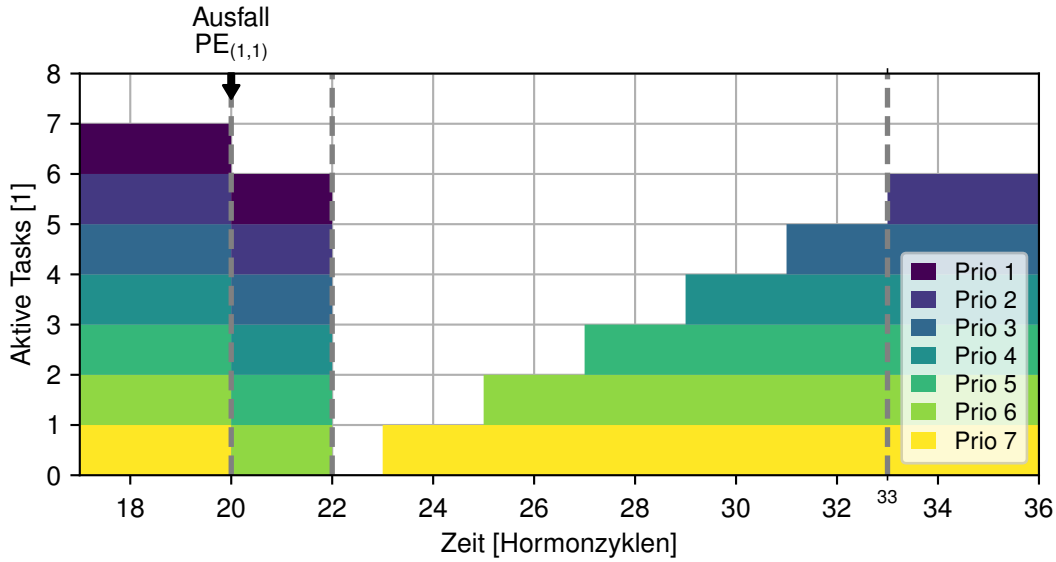


Abbildung 9.11: Selbstheilung in einer Überlastsituation für $m = 1$ und $v = 6$. Die Graphen für das Eager Task Dropping sowie die naive Task-Dropping-Strategie sind identisch.

größer als Eins sind und das Eager Task Dropping daher gemäß den Resultaten aus Kapitel 8.7.2 stets die überlegene Strategie darstellt.

9.4.2. Worst-Case-Szenario für beide Strategien

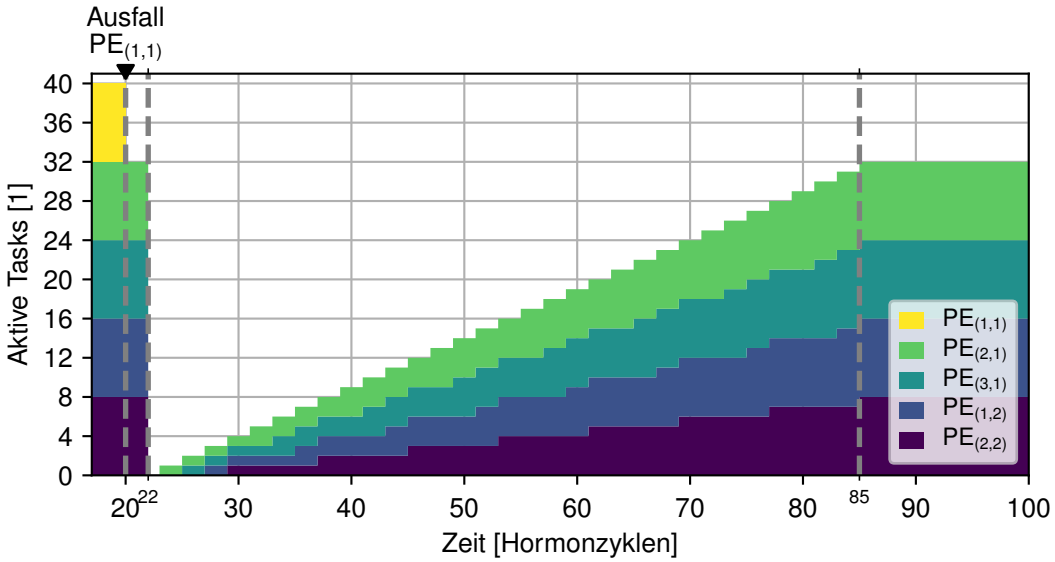
Abbildung 9.12 zeigt die Selbstheilung in einer Überlastsituation für $m = 8$ und $v = 4$ sowohl für die naive Strategie (Abbildung 9.12a) als auch das Eager Task Dropping (Abbildung 9.12b).

Die initiale Taskverteilung entspricht hier abermals dem verallgemeinerten Szenario aus Beispiel 8.43, das für beide Strategien den Worst Case darstellt. Entsprechend stoppt die naive Strategie bei Erkennung der Überlastsituation *alle* zugeteilten Tasks, gefolgt von einer vollständigen Selbstkonfiguration des Systems. Die Selbstheilung wird in Zyklus 85 nach insgesamt 66 Hormonzyklen beendet, wie von Satz 8.11 vorhergesagt.

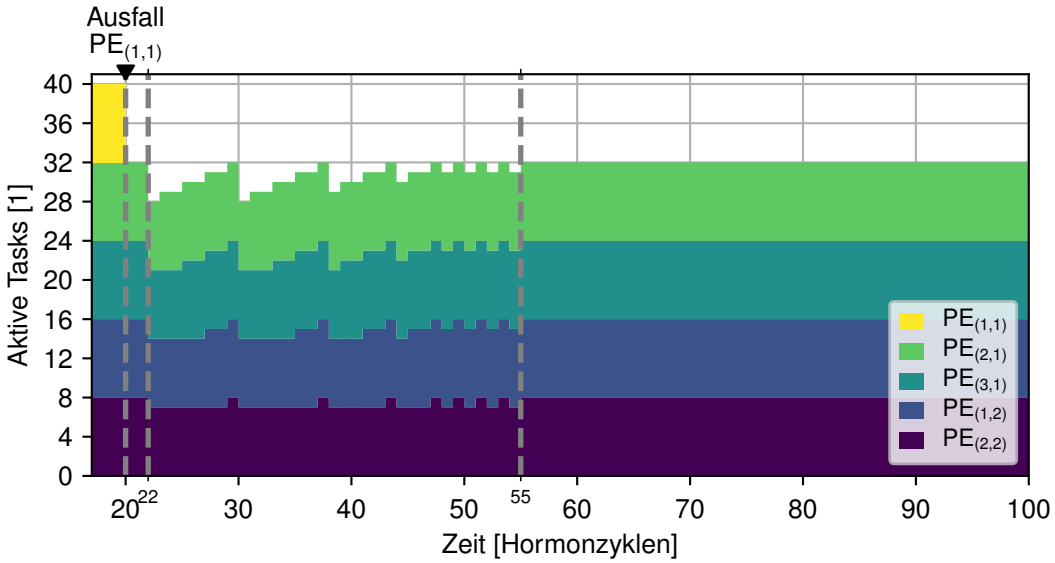
Im Gegensatz dazu ist gut zu erkennen, wie das Eager Task Dropping das System in $m = 8$ Phasen heilt und hierzu nur 36 Hormonzyklen benötigt. Folglich ist das Eager Task Dropping der naiven Strategie in diesem Szenario deutlich überlegen, was von den theoretischen Betrachtungen in Kapitel 8.7.2 ebenso vorhergesagt wurde.

9.4.3. Worst-Case-Szenario der naiven Strategie

Ein weiteres Beispiel für die Überlegenheit des Eager Task Dropping stellt Abbildung 9.13 dar. Hier wird eine Selbstheilung mit $m = 4$ und $v = 4$ für beide Strategien dargestellt, deren jeweilige Ausgangssituation das aus Abbildung 8.2 bekannte Worst-



(a) Bei Verwendung der naiven Task-Dropping-Strategie



(b) Bei Verwendung des Eager Task Dropping

Abbildung 9.12: Selbstheilung in einer Überlastsituation für $m = 8$ und $v = 4$

Case-Szenario der naiven Task-Dropping-Strategie ist: Im Falle dieser Strategie werden entsprechend *alle* zugeteilten Tasks gestoppt, wie in Abbildung 9.13a zu sehen ist. Die Selbstheilung wird in Zyklus 53 nach insgesamt 34 Hormonzyklen beendet.

Das in Abbildung 9.13b gezeigte Verhalten des Eager Task Dropping ist dagegen bereits aus Abbildung 8.3 bekannt: Für das Eager Task Dropping stellt die gegebene Situation keinesfalls den Worst Case dar, vielmehr ist nur eine einzige Selbstheilungsphase nötig und die Selbstheilung wird in Zyklus 29 nach nur 10 Hormonzyklen beendet. Selbst im Worst Case wären jedoch nur 20 Hormonzyklen nötig gewesen, womit die Selbstheilung in Zyklus 39 beendet worden wäre.

Folglich verdeutlicht dieses Szenario nicht nur ein weiteres Mal die Überlegenheit des Eager Task Dropping, sondern bestätigt auch, dass ein Worst-Case-Szenario für die naive Strategie nicht notwendigerweise auch für das Eager Task Dropping den Worst Case darstellt. Diese Eigenschaft wird uns im folgenden Abschnitt erneut begegnen.

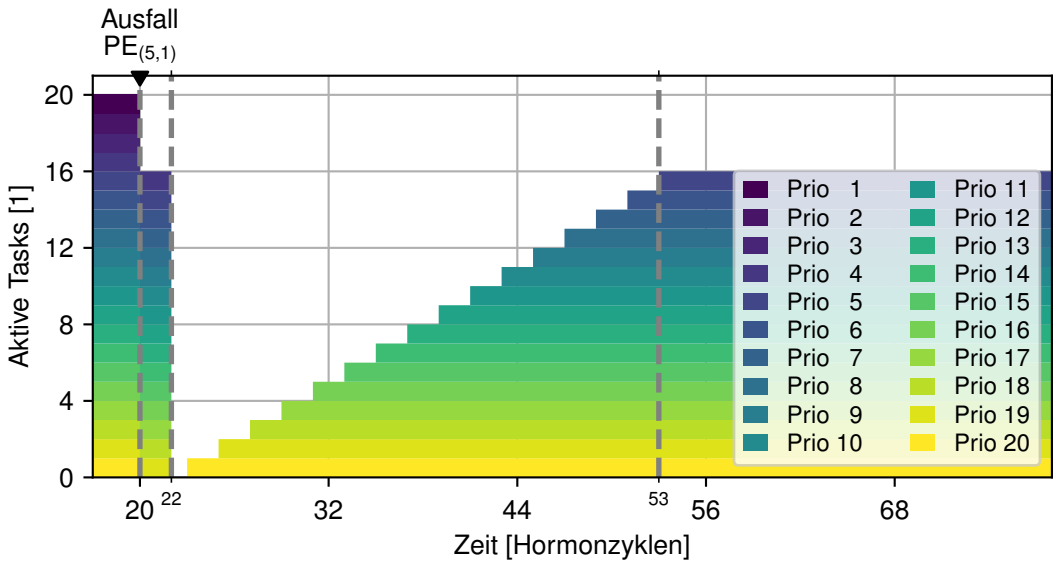
9.4.4. Mehrere PE-Ausfälle nacheinander

Während die bisherigen Evaluationen stets den Ausfall eines einzigen PEs betrachteten, möchten wir nun ein Szenario untersuchen, in dem mehrere PEs nacheinander ausfallen. Hierzu nehmen wir jedoch wie zuvor in Kapitel 8.6.3.1 an, dass PEs nur ausfallen, während sich das System in einem stabilen Zustand befindet, im Umkehrschluss also kein weiteres PE ausfällt, bevor die aktuelle Selbstheilung beendet ist.

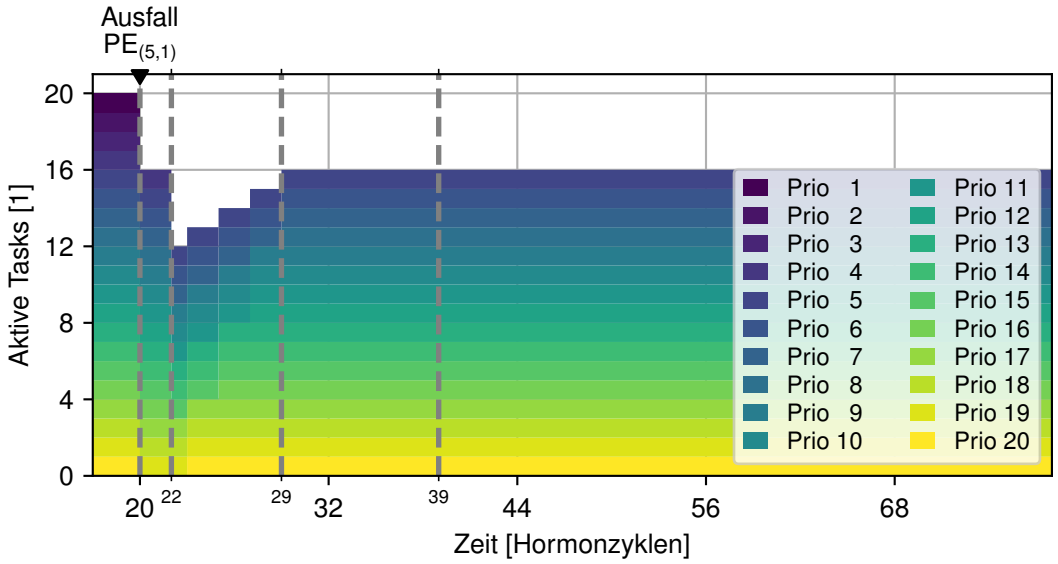
Das konkret verwendete Szenario sieht hierzu $m = 5$ Tasks pro PE bei initial sieben PEs vor, die nacheinander ausfallen. Folglich finden mehrere Selbstheilungen in Überlastsituationen mit $v = 6, \ldots, 1$ verbleibenden PEs statt, bis schließlich ein weiterer PE-Ausfall zum vollständigen Ausfall des Systems führen würde. Als initiale Taskverteilung kommt erneut das verallgemeinerte Szenario aus Beispiel 8.43 zum Einsatz, wodurch garantiert wird, dass der erste PE-Ausfall zu einem Worst-Case-Szenario beider Task-Dropping-Strategien führt. Entsprechend wurde dieses Szenario sowohl für das Eager Task Dropping als auch für die naive Strategie simuliert, um letztere als Vergleichsnormal heranziehen zu können. Hierzu wurde ferner sichergestellt, dass stets gerade jenes PE ausfiel, auf dem die höchstpriore Task 40 zugeteilt war, um die naive Strategie immer in ihren Worst Case zu zwingen.

Abbildung 9.14 zeigt nun den jeweiligen Verlauf der Selbstheilungen für das Eager Task Dropping sowie die naive Strategie. Qualitativ ist gut zu erkennen, dass jeder PE-Ausfall eine *Graceful degradation* des Systems zur Folge hat: Zwar können mit jedem ausfallenden PE insgesamt weniger Tasks im System ausgeführt werden, doch wird sukzessive die Ausführung niedrigpriorer Tasks zugunsten höherpriorer Tasks aufgegeben, um die wichtigsten Systemfunktionen so lange wie möglich lauffähig zu halten.

Quantitativ lässt sich auf den ersten Blick feststellen, dass das System bei Verwendung des Eager Task Dropping nach den ersten fünf PE-Ausfällen jeweils deutlich schneller geheilt wird, als dies bei Verwendung der naiven Task-Dropping-Strategie der Fall ist. Lediglich nach dem letzten Ausfall muss sich das Eager Task Dropping im bereits in Abschnitt 9.3.1.3 diskutierten Sonderfall $v = 1$ der naiven Strategie geschlagen geben.



(a) Bei Verwendung der naiven Task-Dropping-Strategie. Das Verhalten entspricht Abbildung 8.2.



(b) Bei Verwendung des Eager Task Dropping. Das Verhalten entspricht Abbildung 8.3.

Abbildung 9.13: Selbstheilung in einer Überlastsituation für $m = 4$ und $v = 4$

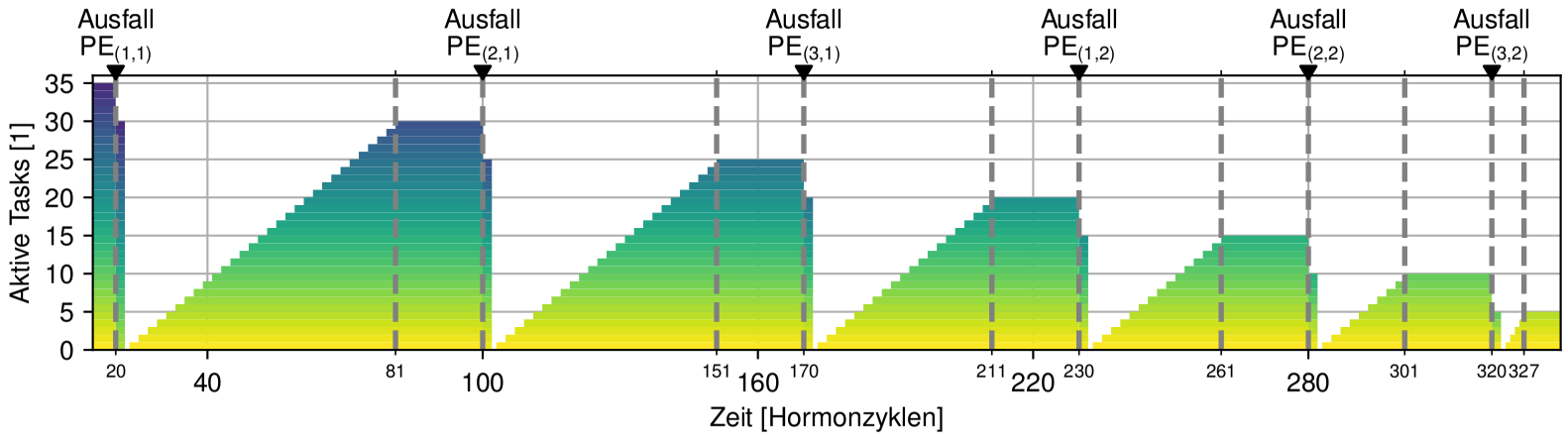
Tabelle 9.1: Dauern der Selbstheilungen aus Abbildung 9.14 sowie Abschätzungen gemäß Satz 8.11 bzw. Satz 8.37. Alle Angaben in Hormonzyklen.

		Naive Strategie			Eager Task Dropping		
v	t_{Start}	t_{Ende}	Dauer	Abschätzung	t_{Ende}	Dauer	Abschätzung
6	20	81	62	≤ 62	49	30	≤ 30
5	100	151	52	≤ 52	123	24	≤ 28
4	170	211	42	≤ 42	189	20	≤ 26
3	230	261	32	≤ 32	243	14	≤ 22
2	280	301	22	≤ 22	289	10	≤ 18
1	320	327	8	≤ 8	329	10	≤ 12

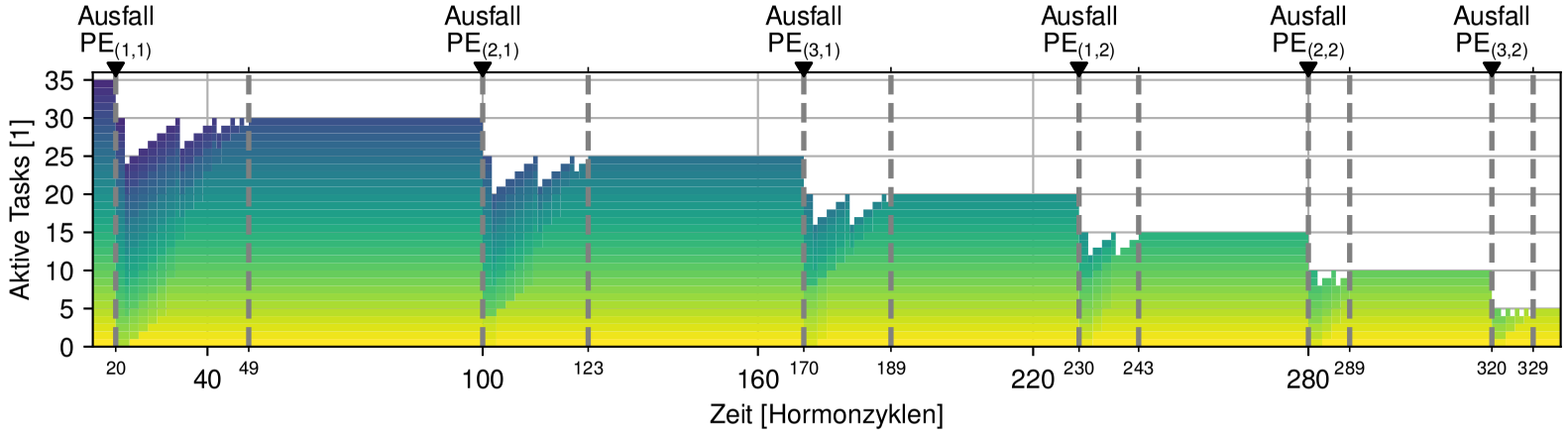
Eine genauere Betrachtung der Selbstheilungsdauern ermöglicht Tabelle 9.1, welche die Zeitpunkte t_{Start} und t_{Ende} des Beginns bzw. Endes der einzelnen Selbstheilungen, die daraus resultierenden Dauern sowie die zugehörigen Abschätzungen gemäß Satz 8.11 bzw. Satz 8.37 gegenüberstellt. Wie aufgrund der Konstruktion des Szenarios zu erwarten war, erreicht die naive Strategie stets exakt die abgeschätzte Zeitschranke. Im Fall des Eager Task Dropping wird die zugehörige Abschätzung dagegen nur für die erste Selbstheilung exakt erreicht, alle späteren Selbstheilungen eilen ihren jeweiligen Abschätzungen voraus. Dies zeigt erneut, dass ein Szenario, das für die naive Strategie den Worst Case darstellt, sich bei Verwendung des Eager Task Dropping unter Umständen deutlich gutmütiger verhält.[1] Auch eine Betrachtung der Anzahl der tatsächlich benötigten Selbstheilungsphasen macht dies deutlich: Während für die erste, dem Worst Case entsprechende Selbstheilung noch alle $m = 5$ theoretisch möglichen Selbstheilungsphasen nötig sind, werden für die folgenden Selbstheilungen nur jeweils 3, 3, 2, 2 und 4 Phasen benötigt, um die Selbstheilung abzuschließen.

Das hier betrachtete Szenario ist insbesondere für praktische Anwendungen relevant: Mithilfe der prioritätsbasierten Task-Decision-Strategie ist es trotz mehrerer aufeinanderfolgender PE-Ausfälle möglich, innerhalb einer beschränkten Zeitspanne sicherzustellen, dass in einer Überlastsituation die höchstprioren $m \cdot v$ Tasks ausgeführt werden. Im Vergleich dazu kann diese Eigenschaft bei Verwendung der *aggressiven* Task-Decision-Strategie nicht garantiert werden, wie Abbildung 9.14d verdeutlicht: Da diese Strategie keine Taskprioritäten beachtet, werden infolge der einzelnen PE-Ausfälle keinerlei Maßnahmen eingeleitet, da alle PEs aufgrund der Überlastsituation voll ausgelastet sind. Bedingt durch die initiale Taskverteilung im betrachteten Worst-Case-Szenario fallen hier sogar die *höchstprioren* Tasks zuerst aus, wohingegen die *niedrigprioren* Tasks weiter ausgeführt werden. Da im Allgemeinen jedoch weder die

[1] Umgekehrt gilt dies jedoch nicht: Nach Satz 8.53 kann das Eager Task Dropping niemals mehr Tasks als die naive Strategie stoppen. Lediglich im Fall $v = 1$ mit $m > 1$, der in Kapitel 8.7.2 theoretisch untersucht und in Abschnitt 9.4.1.2 sowie anhand des gerade betrachteten Szenarios experimentell bestätigt wurde, ist die naive Strategie dem Eager Task Dropping in Bezug auf die *Dauer* der Selbstheilung überlegen; in allen übrigen Fällen ist sie dagegen stets unterlegen.

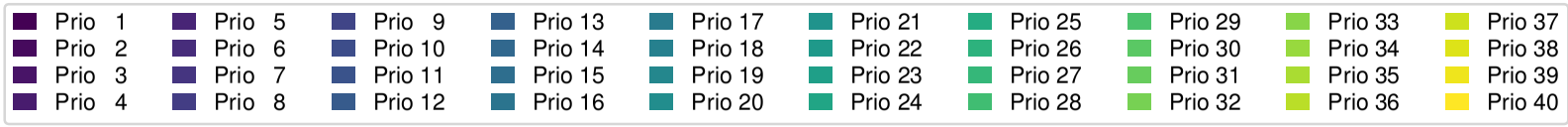


(a) Bei Verwendung der naiven Task-Dropping-Strategie. Die Legende befindet sich in Abbildung 9.14c.

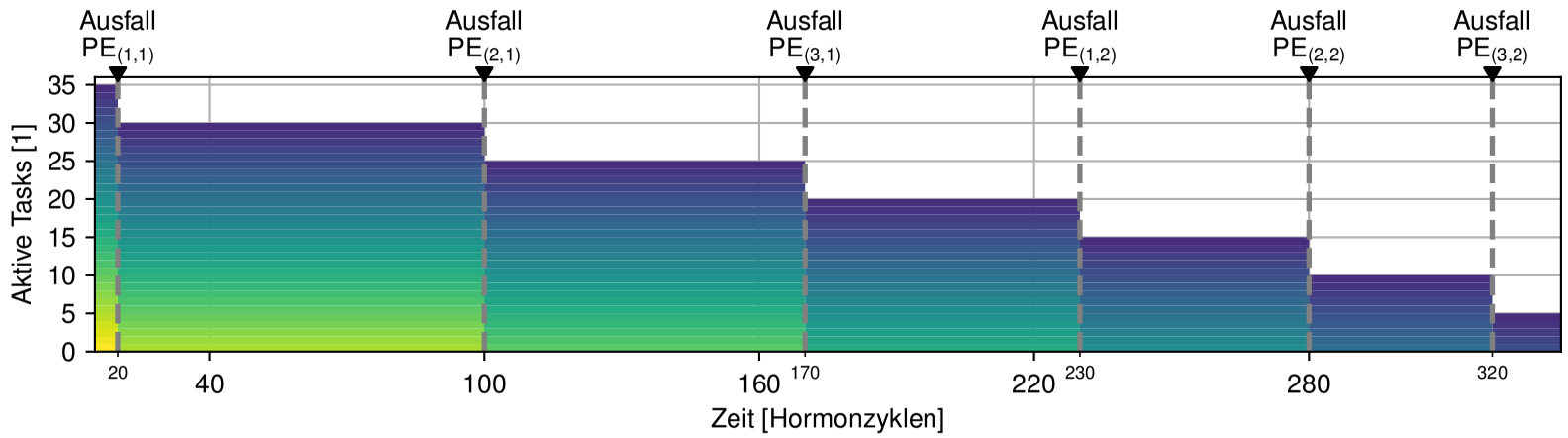


(b) Bei Verwendung des Eager Task Dropping. Die Legende befindet sich in Abbildung 9.14c.

Abbildung 9.14: Selbstheilungen in Überlastsituationen für $m = 5$ und $v = 6, \ldots, 1$



(c) Legende der Graphen aus den Abbildungen 9.14a, 9.14b und 9.14d



(d) Bei Verwendung der aggressiven Task-Decision-Strategie. Die Legende befindet sich in Abbildung 9.14c.

Abbildung 9.14: Selbstheilungen in Überlastsituationen für $m = 5$ und $v = 6, \ldots, 1$ (Fortsetzung)

initiale Taskverteilung noch die Reihenfolge der PE-Ausfälle vorhersagbar sind, lässt sich auch die Reihenfolge der Taskausfälle nicht vorhersagen, entsprechend muss also keineswegs wie hier der für das System fatale Worst Case eintreten. Fest steht jedoch, dass die aggressive Task-Decision-Strategie nicht imstande ist, auf Überlastsituationen angemessen zu reagieren.

Bei Verwendung der prioritätsbasierten Task-Decision-Strategie dagegen vermochten es sowohl die naive Task-Dropping-Strategie als auch das Eager Task Dropping, das System stets innerhalb einer bestimmten Zeit in einen definierten Zustand zu überführen und damit zu stabilisieren. Die in Kapitel 8.7.2 theoretisch hergeleitete Überlegenheit des Eager Task Dropping für $v > 1$ verbleibende PEs wurde im hier betrachteten Szenario vollumfänglich bestätigt. Überdies ergab sich, dass das Eager Task Dropping hier nur sehr selten seinen Worst Case erreichte, obwohl dieser im Falle der naiven Strategie stets auftrat. Für die Analyse im Rahmen der Entwicklung eines Echtzeitsystems muss zwar dennoch stets das Auftreten des Worst Case angenommen werden, trotzdem zeigt dessen hier seltenes Auftreten ein weiteres Mal die Überlegenheit des Eager Task Dropping.

9.5. Abschließende Bemerkungen

In diesem Kapitel wurde die Selbstheilung in Überlastsituationen unter Verwendung zweier Task-Dropping-Strategien evaluiert: der aus Kapitel 8.5 bekannten *naiven* Task-Dropping-Strategie sowie dem in Kapitel 8.6 vorgestellten *Eager Task Dropping*. Die betrachteten Szenarien bestätigten hierbei vollumfänglich die theoretisch hergeleiteten Zeitschranken. Insbesondere bedeutet dies, dass das Eager Task Dropping der naiven Strategie aufgrund ihres deutlich besseren Zeitverhaltens zumeist vorzuziehen ist. Lediglich im Fall, dass nur ein einziges funktionsfähiges PE verbleibt, kann die naive Strategie eine schnellere Selbstheilung garantieren, wobei in einem solchen Fall jedoch die Frage berechtigt ist, ob die auf nur noch einem einzigen PE zuteilbaren Tasks überhaupt noch eine ausreichend große Teilfunktionalität des Systems realisieren können oder das System bereits effektiv als vollständig ausgefallen betrachtet werden muss.

In [HB21b] wurden zudem weitere Szenarien evaluiert, die jedoch aus Platzgründen an dieser Stelle nicht noch einmal behandelt werden sollen, zumal sie vollumfänglich in Einklang mit den theoretischen Ergebnissen stehen.

Auf einen abschließenden Vergleich zwischen der AHS-Middleware und dem Simulator, in dem die hier beschriebenen Evaluationen durchgeführt wurden, wurde aus Platzgründen ebenfalls verzichtet, da bereits in Kapitel 7.6 gezeigt wurde, dass sich beide Implementierungen in denselben Szenarien identisch verhalten.

Bevor wir nun in Kapitel 11 die wesentlichen Ergebnisse dieser Dissertation zusammenfassen und Anknüpfungspunkte für zukünftige Forschungen identifizieren, wollen wir im nächsten Kapitel noch einige Konzepte betrachten, die ähnliche Ziele verfolgen wie das AHS und die im Rahmen dieser Dissertation entwickelte prioritätsbasierte Task-Decision-Strategie.

10 Verwandte Konzepte und Arbeiten

Im Rahmen dieser Dissertation wurde eine prioritätsbasierte Erweiterung des AHS entwickelt. Dieses dient der Zuteilung von Tasks an die Knoten eines verteilten Systems, während die prioritätsbasierte Erweiterung es ermöglicht, die Tasks gemäß einer anhand von *Zuteilungsprioritäten* definierten Reihenfolge zuzuteilen. Hierbei wird insbesondere ein Augenmerk auf die Erhöhung der Toleranz gegenüber Knotenausfällen, die zumeist mehrere Tasks betreffen, sowie Ausfällen einzelner Tasks gelegt: In solchen Fällen werden die betroffenen Tasks automatisch auf die verbleibenden Knoten migriert, wobei gegebenenfalls Tasks niedriger Priorität gestoppt werden, um die nötigen Rechenkapazitäten für höherpriore Tasks zu schaffen.

Das Problem der Taskverteilung in verteilten Systemen wurde in der Literatur bereits häufig für verschiedene Szenarien untersucht. Im Folgenden wollen wir daher einige Lösungsansätze betrachten sowie Gemeinsamkeiten und insbesondere Unterschiede zum prioritätsbasierten AHS herausarbeiten. Hierzu wurden diese Lösungsansätze in zwei Kategorien unterteilt: Die im nächsten Abschnitt vorgestellten Offline-Methoden sowie die in Abschnitt 10.2 beschriebenen Online-Methoden.

10.1. Offline-Methoden

Die erste betrachtete Kategorie bilden Ansätze zur Taskverteilung, die eine *feste* Zuteilung der Tasks an die Knoten des verteilten Systems vorsehen. Auf Knotenausfälle kann zur Laufzeit dann reagiert werden, indem auf eine für das konkrete Ausfallszenario angepasste Taskverteilung gewechselt wird. Diese muss im Rahmen des Systementwurfs vorberechnet worden sein, entsprechend werden die dieser Kategorie zugehörigen Methoden im Folgenden als *Offline-Methoden* bezeichnet.

Diese Vorgehensweise unterscheidet sich grundlegend vom AHS, das statt starrer und zur *Designzeit* vorgegebener Taskverteilungen auf die Selbstorganisation der PEs setzt, um die Tasks zur *Laufzeit* zu verteilen. Entsprechend erlaubt das AHS eine höhere Dynamik: Offline-Methoden können zur Laufzeit nur auf bereits zur Designzeit vorausgesehene Ereignisse reagieren, da ansonsten keine vorberechnete Adaptionsstrategie existieren kann. Das AHS dagegen benötigt aufgrund der Nutzung von Selbstorganisation keine Antizipation aller möglichen Szenarien – stattdessen ergibt sich die Adaption des Systems automatisch aus der selbstorganisierenden Interaktion der PEs.

Nichtsdestotrotz können Offline-Methoden dynamischen Methoden wie dem AHS in einigen Aspekten überlegen sein, erlauben sie doch eine *optimale* Reaktion auf Knotenausfälle: Sofern die entsprechenden Szenarien zur Designzeit vorausgesehen werden, so kann unter Einsatz von genügend Rechenzeit eine optimale Taskverteilung gefunden werden. Das AHS muss dagegen zur Laufzeit versuchen, eine möglichst gute Reaktion

auf geänderte Systemzustände zu finden. Im Anschluss kann im Rahmen der Selbstoptimierung versucht werden, diese weiter zu verbessern. Aufgrund des Rechenaufwandes sowie der beschränkten Sichtweise der einzelnen PEs auf das Gesamtsystems ist eine *optimale* Reaktion dagegen im Allgemeinen nicht möglich, zumal der zum Finden einer solchen nötige Rechenaufwand zumeist nicht mit den geforderten Echtzeiteigenschaften vereinbar wäre.

