Future Internet **2014**, *6*, 518-541; doi:10.3390/fi6030518

OPEN ACCESS

future internet

ISSN 1999-5903

www.mdpi.com/journal/futureinternet

Article

ARCOMEM Crawling Architecture

Vassilis Plachouras [1,*], Florent Carpentier [2], Muhammad Faheem [3], Julien Masanès [2], Thomas Risse [4], Pierre Senellart [3], Patrick Siehndel [4] and Yannis Stavrakas [1]

[1] Institute for the Management of Information Systems, Athena Research and Innovation Center, Artemidos 6 & Epidavrou, Maroussi 15125, Greece; E-Mail: yannis@imis.athena-innovation.gr

[2] Internet Memory Foundation, 45 ter rue de la Révolution, 93100 Montreuil, France; E-Mails: florent.carpentier@internetmemory.net (F.C.); julien.masanes@internetmemory.org (J.M.)

[3] CNRS LTCI, Institut Mines-Télécom, Télécom ParisTech, 46 rue Barrault, 75634 Paris Cedex 13, France; E-Mails: muhammad.faheem@telecom-paristech.fr (M.F.); pierre@senellart.com (P.Se.)

[4] L3S Research Center, University of Hannover, Appelstr. 9a, 30167 Hannover, Germany; E-Mails: risse@L3S.de (T.R.); siehndel@L3S.de (P.Si.)

* Author to whom correspondence should be addressed; E-Mail: vplachouras@acm.org; Tel.: +30-210-687-5413; Fax: +30-210-685-6804.

Received: 15 April 2014; in revised form: 11 July 2014 / Accepted: 14 July 2014 / Published: 19 August 2014

Abstract: The World Wide Web is the largest information repository available today. However, this information is very volatile and Web archiving is essential to preserve it for the future. Existing approaches to Web archiving are based on simple definitions of the scope of Web pages to crawl and are limited to basic interactions with Web servers. The aim of the ARCOMEM project is to overcome these limitations and to provide flexible, adaptive and intelligent content acquisition, relying on social media to create topical Web archives. In this article, we focus on ARCOMEM's crawling architecture. We introduce the overall architecture and we describe its modules, such as the online analysis module, which computes a priority for the Web pages to be crawled, and the Application-Aware Helper which takes into account the type of Web sites and applications to extract structure from crawled content. We also describe a large-scale distributed crawler that has been developed, as well as the modifications we have implemented to adapt Heritrix, an open source crawler, to the needs of the project. Our experimental results from real crawls show that ARCOMEM's crawling architecture is effective in acquiring focused information about a topic and leveraging the information from social media.

Keywords: web archiving; crawling architecture; content acquisition

1. Introduction

The World Wide Web is the largest information repository. However, this information is very *volatile*: the typical half-life of content referenced by URLs is of a few years [1]; this trend is even aggravated in social media, where social networking APIs sometimes only extend to a week's worth of content [2]. Web archiving [3] deals with the collection, enrichment, curation, and preservation of today's volatile Web content in an archive that remains accessible to tomorrow's historians. Different strategies for Web archiving exist: bulk harvesting, selective harvesting and combinations of both. Bulk harvesting aims at capturing snapshots of entire domains. In contrast, selective harvesting is much more focused, e.g., on an event or a person. Combined strategies include less frequent domain snapshots complemented with regular selective crawls. In the following we will focus on the technical aspects of selective crawls.