Ebenso sind Zuteilungsprioritäten, wie sie für das AHS im Rahmen dieser Dissertation untersucht wurden, im Allgemeinen für Offline-Methoden nicht nötig, da die Taskverteilung nicht erst zur Laufzeit bestimmt werden muss. Auch auf Überlastsituationen kann – Antizipation der konkreten Situation und Vorausberechnung einer passenden Adaptionsstrategie vorausgesetzt – reagiert werden, ohne dass Zuteilungsprioritäten zwingend nötig wären: Die in solchen Situationen zuzuteilenden Tasks könnten auch manuell vom Systementwickler festgelegt werden.

Im Folgenden werden wir zwei Vertreter der Kategorie der Offline-Methoden betrachten. Zuvor sei jedoch angemerkt, dass diese kurze und allgemeine Betrachtung der wesentlichen Unterschiede zwischen Offline-Methoden sowie dem AHS bereits zeigt, dass beide Varianten der Taskverteilung verschiedene Stärken besitzen. Entsprechend existiert keine allgemeingültige Präferenz für eine der beiden Varianten: Vielmehr muss abgewogen werden, mit welcher der Methoden die Systemanforderungen besser zu realisieren sind. Insbesondere das in Abschnitt 10.1.2 vorgestellte ScOSA-Projekt wurde aufgrund des geplanten Einsatzzwecks explizit als Offline-Methode konzipiert. Zuvor wollen wir jedoch das bereits in Kapitel 1.3 erwähnte AutoKonf-Projekt genauer betrachten.

10.1.1. AutoKonf

Im Automotive-Bereich ist die Toleranz gegenüber Fehlern sicherheitskritischer Komponenten wichtig. Insbesondere in Hinblick auf das vollautonome Fahren kann der Ausfall von Lenkung oder Bremse nicht toleriert werden, da mit dem Fahrer die letzte Instanz entfällt, die in einer Notsituation mithilfe der mechanisch betätigten Lenkung und Bremse eingreifen könnte. Das AutoKonf-Projekt[1] [Osz+18; Orl+19] nutzt zur Sicherstellung der Funktionalität von Lenk- und Bremssteuerung ein ausgefeilteres Redundanzkonzept als eine einfache 1:1-Redundanz, bei der jede funktionale Einheit ein identisches Backup an ihre Seite gestellt bekäme. Stattdessen teilen sich hier die Steuergeräte für Lenkung und Bremse ein einziges Backup-Steuergerät, was in einer 2:1-Redundanz resultiert und die Kompensation eines einzigen Ausfalls von Bremse *oder* Lenkung ermöglicht. Während dieses Konzept zwar im Vergleich mit einer 1:1-Redundanz zu Kostenersparnis sowie Gewichtsreduktion beiträgt und dabei dennoch ein gewisser Grad an Fehlertoleranz beibehalten werden kann, so führt der Ausfall von zwei Steuergeräten jedoch unweigerlich zum Verlust einer der beiden Funktionalitäten und ist daher für das Gesamtsystem kritisch.

[1] **Auto**matisch re**konf**igurierbare Aktorikansteuerungen für ausfallsichere automatisierte Fahrfunktionen

Das AutoKonf-Projekt kann als ein Extrembeispiel der Klasse der Offline-Taskverteilungsalgorithmen angesehen werden: Die Tasks „Lenkung“ und „Bremse“ werden standardmäßig auf den für sie vorgesehenen Steuergeräten ausgeführt. Bei einem Ausfall eines dieser Steuergeräte wird dessen Task auf das gemeinsame Backup-Steuergerät migriert. Diese Adaptionsregeln sind bereits im Voraus festgelegt und ergeben sich nicht erst zur Laufzeit. Im AutoKonf-Projekt existieren nur drei Steuergeräte, was in acht möglichen Systemzuständen resultiert, womit die Anzahl der nötigen Adaptionsszenarien sehr übersichtlich ist. Zudem umfassen vier dieser acht Systemzustände weniger als zwei funktionsfähige Steuergeräte, in denen mindestens eine der Teilfunktionen des Systems vollständig ausfällt.

Prinzipiell ließe sich dieses Szenario auch mithilfe des AHS unter Verwendung von zwei Tasks – je eine für die Systemfunktionen „Lenkung“ und „Bremse“ – und drei PEs nachbilden. Dies würde einen vergleichbaren Grad der Fehlertoleranz ermöglichen. Nichtsdestotrotz wäre es denkbar, die Funktionen deutlich feingranularer in einzelne Tasks aufzuteilen, die flexibler auf die zur Verfügung stehenden PEs verteilt werden könnten: Hierzu wurde in [BF19; BHF19; HBF20] gezeigt, dass das Grundkonzept des AHS sich prinzipiell für den Einsatz in Automotive-Umgebungen eignet und hierbei bei gleicher Anzahl PEs sogar ein höherer Grad der Fehlertoleranz erreicht werden kann als mit einem vergleichbaren n:1-Redundanzkonzept, wie es für $n = 2$ vom AutoKonf-Projekt realisiert wird.

10.1.2. ScOSA

Als weiteren Vertreter der Offline-Methoden wollen wir die Taskverteilung im ScOSA-Projekt[1] [Deub] des Deutschen Zentrums für Luft- und Raumfahrt (DLR) betrachten, welche in den Publikationen [Kov+17; Kov+20] beschrieben wird.

Das Ziel des ScOSA-Projektes ist es, eine zuverlässige Plattform für zukünftige Raumfahrzeuge zu entwickeln. Diese haben für ihre Missionen einen großen Bedarf an Rechenkapazität. Gleichzeitig sind sie auf den Missionen kosmischer Strahlung ausgesetzt, die beispielsweise *Single Event Upsets* auslösen kann, wie bereits in Kapitel 1.3 beschrieben. Folglich ist zur Vermeidung von Fehlverhalten sowie Ausfällen der Einsatz strahlungsgehärteter Hardware sinnvoll. Die auf dem Markt verfügbaren strahlungsgehärteten Mikroprozessoren bieten jedoch nur eine geringe Performance, weshalb es aufwendig und teuer wäre, die Anforderungen mit dieser Hardware abzudecken. Aus diesem Grund werden für ScOSA strahlungsgehärtete Einheiten mit aus Standardhardware *(Commercial off-the-shelf, COTS)* aufgebauten Einheiten kombiniert, die eine deutlich höhere Rechenleistung bieten. Da diese Einheiten nicht strahlungsgehärtet und insbesondere nicht für den Weltraumeinsatz vorgesehen sind, ist für diese mit einer erhöhten Ausfallwahrscheinlichkeit zu rechnen. Daher legt das ScOSA-Projekt sein Augenmerk auf die Toleranz gegenüber Ausfällen COTS-basierter Einheiten.

Konkret bilden alle diese Einheiten jeweils Knoten eines verteilten Systems, dessen Topologie jedoch nicht festgeschrieben ist und sich insbesondere durch Ausfälle ändern

[1] **Sc**alable **o**n-board Computing for **S**pace **A**vionics

kann. Alle zur aktuell auszuführenden Mission gehörenden Tasks werden auf die einzelnen Knoten verteilt, wobei eine offline vorberechnete Zuweisung verwendet wird. Um Ausfälle zu erkennen, existiert ein Masterknoten, der die übrigen Knoten überwacht und selbst wiederum von einem oder mehreren Observerknoten überwacht werden kann. Wird ein Ausfall erkannt, so erfolgt eine Rekonfiguration durch Anwendung eines Adaptionsszenarios, das ebenfalls offline vorberechnet wurde.

Hierbei existieren zwei Gründe für die Nutzung einer Offline-Methode zur Taskverteilung anstatt einer dynamischen Lösung wie beispielsweise dem AHS: Zum einen soll ein deterministisches Systemverhalten erzielt werden, das jederzeit in der Bodenkontrollstation nachvollzogen werden kann. Zum anderen wird die Nutzung einer *optimalen* Taskverteilung präferiert, was praktisch nur durch eine Offline-Vorberechnung möglich ist [Kov+17].

Konkret werden alle möglichen Adaptionsszenarien in einem *Rekonfigurationsgraphen* vorgehalten: Zu jedem Systemzustand existiert eine vorberechnete Verteilung der Tasks auf die funktionsfähigen Knoten, bei einem Knotenausfall wird anhand der entsprechenden Kante des Rekonfigurationsgraphen die neue Taskverteilung geladen und im System realisiert. Hierbei werden insbesondere auch Überlastsituationen betrachtet: Blätter des Rekonfigurationsgraphen stellen sogenannte *„safe modes“* dar, in denen die volle Systemfunktionalität nicht mehr realisiert werden kann. In diesen Situationen werden nur Minimalfunktionen des Raumfahrzeugs bereitgestellt und Kommandos der Bodenkontrollstation erwartet; welche Tasks in den *safe modes* ausgeführt werden sollen, muss wiederum zur Designzeit festgelegt worden sein.

Die einzelnen Taskverteilungen des Rekonfigurationsgraphen werden hierbei je nach Problemgröße mithilfe zweier Methoden gewonnen [Kov+20]: Für kleine Probleminstanzen wird das Problem in ein Satisfiability-Modulo-Theories-Problem (SMT) [Bar+09] umgewandelt und mithilfe eines SMT-Solvers gelöst. Für große Probleminstanzen ist dies dagegen aufgrund des hohen Rechenaufwandes nicht mehr praktisch durchführbar, weshalb in diesen Fällen auf einen genetischen Algorithmus zurückgegriffen wird, um eine möglichst gute Näherungslösung zu erhalten.

Offensichtlich weist das ScOSA-Projekt als Vertreter der Klasse der Offline-Methoden zunächst deren wesentliche allgemeine Unterschiede zum AHS auf. Konkret werden insbesondere prinzipbedingt keine Prioritäten benötigt; lediglich die Auswahl, welche Tasks in den *safe modes* ausgeführt werden sollen, könnte man als *binäre* Prioritäten auffassen. Einen weiteren wesentlichen Unterschied stellt jedoch die Kommunikationsstruktur des Systems dar: Das AHS nimmt eine Netzwerktopologie an, in der insbesondere Broadcasts zum Austausch der Hormone unterstützt werden. Im ScOSA-Projekt wird eine solche Eigenschaft dagegen nicht vorausgesetzt; die Topologie des Netzwerkes kann sich durch Knotenausfälle sogar zur Laufzeit ändern. Dieser Umstand stellt – in Kombination zu den genannten Gründen, die zur Festlegung auf eine Offline-Methode geführt haben – ein Hindernis für den Einsatz des AHS in seiner derzeitigen Form als alternative Taskverteilungsmethode für das ScOSA-Projekt dar.

Nach dieser kurzen Betrachtung der AutoKonf- und ScOSA-Projekte als Vertreter der Klasse von *Offline*-Methoden zur Taskverteilung in verteilten Systemen möchten wir im Folgenden einige *Online*-Methoden zur Taskverteilung untersuchen.

10.2. Online-Methoden

Bei Verwendung einer Online-Methode wird die Taskverteilung nicht bereits zur Designzeit, sondern erst zur Laufzeit des Systems berechnet. Entsprechend fallen das AHS sowie generell auf Prinzipien des Organic Computing setzende Methoden in diese Kategorie: Die Verlagerung von Entscheidungen von der Designzeit *(offline)* in die Laufzeit *(online)* ist gerade eines der Klassifikationsmerkmale des Organic Computing [TSM17; MT17].

Eine der direkten Konsequenzen dieser Verlagerung ist, dass Online-Methoden im Gegensatz zu Offline-Methoden zur Laufzeit in begrenztem Maße auch auf Situationen reagieren können, die zur Designzeit nicht vorhergesehen wurden – vorausgesetzt, das System verfügt über die nötigen Mechanismen, dies selbstständig bewerkstelligen zu können. Dafür wird im Allgemeinen jedoch keine *optimale* Taskverteilung erreicht, da die Komponenten des Systems zum einen meist nur eine begrenzte Sicht auf das Gesamtsystem besitzen und zum anderen insbesondere in Echtzeitsystemen nur begrenzt viel Zeit zur Verfügung steht, um die Tasks zu verteilen oder gar im Fall eines Ausfalls neu zu verteilen. Offline-Methoden dagegen steht zur Designzeit potenziell nahezu unbegrenzt viel Rechenzeit zur Verfügung, um optimale Taskverteilungen für alle betrachteten Systemzustände zu berechnen.

Im Folgenden wollen wir nun einige Vertreter der Klasse der Online-Methoden betrachten.

10.2.1. Multiagentensysteme

Multiagentensysteme sind Systeme, die aus mehreren interagierenden *Agenten* bestehen. Ein Agent ist hierbei eine in einer bestimmten Umgebung befindliche Einheit, die selbstständig Aktionen durchführen kann, um ein bestimmtes Ziel auszuführen [Woo09]. Diese Definition ist sehr allgemein, entsprechend kann auch das AHS als Multiagentensystem aufgefasst werden: Die PEs entscheiden selbstständig über die Übernahme von Tasks (*Aktionen*) mit dem *Ziel*, im Gesamtsystem (*Umgebung*) alle Tasks auszuführen. Hierzu interagieren sie durch den Austausch von Hormonen miteinander.

Für verschiedene Arten von Multiagentensysteme wurden in der Literatur unterschiedliche Methoden zur Taskverteilung beschrieben, von denen im Folgenden zwei der wichtigsten kurz vorgestellt werden sollen.

10.2.1.1. Contract-Net-Protokolle

Die Klasse der *Contract-Net-Protokolle (CNP)* geht zurück auf das ursprüngliche Contract-Net-Protokoll nach Smith [Smi80]. In diesem Protokoll können *Manager*-Agenten Tasks ausschreiben, woraufhin *Contractor*-Agenten sich mit Geboten um deren Ausführung bewerben können. Der Manager wählt das beste Gebot aus und erteilt dem Contractor den Zuschlag. Hierbei können Contractor-Agenten gleichzeitig die Rolle des Managers für andere Tasks einnehmen oder eine gewonnene Task in Untertasks zerlegen und diese wiederum selbst ausschreiben. Nach Abarbeitung einer Task werden dem Manager die Ergebnisse der Ausführung übermittelt.

Im Vergleich zum AHS fällt hier auf, dass das CNP nicht vollständig dezentral ist: Der Ausfall eines Contractors kann zwar durch den Manager erkannt werden, woraufhin er die zugehörige Task erneut ausschreiben kann. Fällt jedoch ein Manager aus, so sind die Contractors nicht imstande, seine Tasks neu auszuschreiben, weshalb ein Manager einen Single-Point-of-Failure für die von ihm ausgeschriebenen Tasks darstellt. Erweiterungen wie [KRD03] versuchen, dieses Problem in bestimmten Situationen durch Einführung eines Fehlerbehandlungsdienstes zu entschärfen, doch führt ein Ausfall dieses zentralisiert ausgelegten Dienstes trotzdem zu einem Rückfall in das ursprüngliche CNP.

Ferner ist im CNP zwar prinzipiell denkbar, für jede Taskausschreibung eine Frist für die Abgabe von Geboten anzugeben, doch garantiert dies keine echtzeitfähige Taskverteilung, da nicht sichergestellt werden kann, dass innerhalb dieser Frist tatsächlich ein Contractor ein Gebot abgibt. Ebenfalls dürften Tasks zur Wahrung von Echtzeiteigenschaften nicht weiter unterteilt werden, da die Ausschreibung von Untertasks die Zeit bis zur vollständigen Zuteilung aller Teilfunktionen der Ursprungstask weiter verzögert.

Ein verbessertes Contract-Net-Protokoll Im ursprünglichen CNP sind keine Taskprioritäten vorgesehen, wobei es jedem Manager natürlich dennoch frei steht, die Tasks in einer bestimmten Reihenfolge auszuschreiben. Dies führt jedoch nicht zu einer global geordneten Reihenfolge der Taskzuteilung, da mehrere Manager Tasks verschiedenster Prioritäten gleichzeitig ausschreiben können. Dennoch wird diese Idee in [ZWS19] aufgegriffen, um ein verbessertes CNP zu realisieren, das jedoch vor allem eine Senkung der Kommunikationslast zum Ziel hat. Hierzu besitzt jeder Agent eine bestimmte *Antwortschwelle*, die sich dynamisch ändern kann. Ebenso enthält jede Taskausschreibung einen *Stimuluswert*; ein Agent bietet nur auf Tasks, deren Stimulus größer als seine Antwortschwelle ist. Die Kommunikation in diesem Protokoll verwendet Ausschreibungs- und Gebots-*Pheromone*, die – ähnlich den Hormonen des AHS – in die Umgebung ausgeschüttet werden und nach einer bestimmten Zeit verfallen.

Trotz dieser Modifikationen gegenüber dem ursprünglichen CNP ist weiterhin nur eine begrenzte Selbstheilung eines Contractor-Ausfalls durch Neuausschreibung der betroffenen Tasks möglich, da die Manager weiterhin Single-Points-of-Failure für die von ihnen ausgeschriebenen Tasks sind. Auch Echtzeitverhalten kann weiterhin nur schwer garantiert werden: Unter Umständen muss ein Manager eine Task mehrfach mit steigenden Stimuli ausschreiben, bis mindestens ein Agent darauf bietet. Zudem dürfen Agenten eine Task nach Erhalt des Zuschlags sogar ablehnen, was ebenfalls zu einer Verzögerung führt, da die Task dann entweder an den zweitbesten Contractor vergeben oder neu ausgeschrieben werden muss.

Aufgrund der Verwendung biologisch inspirierter Kommunikation über Pheromone sowie der Berücksichtigung von Vergabeprioritäten stellt dieses Protokoll eine erwähnenswerte Erweiterung des ursprünglichen CNP dar, obwohl die besagte Publikation im Detail einige Unklarheiten und Widersprüche aufweist. Nichtsdestotrotz möchte der Autor sich an dieser Stelle entschieden von der darin beschriebenen Motivation distanzieren, welche in der Verteilung von Tasks an auf einem Schlachtfeld autonom

agierende Agenten besteht.

10.2.1.2. Konsensbasierte parallele Auktion und Ausführung

In [Das+11] wird ein verteiltes Verfahren zur Taskverteilung in einem Multirobotersystem vorgestellt. Hierbei wird – wie im CNP – ebenfalls der Mechanismus der Taskauktion verwendet. Nach der Gebotsabgabe für eine Task wird in einer verteilten Konsensfindungsphase entschieden, welcher Roboter die Task übernimmt, woraufhin dieser sofort mit deren Ausführung beginnt. Ein Roboter kann in diesem Verfahren höchstens eine Task gleichzeitig ausführen und bemüht sich schon während deren Fertigstellung um potenzielle Anschlusstasks. Dadurch soll im System möglichst früh ein Konsens für diese Tasks entstehen, damit sie von je einem Roboter sofort nach Beendigung seiner aktuellen Task übernommen werden können.

Während besagte Publikation von homogenen Robotern ausgeht und keine Taskprioritäten unterstützt, wird die Methode in [Das+15] erweitert: Hier können die Tasks verschiedene Prioritäten haben, welche die Vergabereihenfolge beeinflussen und daher den Prioritäten im AHS sehr ähnlich sind. Außerdem entfällt die Beschränkung auf homogene Roboter, stattdessen können nun heterogene Roboter mit unterschiedlichen Fähigkeiten zum Einsatz kommen, die verschiedenartige Tasks ausführen können; als Anwendungs- und Evaluationsszenario wird die Ausführung von Aufgaben in einem Pflegeheim durch verschieden spezialisierte Roboter angeführt.

Auch wenn diese Methode auf den ersten Blick prinzipiell mit dem prioritätsbasierten AHS vergleichbar erscheint, existieren dennoch einige Unterschiede: Zum einen kann ein PE des AHS prinzipiell mehrere Tasks gleichzeitig ausführen, während ein Roboter maximal eine Aufgabe gleichzeitig durchführen kann. Zum anderen sieht die vorgestellte Methode weder Selbstoptimierung noch Selbstheilung vor.

10.2.2. Taskverteilung in Sensornetzen

Ein weiteres Anwendungsgebiet, in dem Tasks auf die Knoten eines verteilten Systems verteilt werden müssen, stellen Sensornetze dar. Ein Sensornetz, auch *Wireless Sensor Network (WSN)* genannt, besteht aus vielen Sensorknoten, die drahtlos untereinander kommunizieren können. Diese sind typischerweise batteriebetrieben und können ihren jeweiligen Fähigkeiten entsprechend verschiedene Messdaten erfassen. Die erfassten Daten werden typischerweise innerhalb des Netzwerks vorverarbeitet und anschließend an eine zentrale Basisstation weitergeleitet [Amm+16].

Um die gewünschten Daten erfassen und vorverarbeiten zu können, müssen in einem WSN entsprechende Tasks auf die Sensorknoten verteilt werden. Aufgrund der drahtlosen Vernetzung sowie dem batteriebedingt beschränkten Energievorrat der Knoten ist stets mit temporären oder gar permanenten Verbindungsstörungen sowie Knotenausfällen zu rechnen. Da diese Ereignisse sich im Allgemeinen nicht vorhersagen lassen, bieten sich hier Online-Methoden zur Taskverteilung an und es überrascht nicht, dass auch die Nutzung von Selbstorganisationsmechanismen in WSNs bereits vergleichsweise früh in Betracht gezogen wurde. Beispielsweise wird in [Dre+05] angeregt, Tasks unter

Verwendung biologisch inspirierter Prinzipien zu verteilen, ohne jedoch detaillierte Lösungsansätze zu liefern. Auf einige konkrete Beispiele selbstorganisierender Mechanismen zur Taskverteilung in WSNs wollen wir im Folgenden eingehen. Auch wenn diese nur selten von ihren Autoren explizit als solche bezeichnet werden, decken sie dennoch ein breites Spektrum an Maßnahmen zur – mehr oder weniger stark ausgeprägten – Selbstorganisation ab.

Eine auktionsbasierte Strategie In [ETX12] wird eine auktionsbasierte Strategie zur Taskverteilung in Sensornetzen vorgestellt: Ein Auktionator schreibt Tasks im Rahmen einer Auftragsauktion *(reverse auction)* aus, der Knoten mit dem geringsten Gebot erhält jeweils den Zuschlag. Hierbei nutzen die Knoten eine spieltheoretisch optimale Strategie, um die durch die Energiebudgets der einzelnen Knoten beschränkte Gesamtlaufzeit des Netzwerks zu maximieren. Die Tasks werden dabei gemäß ihrer durch eine Präzedenzrelation gegebenen Reihenfolge versteigert, um eine korrekte Taskausführung zu gewährleisten: Tasks besitzen hier also eine Priorität, die sich implizit aus den Taskpräzedenzen ergibt.

Zur Steigerung der Energieeffizienz der Auftragsauktion wird der jeweilige Siegerknoten mittels eines verteilten Entscheidungsfindungsprotokolls [Eda+10] bestimmt: Der Auktionator sendet zusammen mit der Ausschreibung einen Höchstpreis, aufgrund dessen alle Knoten, die diesen Preis überschreiten, ihre Gebote nicht senden. Alle übrigen Knoten geben ihre Gebote nach einer Wartezeit ab, die vom Gebotswert abhängt – in der Folge wird das kleinste Gebot zuerst im System gesendet und von allen übrigen Knoten empfangen, woraufhin diese ihre höheren Gebote nicht abgeben. In der Folge erhält der Auktionator höchstens ein einziges Gebot, das jedoch nicht über seinem Höchstpreis liegen kann.

Obwohl dieses Verfahren vor dem Hintergrund der Energieeffizienz zunächst sinnvoll erscheint, offenbaren sich bei genauerer Betrachtung jedoch einige Probleme: Zunächst einmal stellt der Auktionator einen Single-Point-of-Failure dar. Zusätzlich erschwert das auf Vermeidung redundanter Kommunikation ausgelegte Protokoll die Erkennung von Knotenausfällen. In [Eda+09] wird zwar eine *Recovery*-Phase beschrieben, welche bei Knotenausfällen aus den aktuell laufenden Tasks die neu zuzuteilenden Vorgängertasks rekonstruiert und diese möglichst energieeffizient neu vergibt. Diese Strategie ist jedoch wiederum vom zentralen Auktionator abhängig, der zudem den Ausfall der Knoten erkennen können muss; entsprechend ist eine Selbstheilung bei dessen Ausfall nicht mehr möglich.

Ein Contract-Net-basiertes Protokoll Die zuvor beschriebenen Contract-Net-Protokolle wurden auch für die Verwendung in Sensornetzen adaptiert, beispielsweise in [Che+12]. Hier werden die Knoten dynamisch in Clustern organisiert. Um den Kommunikationsaufwand – und damit den Energieverbrauch – zu minimieren, werden Tasks von einem zentralen *Taskknoten* an festgelegte Repräsentanten dieser Cluster ausgeschrieben, welche die Tasks wiederum innerhalb ihres Clusters nochmals ausschreiben. Während einmal abgegebene Gebote dabei nicht mehr revidiert werden

können, kann sich jedoch der Zustand der Knoten prinzipiell während der Laufzeit der Ausschreibungen ändern. Daher versuchen die Knoten, ihren *zukünftigen* Zustand vorherzusagen und diese Vorhersage in ihre Gebote einfließen zu lassen.

Ähnlich zum ursprünglichen CNP sind mit dem Taskknoten sowie den Clusterrepräsentanten auch bei diesem Verfahren Single-Points-of-Failure vorhanden, zudem werden Knotenausfälle in der besagten Publikation nicht behandelt.

Ein genetischer Algorithmus In [WSM20] wird zur Taskverteilung ein genetischer Algorithmus genutzt, der zur Laufzeit des Systems auf einem zentralen Knoten ausgeführt wird. Da die periodische Überwachung aller Knotenparameter mit einem zu hohen Energieverbrauch einhergehen würde, stehen dem Algorithmus jedoch nur begrenzte Informationen zur Verfügung.

Der genetische Algorithmus verteilt die Tasks unter Beachtung ihrer Abhängigkeiten, Zeitschranken und der Eignung der einzelnen Knoten. Zusätzlich wird versucht, kritische Tasks auf möglichst zuverlässigen Knoten und häufig miteinander kommunizierende Tasks auf Knotenpaaren mit möglichst stabiler Kommunikationsverbindung zuzuteilen, um aufgrund von Störungen nötige Neuübertragungen von Nachrichten zu vermeiden.

Bei Knotenausfällen kann der genetische Algorithmus die Taskverteilung entsprechend anpassen; der zentrale Knoten, auf dem er ausgeführt wird, stellt jedoch auch hier einen Single-Point-of-Failure dar.

Eine kooperative Strategie Eine von ihren Autoren explizit als selbstorganisierend bezeichnete Strategie wird in [Yin+17] vorgestellt. Hier werden Tasks mithilfe eines dezentralen Algorithmus auf heterogene Knoten verteilt, die verschiedene Fähigkeiten besitzen. Das zugrunde liegende Taskmodell verwendet voneinander unabhängige Tasks ohne Präzedenzrelationen. Während dies auf den ersten Blick dem Taskmodell des AHS sehr ähnlich erscheint, existieren dennoch einige wesentliche Unterschiede: Zum einen besitzen die Tasks keine Prioritäten, zum anderen werden Tasks immer von einer *Gruppe* von Knoten bearbeitet, die jeweils genau eine der zur Ausführung der Task nötigen Fähigkeiten haben. Im betrachteten Netzwerkmodell können Tasks hierbei entweder direkt mit ihren jeweiligen Nachbarn kommunizieren oder über mehrere Hops mit weiter entfernten Knoten. Hierbei sind jedem Knoten initial keine Informationen über die übrigen Knoten bekannt.

Die Taskverteilung nutzt einen dezentralen Algorithmus, bei dem jeder Knoten selbst über die Mitarbeit an einer Task entscheidet. Hierbei verhalten sich alle Knoten kooperativ und beschließen stets, an einer Task mitzuarbeiten, sofern sie die hierfür notwendige Fähigkeit besitzen. Der Algorithmus selbst ist im Wesentlichen eine verteilte Breitensuche entlang der direkten Kommunikationsverbindungen, die von jenem Knoten ausgeht, der eine neue Task ins System einbringt. Folglich arbeiten bevorzugt lokal benachbarte Knoten gemeinsam an den einzelnen Tasks. Um den Energieverbrauch sowie den Kommunikationsaufwand zu verringern, werden die durch vergangene Verteilungen gewonnenen Informationen über entfernte Knoten bei späteren Taskverteilungen zur Optimierung der Kommunikationsrouten genutzt.

Da im schlimmsten Fall alle Knoten durchlaufen werden müssen, bis eine Task vollständig zugeteilt ist, kann die hierfür nötige Zuteilungsdauer folglich beschränkt werden. Fallen Knoten aus, werden diese bei der Zuteilung zukünftiger Tasks übersprungen, wodurch das Gesamtsystem funktionsfähig bleibt; die Neuzuteilung der vom Ausfall betroffenen Tasks wird dagegen nicht betrachtet, weshalb diese Strategie nicht als selbstheilend bezeichnet werden kann. Ebenso existiert kein Selbstoptimierungsmechanismus für bereits zugeteilte Tasks; es wird lediglich versucht, die Zuteilung zukünftig eintreffender Tasks zu optimieren.

Vergleich mit dem AHS Die genannten Beispiele stellen eine Auswahl der wichtigsten Forschungsaspekte auf dem Gebiet der Taskverteilung in Sensornetzen dar. Im Folgenden sollen einige Merkmale identifiziert werden, anhand derer sich Sensornetze im Allgemeinen sowie die betrachteten Ansätze im Speziellen vom AHS abgrenzen lassen.

Zunächst einmal ist hierzu festzuhalten, dass die betrachteten Strategien stets einen starken Fokus auf die Energieeffizienz legen, da die Knoten eines Sensornetzes typischerweise batteriebetrieben sind und daher einen begrenzen Energievorrat aufweisen. Das Hauptaugenmerk des AHS ist dagegen nicht die Energieeffizienz, sondern die Einhaltung harter Zeitschranken für die Taskverteilung, was die meisten der betrachteten Strategien dagegen nicht gewährleisten können. Nichtsdestotrotz ließen sich beschränkte Energievorräte der einzelnen PEs im AHS mithilfe von Monitoring-Suppressoren und -Acceleratoren abbilden.

Einen weiteren Unterschied stellt das zugrunde liegende Netzwerkmodell dar. Während das AHS zum Hormonaustausch in jedem Hormonzyklus Nachrichten durch Broadcasts austauscht, sind häufige Broadcasts in Sensornetzen unter Umständen problematisch: Die zur drahtlosen Übertragung einer Information nötige Sendeleistung ist proportional zum Quadrat der Distanz der beteiligten Knoten. Entsprechend ist es energetisch häufig sinnvoller, die Kommunikation über große Distanzen stattdessen durch ein Multi-Hop-Routing zu realisieren [DP10]. Dies spiegelt sich bei den betrachteten Methoden insbesondere darin wider, dass zumeist nur lokale Kommunikation verwendet wird. Ferner können Knotenausfälle sowie Kommunikationsstörungen jederzeit die Netzwerktopologie verändern. Folglich liegen dem AHS andere Annahmen über das verwendete Kommunikationsmedium zugrunde, weshalb eine Adaption des AHS auf Sensornetze versuchen müsste, die Verwendung von Broadcasts auf ein Minimum zu reduzieren.

Zudem enthalten die betrachteten Methoden zumeist einen zentralisierten Knoten, der besondere Aufgaben ausführt und damit einen Single-Point-of-Failure darstellt, während das vollständig dezentral konzipierte AHS einen solchen explizit vermeidet. Dies ist jedoch ohnehin ein in Sensornetzen häufig anzutreffendes Merkmal, da diese zumeist eine zentrale Basisstation zur Sammlung der vorverarbeiteten Daten besitzen [Amm+16]. Entsprechend führt beispielsweise die Verwendung dieser Basisstation als zentraler Auktionator nicht zwangsläufig zu einer gesteigerten Anfälligkeit für Systemausfälle, als dies bei Verwendung einer zentralen Basisstation ohnehin bereits der Fall ist.

Zu guter Letzt unterscheiden sich die jeweils zugrunde liegenden Taskmodelle vom im AHS verwendeten Modell. Zum einen sehen die meisten Strategien zur Taskverteilung in Sensornetzen vor, dass zur Laufzeit neue Tasks zentral von der Basisstation oder dezentral von Knoten, die ein Ereignis beobachten, ins System eingebracht werden können. Im Taskmodell, das dem AHS im Rahmen dieser Dissertation zugrunde gelegt wurde, wird jedoch davon ausgegangen, dass jedem PE von Beginn an alle Tasks sowie insbesondere deren Prioritäten bekannt sind. Zum anderen gehen viele der betrachteten Methoden von *abhängigen* Tasks aus. Taskabhängigkeiten sind im AHS zunächst nicht vorgesehen, können aber mithilfe verschiedener AHS-Erweiterungen [Pac10; PB10; HPB20] realisiert werden. Aufgrund *zeitlicher* Abhängigkeiten zwischen Tasks beinhalten die meisten der vorgestellten Methoden auch das zeitliche Scheduling der verteilten Tasks. Das AHS befasst sich dagegen primär mit der Verteilung der Tasks auf die einzelnen PEs, während jedes PE selbst für das Scheduling der auf ihm zugeteilten Tasks zuständig ist. Nichtsdestotrotz wäre eine Kombination der prioritätsbasierten Taskverteilung im AHS mit den erwähnten Ansätzen zur Realisierung abhängiger Tasks eine interessante zukünftige Forschungsrichtung.

Folglich erklären diese prinzipiellen Unterschiede zwischen Sensornetzen und dem AHS viele der Eigenschaften der vorgestellten Methoden zur Taskverteilung. Nichtsdestotrotz könnten Anpassungen des AHS an die Besonderheiten von Sensornetzen ebenfalls eine interessante Richtung für zukünftige Forschungen darstellen.

10.3. Abschließende Betrachtungen

In diesem Kapitel wurden verschiedene Methoden zur Taskverteilung in verteilten Systemen betrachtet und mit dem AHS verglichen. Keine der betrachteten Methoden konnte vollumfänglich mit dem AHS vergleichbare Eigenschaften zusichern: Das Spektrum der Unterscheidungsmerkmale reichte von Single-Points-of-Failure, fehlender Unterstützung von Prioritäten oder Echtzeiteigenschaften bis hin zu unzureichender oder gänzlich fehlender Selbstheilungs- oder Selbstoptimierungsfähigkeiten. Obwohl einige dieser Einschränkungen zumeist mit Besonderheiten des jeweiligen Einsatzzwecks begründet werden können, wären Anpassungen des AHS an eben jene Besonderheiten dennoch eine interessante Möglichkeit für zukünftige Forschungsarbeiten.

11 Zusammenfassung und Ausblick

Diese Dissertation betrachtete eine prioritätsbasierte Erweiterung für das künstliche Hormonsystem (AHS). Das AHS ist eine Middleware, um Tasks auf die Prozessorelemente (PEs) eines verteilten Systems zu verteilen. Hierbei kommen Prinzipien des in Kapitel 3 vorgestellten *Organic Computing* zum Einsatz. Ein mithilfe des AHS realisiertes System ist *selbstkonfigurierend*, da die PEs selbstständig eine initiale Verteilung der Tasks auf die PEs finden, die sich während der Laufzeit des Systems zudem geänderten Rahmenbedingungen anpassen kann, indem Tasks zum Zwecke der *Selbstoptimierung* automatisch auf andere PEs migriert werden. Eine solche Migration findet ebenfalls im Falle des Ausfalls einzelner Tasks oder gar PEs statt: Solchen Ausfällen wird, sobald diese bemerkt wurden, durch Migration der betroffenen Tasks auf funktionsfähige PEs begegnet, sofern diese in der Lage sind, jene Tasks zusätzlich auszuführen. Das AHS besitzt somit keinen Single-Point-of-Failure und ist als *selbstheilend* zu klassifizieren.

Zudem eignet sich das AHS für den Einsatz in Echtzeitsystemen, da für die Dauer der Selbstkonfiguration, -optimierung und -heilung jeweils die Einhaltung harter Zeitschranken garantiert werden kann.

Nachdem das AHS in Kapitel 4 im Detail vorgestellt wurde, befasste sich Kapitel 5 mit einer detaillierten Betrachtung seiner Zeitschranken: Es konnte gezeigt werden, dass die bisher bekannten Zeitschranken unter bestimmen Voraussetzungen in unerwarteter Weise degenerieren können. Die Ursache dieses Phänomens wurde identifiziert, woraufhin die bekannten Zeitschranken verfeinert wurden, um diesem Umstand Rechnung zu tragen.

Kapitel 6 stellte eine prioritätsbasierte Erweiterung des AHS vor, um Tasks in einer anhand von *Vergabeprioritäten* definierten Reihenfolge im System zuzuteilen. Dies stellt eine Verbesserung gegenüber dem ursprünglichen AHS dar, das keine solche Reihenfolge garantieren kann. Auch unter Verwendung dieser Erweiterung kann das AHS weiterhin die Einhaltung harter Zeitschranken garantieren, bei deren Herleitung insbesondere besagtes Degenerationsphänomen eine tragende Rolle spielte; eine umfassende Evaluation in Kapitel 7 bestätigte die theoretischen Betrachtungen.

Das bisherige, prioritätslose AHS ermöglicht eine Selbstheilung nur, wenn die Kapazitäten der funktionsfähigen PEs ausreichend sind, um die vom Ausfall betroffenen Tasks *zusätzlich* zu übernehmen. Ist dies in einer *Überlastsituation* dagegen nicht mehr der Fall, versagen die Selbstheilungsmechanismen des klassischen AHS, da in einem solchen Fall nicht definiert ist, welche Tasks noch zur Ausführung kommen. Demgegenüber erlaubt die Einführung von Taskprioritäten es jedoch, in Überlastsituationen Tasks niedriger Priorität zu stoppen, um Kapazitäten für die Ausführung von Tasks höherer Priorität zu schaffen. Entsprechend befasste sich Kapitel 8 im Detail mit der Selbstheilung in Überlastsituationen. Die Dauer einer solchen Selbstheilung ist insbesondere davon abhängig, wie viele und in welcher Reihenfolge Tasks von den verbleibenden PEs gestoppt werden. Diese Entscheidung wird auf jedem PE von einer

sogenannten *Task-Dropping-Strategie* getroffen. Zum Zwecke der späteren Analyse wurde zunächst ein Modell für Überlastsituationen entwickelt. Anschließend wurden zwei Task-Dropping-Strategien vorgestellt und im Detail analysiert:

Bei Verwendung der *naiven* Task-Dropping-Strategie stoppt jedes PE bei Erkennung einer Überlastsituation *alle* seine zugeteilten Tasks, die eine geringere Priorität haben als die höchstpriore Task, die aufgrund der Überlastsituation keinem PE zugeteilt werden kann. Es wurde gezeigt, dass diese Strategie im Worst Case *alle* zugeteilten Tasks stoppt (was in einer anschließenden vollständigen Selbstkonfiguration des Systems resultiert) und sich auch ihr Average Case nicht signifikant besser verhält.

Das *Eager Task Dropping* dagegen stoppt Tasks niedriger Priorität nacheinander in mehreren Phasen. Es zeigte sich, dass hier zumeist deutlich weniger Tasks gestoppt werden müssen, bis die Selbstheilung beendet ist. Konkret ist diese Strategie in den meisten Fällen sogar dem *Average Case* der naiven Strategie überlegen, lediglich in einigen Grenzfällen muss sie sich dieser Strategie geschlagen geben, beispielsweise, wenn nur noch ein einziges funktionsfähiges PE im System verbleibt.

Insbesondere konnte gezeigt werden, dass viele der hergeleiteten Zeitschranken sogar *kleinste* obere Schranken sind, es also Szenarien gibt, in denen die hergeleiteten Zeitschranken exakt erreicht werden. Die in Kapitel 9 durchgeführten Evaluationen des Systemverhaltens in Überlastsituationen bestätigten die theoretischen Betrachtungen vollumfänglich.

In Kapitel 10 wurden schließlich Vergleiche zu verwandten Konzepten und Projekten gezogen und die Unterschiede zum (prioritätsbasierten) AHS herausgearbeitet.

Fazit

Zusammenfassend stellt die vorliegende Dissertation eine Erweiterung des ursprünglichen AHS vor, welche dieses um eine völlig neue Qualität ergänzt: die Gewährleistung einer definierten Vergabereihenfolge aller im System befindlichen Tasks. Hierbei werden die Selbst-X-Eigenschaften des ursprünglichen AHS sowie dessen Eignung für Echtzeitanforderungen beibehalten und um die Funktionalität einer Selbstheilung in zuvor nicht betrachteten Überlastsituationen ergänzt: Fallen zu viele PEs aus, sodass das System nicht mehr *alle* seine Funktionalitäten realisieren kann, können nun – anhand der für die einzelnen Tasks vergebenen Prioritäten – Tasks niedriger Priorität zugunsten solcher höherer Priorität gestoppt werden. Dies ermöglicht die Realisierung der wichtigsten *Teilfunktionalitäten* des Systems, um beispielsweise einen zwar eingeschränkten, aber dafür sicheren Systemzustand einnehmen zu können. Da das ursprüngliche AHS in einer solchen Überlastsituation keinerlei Garantien geben kann, stellt die vorgestellte prioritätsbasierte Erweiterung eine signifikante Verbesserung des status quo dar.

Ausblick

Nichtsdestotrotz lassen sich bei genauerer Betrachtung einige Aspekte ausmachen, in denen das um Prioritäten erweiterte AHS im Rahmen zukünftiger Forschung erweitert werden könnte:

Kommunikationsstörungen Zum einen nahm diese Dissertation ein zuverlässiges Kommunikationsmedium zwischen den PEs an; Kommunikationsausfälle, die sich beispielsweise in fehlenden Hormonen äußern könnten, wurden nicht betrachtet. Obwohl das AHS erfahrungsgemäß vereinzelte fehlende Hormone vergleichsweise gut verkraftet, sind hierzu genauere Analysen nötig, um insbesondere festzustellen, welche Garantien beim Auftreten von Kommunikationsstörungen gegeben werden können. Erste Ansätze hierzu wurden für den Fall des ursprünglichen, prioritätslosen AHS in [BP21] unternommen, im Rahmen zukünftiger Forschung sollte daher insbesondere überprüft werden, inwiefern diese Resultate auch für das prioritätsbasierte AHS gelten.

Beschleunigung der Selbstkonfiguration Im Vergleich zum ursprünglichen AHS ohne Prioritätsunterstützung dauert die Selbstkonfiguration eines Systems bei Verwendung von $|P|$ verschiedenen Prioritäten im Worst Case $|P| - 1$ Hormonzyklen länger, da sich an die vollständige Zuteilung jedes Prioritätslevels ein Hormonzyklus anschließt, in dem keine Taskzuteilung erfolgen kann. Im Rahmen künftiger Forschungsarbeiten sollte daher untersucht werden, ob sich diese *Wartezyklen* unter bestimmten Umständen vermeiden lassen, um die Selbstkonfiguration zu beschleunigen. Da die Wartezyklen jedoch in der derzeitigen Implementierung des AHS begründet sind, müssen möglicherweise größere Änderungen an der Kommunikationsstruktur des AHS vorgenommen werden, um diese zu vermeiden.

Verbesserung der Selbstheilung in Überlastsituationen Da die Selbstkonfiguration den zentralen Mechanismus des AHS darstellt, der insbesondere auch zur Selbstheilung genutzt wird, würden besagte Verbesserungen der Selbstkonfiguration auch die Selbstheilung beschleunigen. In Hinblick auf Überlastsituationen könnten jedoch zusätzlich die hierauf spezialisierten Mechanismen weiter verbessert werden. Konkret wäre beispielsweise denkbar, in jenen Situationen, in denen die naive Task-Dropping-Strategie dem Eager Task Dropping überlegen ist, dynamisch auf diese Strategie zu wechseln. Ebenso wären die Entwicklung weiterer Task-Dropping-Strategien sowie – insbesondere für reale Anwendungsszenarien des AHS – Verfeinerungen des für die Analysen verwendeten Überlastmodells denkbar, da das aktuelle Modell Szenarien, die nur wenige verschiedene Prioritätslevel besitzen, nur in Ansätzen berücksichtigt, was in zu groben Abschätzungen der maximalen Selbstheilungsdauer resultieren kann. Ebenso ist denkbar, Tasks zu betrachten, die unterschiedliche Last auf ihren PEs hervorrufen, sodass unter Umständen mehr als eine Task gestoppt werden muss, um eine andere Task übernehmen zu können.

Anwendungen des prioritätsbasierten AHS Zu guter Letzt ist die Anwendung des AHS zur Umsetzung realer Systeme eine interessante Forschungsrichtung, um insbesondere Vergleiche mit anderen etablierten Architekturen durchführen zu können. Hierzu könnten sich insbesondere die in Kapitel 10 vorgestellten Einsatzgebiete eignen, wie beispielsweise automotive Steuergeräte sowie Sensornetze.

Anhang

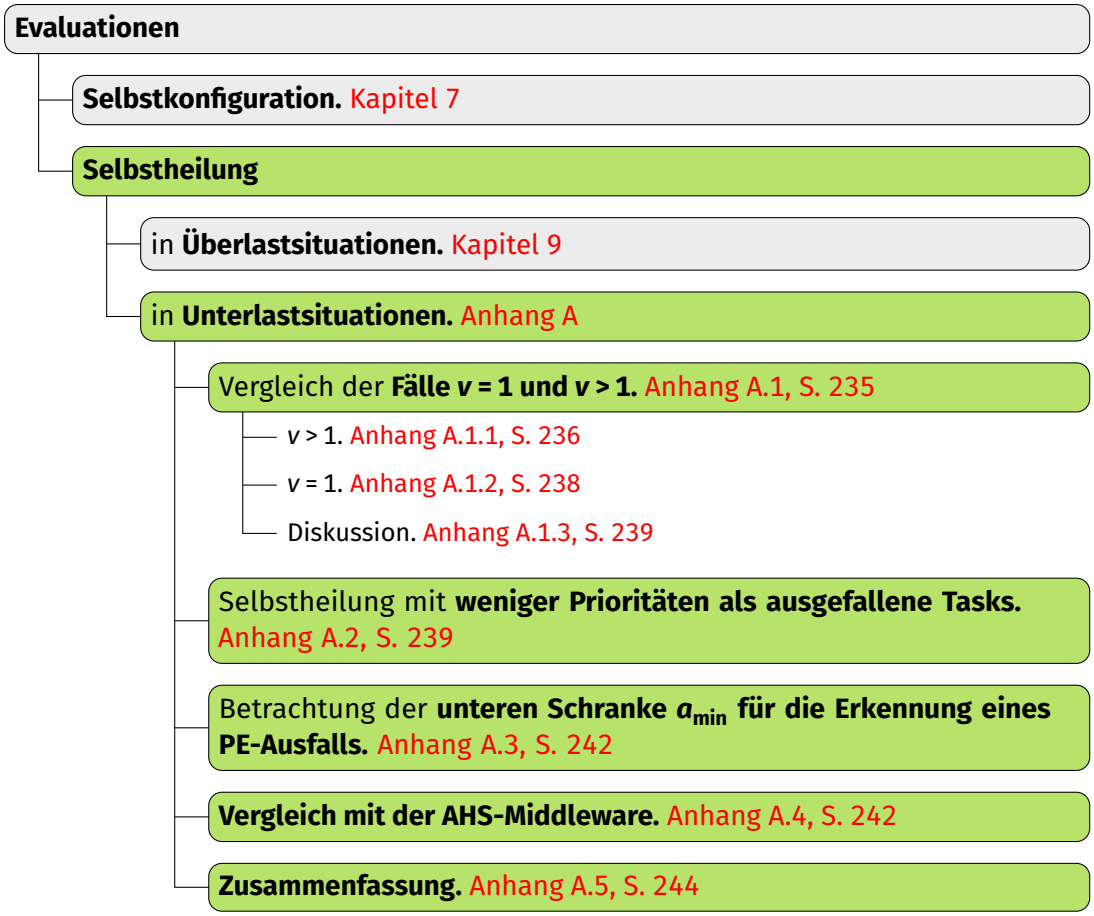


Abbildung A.1: Überblick über die Evaluationen der Selbstheilung in Unterlastsituationen

A Evaluation der Selbstheilung in Unterlastsituationen

Die Evaluationen in Kapitel 7 beschränkten sich auf die Selbstkonfiguration des AHS bei Verwendung der prioritätsbasierten Task-Decision-Strategie, da der Selbstkonfigurationsmechanismus allen weiteren Mechanismen zugrunde liegt. Eine Selbstheilung in einer Unterlastsituation besteht beispielsweise aus der Erkennung eines PE-Ausfalls, an den sich automatisch eine Selbstkonfiguration der mit dem PE ausgefallenen Tasks anschließt. Im Folgenden wollen wir daher die Erkennung eines solchen Ausfalls genauer untersuchen und überprüfen, dass im Anschluss tatsächlich eine Neuzuteilung der ausgefallenen Tasks stattfindet. Einige der vorgestellten Evaluationsergebnisse wurden hierbei bereits in [HB21a] veröffentlicht.

Im Folgenden nehmen wir an, dass nach dem Ausfall von $\mathrm{PE}_\times$ noch $v \geq 1$ funktionsfähige PEs verbleiben. Zudem finden während der Selbstheilung weder Selbstkonfiguration, Selbstoptimierung noch eine weitere Selbstheilung statt. Für ein solches Szenario sagt Satz 6.15 voraus, dass die Selbstheilung höchstens

$$m_f + (|P_f| - 1) \cdot [v > 1] + a \text{ Hormonzyklen}$$

dauert, wobei m_f die Anzahl der ausgefallenen Tasks (die zuvor auf $\mathrm{PE}_\times$ zugeteilt waren) sowie P_f die Menge deren Prioritäten bezeichnet. Die Konstante a gibt die zur Erkennung des Ausfalls von $\mathrm{PE}_\times$ benötigte und in Hormonzyklen gemessene Zeit an. Gemäß Bemerkung 6.16 wird der Ausfall nach frühestens $a_{\min} = 1$ und spätestens nach $a = 2$ Hormonzyklen erkannt.

Im Folgenden werden wir diese Zeitschranke für einige exemplarisch ausgewählte Szenarien experimentell überprüfen. Hierzu kommen Abwandlungen der bereits in Kapitel 7 verwendeten Taskkonfigurationen zum Einsatz, die erneut im Hormonsimulator betrachtet werden. Für einen Überblick über die Struktur dieses Evaluationskapitels sei auf Abbildung A.1 verwiesen.

A.1. Vergleich der Fälle $v = 1$ und $v > 1$

Die eingangs erwähnte Zeitschranke unterscheidet zwei Fälle: Falls aufgrund des Ausfalls von $\mathrm{PE}_\times$ noch $v > 1$ PEs funktionsfähig bleiben, entspricht die resultierende Zeitschranke von höchstens $m_f + |P_f| - 1 + a$ Hormonzyklen abzüglich der Konstanten a gerade jener der Selbstkonfiguration nach Satz 6.9, wobei jedoch nur eine Teilmenge $\mathcal{T}_f \subseteq \mathcal{T}$ der Menge aller im System vorhandenen Tasks betrachtet wird.

Für $v = 1$ – es bleibt also nur noch genau ein PE funktionsfähig – entfallen dagegen die Wartezyklen nach der vollständigen Zuteilung jedes Prioritätslevels, weshalb sich die Zeitschranke zu höchstens $m_f + a$ Hormonzyklen ergibt. Obwohl dieses Verhalten von Bemerkung 6.8 vorhergesagt und in Kapitel 7.2.1.2 experimentell bestätigt wurde, wird

es in der Zeitschranke der Selbst*konfiguration* nicht berücksichtigt: Diese setzt implizit die Existenz von mindestens zwei PEs voraus, da das AHS eine *verteilte* Middleware ist. Im Falle der Selbstheilung wird der Spezialfall $v = 1$ dagegen explizit betrachtet, da es durchaus vorkommen kann, dass nach dem Ausfall eines oder mehrerer PEs nur noch ein einziges funktionsfähig bleibt.

Daher soll in diesem Abschnitt das Verhalten beider Fälle betrachtet werden. Hierzu kommt das aus Abbildung 7.2 bekannte, aus zehn Tasks mit zehn Prioritäten bestehende Evaluationsszenario auf drei gleichartigen, in einem 1×3-Gitter angeordneten PEs zum Einsatz. Nach einer anfänglichen Selbstkonfiguration wird nacheinander der Ausfall von zwei der drei PEs simuliert, was zu zwei Selbstheilungen mit $v = 2$ bzw. $v = 1$ verbleibenden PEs führt. Damit keine Überlastsituation vorliegt, wurde sichergestellt, dass jedes PE für sich genommen alle $m = 10$ Tasks übernehmen kann.

Weil alle Tasks verschiedene Prioritäten haben, gilt für jede Selbstheilung automatisch $|P_f| = m_f$, was im Fall von $v > 1$ verbleibenden PEs zu $m_f - 1$ Wartezyklen führt und damit dem Worst Case entspricht. Ebenso findet jeder simulierte PE-Ausfall stets direkt *nach* dem Senden der Hormone statt, was gemäß der Beobachtungen aus Kapitel 5.3 dazu führt, dass der Ausfall erst nach $a = 2$ Hormonzyklen bemerkt wird und damit gleichsam dem Worst Case entspricht.

Der zeitliche Verlauf der so resultierenden Selbstheilungen ist in Abbildung A.2 gestapelt nach der Anzahl Tasks pro PE sowie den Taskprioritäten dargestellt; die vorangegangene Selbstkonfiguration ist dagegen nicht abgebildet.

A.1.1. Erste Selbstheilung mit $v > 1$

Wie aus der Abbildung zu entnehmen ist, fällt in Hormonzyklus 30 mit $\mathrm{PE}_{(1,1)}$ das erste PE aus, auf dem zuvor die vier Tasks 1, 2, 8 und 10 zugeteilt waren.[1] Mit $v = 2$ verbleibenden PEs, $m_f = |P_f| = 4$ ausgefallenen Tasks und $a = 2$ Hormonzyklen zur Ausfallerkennung sollte die Selbstheilung laut Satz 6.15 daher höchstens $m_f + |P_f| - 1 + a = 9$ Hormonzyklen dauern.

Diese Zeitschranke wird hier exakt erreicht: In Hormonzyklus 32 wird (nach $a = 2$ Zyklen) der Ausfall von $\mathrm{PE}_{(1,1)}$ bemerkt und mit Task 10 die erste ausgefallene Task erneut zugeteilt. Nach der letzten Zuteilung ist das System am Ende von Zyklus 38 vollständig geheilt, die Selbstheilung dauerte daher exakt neun Hormonzyklen.

Dies zeigt, dass die Selbstheilung nach der Erkennung des PE-Ausfalls wie erwartet einer Selbstkonfiguration der ausgefallenen Tasks entspricht, entsprechend tritt wegen $m_f = |P_f|$ nach der Zuteilung *jeder* Task (außer der letzten) ein Wartezyklus auf, wie es bereits in Kapitel 7.2.1 im Fall der Selbstkonfiguration betrachtet wurde.

Im Falle der Selbstheilung in einer Unterlastsituation ist zudem die Zuteilungsdauer eine relevante Größe, um die Zeitspanne abzuschätzen, nach der eine *bestimmte* Task garantiert wieder einem PE zugeteilt ist. Im Gegensatz zur Dauer der *gesamten* Selbstheilung (welche abschätzt, wann *alle* Tasks des Systems einem PE zugeteilt sind) ermöglicht dies im Allgemeinen eine genauere Abschätzung, da bestimmt werden kann,

[1] Hierbei gelte wieder $\mathrm{prio}(T) = T$ für jede Task T, die Nummer einer Task entspricht also gerade ihrer Priorität.

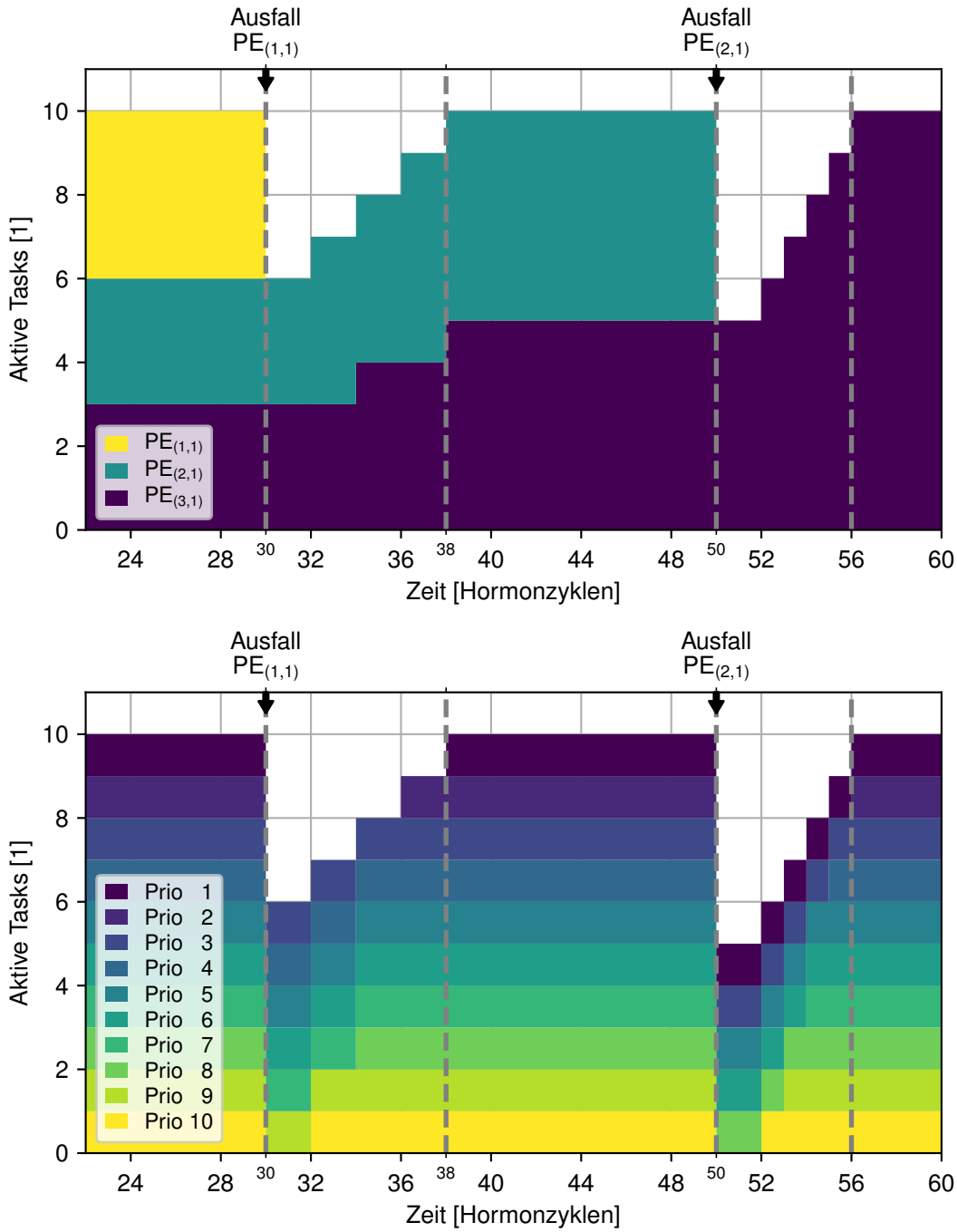


Abbildung A.2: Zeitlicher Verlauf zweier Selbstheilungen mit $v = 2$ bzw. $v = 1$ verbleibenden PEs sowie $|P_f| = m_f$

Tabelle A.1: Zuteilungsdauern der Tasks gemäß Abbildung A.2 und durch D(T) abgeschätzte Werte. Alle Angaben in Hormonzyklen.

(a) Erste Selbstheilung

Task	Beobachtung	Abschätzung
10	3	≤ 3
8	5	≤ 5
2	7	≤ 7
1	9	≤ 9

(b) Zweite Selbstheilung

Task	Beobachtung	Abschätzung
10	3	≤ 3
9	4	≤ 4
7	5	≤ 5
4	6	≤ 6
2	7	≤ 7

wann zumindest eine bestimmte *Teilfunktionalität* des Systems wiederhergestellt ist. In Tabelle A.1a sind daher die gemäß Satz 6.19 abgeschätzten Zuteilungsdauern der einzelnen ausgefallenen Tasks den beobachteten Werten gegenübergestellt. Hierbei sind alle Angaben relativ auf den Ausfallzeitpunkt von $\text{PE}_{(1,1)}$ in Hormonzyklus 30 zu interpretieren. Es fällt auf, dass die beobachteten Zeitpunkte, zu denen die ausgefallenen Tasks wieder einem PE zugeteilt sind, genau den durch D(T) abgeschätzten oberen Schranken entsprechen, ebenso wie bereits die Gesamtdauer der Selbstheilung (die gerade der Zuteilungsdauer der niedrigstprioren ausgefallenen Task entspricht) exakt der theoretischen Zeitsprache entsprach.

A.1.2. Zweite Selbstheilung mit $v = 1$

Mit dem Ausfall von $\text{PE}_{(2,1)}$ in Zyklus 50 wird die zweite Selbstheilung des Systems nötig. Auf diesem PE waren zuvor die fünf Tasks 2, 4, 7, 9 und 10 zugeteilt. Mit $v = 1$ verbleibenden PEs, $m_f = 5$ ausgefallenen Tasks und $a = 2$ schätzt Satz 6.15 die Dauer der Selbstheilung zu $m_f + a = 7$ Hormonzyklen ab.

Auch hier wird die theoretische Zeitschranke exakt erreicht: Beginnend mit Hormonzyklus 52 werden die ausgefallenen Tasks erneut zugeteilt. Weil wegen $v = 1$ stets nur *ein* PE positive modifizierte Eignungswerte senden kann, kann kein anderes PE den Almost-Winner-Zustand einnehmen, weshalb keine Wartezyklen auftreten. Dies ist wiederum konsistent mit Beobachtung 6.8, die für den Fall der Selbstkonfiguration bereits in Kapitel 7.2.1.2 experimentell bestätigt wurde.

Entsprechend ist die Selbstheilung mit der Zuteilung von Task 2 in Hormonzyklus 56 beendet und dauerte insgesamt 7 Hormonzyklen: Obwohl bei der zweiten Selbstheilung mehr Tasks neu zugeteilt werden mussten, dauerte die gesamte Selbstheilung aufgrund der weggefallenen Wartezyklen kürzer als jene nach dem ersten PE-Ausfall.

In Tabelle A.1b sind die beobachteten Zuteilungsdauern der einzelnen Tasks erneut den mithilfe von Satz 6.19 abgeschätzten Werten gegenübergestellt. Abermals stimmen die Beobachtungen exakt mit den oberen Schranken überein.

A.1.3. Diskussion der Ergebnisse

Das betrachtete Szenario zeigt, dass die Selbstheilung in Unterlastsituationen wie gewünscht funktioniert: Nach der Erkennung des Ausfalls von $\mathrm{PE}_{\times}$ – was hier nach $a = 2$ Hormonzyklen geschah und damit dem Worst Case entsprach – startet eine Selbstkonfiguration der nun im System nicht mehr zugeteilten Tasks. Die theoretisch hergeleiteten Zeitschranken für die Gesamtdauer der Selbstheilung sowie die individuelle Zuteilungsdauer einzelner Tasks wurden hier jeweils exakt eingehalten. Dies ist jedoch nicht weiter verwunderlich, da dieses Verhalten im hier vorliegenden Fall, in dem alle Tasks verschiedene Prioritäten haben, bereits für die zugrunde liegende Selbstkonfiguration beobachtet und erklärt werden konnte. Die Zeitschranken der Selbstheilung bauen auf jenen der Selbstkonfiguration auf und berücksichtigen zusätzlich die Konstante a sowie den Entfall der Wartezyklen für lediglich $v = 1$ verbleibende PEs, folglich sind die Evaluationsergebnisse der Selbstkonfiguration auch auf die Selbstheilung übertragbar. Aus diesem Grund beschränken wir uns im Folgenden auf den Fall einer Selbstheilung mit weniger Prioritäten als Tasks sowie eine Betrachtung der zur Ausfallerkennung *mindestens* benötigten Zeit $a_{\min}$.

A.2. Selbstheilung mit weniger Prioritäten als ausgefallene Tasks ($|P_f| < m_f$)

Um das Verhalten der Selbstheilung zu untersuchen, wenn die ausgefallenen Tasks nicht alle verschiedene Prioritäten haben, wurde das bereits aus Abbildung 7.10 bekannte und aus 64 Tasks bestehende Szenario verwendet, dessen Tasks wie in Kapitel 7.4.3.1 beschrieben in vier Prioritätslevel zu je 16 Tasks unterteilt wurden. Im Gegensatz zu den Evaluationen der Selbstkonfiguration, in denen ein solches Szenario auf 16 PEs simuliert wurde, wurden hier nur acht gleichartige, in einem 2×4-Gitter angeordnete PEs verwendet, um bei einem PE-Ausfall eine größere Zahl Tasks neu zuteilen zu müssen.

Abbildung A.3 zeigt den zeitlichen Verlauf der Selbstheilung, die aufgrund des Ausfalls von $\mathrm{PE}_{(1,1)}$ in Hormonzyklus 60 nötig wird: Nach $a = 2$ Hormonzyklen – der Ausfall geschah auch hier wieder zum Worst-Case-Zeitpunkt – wird in Zyklus 62 die erste Task erneut zugeteilt, mit der Zuteilung der letzten Task in Zyklus 72 endet die damit insgesamt 13 Hormonzyklen dauernde Selbstheilung.

Bei genauerer Betrachtung des unteren, nach Prioritäten gestapelten Graphen ergibt sich, dass hier insgesamt $m_f = 9$ Tasks geheilt wurden: Je zwei Tasks der Prioritäten 4 (Zyklen 62 und 63), 3 (Zyklen 65 und 66) und 2 (Zyklus 68) sowie drei Tasks der Priorität 1 (Zyklen 70, 71 und 72), folglich gilt $|P_f| = 4$. In einem solchen Fall dürfte die Selbstheilung gemäß Satz 6.15 höchstens $m_f + |P_f| - 1 + a = 14$ Hormonzyklen dauern. Dass die Selbstheilung im konkreten Fall sogar einen Zyklus schneller beendet ist, liegt an der in Zyklus 68 stattfindenden gleichzeitigen Zuteilung zweier Tasks der Priorität 2. Würde diese nicht erfolgen, wäre auch hier ein exaktes Erreichen der theoretischen Zeitschranke zu erwarten, zumal erneut zu erkennen ist, dass in den

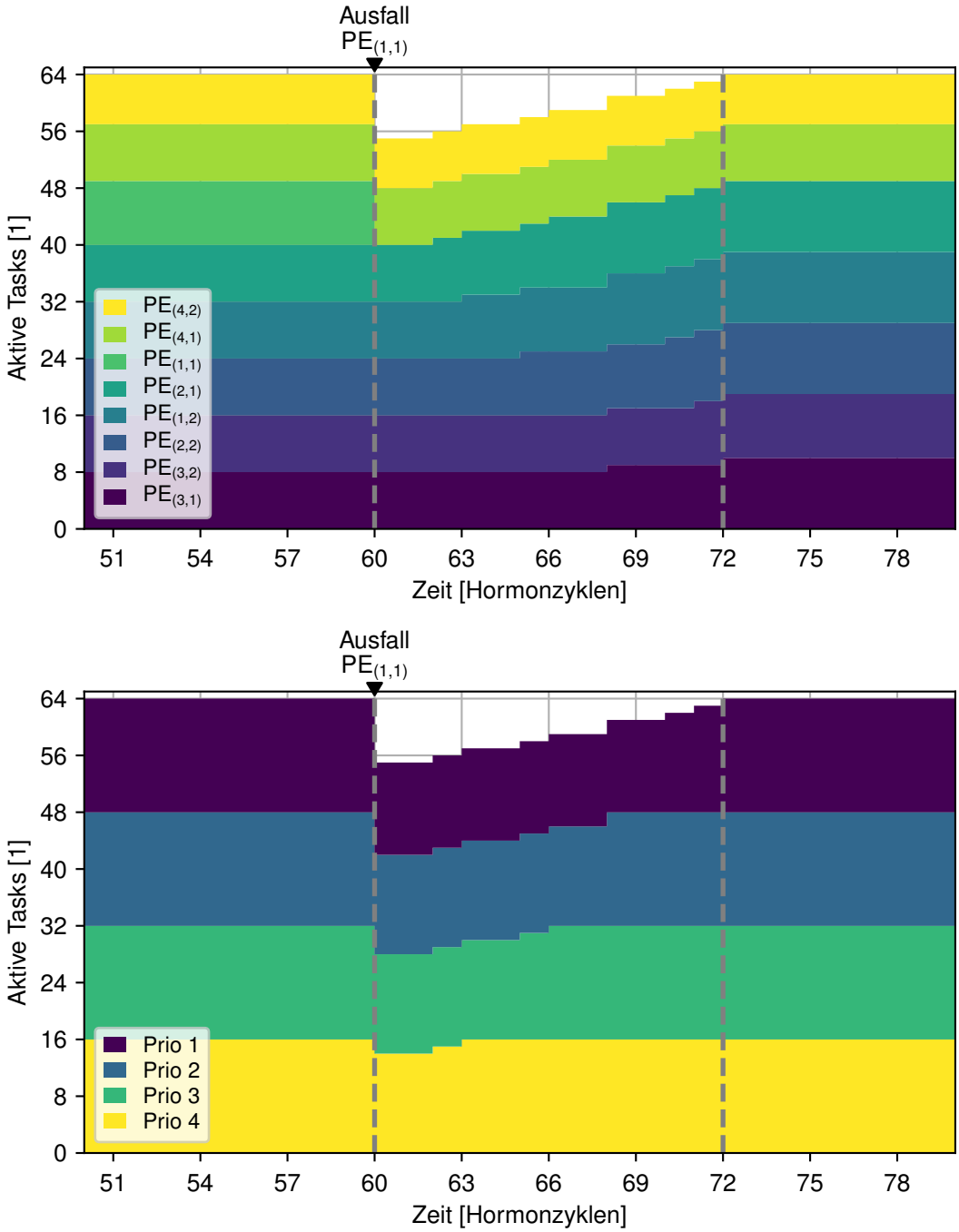


Abbildung A.3: Zeitlicher Verlauf zweier Selbstheilungen mit $v = 7$ verbleibenden PEs sowie $|P_f| < m_f$

Tabelle A.2: Zuteilungsdauern der Prioritätslevel gemäß Abbildung A.3 und durch $D(T_p)$ für einen Repräsentanten T_p des Prioritätslevels p abgeschätzte Werte. Alle Angaben in Hormonzyklen.

Priorität	Beobachtung	Abschätzung
4	4	$\leq$ 4
3	7	$\leq$ 7
2	9	$\leq$ 10
1	13	$\leq$ 14

Zyklen 64, 67 und 69 keine Tasks zugeteilt werden: Dies sind gerade die von den theoretischen Überlegungen vorhergesagten Wartezyklen, die zwischen der Zuteilung von Tasks verschiedener Prioritäten auftreten.

Für die Betrachtung der Zuteilungsdauer werden wir lediglich für jedes Prioritätslevel die Zeitspanne betrachten, nach der *alle* seine Tasks wieder einem PE zugeteilt sind: Wie im Fall der Selbstkonfiguration ist die Reihenfolge der Zuteilung innerhalb desselben Prioritätslevels nicht definiert, weshalb eine Betrachtung individueller Tasks nicht zielführend ist. Hierzu stellt Tabelle A.2 erneut die beobachteten Werte den durch $D(T)$ für einen Repräsentanten jedes Prioritätslevels abgeschätzten Werten gegenüber. Auch hier erreichen die beobachteten Werte zunächst exakt die theoretischen Abschätzungen, bis durch Doppelzuteilung in Prioritätslevel 2 ein Vorsprung von einem Hormonzyklus erarbeitet und bis zum Ende der Selbstheilung gehalten wird. Dies ist konsistent zu den Evaluationen der Selbstkonfiguration, bei denen ein solches Verhalten häufig zu beobachten war.

Auch im hier betrachteten Fall mit $|P_f| < m_f$ führt das System also eine Selbstheilung durch, indem nach der Erkennung eines PE-Ausfalls nach a Hormonzyklen eine Selbstkonfiguration der nun nicht mehr zugeteilten Tasks erfolgt. Das Verhalten dieser Selbstkonfiguration entspricht dabei dem bereits in Kapitel 7 evaluierten Verhalten.

Es sei jedoch angemerkt, dass im Rahmen dieser Evaluation stets Wissen darüber vorhanden war, *welche* Tasks ausgefallen waren. Unter Kenntnis der Anzahl m_f ausgefallener Tasks sowie der Anzahl $|P_f|$ der Prioritäten jener Tasks konnte die maximale Dauer der gesamten Selbstheilung abgeschätzt werden. Für die Abschätzung der Zuteilungsdauern der einzelnen Prioritätslevel muss dagegen sogar die Anzahl der ausgefallenen Tasks *pro Prioritätslevel* bekannt sein. Dieses Wissen ist im Voraus jedoch zumeist nicht vorhanden, was eine exakte Analyse erschwert. Vielmehr müssen die relevanten Größen für praktische Anwendungen anhand der Kapazitäten der einzelnen PEs sowie der konkreten Taskmengen und deren Prioritäten im Rahmen einer Worst-Case-Analyse abgeschätzt werden. Eine vergleichbare Analyse war jedoch auch für das AHS ohne Prioritätsunterstützung nötig: Dort musste zwar nur der Parameter m_f abgeschätzt werden, dafür konnte jedoch keinerlei Garantie über die Zuteilungsreihenfolge der ausgefallenen Tasks gegeben werden. Folglich macht die prioritätsbasierte Task-Decision-Strategie zwar eine komplexere Analyse des Worst Case nötig, bietet

dafür im Rahmen der Selbstheilung aber durch die wohldefinierte Zuteilungsreihenfolge auch völlig neue Qualitäten.

A.3. Betrachtung der unteren Schranke a_{min} für die Erkennung eines PE-Ausfalls

Bisher wurden PE-Ausfälle jeweils *nach* dem Senden der Hormone simuliert, was dem Worst-Case-Zeitpunkt entspricht und dazu führte, dass der Ausfall erst nach $a = 2$ Hormonzyklen erkannt werden konnte. Um auch die zur Ausfallerkennung *mindestens* benötigte Zeit a_{min} zu überprüfen, die gemäß Bemerkung 6.16 lediglich einen Hormonzyklus beträgt, wurde ein weiteres Experiment konzipiert.

Konkret wurden zwei nahezu identische Simulationsläufe durchgeführt: In beiden wurde ein Ausfall von $\text{PE}_{(1,1)}$ in Hormonzyklus 40 simuliert. Im ersten Durchlauf geschah dieser *nach* dem Senden seiner Hormone, im zweiten dagegen *vor* dem Senden. Der erste Durchlauf entspricht daher dem Worst Case, während der zweite ein Best-Case-Szenario liefern sollte.

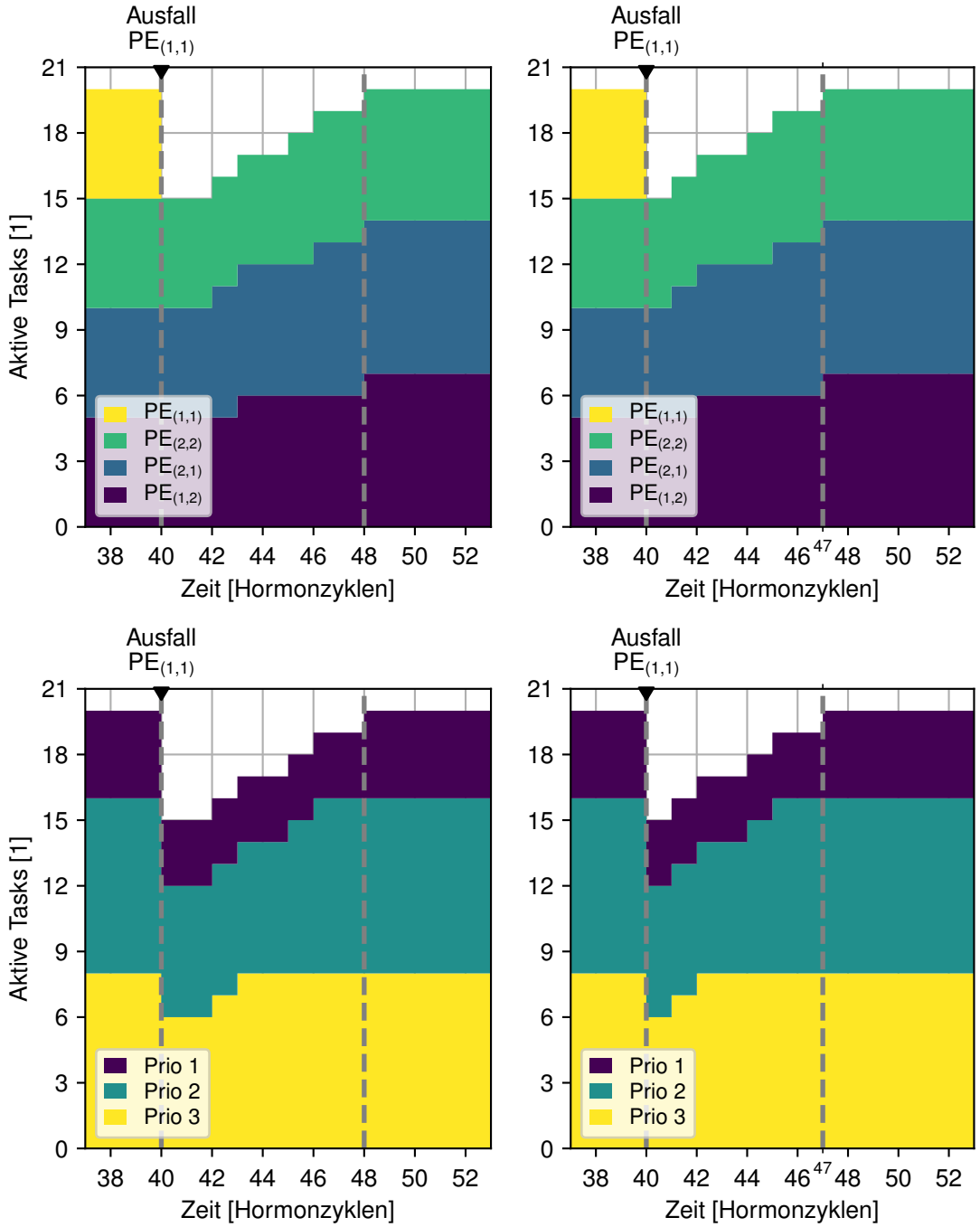
Die Verläufe der resultierenden Selbstheilungen sind in Abbildung A.4 gegenübergestellt. Es fällt auf, dass beide Selbstheilungen nahezu identisch ablaufen. Konkret besteht der einzige Unterschied darin, dass die Zuteilung der ersten Task im zweiten Szenario einen Hormonzyklus früher erfolgt und die Selbstheilung daher ebenfalls einen Zyklus früher beendet ist. Abgesehen von diesem Unterschied laufen jedoch beide Selbstheilungen identisch ab – insbesondere werden alle Tasks auf jeweils denselben PEs zugeteilt, weshalb die Taskverteilungen nach Abschluss der Selbstheilungen in beiden Szenarien identisch sind, ebenso wie dies bereits vor dem Ausfall von $\text{PE}_{(1,1)}$ der Fall war.

Folglich bestätigt dieses Resultat die theoretisch hergeleitete untere Schranke von $a_{\text{min}} = 1$ Hormonzyklus für die Erkennung eines PE-Ausfalls: Im Worst Case, der im ersten Durchlauf vorliegt, wird der Ausfall von $\text{PE}_{(1,1)}$ nach zwei Hormonzyklen erkannt, woraufhin die erste Task erneut zugeteilt wird; im zweiten Durchlauf liegt dagegen der Best Case vor und der Ausfall wird bereits einen Hormonzyklus früher erkannt.

Auf eine genaue Betrachtung des Zeitverhaltens der Selbstheilung wird an dieser Stelle verzichtet. Der Vollständigkeit halber sei lediglich erwähnt, dass die Zeitschranken für die Dauer der gesamten Selbstheilung sowie der Zuteilungsdauer der einzelnen Prioritätslevel im ersten Durchlauf jeweils exakt erreicht werden; der zweite Durchlauf eilt diesen Schranken dagegen stets genau jenen einen Hormonzyklus voraus, welcher bei der Erkennung des Ausfall von $\text{PE}_{(1,1)}$ eingespart werden konnte.

A.4. Vergleich mit der AHS-Middleware

Zum Abschluss dieses Evaluationskapitels wollen wir erneut die Implementierungen von Hormonsimulator und AHS-Middleware miteinander vergleichen, um sicherzustellen,



(a) Worst Case: Ausfall *nach* Senden der Hormone in Zyklus 40

(b) Best Case: Ausfall *vor* Senden der Hormone in Zyklus 40

Abbildung A.4: Zeitlicher Verlauf der Selbstheilung nach einem PE-Ausfall zu verschiedenen Zeitpunkten während eines Hormonzyklus

dass sich die Selbstheilung in beiden Fällen identisch verhält und die im Simulator gewonnenen Evaluationsergebnisse folglich auch auf die Middleware übertragbar sind.

Hierzu wurde erneut das aus Kapitel 7.6 bzw. Abbildung 7.25 bekannte, aus 41 Tasks mit fünf verschiedenen Prioritäten bestehende Szenario verwendet. Um bei einem PE-Ausfall mehr Tasks neu zuteilen zu müssen, wurde jedoch die Anzahl der PEs von neun auf vier reduziert.

Abbildung A.5 zeigt den zeitlichen Verlauf der resultierenden Selbstheilung in der Middleware, wenn PE_3 nach einer anfänglichen Selbstkonfiguration (die nicht abgebildet ist) in Hormonzyklus 56 ausfällt. Der Ausfall geschah hier zum Worst-Case-Zeitpunkt, weshalb er nach $a = 2$ Hormonzyklen erkannt wurde. Im Anschluss findet erneut die gewohnte Selbstkonfiguration der betroffenen Tasks statt; die Wartezyklen nach vollständiger Zuteilung der einzelnen Prioritätslevel sind gut erkennbar.

Auf eine exakte Betrachtung des Zeitverhaltens wird hier erneut verzichtet, die beobachtete Dauer der Selbstheilung sowie die Zuteilungsdauern der einzelnen Prioritätslevel erreichen jedoch wieder exakt die theoretisch hergeleiteten oberen Schranken.

Eine Wiederholung dieses Szenarios im Hormonsimulator, bei welcher in Hormonzyklus 56 der Ausfall von $PE_{(3,1)}$ nach dem Senden seiner Hormone simuliert wurde, lieferte einen identischen Verlauf der Selbstheilung: Es ergaben sich keine Unterschiede im Zeitverhalten, ebenso war die Verteilung der Tasks auf die einzelnen PEs zu jedem Zeitpunkt identisch zur Middleware. Daher wurde auf die Abbildung der Ergebnisse dieses Experiments verzichtet.

Aufgrund des vollkommen identischen Verhaltens darf daher mit hoher Sicherheit angenommen werden, dass sich Hormonsimulator und AHS-Middleware auch in Bezug auf die Selbstheilung in Unterlastsituationen identisch verhalten.

A.5. Zusammenfassung

Die Resultate dieses Kapitels zusammenfassend lässt sich festhalten, dass sich die Selbstheilung in Unterlastsituationen stets so verhält, wie es von den theoretischen Betrachtungen vorhergesagt wird: Der Ausfall eines PEs wird frühestens nach $a_{\min} = 1$ und spätestens nach $a = 2$ Hormonzyklen erkannt, woraufhin eine Selbstkonfiguration aller zusammen mit dem PE ausgefallenen Tasks stattfindet. Die in diesem Kapitel betrachteten Evaluationsszenarien bestätigten die theoretisch hergeleiteten Zeitschranken. Angesichts der Tatsache, dass der zugrunde liegende Selbstkonfigurationsmechanismus bereits in Kapitel 7 ausführlich evaluiert wurde, ist ferner zu erwarten, dass auch eine Untersuchung weiterer Szenarien im Hormonsimulator oder der AHS-Middleware kein abweichendes Verhalten beobachten ließe.

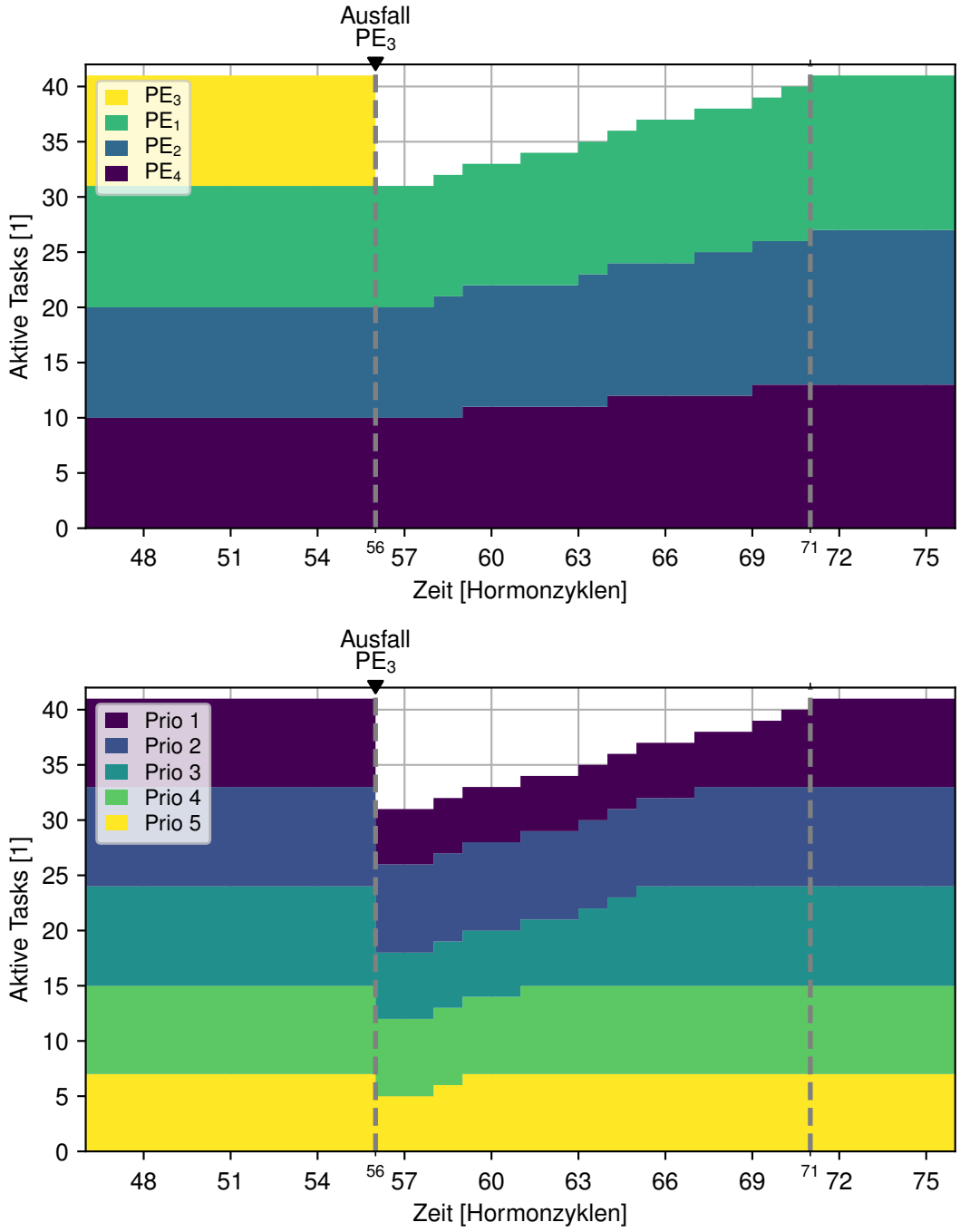


Abbildung A.5: Zeitlicher Verlauf einer Selbstheilung in der AHS-Middleware

B Average-Case-Analyse der naiven Task-Dropping-Strategie

Wie bereits in Kapitel 8.5.2 beschrieben, wurde im Rahmen dieser Dissertation eine Average-Case-Analyse der naiven Task-Dropping-Strategie durchgeführt, um einen Vergleich zum Worst Case ziehen zu können. Die Ergebnisse dieser Analyse werden im Folgenden beschrieben. Zunächst betrachten wir jedoch einige dafür notwendige mathematische Grundlagen.

B.1. Mathematische Grundlagen

B.1.1. Steigende und fallende Faktorielle

Zunächst führen wir zwei Funktionen ein, die als Verallgemeinerungen der Fakultätsfunktion angesehen werden können:

Steigende Faktorielle, $n^{\overline{k}}$

Definition B.1. Sei $n \in \mathbb{Z}$ und $k \in \mathbb{N}$. Dann sei die *steigende Faktorielle* $n^{\overline{k}}$ definiert als

$$n^{\overline{k}} := \underbrace{n \cdot (n+1) \cdots (n+k-1)}_{k \text{ Faktoren}} = \prod_{i=n}^{n+k-1} i.$$

Bemerkung B.2. Für $n \geq 1$ lässt sich die steigende Faktorielle als Quotient von Fakultäten ausdrücken:

$$n^{\overline{k}} = \frac{(n+k-1)!}{(n-1)!}.$$

Fallende Faktorielle, $n^{\underline{k}}$

Definition B.3. Sei $n \in \mathbb{Z}$ und $k \in \mathbb{N}$. Dann sei die *fallende Faktorielle* $n^{\underline{k}}$ definiert als

$$n^{\underline{k}} := \underbrace{n \cdot (n-1) \cdots (n-k+1)}_{k \text{ Faktoren}} = \prod_{i=n-k+1}^{n} i.$$

Bemerkung B.4. Für $n \geq 1$ und $k \leq n$ lässt sich die fallende Faktorielle als Quotient von Fakultäten ausdrücken:

$$n^{\underline{k}} = \frac{n!}{(n-k)!}.$$

Bemerkung B.5. Gemäß obiger Definitionen lassen sich steigende und fallende Faktorielle wie folgt ineinander umwandeln:

$$n^{\overline{k}} = (n+k-1)^{\underline{k}}, \qquad n^{\underline{k}} = (n-k+1)^{\overline{k}}.$$

B.1.2. Binomialkoeffizienten

Mithilfe steigender bzw. fallender Faktoriellen lassen sich nun Binomialkoeffizienten wie folgt definieren:

Binomialkoeffizient, $\binom{n}{k}$

Definition B.6. Für $n, k \in \mathbb{Z}$ sei der *Binomialkoeffizient* $\binom{n}{k}$ definiert durch

$$\begin{aligned}\binom{n}{k} &= \begin{cases} \frac{n^{\underline{k}}}{k!}, & k \geq 0 \\ 0, & k < 0 \end{cases} \\ &= \begin{cases} \frac{(n-k+1)^{\overline{k}}}{k!}, & k \geq 0 \\ 0, & k < 0. \end{cases}\end{aligned}$$

Bemerkung B.7. Umgekehrt gilt insbesondere

$$\frac{n^{\overline{k}}}{k!} = \binom{n+k-1}{k},$$

diese Gleichung wird in Anhang C im Beweis von Lemma C.3 verwendet.

Bemerkung B.8. Für $n, k \in \mathbb{N}$ mit $0 \leq k \leq n$ ergibt sich die bekannte kombinatorische Definition

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

als Spezialfall von Definition B.6.

B.1.2.1. Eigenschaften und Identitäten

Ferner werden die folgenden Eigenschaften von Binomialkoeffizienten vorausgesetzt:

Lemma B.9. Seien $n, k \in \mathbb{N}$ mit $k > n$.
Dann gilt

$$\binom{n}{k} = 0.$$

Beweis. Wegen $k \geq 0$ gilt

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!} = \frac{\prod_{i=n-k+1}^{n} i}{k!}.$$

Wegen $k > n$ gilt $k \geq n+1$, also $n-k+1 \leq 0$. Weil n und k ganzzahlig sind, enthält der Zähler also insbesondere den Faktor Null, woraus die Behauptung folgt. □

Lemma B.10. Seien $n \in \mathbb{N}$ und $k \in \mathbb{Z}$.
Dann gilt

$$\binom{n}{k} = \binom{n}{n-k}.$$

Beweis. Es sind drei Fälle zu unterscheiden:

a) $k < 0$. Gemäß Definition gilt dann $\binom{n}{k} = 0$. Wegen $n - k > n \geq 0$ gilt nach Lemma B.9 jedoch auch $\binom{n}{n-k} = 0$.

b) $0 \leq k \leq n$. In diesem Fall kann die kombinatorische Definition des Binomialkoeffizienten verwendet werden:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} = \frac{n!}{(n-(n-k))! \cdot (n-k)!} = \binom{n}{n-k}.$$

c) $k > n$. Wegen $n - k < 0$ gilt dann nach Definition $\binom{n}{n-k} = 0$. Wegen $k > n \geq 0$ gilt nach Lemma B.9 jedoch auch $\binom{n}{k} = 0$. □

Definition B.6 definiert $\binom{n}{k}$ auch für $n < 0$; das folgende Lemma ermöglicht in solchen Fällen eine Transformation auf Binomialkoeffizienten mit nichtnegativem n:

Lemma B.11. Seien $n, k \in \mathbb{Z}$. Dann gilt

$$\binom{n}{k} = (-1)^k \cdot \binom{k-n-1}{k}. \tag{B.1}$$

Beweis. Für $k \geq 0$ gilt

$$\begin{aligned}
\binom{n}{k} &= \frac{n^{\underline{k}}}{k!} \\
&= \frac{\overbrace{n \cdot (n-1) \cdots (n-k+1)}^{k \text{ Faktoren}}}{k!} \\
&= (-1)^k \cdot \frac{(-n) \cdot (-n+1) \cdots (-n+k-1)}{k!} \\
&= (-1)^k \cdot \frac{(k-n-1) \cdot (k-n-2) \cdots \overbrace{(k-n-1-k+1)}^{-n}}{k!} \\
&= (-1)^k \cdot \frac{(k-n-1)^{\underline{k}}}{k!} \\
&= (-1)^k \cdot \binom{k-n-1}{k}.
\end{aligned}$$

Für $k < 0$ gilt dagegen

$$\binom{n}{k} = 0 = (-1)^k \cdot 0 = (-1)^k \cdot \binom{k-n-1}{k}.$$ □

Zu guter Letzt erweist sich folgendes Lemma als nützlich:

Lemma B.12. Seien $k, l, m, n \in \mathbb{N}$ sowie $l \geq n$.
Dann gilt

$$\sum_{i=0}^{m} \binom{m-i}{k} \cdot \binom{n+i}{l} = \binom{m+n+1}{k+l+1}.$$

Beweis. Siehe [GKP94, ex. 5.14] oder [Knu97, 1.2.6, ex. 20]. □

Bemerkung

Abgesehen von einer angepassten Variablenbezeichnung entstammt Lemma B.12 dem *ersten* Druck der zweiten Auflage von [GKP94, 5.26] aus dem Jahre 1994. In späteren Drucken derselben Auflage findet sich stattdessen die Gleichung

$$\sum_{i=-n}^{m} \binom{m-i}{k} \cdot \binom{n+i}{l} = \binom{m+n+1}{k+l+1}$$

mit den Bedingungen $k, l \in \mathbb{N}$ und $m, n \in \mathbb{Z}$ mit $m + n \geq 0$, wobei die Variablenbezeichnung hier abermals angepasst wurde.

Laut der zum Buch gehörenden Errata-Website [Knu] stellt diese Form eine Verallgemeinerung der vorherigen Gleichung dar und ersetzte diese zu einem nicht näher spezifizierten Zeitpunkt zwischen Januar 1998 und Mai 2013.

Im Rahmen dieser Dissertation ist die speziellere Form der Gleichung, wie sie durch Lemma B.12 gegeben ist, jedoch ausreichend. In dieser Form findet sich die Gleichung ebenfalls in [Knu97, 1.2.6, eq. 25].

B.2. Annahmen

Mit diesen mathematischen Grundlagen kann nun der Average Case der naiven Task-Dropping-Strategie analysiert werden. Hierzu wird zum einen das aus Kapitel 8 bekannte Überlastmodell zugrunde gelegt, zum anderen basieren alle folgenden Überlegungen auf einer zusätzlichen fundamentalen Annahme, die bereits in Kapitel 8.5.2 erwähnt wurde:

Annahme B.13. Alle Verteilungen der $m(v+1)$ Tasks auf die $v+1$ PEs sind gleich wahrscheinlich.

Bemerkung B.14

Diese Annahme ist eine Voraussetzung, um die Komplexität einer Average-Case-Analyse beherrschen zu können.

Im Allgemeinen unterliegt die Verteilung der Tasks auf die verfügbaren PEs vielen verschiedenen Faktoren; unter anderem sind als Einflussfaktoren auf die Taskverteilung

- die statischen Eignungswerte der PEs für die verschiedenen Tasks,

- die Stärke der Acceleratoren, welche der Bildung virtueller Organe verwandter Tasks auf benachbarten PEs dienen,
- Verwandtschaftsverhältnisse zwischen den Tasks sowie
- Nachbarschaftsbeziehungen zwischen den PEs

zu nennen.

Da sich diese Faktoren – wenn überhaupt – nur sehr schwer allgemein modellieren lassen, wurde auf eine solche Modellierung im Rahmen der Average-Case-Analyse verzichtet.

Abgesehen von diesen Annahmen laufe die Selbstheilung in einer Überlastsituation unter Verwendung der naiven Task Dropping Strategie wie in Kapitel 8.5 beschrieben ab: Nach der Erkennung des Ausfalls von $\mathrm{PE}_\times$ stoppen $\mathrm{PE}_1 \dots \mathrm{PE}_v$ jeweils alle Tasks, die eine geringere Priorität haben, als die höchstpriore nicht zugeteilte Task $T_{\max}$ (die zuvor auf $\mathrm{PE}_\times$ zugeteilt war). Im Anschluss wird eine Selbstkonfiguration für die k gerade gestoppten Tasks sowie die m zuvor auf $\mathrm{PE}_\times$ zugeteilten Tasks durchgeführt, wobei aufgrund der Überlastsituation jedoch nur noch k Tasks zugeteilt werden können.

B.3. Erwartete Anzahl Taskstopps

Für eine gemäß Annahme B.13 zufällige Verteilung der $m(v+1)$ Tasks auf die PEs $\mathrm{PE}_1 \dots \mathrm{PE}_v$ sowie $\mathrm{PE}_\times$ lässt sich nun zunächst die erwartete Anzahl Taskstopps bei Verwendung der naiven Task-Dropping-Strategie bestimmen:

Satz B.15. Sei X eine Zufallsvariable, welche die Anzahl der von der naiven Task-Dropping-Strategie gestoppten Tasks angibt.

Dann gilt für den Erwartungswert $\mathbf{E}[X]$ dieser Zufallsvariablen

$$\mathbf{E}[X] = \frac{m^2 v}{1+m}.$$

Im Rahmen dieser Dissertation wurden verschiedene Beweise dieser Aussage entwickelt. Der folgende Beweis, der bereits in [HB20] veröffentlicht wurde, berechnet den Erwartungswert direkt und kommt daher mit vergleichsweise wenigen zusätzlichen Lemmata aus. Anhang D enthält dagegen zwei weitere Beweise dieser Aussage, die eine andere Sichtweise auf das zugrunde liegende Problem einnehmen und Teile der Herleitung des folgenden Beweises ersetzen können.

Beweis. Wir bestimmen zunächst die Wahrscheinlichkeitsverteilung von X. Dazu nehmen wir an, dass die $m(v+1)$ Tasks, die aufgrund der Annahmen des Überlastmodells alle verschiedene Prioritäten haben, nacheinander in der Reihenfolge *absteigender* Prioritäten *zufällig* auf

- mv Positionen auf $\mathrm{PE}_1 \dots \mathrm{PE}_v$ sowie

- m Positionen auf $\mathrm{PE}_\times$

verteilt werden.

Offensichtlich kann X Werte zwischen 0 und mv annehmen: Im Best Case muss keine Task gestoppt werden, da zum Zeitpunkt des Ausfalls die m Tasks mit den kleinsten Prioritäten auf $\mathrm{PE}_\times$ zugeteilt waren; im Worst Case werden dagegen mv Tasks gestoppt.

Es gilt hierbei:

- $X = mv$ $\Leftrightarrow$ Die höchstpriore Task wird auf eine der m Positionen auf $\mathrm{PE}_\times$ gesetzt (von insgesamt $m(v+1)$ freien Positionen auf allen PEs).
- $X = mv - 1 \Leftrightarrow$ Die zweit-höchstpriore Task ist die erste Task, die auf eine der m Positionen auf $\mathrm{PE}_\times$ gesetzt wird.

  Entsprechend muss zunächst die höchstpriore Task auf eine beliebige der mv Positionen auf $\mathrm{PE}_1 \ldots \mathrm{PE}_v$ gesetzt werden (von insgesamt $m(v+1)$ freien Positionen auf allen PEs), anschließend muss die zweit-höchstpriore Task auf eine beliebige der m Positionen auf $\mathrm{PE}_\times$ gesetzt werden (von nun insgesamt nur noch $m(v+1) - 1$ freien Positionen auf allen PEs).
- $X = mv - 2 \Leftrightarrow$ Die dritt-höchstpriore Task ist die erste Task, die auf eine der m Positionen auf $\mathrm{PE}_\times$ gesetzt wird.

  Entsprechend müssen zunächst die höchst- und zweit-höchstprioren Tasks jeweils auf beliebige Positionen auf $\mathrm{PE}_1 \ldots \mathrm{PE}_v$ gesetzt werden (die höchstpriore Task daher auf eine von mv Positionen auf diesen PEs bei insgesamt $m(v+1)$ freien Positionen auf allen PEs; die zweit-höchstpriore Task im Anschluss auf eine der verbleibenden $mv-1$ Positionen bei insgesamt noch $m(v+1)-1$ freien Positionen auf allen PEs). Im Anschluss muss die dritt-höchstpriore Task auf eine beliebige der m Positionen auf $\mathrm{PE}_\times$ gesetzt werden (bei insgesamt noch $m(v+1)-2$ freien Positionen auf allen PEs).

  $\vdots$
- $X = 0$ $\Leftrightarrow$ Die $(mv+1)$-höchstpriore (bzw. die m-niedrigstpriore) Task ist die erste Task, die auf eine der m Positionen auf $\mathrm{PE}_\times$ gesetzt wird.

  Entsprechend müssen die mv höchstprioren Tasks zunächst jeweils auf freie Positionen von $\mathrm{PE}_1 \ldots \mathrm{PE}_v$ gesetzt werden. Die $(mv+1)$-höchstpriore Task muss anschließend auf eine der m Positionen von $\mathrm{PE}_\times$ gesetzt werden (dies sind nun ohnehin *alle* noch freien Positionen, da alle mv Positionen der übrigen PEs zuvor mit den mv höchstprioren Tasks gefüllt wurden).

Für die Wahrscheinlichkeitsverteilung von X folgt somit:

$$\mathbf{P}(X = mv) = \underbrace{\frac{m}{m(v+1)}}_{\substack{\text{höchstpriore} \\ \text{Task: auf } \mathrm{PE}_\times}}$$

$$\mathbf{P}(X = mv-1) = \underbrace{\frac{mv}{m(v+1)}}_{\substack{\text{höchstpriore}\\ \text{Task: } \textit{nicht}\\ \text{auf } \mathrm{PE}_\times}} \cdot \underbrace{\frac{m}{m(v+1)-1}}_{\substack{\text{zweit-höchstpriore}\\ \text{Task: auf } \mathrm{PE}_\times}}$$

$$\mathbf{P}(X = mv-2) = \underbrace{\frac{mv}{m(v+1)} \cdot \frac{mv-1}{m(v+1)-1}}_{\substack{\text{höchst- und zweit-höchstpriore}\\ \text{Tasks: } \textit{nicht} \text{ auf } \mathrm{PE}_\times}} \cdot \underbrace{\frac{m}{m(v+1)-2}}_{\substack{\text{dritt-höchstpriore}\\ \text{Task: auf } \mathrm{PE}_\times}}$$

$$\vdots$$

$$\mathbf{P}(X = 0) = \underbrace{\frac{mv}{m(v+1)} \cdot \frac{mv-1}{m(v+1)-1} \cdots \frac{1}{m+1}}_{\substack{mv \text{ höchstpriore Tasks: } \textit{nicht} \text{ auf } \mathrm{PE}_\times\\ \text{(sondern auf } \mathrm{PE}_1 \ldots \mathrm{PE}_v)}} \cdot \underbrace{\frac{m}{m}}_{\substack{(mv+1)\text{-höchstpriore}\\ \text{Task: auf } \mathrm{PE}_\times}}$$

Verallgemeinert ergibt sich

$$\mathbf{P}(X = j) = \frac{\prod_{i=j+1}^{mv} i}{\prod_{i=m+j+1}^{mv+m} i} \cdot \frac{m}{m+j} = m \cdot \frac{\prod_{i=j+1}^{mv} i}{\prod_{i=m+j}^{mv+m} i} \tag{B.2}$$

für $0 \leq j \leq mv$.

Damit lässt sich nun der Erwartungswert von X berechnen:

$$\begin{aligned}
\mathbf{E}[X] &= \sum_{j=0}^{mv} j \cdot \mathbf{P}(X = j) \\
&= \sum_{j=1}^{mv} j \cdot m \cdot \frac{\prod_{i=j+1}^{mv} i}{\prod_{i=m+j}^{mv+m} i} \\
&= m \cdot \sum_{j=1}^{mv} j \cdot \frac{(mv)!/j!}{(mv+m)!/(m+j-1)!} \\
&= m \cdot \sum_{j=1}^{mv} j \cdot \frac{(mv)!\,(m+j-1)!}{(mv+m)!\,j!} \\
&= m \cdot \sum_{j=1}^{mv} \frac{(mv)!\,(m+j-1)!}{(mv+m)!\,(j-1)!} \\
&= m \cdot \sum_{j=1}^{mv} \frac{(m+j-1)!/(j-1)!}{(mv+m)!/(mv)!}
\end{aligned}$$

$$= \frac{m}{\prod\limits_{i=mv+1}^{mv+m} i} \cdot \sum_{j=1}^{mv} \left(\prod_{i=j}^{m+j-1} i \right)$$
$$= \frac{m}{(mv+1)^{\overline{m}}} \cdot \sum_{j=1}^{mv} j^{\overline{m}}$$

Unter Anwendung von Lemma C.3 ergibt sich schließlich

$$= \frac{m}{(mv+1)^{\overline{m}}} \cdot \frac{mv \cdot (mv+1)^{\overline{m}}}{1+m}$$
$$= \frac{m^2 v}{1+m}.$$

□

B.4. Erwartete Dauer der Selbstheilung

Die erwartete Dauer der Selbstheilung in einer Überlastsituation ist nun unmittelbar abhängig von der erwarteten Anzahl Taskstopps: Werden $k > 0$ Tasks gestoppt, so ist die Selbstheilung beendet, sobald die k höchstprioren nicht zugeteilten Tasks zugeteilt wurden; werden dagegen *keine* Tasks gestoppt, so ist die Selbstheilung gemäß Definition 8.2 jedoch erst beendet, sobald der Ausfall von $\mathrm{PE}_\times$ erkannt wurde.

Bevor im ersten Fall $k > 0$ Tasks gestoppt werden, muss die Überlastsituation jedoch ebenfalls zunächst erkannt werden. Im Falle der Worst-Case-Analysen wurde die hierzu nötige Zeit durch eine konstante Zahl von $a + 1$ Hormonzyklen abgeschätzt. Satz 8.4 zeigte ferner, dass eine Überlastsituation frühestens nach $a_{\min} + 1$ Hormonzyklen erkannt werden kann. Da die Werte von a und $a_{\min}$ nicht identisch sind, muss die zur Erkennung des Ausfalls von $\mathrm{PE}_\times$ nötige Zeit im Rahmen der Average-Case-Analyse durch eine Zufallsvariable A modelliert werden.

Der Zeitpunkt dieser Erkennung ist jedoch vom genauen Ausfallzeitpunkt von $\mathrm{PE}_\times$ innerhalb des Hormonzyklus abhängig. Um die Analyse überschaubar zu halten, wird A daher eine unbekannte Verteilung zugrunde gelegt: Die Bestimmung der Verteilung von A würde zum einen eine noch genauere Betrachtung der zeitlichen Aspekte eines Hormonzyklus (wie beispielsweise in [vRBP11a] beschrieben) erfordern. Zum anderen müsste hierzu möglicherweise ein konkretes Kommunikationsmedium angenommen werden, was zu einer weniger allgemeinen Analyse führen würde. Daher wird A im Folgenden lediglich nach oben und unten hin abgeschätzt; es ist klar, dass der Wert von A zwischen beiden möglichen Extremen liegen muss, es gilt also $a_{\min} \leq A \leq a$.

Da die Selbstheilung in einer Überlastsituation gemäß Definition 8.2 erst dann beendet ist, wenn der Ausfall von $\mathrm{PE}_\times$ erkannt wurde, dauert eine Selbstheilung mindestens $a_{\min}$ Hormonzyklen (nämlich dann, wenn der Ausfall von $\mathrm{PE}_\times$ frühestmöglich erkannt werden kann und keine Tasks gestoppt werden müssen).

Zunächst muss jedoch die Dauer einer Selbstheilung in einer Überlastsituation in Abhängigkeit der Anzahl gestoppter Tasks möglichst exakt abgeschätzt werden. Dies leistet der folgende Satz:

Satz B.16. Sei k die Anzahl der von der naiven Task-Dropping-Strategie gestoppten Tasks und sei t_{SH} die sich daraus ergebende und in Hormonzyklen gemessene Dauer der Selbstheilung in einer Überlastsituation.

Dann gelten

$$t_{\mathrm{SH}} \leq k + \max\{0, k-1\} \cdot [v > 1] + \min\{k, 1\} + a \text{ und}$$
$$t_{\mathrm{SH}} \geq 2k - \min\{m, k\} + \min\{k, 1\} + a_{\min}.$$

Beweis. Die obere Schranke $t_{\mathrm{SH}} \leq k + \max\{0, k-1\} \cdot [v > 1] + \min\{k, 1\} + a$ ergibt sich mit einer ähnlichen Argumentation wie im Beweis zu Satz 8.11, wobei hier jedoch nicht vom Worst Case $k = mv$ ausgegangen wird:

Spätestens nach a Hormonzyklen wird der Ausfall von $\mathrm{PE}_\times$ bemerkt. Nun sind zwei Fälle zu unterscheiden:

a) $k = 0$: Es werden keine Tasks gestoppt, die Dauer der Selbstheilung beträgt daher gemäß Definition 8.2 gerade die zur Erkennung des Ausfalls von $\mathrm{PE}_\times$ nötigen a Hormonzyklen. Wegen $k = 0$ stimmt dieser Term jedoch – unabhängig von v – mit der Behauptung $k + \max\{0, k-1\} \cdot [v > 1] + \min\{k, 1\} + a$ überein.

b) $k > 0$: Im Zyklus nach der Erkennung des Ausfalls werden das Vorliegen der Überlastsituation erkannt sowie $k > 0$ Tasks gestoppt. Beginnend mit dem darauffolgenden Hormonzyklus werden die k höchstprioren nicht zugeteilten Tasks zugeteilt. Dies entspricht einer Selbstkonfiguration mit k Tasks. Da gemäß der Annahmen alle Tasks verschiedene Prioritäten haben, sind hierzu k Hormonzyklen mit Zuteilungen sowie $k - 1$ Wartezyklen nötig, die aber gemäß Bemerkung 6.8 nur anfallen, wenn $v > 1$ gilt. Zusammengefasst dauert die Selbstkonfiguration daher höchstens $k + (k-1) \cdot [v > 1]$ Hormonzyklen.

   In Summe ergeben sich $t_{\mathrm{SH}} \leq k + (k-1) \cdot [v > 1] + 1 + a$ Hormonzyklen, wegen $k > 0$ ist dies identisch mit der Behauptung $t_{\mathrm{SH}} \leq k + \max\{0, k-1\} \cdot [v > 1] + \min\{k, 1\} + a$.

Der unteren Schranke liegen ebenfalls die Beobachtungen aus Bemerkung 6.8 zugrunde: Im vorliegenden Szenario haben alle Tasks verschiedene Prioritäten. Wird in einem Zyklus i eine Task T zugeteilt, so ist Zyklus $i + 1$ nur dann ein Wartezyklus, wenn in Zyklus i mindestens zwei PEs einen positiven modifizierten Eignungswert für T gesendet hatten.

Falls das System sich nach dem Ausfall von $\mathrm{PE}_\times$ bereits in einem stabilen Zustand befindet (weil keine Tasks gestoppt werden müssen, also $k = 0$ gilt), ist die Selbstheilung beendet, sobald der Ausfall erkannt wurde. Dies ist frühestens nach $a_{\min}$ Hormonzyklen der Fall. Wegen $k = 0$ (und $m \geq 1$) gilt daher $t_{\mathrm{SH}} \geq a_{\min} = 2k - \min\{m, k\} + \min\{k, 1\} + a_{\min}$, womit die Behauptung folgt.

Für $k > 0$ ist dagegen zu unterscheiden, wie viele PEs nach dem Ausfall von $\mathrm{PE}_\times$ verbleiben:

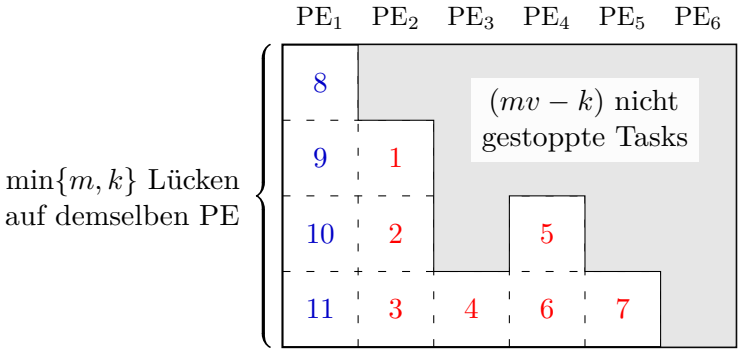
a) $v = 1$: Der Ausfall von $\mathrm{PE}_\times$ wird frühestens nach $a_{\min}$ Hormonzyklen bemerkt. Im darauffolgenden Hormonzyklus wird das Vorliegen der Überlastsituation erkannt

und es werden $k > 0$ Tasks gestoppt. Bei der anschließenden Selbstkonfiguration der k höchstprioren nicht zugeteilten Tasks treten keinerlei Wartezyklen auf, da nur noch ein PE verbleibt, folglich dauert die Zuteilung dieser Tasks gerade k Hormonzyklen.

In der Summe ergeben sich also mindestens $k + 1 + a_{\min}$ Hormonzyklen. Wegen $v = 1$ gilt jedoch $k \leq m$ und daher insbesondere $\min\{m, k\} = k$, wegen $k > 0$ gilt ferner $\min\{k, 1\} = 1$. Damit folgt die Behauptung $t_{\text{SH}} \geq 2k - \min\{m, k\} + \min\{k, 1\} + a_{\min}$.

b) $v > 1$: In diesem Fall können Wartezyklen auftreten, da mehr als ein PE positive modifizierte Eignungswerte senden kann.

Im Best Case stammen jedoch möglichst viele der k gestoppten Tasks vom selben PE und die Reihenfolge der Zuteilung der Tasks auf die PEs ist günstig, sodass dieses PE möglichst viele Tasks ohne andere PEs als Konkurrenten übernehmen kann. Dies ist im folgenden Beispiel illustriert, in dem die abgebildeten Zahlen jedoch im Gegensatz zu bisherigen Beispielen *nicht* für Tasks (bzw. deren Prioritäten) stehen, sondern die durch Stoppen von k Tasks entstandenen „Lücken“ nummerieren:



Im Best Case existiert also ein PE, auf dem genau $\min\{m, k\}$ Tasks gestoppt wurden; im Beispiel ist dies PE_1 und es gilt zudem $\min\{m, k\} = m$. Der Übersicht halber sind die Lücken auf PE_1 in blau markiert, während die Lücken auf den übrigen PEs in rot dargestellt werden.

Je nachdem, in welcher Reihenfolge die insgesamt k Lücken nun mit den k höchstprioren nicht zugeteilten Tasks gefüllt werden, konkurrieren unterschiedlich viele PEs um die Übernahme der einzelnen Tasks:

- Werden die Lücken beispielsweise in der Reihenfolge

$$8, 9, 10, 11, 1, 2, 3, 4, 5, 6, 7$$

gefüllt, konkurrieren bei den *ersten zehn Zuteilungen* immer mindestens zwei PEs um die Übernahme der jeweils aktuell höchstprioren nicht zugeteilten Task $T_{\max}$: Zu jedem Zeitpunkt existieren Lücken auf mindestens zwei verschiedenen PEs, weshalb $T_{\max}$ prinzipiell von jedem dieser PEs übernommen

werden könnte. Entsprechend treten in diesen Fällen aufgrund der um $T_{\max}$ konkurrierenden PEs Wartezyklen nach der jeweiligen Taskübernahme auf.

Beim Füllen der *letzten* Lücke existiert dagegen nur noch ein einziges PE, auf dem $T_{\max}$ zugeteilt werden könnte. In diesem Fall tritt also kein Wartezyklus auf, da keine PEs um die Übernahme von $T_{\max}$ konkurrieren.

Als Resultat treten bei k Lücken daher gerade $k-1$ Wartezyklen auf, diese Zuteilungsreihenfolge entspricht also dem Worst Case.

- Werden die Lücken dagegen in der Reihenfolge

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11$$

gefüllt, so treten nur *bei den ersten sieben Zuteilungen* Wartezyklen auf, da nur hier jeweils mehrere PEs um die Zuteilung von $T_{\max}$ konkurrieren. Die anschließenden Taskübernahmen, welche die vier verbleibenden Lücken füllen, rufen dagegen keine weiteren Wartezyklen hervor: Da sich diese Lücken allesamt auf PE_1 befinden, konkurriert nun kein weiteres PE mehr um die zu übernehmenden Tasks.

Somit treten hier nur Wartezyklen für jene Tasks auf, welche von $\mathrm{PE}_2 \ldots \mathrm{PE}_5$ übernommen werden.

Diese Zuteilungsreihenfolge entspricht daher dem Best Case.

Verallgemeinert man diese Best-Case-Zuteilungsreihenfolge, so ergibt sich die Dauer der Selbstheilung für $k > 0$ zur Summe aus *mindestens*

- $a_{\min}$ Hormonzyklen zur Erkennung des Ausfalls von $\mathrm{PE}_\times$,
- einem Hormonzyklus zur Erkennung der Überlastsituation und Stopp von $k > 0$ Tasks,
- k Hormonzyklen für die eigentliche Zuteilung der k höchstprioren nicht zugeteilten Tasks (hierbei sind noch keine Wartezyklen berücksichtigt) sowie
- $k - \min\{m, k\}$ Wartezyklen: Im Best Case existieren $\min\{m, k\}$ Lücken auf demselben PE und nur die Zuteilung von Tasks in die *übrigen* $k - \min\{m, k\}$ Lücken ruft Wartezyklen hervor.

In Summe ergibt sich daher $t_{\mathrm{SH}} \geq k + (k - \min\{m, k\}) + 1 + a_{\min}$, wegen $k > 0$ ist dies identisch zur Behauptung $t_{\mathrm{SH}} \geq 2k - \min\{m, k\} + \min\{k, 1\} + a_{\min}$.

Da beide Schranken gezeigt wurden, folgt die Behauptung. □

Bemerkung B.17

Wie im vorigen Beweis gesehen, ist die Dauer der Selbstheilung abhängig von der Verteilung der Lücken auf die einzelnen PEs sowie der Reihenfolge, in der diese Lücken mit den k höchstprioren nicht zugeteilten Tasks gefüllt werden.

Beide Faktoren lassen sich im Allgemeinen nicht vorhersagen: Die Position der

Lücken *nach* dem Stopp der k Tasks ist primär abhängig von der Verteilung der einzelnen Tasks auf die PEs *vor* dem Stopp der Tasks. Gemäß Annahme B.13 werden jedoch alle Taskverteilungen als gleich wahrscheinlich betrachtet.

Die Reihenfolge des Füllens der durch die Taskstopps entstandenen Lücken ist dagegen unter anderem abhängig von den konkret beteiligten Hormonwerten in Kombination mit den k tatsächlich (erneut) zuzuteilenden Tasks:

- Ist beispielsweise ein PE durch einen hohen statischen Eignungswert für bestimmte Tasks deutlich besser geeignet als andere PEs, so können seine Lücken unter Umständen deutlich früher gefüllt werden als jene der anderen PEs.
- Auch von den $mv - k$ nicht gestoppten Tasks gebildete Organe können Einfluss auf die Reihenfolge nehmen, in der die Lücken gefüllt werden, da verwandte Tasks bevorzugt in der Nachbarschaft der bestehenden Organe zugeteilt werden.

  Hierbei nehmen jedoch unter anderem

  – Verwandtschaftsbeziehungen von Tasks, sowohl zwischen den $mv - k$ nicht gestoppten Tasks und den k zuzuteilenden Tasks untereinander als auch zwischen diesen beiden Gruppen von Tasks,
  – Nachbarschaftsbeziehungen zwischen den PEs,
  – die Stärke der Acceleratoren sowie
  – die Verteilung der $mv - k$ nicht gestoppten Tasks auf die PEs

  Einfluss auf die Positionen der k Tasks auf den PEs.

Damit sind diese Einflussfaktoren sehr ähnlich zu den in Bemerkung B.14 genannten.

Als Resultat lässt sich die Reihenfolge des Füllens der Lücken und damit auch die Dauer t_{SH} der Selbstheilung innerhalb des verwendeten Modells im Allgemeinen nicht genauer als durch Satz B.16 abschätzen. Zu beachten ist jedoch, dass die gegebenen Schranken im verwendeten Modell dennoch die größte untere bzw. die kleinste obere Schranke darstellen, wie im Beweis des Satzes begründet wurde.

Wie gerade begründet, kann die Dauer t_{SH} der Selbstheilung also nicht genauer abgeschätzt werden als durch Satz B.16, der hierzu zwei Schranken angibt. Als Konsequenz kann daher auch die *erwartete* Dauer der Selbstheilung – und damit der Average-Case der naiven Task-Dropping-Strategie – nur innerhalb zweier Schranken angegeben werden. Stochastisch lässt sich die Dauer der Selbstheilung daher als Zufallsvariable auffassen, deren Verteilung unbekannt ist, jedoch gemäß Satz B.16 Beschränkungen unterliegt und insbesondere von der Anzahl gestoppter Tasks abhängig ist, die von Satz B.15 durch eine Zufallsvariable X modelliert wurde. Diese Überlegung bildet die Grundlage des folgenden Satzes:

Satz B.18. Es bezeichne $\mathrm{ACET}_{\mathrm{naiv}}$ den Average Case der naiven Task-Dropping-Strategie, welcher der erwarteten Dauer einer Selbstheilung in einer Überlastsituation bei Verwendung dieser Strategie entspreche.

Dann gilt

$$2\frac{m^2v}{1+m} - m + \frac{\binom{2m}{m+1}-1}{\binom{mv+m}{m}} + 1 + a_{\min} \leq \mathrm{ACET}_{\mathrm{naiv}} \leq a + \begin{cases} \frac{m^2v}{1+m} + 1 - \frac{1}{\binom{mv+m}{m}}, & v = 1 \\ 2\frac{m^2v}{1+m}, & v > 1. \end{cases}$$

Beweis. Sei Y eine Zufallsvariable, welche die Dauer der Selbstheilung in Hormonzyklen angibt, sofern die naive Task-Dropping-Strategie verwendet wird. Dann ergibt sich gerade $\mathrm{ACET}_{\mathrm{naiv}} = \mathbf{E}[Y]$.

Sei ferner X eine Zufallsvariable, welche die Anzahl der von der naiven Task-Dropping-Strategie gestoppten Tasks angibt.

Satz B.16 liefert nun die Abschätzung

$$2X - \min\{m, X\} + \min\{X, 1\} + a_{\min} \leq Y \leq X + \max\{0, X-1\} \cdot [v > 1] + \min\{X, 1\} + a. \tag{B.3}$$

Gemäß Definition gilt $\mathbf{E}[Y] = \mathrm{ACET}_{\mathrm{naiv}}$, das Bilden der Erwartungswerte liefert daher aufgrund der Monotonie des Erwartungswerts

$$\begin{aligned} \mathbf{E}[2X - \min\{m, X\} + \min\{X, 1\} + a_{\min}] &\leq \mathrm{ACET}_{\mathrm{naiv}} \\ &\leq \mathbf{E}[X + \max\{0, X-1\} \cdot [v > 1] + \min\{X, 1\} + a]. \end{aligned}$$

Wir werden nun zeigen, dass diese beiden Ungleichungen mit jenen aus der Behauptung identisch sind:

1. Behauptung (untere Schranke): Es gilt

   $$\mathbf{E}[2X - \min\{m, X\} + \min\{X, 1\} + a_{\min}] = 2\frac{m^2v}{1+m} - m + \frac{\binom{2m}{m+1}-1}{\binom{mv+m}{m}} + 1 + a_{\min}.$$

   Beweis. Aufgrund der Linearität des Erwartungswerts gilt

   $$\begin{aligned} &\mathbf{E}[2X - \min\{m, X\} + \min\{X, 1\} + a_{\min}] \\ &\quad = 2\,\mathbf{E}[X] - \mathbf{E}[\min\{m, X\}] + \mathbf{E}[\min\{X, 1\}] + \mathbf{E}[a_{\min}], \end{aligned}$$

   $a_{\min}$ ist jedoch eine Konstante und gemäß Satz B.15 gilt $\mathbf{E}[X] = (m^2v)/(1+m)$, damit folgt

   $$= 2\frac{m^2v}{1+m} - \mathbf{E}[\min\{m, X\}] + \mathbf{E}[\min\{X, 1\}] + a_{\min}. \tag{B.4}$$

   Zur weiteren Umformung zeigen wir nun einige Hilfsaussagen:

   i. Es gilt $\mathbf{E}[\min\{m, X\}] = m - \sum_{i=0}^{m-1} (m-i) \cdot \mathbf{P}(X = i)$.

      Beweis. Da X die Anzahl gestoppter Tasks angibt, muss der Wert von X eine ganze Zahl zwischen 0 und mv sein. Folglich gilt $\min\{m, X\} \in \{0, 1, \ldots, m\}$

und die Wahrscheinlichkeitsverteilung von $\min\{m, X\}$ ergibt sich zu

$$\begin{aligned}
\mathbf{P}(\min\{m, X\} = 0) &= \mathbf{P}(X = 0) \\
\mathbf{P}(\min\{m, X\} = 1) &= \mathbf{P}(X = 1) \\
&\vdots \\
\mathbf{P}(\min\{m, X\} = m - 1) &= \mathbf{P}(X = m - 1) \\
\mathbf{P}(\min\{m, X\} = m) &= \mathbf{P}(X \geq m) \\
&= 1 - \mathbf{P}(X \leq m - 1) \\
&= 1 - \sum_{i=0}^{m-1} \mathbf{P}(X = i).
\end{aligned}$$

Damit berechnet sich der Erwartungswert von $\min\{m, X\}$ zu

$$\begin{aligned}
\mathbf{E}[\min\{m, X\}] &= \sum_{i=0}^{m} i \cdot \mathbf{P}(\min\{m, X\} = i) \\
&= \sum_{i=0}^{m-1} i \cdot \mathbf{P}(X = i) + m \cdot \left(1 - \sum_{i=0}^{m-1} \mathbf{P}(X = i)\right) \\
&= m + \sum_{i=0}^{m-1} (i - m) \cdot \mathbf{P}(X = i) \\
&= m - \sum_{i=0}^{m-1} (m - i) \cdot \mathbf{P}(X = i). \qquad \blacklozenge
\end{aligned}$$

ii. Es gilt $\sum_{i=0}^{m-1} (m - i) \cdot \mathbf{P}(X = i) = \binom{2m}{m+1} / \binom{mv+m}{m}$.

Beweis. Unter Verwendung von Gleichung (B.2) aus dem Beweis von Satz B.15 (S. 253) folgt

$$\begin{aligned}
&\sum_{i=0}^{m-1} (m - i) \cdot \mathbf{P}(X = i) \\
&\quad = \sum_{i=0}^{m-1} (m - i) \cdot m \cdot \frac{\prod_{j=i+1}^{mv} j}{\prod_{j=m+i}^{mv+m} j} \\
&\quad = \sum_{i=0}^{m-1} (m - i) \cdot m \cdot \frac{(mv)!\,(m + i - 1)!}{i!\,(mv + m)!} \cdot \frac{(m - 1)!}{(m - 1)!} \\
&\quad = \frac{m!\,(mv)!}{(mv + m)!} \cdot \sum_{i=0}^{m-1} (m - i) \cdot \frac{(m + i - 1)!}{i!\,(m - 1)!}
\end{aligned}$$

$$= \frac{1}{\binom{mv+m}{m}} \cdot \sum_{i=0}^{m-1} (m-i) \cdot \binom{m-1+i}{i}.$$

Der Faktor $(m-i)$ kann nun als Binomialkoeffizient $\binom{m-i}{1}$ dargestellt werden, der Binomialkoeffizient $\binom{m+i-1}{i}$ kann dagegen gemäß Lemma B.10 umgeschrieben werden:

$$= \frac{1}{\binom{mv+m}{m}} \cdot \sum_{i=0}^{m-1} \binom{m-i}{1} \cdot \binom{m-1+i}{m-1}.$$

Lässt man die Summe bis m laufen, so ändert sich ihr Wert wegen des Faktors $\binom{m-m}{1} = \binom{0}{1} = 0$ nicht, damit folgt

$$= \frac{1}{\binom{mv+m}{m}} \cdot \sum_{i=0}^{m} \binom{m-i}{1} \cdot \binom{m-1+i}{m-1}.$$

Auf diese Summe kann nun Lemma B.12 angewandt werden:

$$= \frac{1}{\binom{mv+m}{m}} \cdot \binom{m+m-1+1}{1+m-1+1}$$
$$= \frac{\binom{2m}{m+1}}{\binom{mv+m}{m}}.$$ ♦

iii. Es gilt $\mathbf{E}[\min\{X,1\}] = 1 - \mathbf{P}(X=0)$.

Beweis. Wegen $X \geq 0$ gilt $\min\{X,1\} \in \{0,1\}$, folglich ergibt sich die Wahrscheinlichkeitsverteilung von $\min\{X,1\}$ zu

$$\mathbf{P}(\min\{X,1\}=0) = \mathbf{P}(X=0)$$
$$\mathbf{P}(\min\{X,1\}=1) = \mathbf{P}(X \geq 1) = 1 - \mathbf{P}(X=0).$$

Damit berechnet sich der Erwartungswert von $\min\{X,1\}$ zu

$$\mathbf{E}[\min\{X,1\}] = 0 \cdot \mathbf{P}(X=0) + 1 \cdot \mathbf{P}(X=1)$$
$$= 1 - \mathbf{P}(X=0).$$ ♦

iv. Es gilt $\mathbf{P}(X=0) = 1/\binom{mv+m}{m}$.

Beweis. Nach Gleichung (B.2) gilt

$$\mathbf{P}(X=0) = m \cdot \frac{\prod_{i=1}^{mv} i}{\prod_{i=m}^{mv+m} i}$$

$$
\begin{aligned}
&= m \cdot \frac{(mv)!\,(m-1)!}{(mv+m)!} \\
&= \frac{(mv)!\,m!}{(mv+m)!} \\
&= \frac{1}{\binom{mv+m}{m}}.
\end{aligned}
$$

♦

Diese Hilfsaussagen können nun zum Beweis der unteren Schranke verwendet werden. Gemäß Gleichung (B.4) gilt

$$
\begin{aligned}
&\mathbf{E}[2X - \min\{m, X\} + \min\{X, 1\} + a_{\min}] \\
&\quad = 2\frac{m^2 v}{1+m} - \mathbf{E}[\min\{m, X\}] + \mathbf{E}[\min\{X, 1\}] + a_{\min}.
\end{aligned}
$$

Mit den Hilfsaussagen i. und iii. folgt

$$
= 2\frac{m^2 v}{1+m} - \left(m - \sum_{i=0}^{m-1} (m-i) \cdot \mathbf{P}(X = i) \right) + 1 - \mathbf{P}(X = 0) + a_{\min},
$$

die Hilfsaussagen ii. und iv. liefern schließlich

$$
\begin{aligned}
&= 2\frac{m^2 v}{1+m} - m + \frac{\binom{2m}{m+1}}{\binom{mv+m}{m}} + 1 - \frac{1}{\binom{mv+m}{m}} + a_{\min} \\
&= 2\frac{m^2 v}{1+m} - m + \frac{\binom{2m}{m+1} - 1}{\binom{mv+m}{m}} + 1 + a_{\min},
\end{aligned}
$$

was zu zeigen war. ◊

2. Behauptung (obere Schranke): Es gilt

$$
\begin{aligned}
&\mathbf{E}[X + \max\{0, X-1\} \cdot [v > 1] + \min\{X, 1\} + a] \\
&\quad = a + \begin{cases} \frac{m^2 v}{1+m} + 1 - \frac{1}{\binom{mv+m}{m}}, & v = 1 \\ 2\frac{m^2 v}{1+m}, & v > 1. \end{cases}
\end{aligned}
$$

Beweis. Es ist leicht einzusehen, dass $\max\{0, X-1\} = X - \min\{X, 1\}$ gilt, damit ergibt sich

$$
\begin{aligned}
&\mathbf{E}[X + \max\{0, X-1\} \cdot [v > 1] + \min\{X, 1\} + a] \\
&\quad = \mathbf{E}[X + (X - \min\{X, 1\}) \cdot [v > 1] + \min\{X, 1\} + a].
\end{aligned}
$$

Aufgrund der Linearität des Erwartungswerts ($[v > 1]$ kann als Konstante betrachtet werden) ergibt sich

$$
= \mathbf{E}[X] + (\mathbf{E}[X] - \mathbf{E}[\min\{X, 1\}]) \cdot [v > 1] + \mathbf{E}[\min\{X, 1\}] + \mathbf{E}[a].
$$

Analog zur unteren Schranke ist a eine Konstante, ferner gilt gemäß Satz B.15 $\mathbf{E}[X] = (m^2 v)/(1+m)$. Damit folgt

$$= \frac{m^2 v}{1+m} + \left(\frac{m^2 v}{1+m} - \mathbf{E}[\min\{X, 1\}] \right) \cdot [v > 1] + \mathbf{E}[\min\{X, 1\}] + a.$$

Nun kann die bereits für die untere Schranke hergeleitete Hilfsaussage iii. genutzt werden, damit ergibt sich

$$= \frac{m^2 v}{1+m} + \left(\frac{m^2 v}{1+m} - (1 - \mathbf{P}(X = 0)) \right) \cdot [v > 1] + 1 - \mathbf{P}(X = 0) + a$$

und Hilfsaussage iv. liefert

$$= \frac{m^2 v}{1+m} + \left(\frac{m^2 v}{1+m} - 1 + \frac{1}{\binom{mv+m}{m}} \right) \cdot [v > 1] + 1 - \frac{1}{\binom{mv+m}{m}} + a$$

$$= a + \begin{cases} \frac{m^2 v}{1+m} + 1 - \frac{1}{\binom{mv+m}{m}}, & v = 1 \\ 2\frac{m^2 v}{1+m}, & v > 1 \end{cases}$$

und das war zu zeigen. ◊

Die Behauptung folgt nun aus der Korrektheit der unteren Schranke (Behauptung 1) sowie der oberen Schranke (Behauptung 2). □

Korollar B.19. Mit den bekannten Werten $a_{\min} = 1$ und $a = 2$ ergibt sich die Abschätzung aus Satz B.18 zu

$$2\frac{m^2 v}{1+m} + 2 - m + \frac{\binom{2m}{m+1} - 1}{\binom{mv+m}{m}} \leq \mathrm{ACET}_{\mathrm{naiv}} \leq \begin{cases} \frac{m^2 v}{1+m} + 3 - \frac{1}{\binom{mv+m}{m}}, & v = 1 \\ 2\frac{m^2 v}{1+m} + 2, & v > 1. \end{cases}$$

B.4.1. Vergleich der Schranken

Satz B.18 gibt für $\mathrm{ACET}_{\mathrm{naiv}}$ nur ein Intervall an. Im Folgenden wollen wir betrachten, wie groß dieses ist:

Satz B.20. Es bezeichnen $\mathrm{ACET}_{\mathrm{naiv}}^{\min}$ und $\mathrm{ACET}_{\mathrm{naiv}}^{\max}$ die untere bzw. obere Schranke von $\mathrm{ACET}_{\mathrm{naiv}}$ gemäß Satz B.18.

Dann gilt für die Differenz beider Werte

$$\mathrm{ACET}_{\mathrm{naiv}}^{\max} - \mathrm{ACET}_{\mathrm{naiv}}^{\min} = a - a_{\min} + \begin{cases} 0, & v = 1 \\ m - 1 + c_{m,v}, & v > 1, \end{cases}$$

wobei

$$c_{m,v} = \frac{1 - \binom{2m}{m+1}}{\binom{mv+m}{m}}. \tag{B.5}$$

Für $v > 1$ gilt ferner $-1 < c_{m,v} \leq 0$.

Beweis. Der Fall $v = 1$ kann zwar durch algebraische Vereinfachungen gezeigt werden,[1] anhand der folgenden Überlegungen wird jedoch klar, *warum* die Aussage gilt: Hierzu betrachten wir Ungleichung (B.3) aus dem Beweis von Satz B.18, die sich in diesem Fall zu

$$2X - \min\{m, X\} + \min\{X, 1\} + a_{\min} \leq Y \leq X + \min\{X, 1\} + a$$

ergibt. Wegen $v = 1$ können jedoch höchstens m Tasks gestoppt werden, es gilt daher $\min\{m, X\} = X$, womit

$$X + \min\{X, 1\} + a_{\min} \leq Y \leq X + \min\{X, 1\} + a$$

folgt. Offensichtlich sind beide Abschätzungen bis auf die Konstanten a und $a_{\min}$ identisch, woraus die Behauptung folgt.

Für den Fall $v > 1$ ergibt sich dagegen gemäß Satz B.18

$$\begin{aligned}
&\mathrm{ACET}^{\max}_{\mathrm{naiv}} - \mathrm{ACET}^{\min}_{\mathrm{naiv}} \\
&\quad = a + 2\frac{m^2 v}{1+m} - \left(2\frac{m^2 v}{1+m} - m + \frac{\binom{2m}{m+1} - 1}{\binom{mv+m}{m}} + 1 + a_{\min}\right) \\
&\quad = a - a_{\min} + m - 1 + \underbrace{\frac{1 - \binom{2m}{m+1}}{\binom{mv+m}{m}}}_{c_{m,v}}.
\end{aligned}$$

Dies entspricht dem ersten Teil der Behauptung.

Es muss daher nur noch $-1 < c_{m,v} \leq 0$ für $v > 1$ gezeigt werden. Wegen $m > 0$ lassen sich die Binomialkoeffizienten jedoch wie folgt auflösen:

$$\begin{aligned}
c_{m,v} = \frac{1 - \binom{2m}{m+1}}{\binom{mv+m}{m}} &= \frac{1 - \frac{(2m)^{\underline{m+1}}}{(m+1)!}}{\frac{(mv+m)^{\underline{m}}}{m!}} \\
&= \frac{m! - \frac{1}{m+1} \cdot (2m)^{\underline{m+1}}}{(mv+m)^{\underline{m}}}
\end{aligned}$$

[1] Konkret ergibt sich der in der Differenz auftretende Term $m - \frac{m^2 v}{1+m} - \frac{\binom{2m}{m+1}}{\binom{mv+m}{m}}$ für $v = 1$ zu Null, da ein Auflösen des Quotienten der Binomialkoeffizienten in diesem Fall $m/(m+1)$ ergibt.

$$
\begin{aligned}
&= \frac{\prod_{i=1}^{m} i - \frac{1}{m+1} \cdot \prod_{i=m}^{2m} i}{\prod_{i=mv+1}^{mv+m} i} \\
&= \frac{\prod_{i=1}^{m} i - \frac{m}{m+1} \cdot \prod_{i=m+1}^{2m} i}{\prod_{i=mv+1}^{mv+m} i} \\
&= \frac{\prod_{i=1}^{m} i - \frac{m}{m+1} \cdot \prod_{i=1}^{m} (i+m)}{\prod_{i=1}^{m} (i+mv)} \\
&= \frac{\prod_{i=1}^{m} i}{\prod_{i=1}^{m} (i+mv)} - \frac{m}{m+1} \cdot \frac{\prod_{i=1}^{m} (i+m)}{\prod_{i=1}^{m} (i+mv)} \\
&= \prod_{i=1}^{m} \frac{i}{i+mv} - \frac{m}{m+1} \cdot \prod_{i=1}^{m} \frac{i+m}{i+mv}. \qquad \text{(B.6)}
\end{aligned}
$$

Weil m und v ganzzahlig sind und zudem $m \geq 1$ sowie $v > 1$ gelten, ist Lemma C.4 anwendbar und es folgt die Behauptung

$$-1 < c_{m,v} \leq 0. \qquad \square$$

Wie zuvor in Abschnitt B.4 begründet, kann die zur Erkennung des Ausfalls von $\mathrm{PE}_\times$ nötige Zeit A im Allgemeinen nicht genauer abgeschätzt werden als durch $a_{\min} \leq A \leq a$. Abgesehen von der hierin begründeten Unsicherheit von $a - a_{\min}$ Hormonzyklen (bzw. ein Zyklus, wenn für diese Konstanten ihre konkreten Werte eingesetzt werden) kann die Dauer der Selbstheilung im Average Case durch Satz B.18 somit für $v = 1$ exakt angegeben werden, für $v > 1$ dagegen nur mit einer Unsicherheit von $m - 1 + c_{m,v}$ Hormonzyklen abgeschätzt werden, wobei jedoch $-1 < c_{m,v} \leq 0$ gilt.

B.4.1.1. Genauere Betrachtung von $c_{m,v}$

Abbildung B.1 stellt die Werte von $c_{m,v}$ gemäß Gleichung (B.5) für $1 \leq m \leq 20$ sowie $2 \leq v \leq 20$ dar. Der kleinste Wert in diesem Bereich tritt für $m = v = 2$ auf: Es gilt $c_{2,2} = -0{,}2$. Zudem lässt sich für alle $v \geq 2$ leicht $c_{1,v} = 0$ zeigen.

Ferner ist in der Abbildung gut erkennbar, dass $c_{m,v}$ für große Werte von m und v vernachlässigbar klein wird: Konkret gilt $c_{m,v} = \prod_{i=1}^{m} \frac{i}{i+mv} - \frac{m}{m+1} \cdot \prod_{i=1}^{m} \frac{i+m}{i+mv}$ gemäß Gleichung (B.6). Die Nenner beider in diesem Term auftretenden Produkte sind für $v \geq 2$ jeweils größer als ihre Zähler. Folglich kann insbesondere für große Werte von m angenommen werden, dass der gesamte Term $c_{m,v}$ vernachlässigbar klein ist.

Für $v > 1$ gilt nach Satz B.20 jedoch in jedem Fall

$$a - a_{\min} + m - 2 < \mathrm{ACET}_{\mathrm{naiv}}^{\max} - \mathrm{ACET}_{\mathrm{naiv}}^{\min} \leq a - a_{\min} + m - 1,$$

die Unsicherheit der Abschätzung von $\mathrm{ACET}_{\mathrm{naiv}}$ beträgt also nie mehr als $a - a_{\min} + m - 1$ Hormonzyklen.

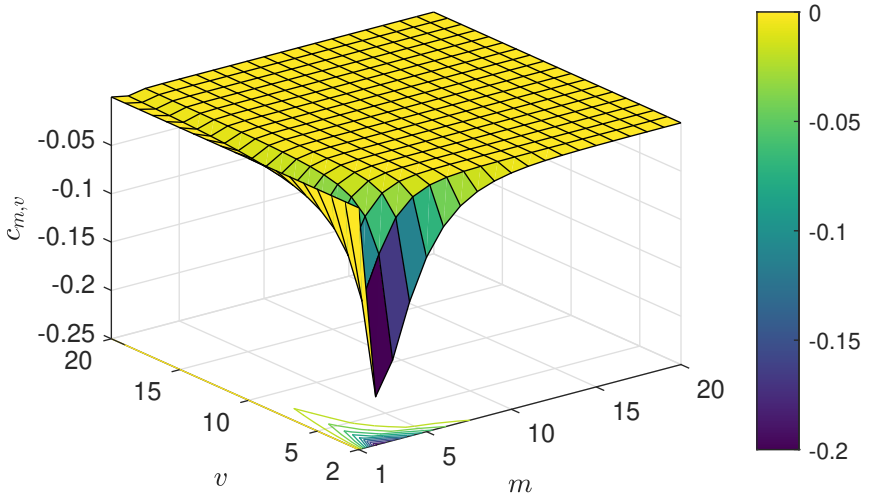


Abbildung B.1: In $\mathrm{ACET}^{\max}_{\mathrm{naiv}} - \mathrm{ACET}^{\min}_{\mathrm{naiv}}$ auftretender Term $c_{m,v}$ gemäß Gleichung (B.5) für $1 \leq m \leq 20$ und $2 \leq v \leq 20$. Der kleinste auftretende Wert ist $c_{2,2} = -0{,}2$.

B.5. Vergleich mit dem Worst Case

Im Folgenden sollen die Abschätzungen für die Anzahl gestoppter Tasks sowie die resultierende Selbstheilungsdauer im Average Case mit den entsprechenden Werten des Worst Case verglichen werden, um die naive Task-Dropping-Strategie besser beurteilen zu können.

B.5.1. Anzahl gestoppter Tasks

Zum Vergleich der Anzahl gestoppter Tasks wird hierzu der Quotient

$$Q(m,v) \coloneqq \frac{\text{Anzahl gestoppter Tasks im Worst Case}}{\text{Anzahl gestoppter Tasks im Average Case}}$$

betrachtet. Im Worst Case werden gerade *alle* mv zugeteilten Tasks gestoppt, der Average Case ergibt sich dagegen gemäß Satz B.15 zu $m^2v/(1+m)$ Taskstopps und es folgt

$$Q(m,v) = \frac{mv}{m^2v/(1+m)} = 1 + \frac{1}{m}.$$

Der Quotient ist also nur von m abhängig, nicht jedoch von der Anzahl v verbleibender PEs. Daher wird er im Folgenden nur noch mit $Q(m)$ bezeichnet.

Für $m = 1$ ergibt sich $Q(m) = 2$, im Average Case werden also nur halb so viele Tasks gestoppt wie im Worst Case (nämlich $v/2$ statt v Tasks). Mit wachsendem m

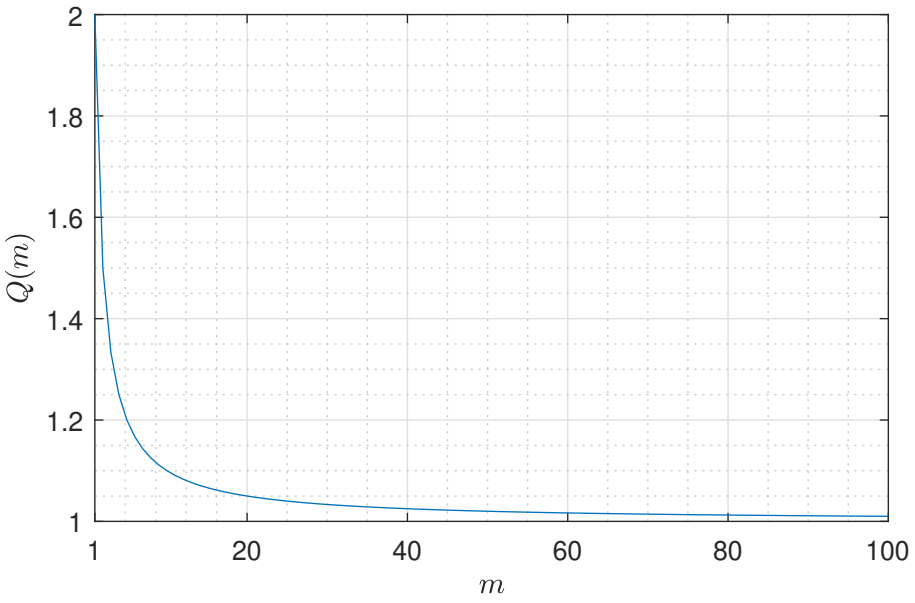


Abbildung B.2: Quotient $Q(m)$ der von der naiven Task-Dropping-Strategie gestoppten Tasks im Worst- und Average Case

nähern sich Average- und Worst Case dagegen einander an:

$$\lim_{m\to\infty} Q(m) = \lim_{m\to\infty} \left(1 + \frac{1}{m}\right) = 1.$$

Abbildung B.2 stellt $Q(m)$ grafisch dar; das Grenzwertverhalten für große m ist gut zu erkennen.

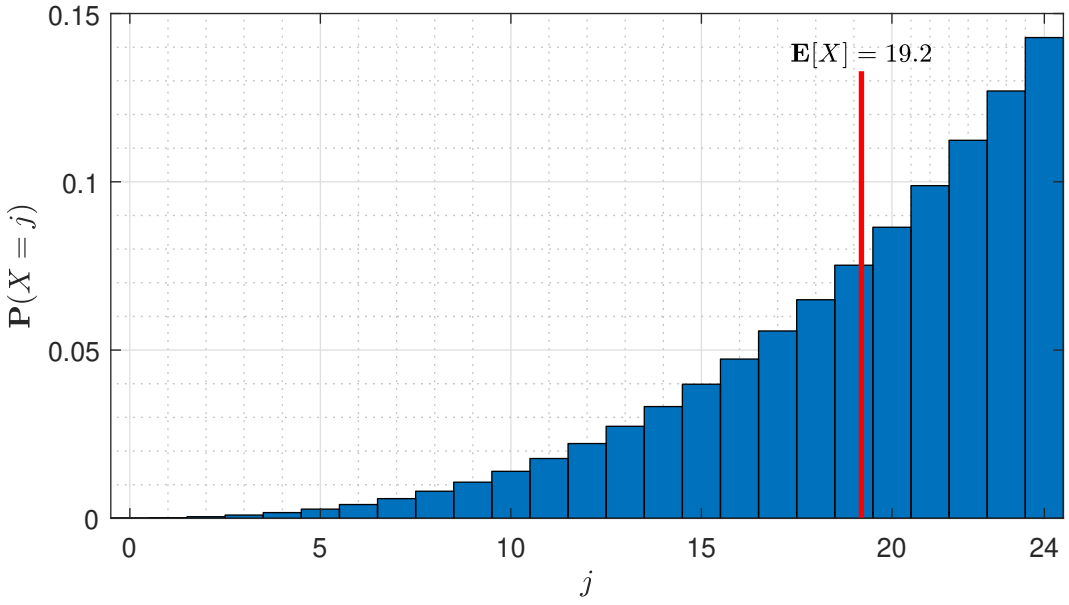
Eine intuitive Begründung für dieses Resultat liefert Abbildung B.3. Diese zeigt die Wahrscheinlichkeitsverteilung der Anzahl X gestoppter Tasks, wie sie im Beweis von Satz B.15 hergeleitet wurde, exemplarisch für jeweils $v = 6$ verbleibende PEs mit $m = 4$ sowie $m = 20$ Tasks pro PE.

Für die erste Parameterkombination gelten $0 \leq X \leq 24$ und $\mathbf{E}[X] = 19{,}2$; der Quotient $Q(m)$ berechnet sich zu 1,25.

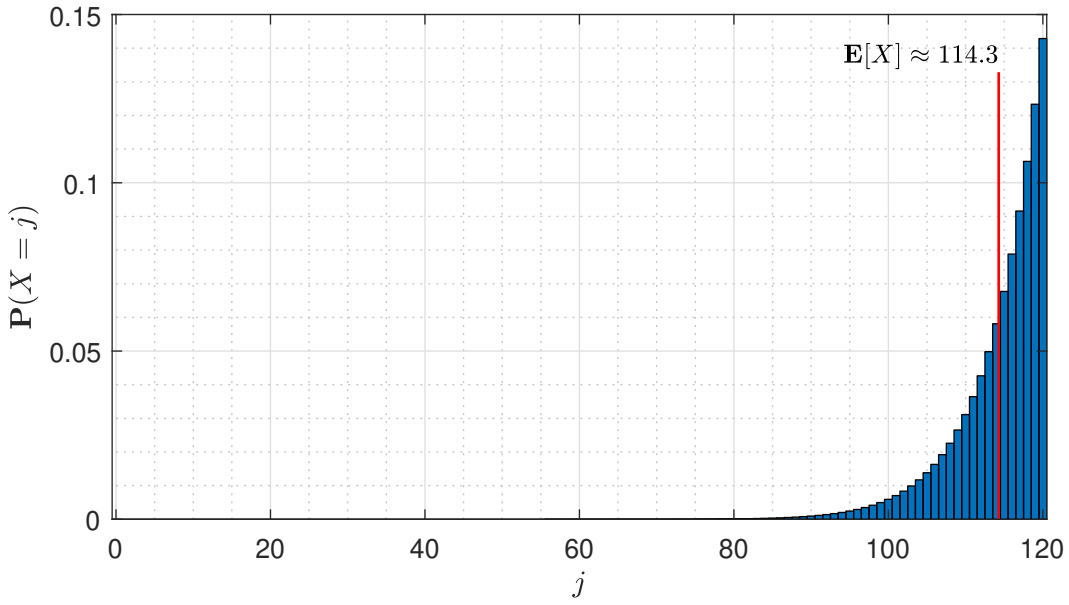
Für die zweite Parameterkombination gelten dagegen $0 \leq X \leq 120$ und $\mathbf{E}[X] \approx 114{,}3$; $Q(m)$ ergibt sich zu 1,05.

Betrachtet man die beiden abgebildeten Histogramme, so wird intuitiv klar, warum der Erwartungswert von X – als Summe der zusätzlich mit den Werten von X gewichteten Wahrscheinlichkeiten – nicht allzu weit entfernt vom maximalen Wert von X liegt; für kleine m (Abbildung B.3a) ist der relative Abstand des Erwartungswerts vom Maximalwert dabei deutlich größer als für große m (Abbildung B.3b).

Zusammenfassend liegen Worst- und Average Case der naiven Task-Dropping-Strategie bezogen auf die Anzahl der gestoppten Tasks also stets lediglich einen Faktor von höchstens Zwei auseinander, der mit wachsendem m zudem gegen Eins



(a) $m = 4$, $v = 6$



(b) $m = 20$, $v = 6$

Abbildung B.3: Wahrscheinlichkeitsverteilung der Anzahl X von der naiven Task-Dropping-Strategie gestoppter Tasks gemäß Gleichung (B.2)

strebt; als Resultat liegen Worst- und Average Case asymptotisch in der gleichen Komplexitätsklasse. Im Folgenden wollen wir uns ferner einem Vergleich der sich aus der Anzahl gestoppter Tasks ergebenden Dauer der Selbstheilung widmen.

B.5.2. Dauer der Selbstheilung

Auch im Fall der Selbstheilungsdauer ist es sinnvoll, nicht die (absolute) Differenz zwischen Worst- und Average Case zu betrachten, sondern die entsprechenden Hormonzyklen in Relation zueinander zu setzen. Daher wird im Folgenden der Quotient

$$Q(m,v) := \frac{\mathrm{WCET}_{\mathrm{naiv}}}{\mathrm{ACET}_{\mathrm{naiv}}}$$

betrachtet. Da $\mathrm{ACET}_{\mathrm{naiv}}$ nicht genauer bestimmt werden kann als durch Satz B.18, kann auch $Q(m,v)$ nicht exakt bestimmt werden. Im Folgenden bezeichnen $\mathrm{ACET}^{\mathrm{min}}_{\mathrm{naiv}}$ und $\mathrm{ACET}^{\mathrm{max}}_{\mathrm{naiv}}$ daher erneut die untere bzw. obere Schranke für $\mathrm{ACET}_{\mathrm{naiv}}$ gemäß Satz B.18. Damit folgt

$$\frac{\mathrm{WCET}_{\mathrm{naiv}}}{\mathrm{ACET}^{\mathrm{max}}_{\mathrm{naiv}}} \leq Q(m,v) \leq \frac{\mathrm{WCET}_{\mathrm{naiv}}}{\mathrm{ACET}^{\mathrm{min}}_{\mathrm{naiv}}}.$$

Für die bekannten Werte $a_{\min} = 1$ und $a = 2$ sind die resultierenden Schranken für $Q(m,v)$ in Abbildung B.4 für $1 \leq m \leq 20$ und $1 \leq v \leq 20$ aufgetragen.

Für $v = 1$ treten keine Wartezyklen auf. Dass ein solcher Sonderfall existiert, lässt sich anhand von Abbildung B.4a bereits erahnen, zur Übersicht stellt Abbildung B.5 die Schranken von $Q(m,v)$ zudem für verschiedene kleine Werte von v separat dar.

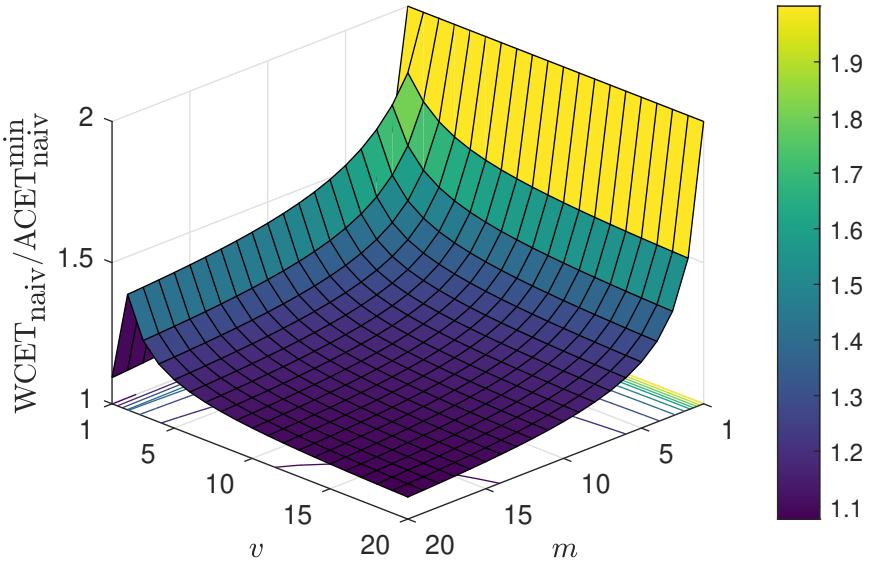
Aus dem Vergleich der Abbildungen B.4a und B.4b ist zudem zu erkennen, dass Average- und Worst Case sich für großes m und großes v einander anzunähern scheinen. Dass dies für große Werte von m der Fall ist, überrascht angesichts des Vergleichs der *Anzahlen* gestoppter Tasks in Worst- und Average Case nicht, die sich für große m ebenfalls kaum voneinander unterscheiden.

Interessant ist dagegen, dass der Wert der oberen Schranke für großes m bei kleinem v mitunter deutlich größer als jener der unteren Schranke ist, wie insbesondere aus Abbildung B.5 hervorgeht. Für $v = 1$ sind obere und untere Schranke von $\mathrm{ACET}_{\mathrm{naiv}}$ gemäß Satz B.20 bis auf die Konstanten a und $a_{\min}$ identisch, weshalb wir uns im Folgenden auf den Fall $v \geq 2$ beschränken.

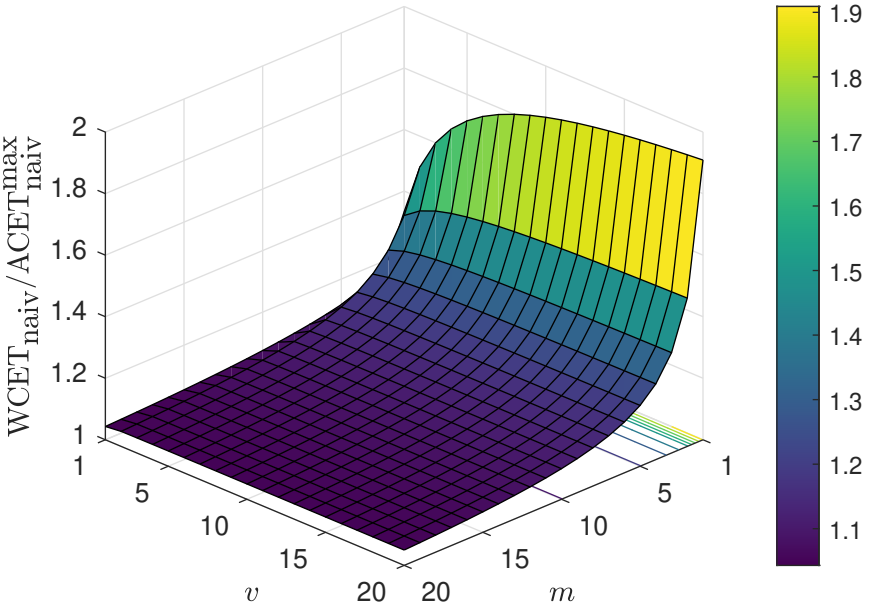
Untersucht man den Nenner der oberen Schranke von $Q(m,v)$, der für $v \geq 2$ gerade $\mathrm{ACET}^{\mathrm{min}}_{\mathrm{naiv}} = 2\frac{m^2 v}{1+m} - m + \frac{\binom{2m}{m+1}-1}{\binom{mv+m}{m}} + 1 + a_{\min}$ beträgt, lassen sich zwei Beobachtungen machen:

- Wie wir zuvor gesehen haben, kann $c_{m,v}$ für $v \geq 2$ insbesondere für große Werte von m vernachlässigt werden. Damit kann in solchen Fällen jedoch auch der Term $\left(\binom{2m}{m+1} - 1\right) / \binom{mv+m}{m} = -c_{m,v}$ vernachlässigt werden.

- Der Term $2\frac{m^2 v}{1+m}$ kann – vor allem für besagte große Werte von m – näherungsweise

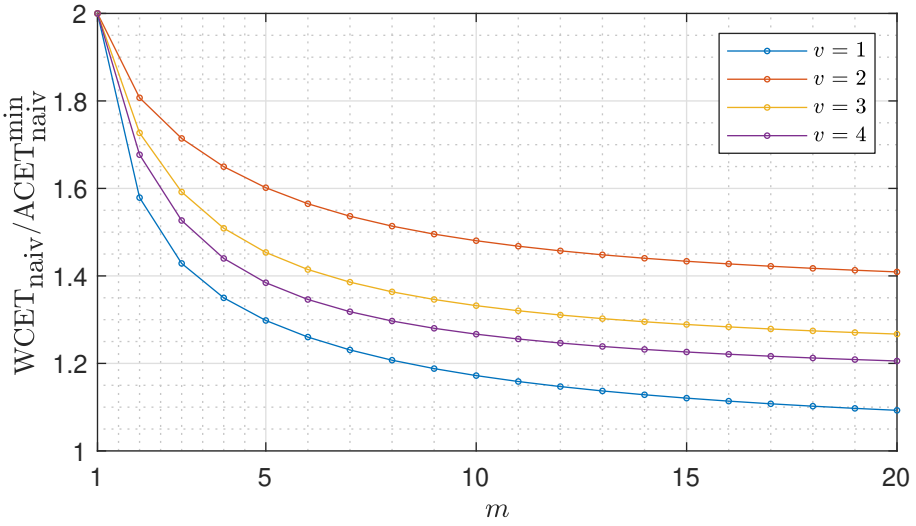


(a) Obere Schranke für $Q(m, v)$

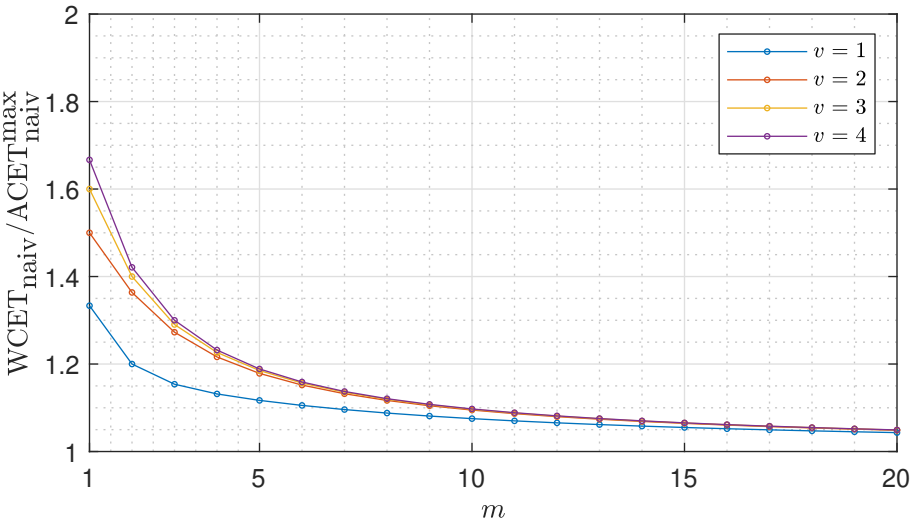


(b) Untere Schranke für $Q(m, v)$

Abbildung B.4: Quotient $Q(m, v)$ der Selbstheilungsdauer der naiven Task-Dropping-Strategie in Worst- und Average Case



(a) Obere Schranke



(b) Untere Schranke

Abbildung B.5: Untere und obere Schranke des Quotienten $Q(m, v)$ für kleine Werte von v

zu $2mv$ abgeschätzt werden.

Folglich berechnet sich die obere Schranke von $Q(m, v)$ näherungsweise zu

$$\frac{\mathrm{WCET}_{\mathrm{naiv}}}{\mathrm{ACET}^{\min}_{\mathrm{naiv}}} \approx \frac{2mv + a}{2mv - m + 1 + a_{\min}} = \frac{2mv + 2}{2mv - m + 2} \quad \text{für große } m \text{ und } v \geq 2.$$

Anhand dieser Näherung wird deutlich, dass der Nenner für kleines v aufgrund des Terms $-m$ deutlich kleiner als der Zähler werden kann. Im Gegensatz dazu enthält $\mathrm{ACET}^{\max}_{\mathrm{naiv}}$ gemäß Satz B.18 keinen vergleichbaren Term, was die signifikanten Unterschiede zwischen unterer und oberer Schranke von $Q(m, v)$ für kleines v erklärt.

Es lassen sich aus dem Vergleich der Graphen noch viele weitere Schlüsse ziehen, im Folgenden wollen wir uns jedoch mit einer einzigen abschließenden Aussage begnügen: Die betrachteten Abbildungen legen nahe, dass $Q(m, v) \leq 2$ zu gelten scheint; das folgende Resultat bestätigt diese Vermutung:

Satz B.21. Für den Quotienten $Q(m, v)$ aus der Dauer der Selbstheilung im Worst- und Average Case gilt $1 \leq Q(m, v) \leq 2$.

Beweis. Dass $Q(m, v) \geq 1$ gelten muss, lässt sich leicht einsehen, da die Dauer der Selbstheilung im Average Case nicht über jener des Worst Case liegen kann.

Die obere Schranke ergibt sich dagegen aus Betrachtungen der Worst-Case-Selbstheilungsdauer sowie $\mathrm{ACET}^{\min}_{\mathrm{naiv}}$; hierbei müssen zwei Fälle betrachtet werden:

1. $v = 1$: In diesem Fall gilt $\mathrm{WCET}_{\mathrm{naiv}} = mv + 1 + a = m + 1 + a$ gemäß Satz 8.11.

   Zur Maximierung des Quotienten benötigen wir $\mathrm{ACET}^{\min}_{\mathrm{naiv}}$. Dieser Term ist zwar aus Satz B.18 bekannt, im Folgenden verwenden wir jedoch einen anderen, äquivalenten Term: Für $v = 1$ sind $\mathrm{ACET}^{\min}_{\mathrm{naiv}}$ und $\mathrm{ACET}^{\max}_{\mathrm{naiv}}$ gemäß Satz B.20 bis auf die Konstanten $a_{\min}$ und a identisch, es folgt also

   $$\mathrm{ACET}^{\min}_{\mathrm{naiv}} = \mathrm{ACET}^{\max}_{\mathrm{naiv}} - a + a_{\min}.$$

   Die Ersetzung von $\mathrm{ACET}^{\max}_{\mathrm{naiv}}$ durch den aus Satz B.18 bekannten Term liefert daher

   $$\mathrm{ACET}^{\min}_{\mathrm{naiv}} = \frac{m^2 v}{1 + m} + 1 - \frac{1}{\binom{mv+m}{m}} + a_{\min} = \frac{m^2}{1 + m} + 1 - \frac{1}{\binom{2m}{m}} + a_{\min}.$$

   Es müssen nun drei Fälle unterschieden werden:

   a) $m = 1$. In diesem Fall gilt $\binom{2m}{m} = \binom{2}{1} = 2$ und daher $\mathrm{ACET}^{\min}_{\mathrm{naiv}} = 1 + a_{\min}$ sowie $\mathrm{WCET}_{\mathrm{naiv}} = 2 + a$.

      Damit folgt die Behauptung

      $$Q(m, v) \leq \frac{2 + a}{1 + a_{\min}} = \frac{2 + 2}{1 + 1} = 2.$$

b) $m = 2$. In diesem Fall gilt $\binom{2m}{m} = \binom{4}{2} = 6$ und daher $\text{ACET}^{\min}_{\text{naiv}} = 13/6 + a_{\min}$ sowie $\text{WCET}_{\text{naiv}} = 4 + a$.

Damit folgt die Behauptung

$$Q(m, v) \leq \frac{4 + a}{13/6 + a_{\min}} = \frac{6}{19/6} = \frac{36}{19} < 2.$$

c) $m \geq 3$. Offensichtlich gilt $0 < 1/\binom{2m}{m} \leq 1$, weshalb $\text{ACET}^{\min}_{\text{naiv}} \geq \frac{m^2}{1+m} + a_{\min}$ folgt.

Der Quotient $Q(m, v)$ wird maximal, wenn sein Nenner minimiert wird. Damit folgt

$$Q(m, v) \leq \frac{m + 1 + a}{\frac{m^2}{1+m} + a_{\min}} = \frac{m + 3}{\frac{m^2}{1+m} + 1}.$$

Die Behauptung folgt also, wenn

$$\frac{m + 3}{\frac{m^2}{1+m} + 1} \leq 2$$

gezeigt werden kann. Äquivalenzumformungen liefern die Ungleichung $m^2 - 2m - 1 \geq 0$. Die Funktion $f(m) \coloneqq m^2 - 2m - 1$ beschreibt jedoch eine nach oben geöffnete Parabel mit den Nullstellen $1 \pm \sqrt{2}$, weshalb die Ungleichung für alle $m \geq 3$ erfüllt ist und die Behauptung $Q(m, v) \leq 2$ folgt.

2. $v > 1$: In diesem Fall gilt $\text{WCET}_{\text{naiv}} = 2mv + a$ gemäß Satz 8.11.

Nach Satz B.18 gilt ferner

$$\text{ACET}^{\min}_{\text{naiv}} = 2\frac{m^2 v}{1 + m} - m + \frac{\binom{2m}{m+1} - 1}{\binom{mv+m}{m}} + 1 + a_{\min}.$$

Folglich muss

$$\frac{2mv + a}{2\frac{m^2 v}{1+m} - m + \frac{\binom{2m}{m+1}-1}{\binom{mv+m}{m}} + 1 + a_{\min}} = \frac{2mv + 2}{2\frac{m^2 v}{1+m} - m + \frac{\binom{2m}{m+1}-1}{\binom{mv+m}{m}} + 2} \leq 2 \tag{B.7}$$

gezeigt werden.

Der Term $\left(\binom{2m}{m+1} - 1\right) / \binom{mv+m}{m}$ entspricht jedoch gerade dem Term $-c_{m,v}$, für den gemäß Satz B.20 $0 \leq -c_{m,v} < 1$ gilt. Als Resultat wird der Nenner der linken Seite von Ungleichung (B.7) für $-c_{m,v} = 0$ minimal und der Quotient damit maximal. Folglich ergibt sich die Behauptung, wenn

$$\frac{2mv + 2}{2\frac{m^2 v}{1+m} - m + 2} \leq 2$$

gezeigt werden kann. Äquivalenzumformungen dieser Ungleichung liefern

$$m^2 - m^2 v + mv - 1 \leq 0. \tag{B.8}$$

Dass diese Ungleichung für $m \geq 1$ und $v > 1$ erfüllt ist, lässt sich jedoch leicht induktiv nachweisen:

Induktionsanfang: Für $m = 1$ und ein beliebiges $v > 1$ gilt

$$m^2 - m^2 v + mv - 1 = 0 \leq 0.$$

Induktionsschritt $m \to m + 1$.

Induktionsannahme. Für ein $m \geq 1$ sowie ein beliebiges $v > 1$ gelte

$$m^2 - m^2 v + mv - 1 \leq 0.$$

Induktionsbehauptung. Wenn die Induktionsannahme gilt, so gilt auch

$$(m+1)^2 - (m+1)^2 v + (m+1)v - 1 \leq 0.$$

Beweis. Es gilt

$$\begin{aligned}(m+1)^2 - (m+1)^2 v + (m+1)v - 1 \\ = -2mv + 2m + 1 + m^2 - m^2 v + mv - 1.\end{aligned}$$

Mit der Induktionsannahme folgt

$$\leq -2mv + 2m + 1 + 0.$$

Wegen $m \geq 1$ und $v > 1$ gilt $2mv > 2m$, also insbesondere $-2mv + 2m + 1 \leq 0$. Daher folgt

$$\leq 0$$

und das war zu zeigen. ◊

Damit gilt Ungleichung (B.8), woraus Ungleichung (B.7) und somit die Behauptung folgt. □

Damit wurde gezeigt, dass die Selbstheilung im Worst Case stets mindestens so lang wie im Average Case dauert, höchstens jedoch doppelt so lang. Folglich liegen Worst- und Average Case der naiven Task-Dropping-Strategie nicht nur bezüglich der Anzahl gestoppter Tasks asymptotisch in der gleichen Komplexitätsklasse, sondern auch in Hinblick auf die resultierende Dauer der Selbstheilung.

C Lemmata

Im Laufe dieser Dissertation wurden verschiedene Hilfsaussagen verwendet, die an dieser Stelle bewiesen werden sollen.

Lemma zu Kapitel 8

Das erste Lemma befasst sich mit der Gleichheit konsekutiver Glieder der Folge $(a_z) \coloneqq \lceil m/z \rceil$ unter bestimmten Voraussetzungen und wird im Beweis von Satz 8.33 zur Herleitung der WCET des Eager Task Dropping genutzt:

Lemma C.1. Für $m, z \in \mathbb{N}$ mit $m \geq 1$ und $z \geq 2$ mit $m - (z-1) \cdot \lceil \frac{m}{z} \rceil \leq 0$ gilt

$$\left\lceil \frac{m}{z} \right\rceil = \left\lceil \frac{m}{z-1} \right\rceil.$$

Beweis. Gemäß der Voraussetzung gilt

$$m \leq (z-1) \cdot \left\lceil \frac{m}{z} \right\rceil,$$

mit $z - 1 \geq 1$ folgt also

$$\frac{m}{z-1} \leq \left\lceil \frac{m}{z} \right\rceil. \tag{C.1a}$$

Allgemein gilt aber

$$\frac{m}{z} \leq \frac{m}{z-1} \text{ für } z - 1 > 0.$$

Folglich gilt wegen der Monotonie der Aufrundungsfunktion auch

$$\left\lceil \frac{m}{z} \right\rceil \leq \left\lceil \frac{m}{z-1} \right\rceil. \tag{C.1b}$$

Mit (C.1a) und (C.1b) folgt daher

$$\frac{m}{z-1} \leq \left\lceil \frac{m}{z} \right\rceil \leq \left\lceil \frac{m}{z-1} \right\rceil.$$

Zwischen $m/(z-1)$ und $\lceil m/(z-1) \rceil$ liegt jedoch keine von $\lceil m/(z-1) \rceil$ verschiedene ganze Zahl, aber $\lceil m/z \rceil$ ist ganzzahlig. Daher muss $\lceil m/z \rceil = \lceil m/(z-1) \rceil$ gelten. □

Lemmata zu Anhang B

Anhang B befasste sich mit der Average-Case-Analyse der naiven Task-Dropping-Strategie, in deren Kontext insbesondere die erwartete Anzahl Taskstopps sowie die daraus resultierende Dauer der Selbstheilung in einer Überlastsituation hergeleitet wurden.

Erwartete Anzahl Taskstopps

Das folgende Lemma wird nicht direkt verwendet, bildet jedoch die Grundlage für den Beweis des darauffolgenden Lemmas C.3:

Lemma C.2. Für $n, r \in \mathbb{N}$ mit $n \geq r$ gilt

$$\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}.$$

Beweis. Durch Induktion über n.

Induktionsanfang. Sei $n = r$ für ein $r \in \mathbb{N}$.

In diesem Fall gilt

$$\sum_{i=r}^{r} \binom{i}{r} = \binom{r}{r} = 1 = \binom{r+1}{r+1}.$$

Induktionsschritt $n \to n+1$.

Induktionsannahme. Für ein $n \geq r$ gelte

$$\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}.$$

Induktionsbehauptung. Dann gilt auch

$$\sum_{i=r}^{n+1} \binom{i}{r} = \binom{(n+1)+1}{r+1} = \binom{n+2}{r+1}.$$

Beweis. Es gilt

$$\sum_{i=r}^{n+1} \binom{i}{r} = \sum_{i=r}^{n} \binom{i}{r} + \binom{n+1}{r}.$$

Mit der Induktionsannahme folgt

$$\sum_{i=r}^{n+1} \binom{i}{r} = \binom{n+1}{r+1} + \binom{n+1}{r},$$

die Identitiät $\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1}$ liefert schließlich

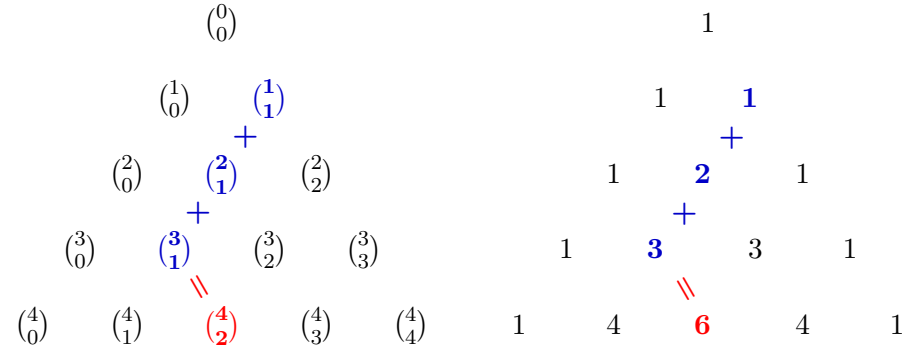
$$\sum_{i=r}^{n+1} \binom{i}{r} = \binom{n+2}{r+1}. \qquad \lozenge$$

Somit gilt die Behauptung für alle $n, r \in \mathbb{N}$ mit $n \geq r$. $\square$

Für elegante, nicht-induktive Beweise einer allgemeineren Variante dieser Behauptung sei an dieser Stelle auf [Knu97, 1.2.6, eq. 11] sowie [GKP94, 5.10] verwiesen.

Bemerkung

Dieses Lemma ist auch als *hockey stick identity* oder *stocking identity* bekannt, da bei einer Visualisierung der beteiligten Binomialkoeffizienten im Pascalschen Dreieck eine entfernte Ähnlichkeit zu einem Hockeyschläger oder einem Strumpf nur schwer von der Hand zu weisen ist (vgl. [Jon96]):



Unter Anwendung von Lemma C.2 lässt sich schließlich das folgende Lemma beweisen, das im Beweis von Satz B.15 verwendet wurde, der den Erwartungswert der Anzahl von der naiven Task-Dropping-Strategie gestoppter Tasks angibt:

Lemma C.3. Für $m \in \mathbb{N}$ und $k \in \mathbb{N}^+$ gilt

$$\sum_{i=1}^{k} i^{\overline{m}} = k \cdot \frac{(k+1)^{\overline{m}}}{1+m}.$$

Beweis. Es gilt:

$$\begin{aligned}
\sum_{i=1}^{k} i^{\overline{m}} &= m! \cdot \sum_{i=1}^{k} \frac{i^{\overline{m}}}{m!} \\
&= m! \cdot \sum_{i=1}^{k} \binom{i+m-1}{m} \\
&= m! \cdot \sum_{i=m}^{k+m-1} \binom{i}{m}
\end{aligned}$$

Unter Anwendung von Lemma C.2 folgt

$$\begin{aligned}
&= m! \cdot \binom{k+m-1+1}{m+1} = m! \cdot \binom{k+m}{m+1} \\
&= m! \cdot \frac{(k+m-(m+1)+1)^{\overline{m+1}}}{(m+1)!} \\
&= \frac{k^{\overline{m+1}}}{m+1} \\
&= k \cdot \frac{(k+1)^{\overline{m}}}{1+m}.
\end{aligned}$$

□

Dauer der Selbstheilung in einer Überlastsituation

Das folgende Lemma wurde in Anhang B.4.1 verwendet, um die unteren und oberen Schranken der ACET der naiven Task-Dropping-Strategie miteinander zu vergleichen:

Lemma C.4. Für $m, v \in \mathbb{N}^+$ gilt

$$-1 < \prod_{i=1}^{m} \frac{i}{i+mv} - \frac{m}{m+1} \cdot \prod_{i=1}^{m} \frac{i+m}{i+mv} \leq 0.$$

Beweis. Die Gültigkeit der unteren Schranke ist leicht einzusehen. Der Kürze halber seien die Summanden bzw. Faktoren hierzu wie folgt bezeichnet:

$$\overbrace{\prod_{i=1}^{m} \frac{i}{i+mv}}^{A} - \overbrace{\frac{m}{m+1}}^{B} \cdot \overbrace{\prod_{i=1}^{m} \frac{i+m}{i+mv}}^{C}$$

A und C sind jeweils Produkte von Faktoren f_i mit $0 < f_i < 1$, weil gemäß der Voraussetzungen $i < i + mv$ bzw. $i + m < i + mv$ gelten. Daher folgen $0 < A < 1$ und $0 < C < 1$, ferner ist wegen $0 < m < m + 1$ leicht $0 < B < 1$ einzusehen. Damit folgt $0 < B \cdot C < 1$ und somit $-1 < A - B \cdot C < 1$, woraus die untere Schranke folgt.

Zum Nachweis der behaupteten oberen Schranke beginnen wir mit der Aussage

$$\prod_{i=1}^{m-1} i \leq \prod_{i=1}^{m-1} (i+1+m),$$

deren Gültigkeit leicht einzusehen ist: Für $m = 1$ sind beide Produkte leer und es gilt die Gleichheit, für $m > 1$ sind dagegen alle Faktoren der rechten Seite größer als die Faktoren der linken Seite. Multiplizieren mit $m > 0$ sowie weitere Äquivalenzumformungen liefern

$$m \cdot \prod_{i=1}^{m-1} i \leq m \cdot \prod_{i=1}^{m-1} (i+1+m)$$

$$\prod_{i=1}^{m} i \leq m \cdot \prod_{i=2}^{m} (i+m)$$
$$\prod_{i=1}^{m} i \leq \frac{m}{m+1} \cdot \prod_{i=1}^{m} (i+m).$$

Für $1 \leq i \leq m$ gilt $i + mv > 0$, also liefert eine Division durch $\prod_{i=1}^{m}(i+mv) > 0$

$$\frac{\prod_{i=1}^{m} i}{\prod_{i=1}^{m}(i+mv)} \leq \frac{m}{m+1} \cdot \frac{\prod_{i=1}^{m}(i+m)}{\prod_{i=1}^{m}(i+mv)}$$

beziehungsweise

$$\prod_{i=1}^{m} \frac{i}{i+mv} \leq \frac{m}{m+1} \cdot \prod_{i=1}^{m} \frac{i+m}{i+mv}.$$

Eine Subtraktion der rechten Seite liefert schließlich die obere Schranke

$$\prod_{i=1}^{m} \frac{i}{i+mv} - \frac{m}{m+1} \cdot \prod_{i=1}^{m} \frac{i+m}{i+mv} \leq 0. \qquad \square$$

Alternativer Beweis von Satz B.15

D

In diesem Kapitel soll eine andere Sichtweise auf die Analyse des Average-Case der naiven Task-Dropping-Strategie eingenommen werden. Dazu wird zunächst ein Problem betrachtet, das sich später als Verallgemeinerung der Average-Case-Analyse der naiven Task-Dropping-Strategie herausstellen und schließlich einen alternativen Beweis für Satz B.15 liefern wird.

D.1. Anzahl der Überschreitungen

Die Analyse des Average-Case der naiven Task-Dropping-Strategie ist eng verwandt mit dem folgenden allgemeinen Problem:

Satz D.1. Gegeben seien L paarweise verschiedene Zahlen, o.B.d.A. wird angenommen, dass dies die Zahlen $1, \ldots, L$ sind.

Es wird nun zufällig und ohne Zurücklegen eine Stichprobe der Größe m aus diesen Zahlen gezogen.

Im Anschluss wird aus den verbleibenden $L - m$ Zahlen eine zweite Stichprobe der Größe n gezogen, ebenfalls zufällig und ohne Zurücklegen. Dabei gelten $m + n \leq L$ sowie $m, n \in \mathbb{N}^+$.

Anzahl der Überschreitungen

Für ein $r \in \mathbb{N}^+$ mit $1 \leq r \leq m$ gebe die Zufallsvariable X nun die *Anzahl der Überschreitungen* an, d.h. die Anzahl der Zahlen in der zweiten Stichprobe, die größer sind als die r-größte Zahl der ersten Stichprobe.

Dann gilt

$$\mathbf{P}(X = x) = \frac{\binom{m}{r}r\binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}}$$

für $x \in \{0, 1, \ldots, n\}$.[1]

Diese *Anzahl der Überschreitungen* geht zurück auf Gumbel und von Schelling [GvS50], die X als *number of exceedances* bezeichneten; dort wurde jedoch gefordert, dass die beiden Stichproben jeweils *unabhängig* aus einer *kontinuierlichen* Verteilung gezogen werden [JKK05, 6.2.2].

Sarkadi merkte in [Sar57] an, dass dieselbe Formel auch gilt, wenn die beiden Stichproben *ohne Zurücklegen* aus einer *diskreten* Population gezogen werden, was der in Satz D.1 gegebenen Formulierung entspricht. Ein formaler Beweis für diese Behauptung wurde dort jedoch nicht gegeben.

Da dieser Fall jedoch als Verallgemeinerung der Average-Case-Analyse der naiven Task-Dropping-Strategie gesehen werden kann, wie wir in Kapitel D.3.2 sehen werden, wollen wir Satz D.1 an dieser Stelle der Vollständigkeit halber zunächst kombinatorisch beweisen.

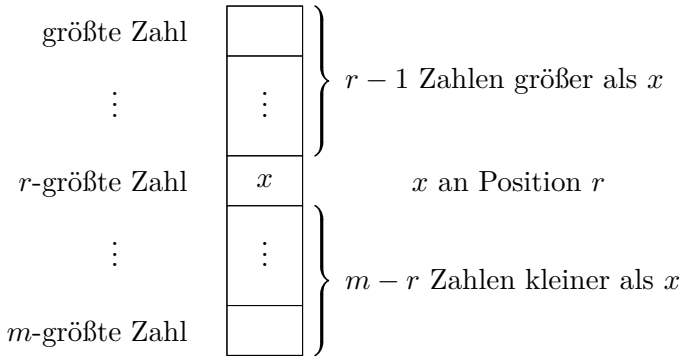
[1] Insbesondere ist $\mathbf{P}(X = x)$ also unabhängig von der Anzahl der Zahlen L.

Beweis. Wir betrachten zunächst eine Zufallsvariable W, die den Wert der r-größten Zahl in der ersten Stichprobe angebe.

Es gibt insgesamt $\binom{L}{m}$ verschiedene Stichproben der Größe m. Damit $W = x$ für ein $x \in \{1, \dots, L\}$ gilt, müssen

- $m - r$ Zahlen der Stichprobe *kleiner* als x sein (es gibt insgesamt $x - 1$ solcher Zahlen),
- $r - 1$ Zahlen der Stichprobe *größer* als x sein (es gibt insgesamt $L - x$ solcher Zahlen),
- x selbst muss in der Stichprobe enthalten sein.

Dies ist in folgender Skizze illustriert:



Mithilfe dieser Überlegungen kann nun für $x \in \{1, \dots, L\}$ die Wahrscheinlichkeitsverteilung von W angegeben werden:

$$\mathbf{P}(W = x) = \frac{\binom{x-1}{m-r}\binom{L-x}{r-1}\binom{1}{1}}{\binom{L}{m}} = \frac{\binom{x-1}{m-r}\binom{L-x}{r-1}}{\binom{L}{m}}. \tag{D.1}$$

Nun betrachten wir eine Zufallsvariable Y, die angebe, wie viele Zahlen, die größer als W sind, sich nach dem Ziehen der ersten Stichprobe noch unter den $L-m$ verbleibenden Zahlen befinden.

Wenn $W = x$ gilt, existieren insgesamt $L - x$ Zahlen, die größer als W sind. Von diesen sind jedoch $r - 1$ Zahlen in der ersten Stichprobe enthalten. Unter allen $L - m$ verbleibenden Zahlen, aus denen die zweite Stichprobe gezogen wird, sind dann folglich nur noch $Y = L - x - (r - 1) = L - x - r + 1$ Zahlen größer als W.

Mit $W = x$ folgt daher $Y = L - W - r + 1$ bzw. $W = L - Y - r + 1$ und folglich

$$\mathbf{P}(Y = k) = \mathbf{P}(W = L - k - r + 1),$$

mithilfe von Gleichung (D.1) ergibt sich

$$= \frac{\binom{L-k-r}{m-r}\binom{k+r-1}{r-1}}{\binom{L}{m}}. \tag{D.2}$$

Anders interpretiert gibt Y nun an, wie viele Zahlen, die größer als W sind, für die Ziehung der zweiten Stichprobe in Betracht kommen. Die gesuchte Zufallsvariable X beschreibt dagegen, wie viele dieser Zahlen *tatsächlich* in der zweiten Stichprobe enthalten sind.

Mithilfe von Y lässt sich nun jedoch die *bedingte* Wahrscheinlichkeitsverteilung von X angeben. Hierzu kann – ähnlich zur Herleitung von Gleichung (D.1) – eine hypergeometrische Verteilung zugrunde gelegt werden:

- Die Grundgesamtheit besteht aus allen $L - m$ verbleibenden Zahlen,
- $Y = k$ gibt die Anzahl möglicher Erfolge an (nämlich die Anzahl der Zahlen mit der Eigenschaft „größer als W“, die sich unter den Elementen der Grundgesamtheit befinden),
- die Stichprobe umfasst n Elemente.

Damit ergibt sich für $x \in \{0, 1, \ldots, n\}$

$$\mathbf{P}(X = x \mid Y = k) = \frac{\binom{k}{x}\binom{L-m-k}{n-x}}{\binom{L-m}{n}}. \tag{D.3}$$

Hierbei muss offensichtlich $x \leq k \leq L-m$ gelten: Wenn $k < x$, so gibt es keine mögliche Stichprobe, in der x Zahlen enthalten sind, die größer als W sind. Besteht die erste Stichprobe dagegen aus den m kleinsten Zahlen $1, \ldots, m$, so sind *alle* verbleibenden $L - m$ Zahlen größer als W.

Aus den Gleichungen (D.2) und (D.3) kann nun mithilfe des Satzes von der totalen Wahrscheinlichkeit die gesuchte Wahrscheinlichkeitsverteilung von X bestimmt werden:

$$\begin{aligned}
&\mathbf{P}(X = x) \\
&\quad = \sum_{k=x}^{L-m} \mathbf{P}(X = x \mid Y = k) \cdot \mathbf{P}(Y = k) \\
&\quad = \sum_{k=x}^{L-m} \frac{\binom{k}{x}\binom{L-m-k}{n-x}}{\binom{L-m}{n}} \cdot \frac{\binom{L-k-r}{m-r}\binom{k+r-1}{r-1}}{\binom{L}{m}}
\end{aligned}$$

Für die roten Binomialkoeffizienten der Form $\binom{a-k}{b}$ kann unter Umständen *nicht* die kombinatorische Definition zugrunde gelegt werden, da für bestimmte Werte von k jeweils $a - k < b$ gelten kann (es gilt jedoch stets $b \geq 0$). Diese werden daher gemäß der allgemeinen Definition expandiert, während die übrigen Binomialkoeffizienten durch ihre jeweilige kombinatorische Darstellung ersetzt werden können:[1]

$$= \sum_{k=x}^{L-m} \frac{\frac{k!}{x!\,(k-x)!} \cdot \frac{(L-m-k)^{\underline{n-x}}}{(n-x)!}}{\frac{(L-m)!}{n!\,(L-m-n)!}} \cdot \frac{\frac{(L-k-r)^{\underline{m-r}}}{(m-r)!} \cdot \frac{(k+r-1)!}{(r-1)!\,k!}}{\frac{L!}{m!\,(L-m)!}}$$

[1] Die im Folgenden verwendete Farbgebung hebt der Übersicht halber jeweils Teilterme hervor, die im nächsten Schritt umgeformt beziehungsweise miteinander verrechnet werden.

$$
\begin{aligned}
&= \frac{m!}{(r-1)!\,(m-r)!} \cdot \frac{r}{r} \cdot \frac{n!}{x!\,(n-x)!} \\
&\qquad \cdot \sum_{k=x}^{L-m} \frac{(L-m-n)!\,(L-m-k)^{\underline{n-x}}\,(L-k-r)^{\underline{m-r}}\,(k+r-1)!}{(k-x)!\,L!} \\
&= \frac{m!}{r!\,(m-r)!} \cdot r \cdot \frac{n!}{x!\,(n-x)!} \\
&\qquad \cdot \sum_{k=x}^{L-m} \frac{(L-m-n)!\,(L-m-k)^{\underline{n-x}}\,(L-k-r)^{\underline{m-r}}\,(k+r-1)!}{(k-x)!\,L!}
\end{aligned}
$$

Die Quotienten der Fakultäten in der oberen Zeile können nun wieder als Binomialkoeffizienten ausgedrückt werden, während die fallenden Faktoriellen in der unteren Zeile umgeformt werden können:

$$
\begin{aligned}
&= \binom{m}{r} r \binom{n}{x} \cdot \sum_{k=x}^{L-m} \frac{(L-m-n)! \cdot \prod_{i=\substack{L-m-k\\-(n-x)+1}}^{L-m-k} i \cdot \prod_{i=\substack{L-k-r\\-(m-r)+1}}^{L-k-r} i \cdot (k+r-1)!}{(k-x)!\,L!} \\
&= \binom{m}{r} r \binom{n}{x} \cdot \sum_{k=x}^{L-m} \frac{(L-m-n)! \cdot \prod_{i=\substack{L-m-n\\-k+x+1}}^{L-m-k} i \cdot \prod_{i=\substack{L-m\\-k+1}}^{L-k-r} i \cdot (k+r-1)!}{(k-x)!\,L!}
\end{aligned}
$$

Die beiden Produkte können nun zusammengefasst und wieder als fallende Faktorielle dargestellt werden:

$$
\begin{aligned}
&= \binom{m}{r} r \binom{n}{x} \cdot \sum_{k=x}^{L-m} \frac{(L-m-n)! \cdot \prod_{i=\substack{L-k-r\\-(m+n-r-x)+1}}^{L-k-r} i \cdot (k+r-1)!}{(k-x)!\,L!} \\
&= \binom{m}{r} r \binom{n}{x} \cdot \sum_{k=x}^{L-m} \frac{(L-m-n)!\,(L-k-r)^{\underline{m+n-r-x}}\,(k+r-1)!}{(k-x)!\,L!}.
\end{aligned}
$$

Ein Erweitern der Gleichung mit $(m+n)\binom{m+n-1}{r+x-1}$ bei gleichzeitiger Verschiebung der Summationsgrenzen liefert

$$
\begin{aligned}
&= \binom{m}{r} r \binom{n}{x} \cdot \frac{(m+n)\binom{m+n-1}{r+x-1}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{(L-m-n)!}{L!} \\
&\qquad \cdot \sum_{k=0}^{L-m-x} \frac{(L-k-x-r)^{\underline{m+n-r-x}}\,(k+x+r-1)!}{(k+x-x)!}
\end{aligned}
$$

$$= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{(L-(m+n))!}{L!} \cdot \overbrace{(m+n)(m+n-1)!}^{(m+n)!} \cdot \sum_{k=0}^{L-m-x} \frac{(L-k-x-r)^{\underline{m+n-r-x}}\,(k+x+r-1)!}{(r+x-1)!\,(m+n-r-x)!\,k!}$$

$$= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{1}{\binom{L}{m+n}} \cdot \sum_{k=0}^{L-m-x} \frac{(L-k-x-r)^{\underline{m+n-r-x}}\,(k+x+r-1)!}{(r+x-1)!\,(m+n-r-x)!\,k!}$$

$$= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{1}{\binom{L}{m+n}} \cdot \sum_{k=0}^{L-m-x} \binom{L-r-x-k}{m+n-r-x} \binom{r+x-1+k}{k}.$$

Auf den Binomialkoeffizienten $\binom{r+x-1+k}{k}$ kann Lemma B.10 angewendet werden, da $r+x-1+k \geq 0$ wegen $r \geq 1$ sowie $x, k \geq 0$ gilt und alle beteiligten Zahlen ganzzahlig sind:

$$= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{1}{\binom{L}{m+n}} \cdot \sum_{k=0}^{L-m-x} \binom{L-r-x-k}{m+n-r-x} \binom{r+x-1+k}{r+x-1}$$

Wegen $1 \leq r \leq m$ gilt zudem $L-m-x \leq L-r-x$, die Summe lässt sich daher als Differenz zweier Summen ausdrücken:

$$= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{1}{\binom{L}{m+n}} \cdot \left(\sum_{k=0}^{L-r-x} \binom{L-r-x-k}{m+n-r-x} \binom{r+x-1+k}{r+x-1} - \sum_{k=L-m-x+1}^{L-r-x} \binom{L-r-x-k}{m+n-r-x} \binom{r+x-1+k}{r+x-1} \right)$$

Die erste Summe kann jedoch nach Lemma B.12 durch einen einzigen Binomialkoeffizienten ersetzt werden:

$$= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{1}{\binom{L}{m+n}} \cdot \left(\binom{L-r-x+r+x-1+1}{m+n-r-x+r+x-1+1} - \sum_{k=L-m-x+1}^{L-r-x} \binom{L-r-x-k}{m+n-r-x} \binom{r+x-1+k}{r+x-1} \right)$$

Eine Verschiebung der Grenzen der verbleibenden Summe liefert

$$= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{1}{\binom{L}{m+n}} \cdot \left(\binom{L}{m+n} - \sum_{k=0}^{m-r-1} \binom{L-r-x-(k+L-m-x+1)}{m+n-r-x} \binom{r+x-1+k+L-m-x+1}{r+x-1} \right)$$

$$= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{1}{\binom{L}{m+n}} \cdot \left(\binom{L}{m+n} - \sum_{k=0}^{m-r-1} \binom{m-r-1-k}{m-r+n-x} \binom{L-m+r+k}{r+x-1} \right)$$

Der Wert der verbleibenden Summe beträgt jedoch Null:

a) $r = m$. In diesem Fall gilt $m - r - 1 = -1 < 0$, die Summe enthält folglich keine Summanden und hat daher den Wert Null.

b) $1 \leq r < m$. In diesem Fall gilt $m - r - 1 \geq 0$, die Summe enthält also mindestens einen Summanden.

   Wegen $0 \leq k \leq m - r - 1$ gilt dann jedoch $0 \leq m - r - 1 - k \leq m - r - 1$.

   Ferner gilt $0 \leq x \leq n$, also gilt auch $m - r \leq m - r + n - x \leq m - r + n$.

   Setzt man die Ungleichungen zusammen, so ergibt sich

   $$0 \leq m - r - 1 - k \leq m - r - 1 < m - r \leq m - r + n - x$$

   und somit insbesondere

   $$0 \leq m - r - 1 - k < m - r + n - x.$$

   Das obere Argument des Binomialkoeffizienten $\binom{m-r-1-k}{m-r+n-x}$ ist folglich stets nichtnegativ, jedoch kleiner als das untere Argument, weshalb der Wert des Binomialkoeffizienten gemäß Lemma B.9 daher Null beträgt.

   Folglich enthält jeder der Summanden den Faktor Null, damit beträgt auch der Wert der gesamten Summe Null.

Mit diesen Überlegungen folgt schließlich

$$\begin{aligned} \mathbf{P}(X = x) &= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \cdot \frac{1}{\binom{L}{m+n}} \cdot \left(\binom{L}{m+n} - 0 \right) \\ &= \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}} \end{aligned}$$

und das war zu zeigen. □

D.2. Beta-Binomialverteilung

Die Wahrscheinlichkeitsverteilung der Anzahl der Überschreitungen lässt sich nun wie folgt darstellen:

Satz D.2. Für die Anzahl der Überschreitungen X gemäß Satz D.1 gilt

$$\mathbf{P}(X = x) = \frac{\binom{-r}{x}\binom{-(m-r+1)}{n-x}}{\binom{-(m+1)}{n}}$$

für $x \in \{0, 1, \ldots, n\}$.

Der folgende Beweis dieses Satzes orientiert sich an [JKK05, 6.2.2]:

Beweis. Nach Satz D.1 gilt

$$\mathbf{P}(X = x) = \frac{\binom{m}{r} r \binom{n}{x}}{(m+n)\binom{m+n-1}{r+x-1}}$$

Da alle Argumente der Binomialkoeffizienten nichtnegative Ganzzahlen und die oberen Argumente nie kleiner als die zugehörigen unteren Argumente sind, kann die kombinatorische Definition der Binomialkoeffizienten angewandt werden:

$$\begin{aligned}
&= \frac{m!}{r!\,(m-r)!} \cdot r \cdot \frac{n!}{x!\,(n-x)!} \cdot \frac{(r+x-1)!\,(m+n-1-r-x+1)!}{(m+n)\,(m+n-1)!} \\
&= \frac{m!}{(r-1)!\,(m-r)!} \cdot \frac{n!}{x!\,(n-x)!} \cdot \frac{(r+x-1)!\,(m+n-r-x)!}{(m+n)!} \\
&= \frac{(-1)^n}{(-1)^n} \cdot \frac{\binom{r+x-1}{x}\binom{m+n-r-x}{n-x}}{\binom{m+n}{n}} \\
&= \frac{(-1)^x \cdot \binom{x-(-r)-1}{x} \cdot (-1)^{n-x} \cdot \binom{(n-x)-(-(m-r+1))-1}{n-x}}{(-1)^n \cdot \binom{n-(-(m+1))-1}{n}}
\end{aligned}$$

Unter Anwendung von Lemma B.11 ergibt sich nun

$$= \frac{\binom{-r}{x}\binom{-(m-r+1)}{n-x}}{\binom{-(m+1)}{n}}$$

und das war zu zeigen. □

Die dieser Form zugrunde liegende Verteilung wird als Beta-Binomialverteilung bezeichnet, die folgende Definition orientiert sich an [JKK05]:

Definition D.3. Eine Zufallsvariable X heiße *beta-binomialverteilt* zu den Parametern $\alpha \geq 0$, $\beta \geq 0$ und $n \in \mathbb{N}^+$, wenn sie für $x \in \{0, 1, \ldots, n\}$ die Wahrscheinlichkeitsvertei- **Beta-Binomialverteilung**

lung

$$\mathbf{P}(X = x) = \frac{\binom{-\alpha}{x}\binom{-\beta}{n-x}}{\binom{-(\alpha+\beta)}{n}}$$

besitzt.

Ferner ergeben sich nach [JKK05, 6.12] Erwartungswert $\mathbf{E}[X]$ und Varianz $\mathrm{Var}(X)$ der Zufallsvariablen X zu

$$\mathbf{E}[X] = \frac{n\alpha}{\alpha+\beta} \tag{D.4}$$

sowie

$$\mathrm{Var}(X) = \frac{n\alpha\beta(\alpha+\beta+n)}{(\alpha+\beta)^2(\alpha+\beta+1)}. \tag{D.5}$$

Vergleicht man nun die Wahrscheinlichkeitsverteilung der Anzahl der Überschreitungen, wie sie in Satz D.2 angegeben ist, mit Definition D.3, so lassen sich folgende Aussagen ableiten:

Satz D.4. Die Anzahl der Überschreitungen X gemäß Satz D.1 ist beta-binomialverteilt zu den Parametern $\alpha = r$, $\beta = m - r + 1$ und n.

Beweis. Einsetzen der angegebenen Parameter liefert die Behauptung. □

Satz D.5. Für die Anzahl der Überschreitungen X gemäß Satz D.1 ergeben sich Erwartungswert und Varianz zu

$$\mathbf{E}[X] = \frac{nr}{m+1}$$
$$\mathrm{Var}(X) = \frac{nr(m-r+1)(m+n+1)}{(m+1)^2(m+2)}.$$

Beweis. Ergibt sich aus Satz D.4 unter Anwendung der Gleichungen (D.4) und (D.5). □

D.3. Anwendung auf ACET der naiven Task-Dropping-Stragegie

Im Beweis von Satz B.15 wurde die Verteilung der Zufallsvariablen X, welche die Anzahl von der naiven Task-Dropping-Strategie gestoppter Tasks angibt, direkt anhand kombinatorischer Überlegungen hergeleitet. Anschließend konnte ihr Erwartungswert $\mathbf{E}[X]$ berechnet werden, der dem Average Case der Anzahl gestoppter Tasks entspricht.

Die Anzahl der Überschreitungen sowie die zugrunde liegende Beta-Binomialverteilung ermöglichen es nun jedoch, den Erwartungswert von X auf zwei alternativen Wegen herzuleiten.

Satz D.6. Sei X eine Zufallsvariable, welche die Anzahl der von der naiven Task-Dropping-Strategie gestoppten Tasks angibt.

Dann gilt: X ist beta-binomialverteilt zu den Parametern $\alpha = m$, $\beta = 1$ und $n = mv$.

Insbesondere gelten

$$\mathbf{E}[X] = \frac{m^2 v}{1+m}$$

sowie

$$\mathrm{Var}(X) = \frac{m^2 v(1+m+mv)}{(m+1)^2(m+2)}.$$

D.3.1. Direkte Herleitung über die Beta-Binomialverteilung

Der erste Weg stützt sich auf die Herleitung der Wahrscheinlichkeitsverteilung von X aus dem Beweis von Satz B.15:

Beweis. Für die genannten Parameter ergibt sich die Wahrscheinlichkeitsverteilung einer beta-binomialverteilten Zufallsvariablen X nach Definition D.3 für $0 \leq x \leq n = mv$ zu

$$\mathbf{P}(X = x) = \frac{\binom{-m}{x}\binom{-1}{mv-x}}{\binom{-(1+m)}{mv}}.$$

Unter Anwendung von Lemma B.11 folgt

$$\begin{aligned}
&= \frac{(-1)^x \cdot \binom{x+m-1}{x} \cdot (-1)^{mv-x} \cdot \binom{mv-x+1-1}{mv-x}}{(-1)^{mv} \cdot \binom{mv+m+1-1}{mv}} \\
&= \frac{\binom{x+m-1}{x}\binom{mv-x}{mv-x}}{\binom{mv+m}{mv}} \\
&= \frac{(x+m-1)^{\underline{x}}\,(mv)!}{x!\,(mv+m)^{\underline{mv}}} \\
&= \frac{\prod_{i=x+m-1-x+1}^{x+m-1} i \cdot \prod_{i=1}^{mv} i}{\prod_{i=1}^{x} i \cdot \prod_{i=mv+m-mv+1}^{mv+m} i}.
\end{aligned}$$

Vereinfachen der Grenzen sowie Erweitern mit m, um die Grenze des zweiten Produktes im Nenner von $m+1$ auf m anzupassen, liefert

$$= m \cdot \frac{\prod_{i=m}^{x+m-1} i \cdot \prod_{i=1}^{mv} i}{\prod_{i=1}^{x} i \cdot \prod_{i=m}^{mv+m} i}.$$

Wegen $mv \geq x$ können die Produkte mit übereinstimmenden Grenzen jeweils zusammengefasst werden und es folgt

$$\mathbf{P}(X = x) = m \cdot \frac{\prod_{i=x+1}^{mv} i}{\prod_{i=m+x}^{mv+m} i}.$$

Dies entspricht jedoch gerade Gleichung (B.2) (siehe S. 253), woraus der erste Teil der Behauptung folgt. Die Formeln für Erwartungswert und Varianz folgen dann unmittelbar aus den Gleichungen (D.4) und (D.5). □

D.3.2. Herleitung über die Anzahl der Überschreitungen

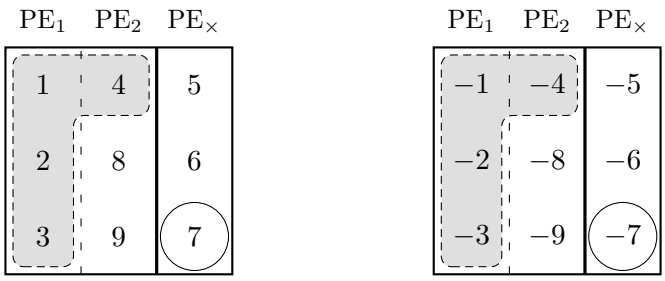
Der zweite alternative Beweis, der bereits in [Hut21] veröffentlicht wurde, nutzt aus, dass die naive Task-Dropping-Strategie als Spezialfall der Anzahl der Überschreitungen gesehen werden kann. Um dies einzusehen, werden die $m(v+1) = mv+m$ verschiedenen Prioritäten $1, \ldots, (mv + m)$ zunächst negiert. Die $mv + m$ resultierenden Zahlen $-(mv + m), \ldots, -1$ werden dann wie folgt verteilt:

- Zunächst wird zufällig eine Stichprobe von m Zahlen ohne Zurücklegen gezogen. Dies entspricht den *negierten* Prioritäten jener Tasks, die zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$ auf diesem PE zugeteilt waren.
- Alle mv verbleibenden Zahlen bilden eine zweite Stichprobe der Größe mv. Dies entspricht den *negierten* Prioritäten jener Tasks, die zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$ auf $\mathrm{PE}_1 \ldots \mathrm{PE}_v$ zugeteilt waren.

Die naive Task-Dropping-Strategie wird nun alle zugeteilten Tasks stoppen, deren Priorität *kleiner* ist als die höchste Priorität aller auf $\mathrm{PE}_\times$ zugeteilten Tasks. Bezogen auf die negierten Prioritäten entspricht dies jedoch gerade allen Tasks, deren *negierte* Priorität *größer* ist als die m-größte negierte Priorität der ersten Stichprobe. Dies wird in folgendem Beispiel illustriert:

Beispiel D.7

Es gelten $m = 3$ sowie $v = 2$ und wir betrachten die auf den einzelnen PEs zugeteilten Tasks zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$:



Auf der linken Seite sind die regulären Prioritäten der Tasks dargestellt. Alle grau unterlegten Tasks haben eine geringere Priorität als die eingekreiste höchstpriore Task $T_{\max}$, die zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$ auf diesem zugeteilt war und werden folglich gestoppt.

Auf der rechten Seite sind dagegen die negierten Prioritäten der Tasks dargestellt. Die grau unterlegten Tasks entsprechen hier gerade allen Tasks auf PE_1 und PE_2, deren negierte Priorität größer ist als die eingekreiste niedrigstpriore Task $T_{\min}$, die zum Zeitpunkt des Ausfalls von $\mathrm{PE}_\times$ auf diesem zugeteilt war. Diese Task ist jedoch gerade die m-höchstpriore Task auf $\mathrm{PE}_\times$.

Diese Überlegungen ermöglichen es nun, Satz D.6 mithilfe der Anzahl der Überschreitungen zu beweisen:

Beweis. Die Anzahl X von der naiven Task-Dropping-Strategie gestoppter Tasks entspricht der Anzahl der Überschreitungen für $r = m$, wobei die erste Stichprobe m Elemente und die zweite Stichprobe mv Elemente umfasst.

Gemäß Satz D.2 und Satz D.4 ergibt sich somit die Wahrscheinlichkeitsverteilung von X für $x \in \{0, 1, \ldots, mv\}$ zu

$$\mathbf{P}(X = x) = \frac{\binom{-m}{x}\binom{-1}{mv-x}}{\binom{-(m+1)}{mv}}.$$

Erwartungswert und Varianz folgen gemäß Satz D.5. □

Alternativ können Erwartungswert und Varianz der zugrunde liegenden Anzahl der Überschreitungen auch direkt hergeleitet werden, anstatt die Verteilung zunächst auf eine Beta-Binomialverteilung zurückzuführen. Hierzu sei auf [GvS50] sowie [Gum54] verwiesen.

D.4. Zusammenfassung

Dieses Kapitel lieferte zwei alternative Sichtweisen auf die erwartete Anzahl Tasks, die von der naiven Task-Dropping-Strategie gestoppt werden, sofern das in Kapitel 8.1 vorgestellte Modell für Überlastsituationen Anwendung findet sowie Annahme B.13 gilt, alle Verteilungen der $m(v + 1)$ Tasks auf die $v + 1$ PEs also gleich wahrscheinlich sind.

Es wurde insbesondere gezeigt, dass die Anzahl X gestoppter Tasks beta-binomialverteilt ist. Diese Beobachtung ermöglicht es insbesondere, Erwartungswert und Varianz von X mithilfe der Gleichungen (D.4) und (D.5) zu berechnen, während in Anhang B lediglich der Erwartungswert hergeleitet wurde.

E Übersicht über die wichtigsten Zeitschranken

In diesem Kapitel soll eine Übersicht über die wichtigsten Zeitschranken gegeben werden, die für die prioritätsbasierte Task-Decision-Strategie hergeleitet wurden.

Hierbei nehmen wir jedoch stets an, dass *keine* mehrfach zuteilbaren Tasks existieren.

E.1. Selbstkonfiguration

Es wird angenommen, dass im System mindestens zwei PEs existieren. Ferner bezeichne

$\boldsymbol{m}$ die Anzahl aller im System vorhandenen Tasks,

$\boldsymbol{P}$ die Menge der Prioritäten dieser Tasks sowie

$\boldsymbol{\mathcal{T}_p}$ die Menge aller Tasks mit Priorität p.

Dann gilt

$$\text{WCET} \leq m + |P| - 1 \qquad \text{Satz 6.9 (S. 79)}$$

$$\text{WCET} \leq 2m - 1 \qquad \text{(falls alle Prioritäten verschieden sind)} \qquad \text{Satz 6.10 (S. 79)}$$

$$\text{D}(T) \leq \sum_{\substack{p \in P \\ p \geq \text{prio}(T)}} (|\mathcal{T}_p| + 1) - 1. \qquad \text{Satz 6.12 (S. 81)}$$

E.2. Selbstoptimierung

Es wird angenommen, dass im System mindestens zwei PEs existieren. Wie zuvor bezeichne m die Anzahl aller im System vorhandenen Tasks und P die Menge ihrer Prioritäten.

Dann gilt $\text{WCET} \leq m + |P| - 1$. Satz 6.14 (S. 82)

E.3. Selbstheilung

E.3.1. In Unterlastsituationen

Es bezeichne

$\boldsymbol{v}$ die Anzahl der verbleibenden PEs ($v \geq 1$),

$\boldsymbol{m_f}$ die Anzahl der Tasks, die auf dem ausgefallenen $\text{PE}_\times$ zugeteilt waren,

$\boldsymbol{P_f}$ die Menge der Prioritäten dieser Tasks und

$\boldsymbol{a}$ die in Hormonzyklen gemessene Zeit, die höchstens benötigt wird, um den PE-Ausfall zu bemerken.

Dann gilt

Satz 6.15 (S. 82) $\quad \mathrm{WCET} \leq m_f + (|P_f| - 1) \cdot [v > 1] + a$

Korollar 6.17 (S. 83) $\quad \mathrm{WCET} \leq m_f + (\,m_f\, - 1) \cdot [v > 1] + a.$ (falls alle Prioritäten verschieden sind)

Zuteilungsdauer Es bezeichne

$\boldsymbol{\mathcal{T}_f}$ die Menge der Tasks, die auf dem ausgefallenen $\mathrm{PE}_\times$ zugeteilt waren,

$\boldsymbol{P_f}$ die Menge der Prioritäten dieser Tasks und

$\boldsymbol{\mathcal{T}_{f,p}}$ die Menge aller Tasks aus $\mathcal{T}_f$ mit Priorität p.

Dann gilt für $T \in \mathcal{T}_f$

Satz 6.19 (S. 83)
$$\mathrm{D}(T) \leq \sum_{\substack{p \in P_f \\ p \geq \mathrm{prio}(T)}} (|\mathcal{T}_{f,p}| + [v > 1]) - [v > 1] + a.$$

E.3.2. In Überlastsituationen

Es werde das Überlastmodell aus Kapitel 8.2 verwendet, demnach bezeichnet

$\boldsymbol{m}$ die Anzahl Tasks, die pro PE übernommen werden können,

$\boldsymbol{v}$ die Anzahl verbleibender PEs ($v \geq 1$),

$\boldsymbol{P}$ die Menge aller im System vorhandenen Taskprioritäten,

$\boldsymbol{a}$ die in Hormonzyklen gemessene Zeit, die *höchstens* benötigt wird, um den PE-Ausfall zu bemerken und

$\boldsymbol{a_{\min}}$ die in Hormonzyklen gemessene Zeit, die *mindestens* benötigt wird, um den PE-Ausfall zu bemerken.

E.3.2.1. Naive Task-Dropping-Strategie

Es bezeichne $\mathbf{E}[X]$ die erwartete Anzahl gestoppter Tasks, falls alle Taskprioritäten verschieden und gemäß Annahme B.13 alle Verteilungen der Tasks auf die PEs gleich wahrscheinlich sind. Ferner bezeichne $\mathrm{ACET}_{\mathrm{naiv}}$ den Average Case der Selbstheilungsdauer gemäß dieser Annahmen.

Dann gilt

$$\text{WCET} \leq mv + (mv-1)\cdot[v>1] + 1 + a \qquad \text{Satz 8.11 (S. 130)}$$

$$\mathbf{E}[X] = \frac{m^2 v}{1+m} \qquad \text{Satz B.15 (S. 251)}$$

$$\text{ACET}_{\text{naiv}} \geq 2\frac{m^2 v}{1+m} - m + \frac{\binom{2m}{m+1}-1}{\binom{mv+m}{m}} + 1 + a_{\min} \qquad \text{Satz B.18 (S. 258)}$$

$$\text{ACET}_{\text{naiv}} \leq a + \begin{cases} \frac{m^2 v}{1+m} + 1 - \frac{1}{\binom{mv+m}{m}}, & v = 1 \\ 2\frac{m^2 v}{1+m}, & v > 1. \end{cases} \qquad \text{Satz B.18 (S. 258)}$$

Falls im System nur $|P| \leq mv$ verschiedene Taskprioritäten existieren, gilt ferner

$$\text{WCET} \leq mv + (\min\{|P|, mv\} - 1)\cdot[v>1] + 1 + a. \qquad \text{Satz 8.12 (S. 131)}$$

E.3.2.2. Eager Task Dropping

Es gilt

$$\text{WCET} \leq 2\cdot\sum_{i=1}^{v}\left\lceil\frac{m}{i}\right\rceil + a \qquad \text{(PE-orientierte Sichtweise)} \qquad \text{Satz 8.37 (S. 152)}$$

$$\text{WCET} \leq 2\cdot\left(v + \sum_{i=2}^{m}\min\left\{v, \left\lfloor\frac{m-1}{i-1}\right\rfloor\right\}\right) + a. \qquad \text{(phasenorientierte Sichtweise)} \qquad \text{Satz 8.40 (S. 161)}$$

Ferner gilt folgende Verallgemeinerung, die für $|P| < v$ eine bessere Abschätzung liefert:

$$\text{WCET} \leq \sum_{i=1}^{v}\left\lceil\frac{m}{i}\right\rceil + \min\{v, |P|\} + \sum_{i=2}^{m}\min\left\{v, \left\lfloor\frac{m-1}{i-1}\right\rfloor, |P|\right\} + a. \qquad \text{Satz 8.51 (S. 183)}$$

Literatur

[Adv20] Advanced Micro Devices, Inc. *AMD EPYC™ 7002 Series Processors: A New Standard for the Modern Datacenter*. Datenblatt. 2020, S. 2.

[All+03] Ralf Allrutz u. a. *VDE/ITG/GI-Positionspapier - Organic Computing: Computer- Und Systemarchitektur Im Jahr 2010*. VDE/ITG/GI-Positionspapier. 2003.

[Amm+16] Habib M. Ammari u. a. „Review of Applications of Wireless Sensor Networks“. In: *Wireless Sensor Networks*. Hrsg. von Shafiullah Khan, Al-Sakib Khan Pathan und Nabil Ali Alrajeh. Boca Raton, FL: CRC Press, 2016, S. 3–31. ISBN: 978-1-138-19987-3.

[Ank] Anker Technology (UK) Ltd. *PowerPort Atom PD 2*. Anker. URL: https://www.anker.com/products/variant/powerport-atom-pd-2/A2029121 (besucht am 21. 02. 2020).

[Arm] Arm Ltd. *Cortex-M0*. ARM Developer. URL: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m0 (besucht am 21. 02. 2020).

[AUT] AUTOSAR development cooperation. *AUTOSAR*. URL: https://www.autosar.org/ (besucht am 28. 02. 2020).

[Bar+09] Clark Barrett u. a. „Satisfiability Modulo Theories“. In: *Handbook of Satisfiability*. Hrsg. von Armin Biere u. a. Bd. 185. Frontiers in Artificial Intelligence and Applications. Amsterdam: IOS Press, 2009, S. 825–885. ISBN: 978-1-60750-376-7.

[BF19] Uwe Brinkschulte und Felix Fastnacht. „Applying the Concept of Artificial DNA and Hormone System to a Low-Performance Automotive Environment“. In: *Architecture of Computing Systems – ARCS 2019*. Architecture of Computing Systems – ARCS 2019. Hrsg. von Martin Schoeberl u. a. Bd. 11479. Cham: Springer International Publishing, 2019, S. 87–99. ISBN: 978-3-030-18655-5. DOI: 10.1007/978-3-030-18656-2_7.

[BHF19] Uwe Brinkschulte, Eric Hutter und Felix Fastnacht. „Adapting the Concept of Artificial DNA and Hormone System to a Classical AUTOSAR Environment“. In: *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*. 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC). Valencia, Spain: IEEE, Mai 2019, S. 35–42. ISBN: 978-1-72810-151-4. DOI: 10.1109/ISORC.2019.00016.

[BP12] Uwe Brinkschulte und Mathias Pacher. „An Agressive Strategy for an Artificial Hormone System to Minimize the Task Allocation Time“. In: *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. 2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW). Shenzhen, China: IEEE, Apr. 2012, S. 188–195. ISBN: 978-0-7695-4669-8. DOI: 10.1109/ISORCW.2012.40.

[BP21] Uwe Brinkschulte und Mathias Pacher. „Handling Communication Dropouts in an Artificial Hormone and DNA System“. In: *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*. 2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC). Daegu, South Korea: IEEE, Juni 2021, S. 123–131. DOI: 10.1109/ISORC52013.2021.00026.

[BPvR07] Uwe Brinkschulte, Mathias Pacher und Alexander von Renteln. „Towards an Artificial Hormone System for Self-organizing Real-Time Task Allocation“. In: *Software Technologies for Embedded and Ubiquitous Systems*. Hrsg. von Roman Obermaisser u. a. Bd. 4761. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 339–347. ISBN: 978-3-540-75664-4. DOI: 10.1007/978-3-540-75664-4_34.

[BPvR08] Uwe Brinkschulte, Mathias Pacher und Alexander von Renteln. „An Artificial Hormone System for Self-Organizing Real-Time Task Allocation in Organic Middleware“. In: Rolf P. Würtz. *Organic Computing*. Bearb. von J.A. Scott Kelso u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, S. 261–283. ISBN: 978-3-540-77657-4. DOI: 10.1007/978-3-540-77657-4_12.

[Bri13] Uwe Brinkschulte. „Increasing the Stability of an Artificial Hormone System for Task Allocation by Accelerator Bounds“. In: *16th IEEE International Symposium on Object/Component/Service-Oriented Real-time Distributed Computing (ISORC 2013)*. 2013 IEEE 16th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC). Paderborn, Germany: IEEE, Juni 2013. ISBN: 978-1-4799-2111-9. DOI: 10.1109/ISORC.2013.6913238.

[Bri16] Uwe Brinkschulte. „Prototypic Implementation and Evaluation of an Artificial DNA for Self-Descripting and Self-Building Embedded Systems“. In: *2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*. 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC). York, United Kingdom: IEEE, Mai 2016, S. 10–18. ISBN: 978-1-4673-9032-3. DOI: 10.1109/ISORC.2016.12.

[BvRP08] Uwe Brinkschulte, Alexander von Renteln und Mathias Pacher. „Measuring the Quality of an Artificial Hormone System Based Task Mapping“. In: *Proceedings of the 2nd International ICST Conference on Autono-*

mic Computing and Communication Systems. 2nd International ICST Conference on Autonomic Computing and Communication Systems. Turin, Italy: ICST, 2008. ISBN: 978-963-9799-34-9. DOI: 10.4108/ICST.AUTONOMICS2008.4662.

[Cal01] Cristian Calude, Hrsg. *Multiset Processing: Mathematical, Computer Science, and Molecular Computing Points of View*. Lecture Notes in Computer Science 2235. Berlin ; London: Springer, 2001. 358 S. ISBN: 978-3-540-43063-6.

[Cha19] ChargerLAB. *Anker PowerPort Atom PD 2 with GaN (A2029) Teardown Review*. 9. Juli 2019. URL: https://www.chargerlab.com/anker-powerport-atom-pd-2-with-gan-a2029-teardown-review/ (besucht am 21. 02. 2020).

[Che+12] L. Chen u. a. „The Contract Net Based Task Allocation Algorithm for Wireless Sensor Network“. In: *2012 IEEE Symposium on Computers and Communications (ISCC)*. 2012 IEEE Symposium on Computers and Communications (ISCC). Juli 2012, S. 600–604. DOI: 10.1109/ISCC.2012.6249362.

[Cyp19] Cypress Semiconductor Corporation. *EZ-PD CCG4 USB Type-C Port Controller*. Datenblatt. 15. Nov. 2019. URL: https://www.cypress.com/file/220266/download (besucht am 21. 02. 2020).

[Das+11] G. P. Das u. a. „A Fast Distributed Auction and Consensus Process Using Parallel Task Allocation and Execution“. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. Sep. 2011, S. 4716–4721. DOI: 10.1109/IROS.2011.6094948.

[Das+15] Gautham P. Das u. a. „A Distributed Task Allocation Algorithm for a Multi-Robot System in Healthcare Facilities“. In: *Journal of Intelligent & Robotic Systems* 80.1 (1. Okt. 2015), S. 33–58. ISSN: 1573-0409. DOI: 10.1007/s10846-014-0154-2.

[Deua] Deutsche Forschungsgemeinschaft. *DFG - GEPRIS - SPP 1183: Organic Computing*. URL: https://gepris.dfg.de/gepris/projekt/5472210 (besucht am 20. 02. 2020).

[Deub] Deutsches Zentrum für Luft- und Raumfahrt. *DLR - Institute for Software Technology - ScOSA*. URL: https://www.dlr.de/sc/en/desktopdefault.aspx/tabid-11139/19481_read-45210/ (besucht am 16. 04. 2021).

[DP10] Waltenegus Dargie und Christian Poellabauer. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. 2010. ISBN: 978-0-470-99765-9.

[Dre+05] Falko Dressler u. a. „Self-Organization in Sensor Networks Using Bio-Inspired Mechanisms“. In: 18th International Conference on Architecture of Computing Systems, Workshops. Hrsg. von Uwe Brinkschulte u. a. Innsbruck, Austria: VDE Verlag, 2005, S. 139–144. ISBN: 978-3-8007-2880-0.

[Eda+09] Neda Edalat u. a. „A Price-based Adaptive Task Allocation for Wireless Sensor Network“. In: *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*. 2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems. Okt. 2009, S. 888–893. DOI: 10.1109/MOBHOC.2009.5337039.

[Eda+10] Neda Edalat u. a. „Distributed Winner Determination Protocol for Reverse Auction-based Task Allocation in Pervasive Computing“. In: *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops). März 2010, S. 780–783. DOI: 10.1109/PERCOMW.2010.5470540.

[ETX12] Neda Edalat, Chen-Khong Tham und Wendong Xiao. „An Auction-Based Strategy for Distributed Task Allocation in Wireless Sensor Networks“. In: *Computer Communications* 35.8 (1. Mai 2012), S. 916–928. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2012.02.004.

[FB15] Simon Fürst und Stefan Bunzel. „AUTOSAR“. In: *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner u. a. 3., überarbeitete und ergänzte Auflage. ATZ / MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2015, S. 105–122. ISBN: 978-3-658-05734-3.

[GJ11] K. P. Girish und Sunil Jacob John. „Rough Multiset and Its Multiset Topology“. In: *Transactions on Rough Sets XIV*. Hrsg. von James F. Peters u. a. Bd. 6600. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 62–80. ISBN: 978-3-642-21563-6. DOI: 10.1007/978-3-642-21563-6_4.

[GKP94] Ronald L. Graham, Donald E. Knuth und Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd edition, 1st printing. Reading, Mass: Addison-Wesley, 1994. 657 S. ISBN: 978-0-201-55802-9.

[Gum54] Emil Julius Gumbel. „Elementare Ableitung Der Momente Für Die Zahl Der Überschreitungen“. In: *Mitteilungsblatt für Mathematische Statistik* 6 (1954), S. 164–169.

[GvS50] E. J. Gumbel und H. von Schelling. „The Distribution of the Number of Exceedances“. In: *The Annals of Mathematical Statistics* 21.2 (Juni 1950), S. 247–262. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729842.

[HB20] Eric Hutter und Uwe Brinkschulte. „Towards a Priority-Based Task Distribution Strategy for an Artificial Hormone System“. In: *Architecture of Computing Systems – ARCS 2020*. Hrsg. von André Brinkmann u. a. Bd. 12155. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, S. 69–81. ISBN: 978-3-030-52794-5. DOI: 10.1007/978-3-030-52794-5_6.

[HB21a] Eric Hutter und Uwe Brinkschulte. „Evaluating a Priority-Based Task Distribution Strategy for an Artificial Hormone System“. In: *Architecture of Computing Systems – ARCS 2021*. Hrsg. von Christian Hochberger, Lars Bauer und Thilo Pionteck. Bd. 12800. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, S. 150–164. ISBN: 978-3-030-81682-7. DOI: 10.1007/978-3-030-81682-7_10.

[HB21b] Eric Hutter und Uwe Brinkschulte. „Handling Assignment Priorities to Degrade Systems in Self-Organizing Task Distribution“. In: *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*. 2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC). Daegu, South Korea: IEEE, Juni 2021, S. 132–140. DOI: 10.1109/ISORC52013.2021.00027.

[HBF20] Eric Hutter, Uwe Brinkschulte und Felix Fastnacht. „Adapting the Concept of Artificial DNA and Hormone System to AUTOSAR Environments“. In: *Journal of Systems Architecture* 104 (März 2020), S. 101700. ISSN: 13837621. DOI: 10.1016/j.sysarc.2019.101700.

[Hel20] Forrest Heller. *Apollo 11 vs USB-C Chargers*. 6. Feb. 2020. URL: https://forrestheller.com/Apollo-11-Computer-vs-USB-C-chargers.html (besucht am 21. 02. 2020).

[HPB20] Eric Hutter, Mathias Pacher und Uwe Brinkschulte. „Evaluation and Complexity Analysis of Task Dependencies in an Artificial Hormone System“. In: *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*. 2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC). Nashville, Tennessee, USA: IEEE, Mai 2020, S. 1–9. DOI: 10.1109/ISORC49007.2020.00011.

[Hun07] John D. Hunter. „Matplotlib: A 2D Graphics Environment“. In: *Computing in Science & Engineering* 9.3 (2007), S. 90–95. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.55.

[Hut21] Eric Hutter. „Degrading Systems by Priority-Based Self-Healing Using an Artificial Hormone System“. In: *Organic Computing: Doctoral Dissertation Colloquium 2020*. Hrsg. von Sven Tomforde und Christian Krupitzer. Intelligent Embedded Systems 18. Kassel: Kassel University Press GmbH, 2021, S. 3–15. ISBN: 978-3-7376-0945-6.

[Ibe15] Eishi H. Ibe. *Terrestrial Radiation Effects in ULSI Devices and Electronic Systems*. Singapore: John Wiley & Sons Inc, 2015. 1 S. ISBN: 978-1-118-47932-2.

[Int] Intel Corporation. *Intel Microprocessor Quick Reference Guide - Product Family*. URL: https://www.intel.com/pressroom/kits/quickreffam.htm (besucht am 02. 03. 2020).

[Int19] Intel Corporation. *Intel® Core™ i9-10980XE Prozessor Extreme Edition*. Intel. 24. Sep. 2019. URL: https://www.intel.com/content/www/de/de/products/processors/core/x-series/i9-10980xe.html (besucht am 28. 02. 2020).

[ISO18] ISO 26262:2018. *Road Vehicles – Functional Safety*. Standard. Geneva, CH: International Organization for Standardization, Dez. 2018.

[Jai+10] Pulkit Jain u. a. „Thermal and Power Delivery Challenges in 3D ICs“. In: *Three Dimensional Integrated Circuit Design: EDA, Design and Microarchitectures*. Hrsg. von Yuan Xie, Jason Cong und Sachin Sapatnekar. Integrated Circuits and Systems. Boston, MA: Springer US, 2010, S. 33–61. ISBN: 978-1-4419-0784-4.

[JKK05] Norman Lloyd Johnson, Adrienne W. Kemp und Samuel Kotz. *Univariate Discrete Distributions*. 3rd ed. Hoboken, N.J: Wiley, 2005. 646 S. ISBN: 978-0-471-27246-5.

[Jon96] Charles H. Jones. „Generalized Hockey Stick Identities and N-Dimensional Blockwalking“. In: *Fibonacci Quarterly* 34.3 (1996), S. 280–288.

[KC03] J.O. Kephart und D.M. Chess. „The Vision of Autonomic Computing“. In: *Computer* 36.1 (Jan. 2003), S. 41–50. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1160055.

[Kim+03] N.S. Kim u. a. „Leakage Current: Moore's Law Meets Static Power“. In: *Computer* 36.12 (Dez. 2003), S. 68–75. ISSN: 1558-0814. DOI: 10.1109/MC.2003.1250885.

[Knu] Donald Ervin Knuth. *Knuth: Concrete Mathematics – Errata*. URL: https://www-cs-faculty.stanford.edu/~knuth/gkperr1.html (besucht am 12. 03. 2020).

[Knu92] Donald E. Knuth. *Two Notes on Notation*. 30. Apr. 1992. arXiv: math/9205211. URL: http://arxiv.org/abs/math/9205211 (besucht am 11. 10. 2019).

[Knu97] Donald Ervin Knuth. *Volume 1: Fundamental Algorithms*. 3rd ed, 19th printing 2005. Bd. 1. The Art of Computer Programming. Reading, Mass: Addison-Wesley, 1997. ISBN: 978-0-201-89683-1.

[Kov+17] Andrii Kovalov u. a. „Task-Node Mapping in an Arbitrary Computer Network Using SMT Solver“. In: *Integrated Formal Methods*. International Conference on Integrated Formal Methods. Hrsg. von Nadia Polikarpova und Steve Schneider. Bd. 10510. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, S. 177–191. ISBN: 978-3-319-66845-1. DOI: 10.1007/978-3-319-66845-1_12.

[Kov+20] Andrii Kovalov u. a. „Model-Based Reconfiguration Planning for a Distributed On-board Computer“. In: *Proceedings of the 12th System Analysis and Modelling Conference*. SAM '20: 12th System Analysis and Modelling Conference. Virtual Event Canada: ACM, 19. Okt. 2020, S. 55–62. ISBN: 978-1-4503-8140-6. DOI: 10.1145/3419804.3420266.

[KR21] Bernhard Kleine und Winfried G Rossmanith. *Hormone und Hormonsystem - Lehrbuch der Endokrinologie*. 4. Aufl. Berlin, Heidelberg: Springer Spektrum, 2021. ISBN: 978-3-662-58501-6.

[KRD03] Mark Klein, Juan-Antonio Rodriguez-Aguilar und Chrysanthos Dellarocas. „Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death“. In: *Autonomous Agents and Multi-Agent Systems* 7.1 (1. Juli 2003), S. 179–189. ISSN: 1573-7454. DOI: 10.1023/A:1024145408578.

[Kum20] Hideyoshi Kume. *Tesla Teardown Finds Electronics 6 Years Ahead of Toyota and VW – Nikkei Asian Review*. Nikkei Asian Review. 17. Feb. 2020. URL: https://asia.nikkei.com/Business/Automobiles/Tesla-teardown-finds-electronics-6-years-ahead-of-Toyota-and-VW2 (besucht am 28. 02. 2020).

[LB18] Andreas Lund und Uwe Brinkschulte. „Task-Allocation in a Large Scaled Hierarchical Many-Core Topology“. In: *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC). Singapore: IEEE, Mai 2018, S. 1–8. ISBN: 978-1-5386-5847-5. DOI: 10.1109/ISORC.2018.00009.

[LMD13] Philippe Lalanda, Julie A. McCann und Ada Diaconescu. *Autonomic Computing: Principles, Design and Implementation*. New York: Springer, 2013. ISBN: 978-1-4471-5006-0.

[LT18] Jens Lienig und Matthias Thiele. *Fundamentals of Electromigration-Aware Integrated Circuit Design*. Cham: Springer, 2018. 159 S. ISBN: 978-3-319-73557-3.

[Lun19] Andreas Lund. „Hierarchical Self-Organizing Systems for Task-Allocation in Large Scaled Distributed Architectures“. Diss. Frankfurt am Main: Goethe-Universität, Juli 2019.

[LV06] G.M. Link und N. Vijaykrishnan. „Thermal Trends in Emerging Technologies“. In: *7th International Symposium on Quality Electronic Design (ISQED'06)*. 7th International Symposium on Quality Electronic Design (ISQED'06). März 2006, 8 pp.–632. DOI: 10.1109/ISQED.2006.136.

[Min08] David A. Mindell. *Digital Apollo: Human and Machine in Spaceflight*. Cambridge, MA: The MIT Press, 2008. 359 S. ISBN: 978-0-262-13497-2.

[MT17] Christian Müller-Schloer und Sven Tomforde. *Organic Computing – Technical Systems for Survival in the Real World*. Autonomic Systems. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-68476-5. DOI: 10.1007/978-3-319-68477-2.

[Mül11] Christian Müller-Schloer, Hrsg. *Organic Computing: A Paradigm Shift for Complex Systems*. Autonomic Systems. Basel: Birkhäuser, 2011. ISBN: 978-3-0348-0129-4.

[Orl+19] S. Orlov u. a. „Automatically Reconfigurable Actuator Control for Reliable Autonomous Driving Functions (AutoKonf)“. In: *10th International Munich Chassis Symposium 2019: Chassis.Tech Plus*. 2019. ISBN: 978-3-658-26435-2. DOI: 10.1007/978-3-658-26435-2_26.

[Osz+18] F. Oszwald u. a. „Dynamic Reconfiguration for Real-Time Automotive Embedded Systems in Fail-Operational Context“. In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). Mai 2018, S. 206–209. DOI: 10.1109/IPDPSW.2018.00039.

[Pac10] Mathias Pacher. „Selbstorganisierende Verteilung von zeitlich abhängigen Tasks in verteilten Systemen“. Diss. Frankfurt am Main: Goethe-Universität, Nov. 2010. URL: http://d-nb.info/1008807079.

[PB10] Mathias Pacher und Uwe Brinkschulte. „Real-Time Distribution of Time-Dependant Tasks in Heterogeneous Environments“. In: *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops. Carmona, Spain: IEEE, 2010, S. 21–28. ISBN: 978-1-4244-7218-5. DOI: 10.1109/ISORCW.2010.9.

[Sar57] K. Sarkadi. „On the Distribution of the Number of Exceedances“. In: *The Annals of Mathematical Statistics* 28.4 (Dez. 1957), S. 1021–1023. ISSN: 0003-4851. DOI: 10.1214/aoms/1177706804.

[Smi80] Reid G. Smith. „The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver“. In: *IEEE Transactions on Computers* C-29.12 (Dez. 1980), S. 1104–1113. DOI: 10.1109/TC.1980.1675516.

[TSM17] Sven Tomforde, Bernhard Sick und Christian Müller-Schloer. *Organic Computing in the Spotlight*. 27. Jan. 2017. arXiv: 1701.08125 [cs]. URL: http://arxiv.org/abs/1701.08125 (besucht am 09. 10. 2019).

[USB] USB Implementers Forum, Inc. *USB Charger (USB Power Delivery)*. URL: https://www.usb.org/usb-charger-pd (besucht am 21. 02. 2020).

[vRB10] Alexander von Renteln und Uwe Brinkschulte. „Implementing and Evaluating the AHS Organic Middleware - A First Approach“. In: *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. Carmona, Spain: IEEE, 2010, S. 163–169. ISBN: 978-1-4244-7083-9. DOI: 10.1109/ISORC.2010.26.

[vRBP11a] Alexander von Renteln, Uwe Brinkschulte und Mathias Pacher. „Introducing a Simplified Implementation of the AHS Organic Middleware“. In: *Proceedings of the 2011 Workshop on Organic Computing - OC '11*. ICAC '11: 8th International Conference on Autonomic Computing. Karlsruhe, Germany: ACM Press, 2011, S. 51. ISBN: 978-1-4503-0736-9. DOI: 10.1145/1998642.1998652.

[vRBP11b] Alexander von Renteln, Uwe Brinkschulte und Mathias Pacher. „The Artificial Hormone System—An Organic Middleware for Self-organising Real-Time Task Allocation“. In: *Organic Computing — A Paradigm Shift for Complex Systems*. Hrsg. von Christian Müller-Schloer, Hartmut Schmeck und Theo Ungerer. Basel: Springer Basel, 2011, S. 369–384. ISBN: 978-3-0348-0130-0. DOI: 10.1007/978-3-0348-0130-0_24.

[Win19] Ally Winning. *Number of Automotive ECUs Continues to Rise*. eeNews Automotive. 15. Mai 2019. URL: https://www.eenewsautomotive.com/news/number-automotive-ecus-continues-rise (besucht am 02. 03. 2020).

[Woo09] Michael J. Wooldridge. *An Introduction to MultiAgent Systems*. 2nd ed. Chichester, U.K: John Wiley & Sons, 2009. 461 S. ISBN: 978-0-470-51946-2.

[WSM20] Dominik Weikert, Christoph Steup und Sanaz Mostaghim. „Multi-Objective Task Allocation for Wireless Sensor Networks“. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2020 IEEE Symposium Series on Computational Intelligence (SSCI). Dez. 2020, S. 181–188. DOI: 10.1109/SSCI47803.2020.9308345.

[Wür08] Rolf P. Würtz, Hrsg. *Organic Computing*. Understanding Complex Systems. Berlin: Springer, 2008. ISBN: 978-3-540-77656-7.

[Yin+17] Xiang Yin u. a. „Cooperative Task Allocation in Heterogeneous Wireless Sensor Networks“. In: *International Journal of Distributed Sensor Networks* 13.10 (1. Okt. 2017). ISSN: 1550-1477. DOI: 10.1177/1550147717735747.

[ZWS19] Jiarui Zhang, Gang Wang und Yafei Song. „Task Assignment of the Improved Contract Net Protocol under a Multi-Agent System“. In: *Algorithms* 12.4 (Apr. 2019), S. 70. DOI: 10.3390/a12040070.