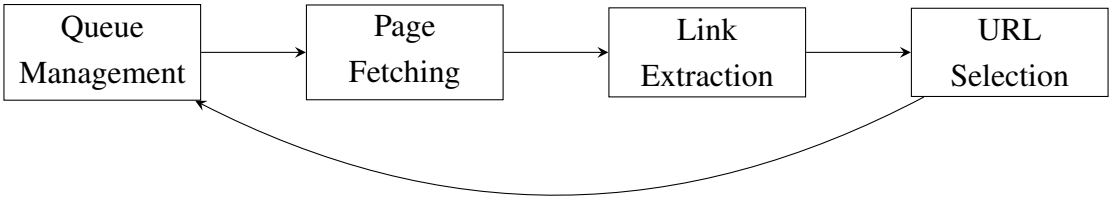
Selective crawls require a lot of manual work for the crawl preparation, crawler control, and quality assurance. On the technical level, current-day archiving crawlers, such as Internet Archive's Heritrix [4], crawl the Web in a conceptually simple manner (See Figure 1). They start from a *seed* list of URLs (typically provided by a Web archivist) to be stored in a queue. Web pages are then fetched from this queue one after the other, stored as is in the archive, and further links are extracted from them. If newly extracted links point to URLs that are in the scope of archiving tasks (usually given by a list or regular expressions of URLs to consider), they are added to the queue. This process ends after a specified time or when there is no interesting URL left to crawl. Due to this simple way of crawling, bulk domain crawls are well supported while selective crawls necessitate additional manual work for the preparation and quality assurance. It is the aim of the ARCOMEM project [5] to support the selective crawling on the technical level by leveraging social media and semantics to build meaningful Web archives [6]. This requires, in particular, a change of paradigm in how content is collected technically via Web crawling, which is the topic of the present article.

This traditional processing chain of a Web crawler like Heritrix [4] has several major limitations:

- Only regular Web pages, accessible through hyperlinks and downloadable with an HTTP GET request, are ever candidates for inclusion in the archive; this excludes other forms of valuable Web information, such as that accessible through Web forms, social networking RESTful APIs, or AJAX applications;
- Web pages are stored as is in the archive, and the granularity of the archive is that of a Web page. Modern Web applications, however, often present individual blocks of information on different parts of a Web page: think of the messages on a Web forum, or the different news items on a news site. These individual *Web objects* can be of independent interest to archive users;
- The crawling process does not vary from one site to another. The crawler is blind to the kind of Web application hosted by this Web site, or to the software (typically, a *content management system*) that powers this Web application. This behavior might lead to resource loss in crawling irrelevant information (e.g., login page, edition page in a wiki system) and prevents any optimization of the crawling strategy within a Web site based on how the Web site is structured;

- The scope of a selective crawl is defined by a crude whitelist and blacklist of URL patterns; there is no way to specify that relevant pages are those that are related to a given semantic entity (say, a person) or that are heavily referenced from influential users in social networks;
- The notion of scope is binary: either a Web page is in the scope or it is not—on the other hand, it is very natural for a Web archivist to consider various degrees of relevance for different pieces of Web content; and ideally content should be crawled by decreasing degree of relevance.

Figure 1. Traditional processing chain of a Web crawler.



The crawling architecture of ARCOMEM aims at solving these different issues by providing flexible, adaptive, intelligent content acquisition. This is achieved by interfacing traditional Web crawlers such as Heritrix with additional modules (complex resource fetching, Web-application-aware extraction and crawling, online and offline analysis of content, prioritization), as well as by adapting the internals of the crawlers when needed (typically for managing priorities of content relevance). The objective of this article is to present an overview of this crawling architecture, and of its performance (both in terms of efficiency and of quality of the archive obtained) on real-Web crawls. This article extends [7].

The remainder of this work is organized as follows. We first discuss in Section 2 the related work. Then we present in Section 3 a high-level view of the ARCOMEM architecture, before reviewing individual modules in Section 4. We present evaluation results that highlight the effectiveness of ARCOMEM's crawling architecture in Section 5. Finally, we present our concluding remarks in Section 6.

2. Related Work

While crawling appears to be a simple process, there are several associated challenges, especially when the aim is to crawl a large number of Web pages [8], in order to create the index of a Web search engine, or to archive them for future reference.

2.1. Web Crawling

Descriptions of early versions of Google's and Internet Archive's large-scale crawler systems appeared in [9,10], respectively. However, one of the first detailed descriptions of a scalable Web crawler is that of Mercator by Heydon and Najork [11], who provide information on the various modules of the crawler and the design options. Najork and Heydon also describe a distributed crawler based on Mercator in [12]. Shkapenyuk and Suel [13] introduce a distributed and robust crawler, managing the failure of individual servers. Heritrix [14] is an archival-quality and modular open source crawler, developed at

the Internet Archive. In Section 4.4 we will describe how we have adapted Heritrix in order to fit in ARCOMEM's crawling architecture. Boldi *et al.* [15] describe UBICrawler, a distributed Web crawler, implemented in Java, which operates in a decentralized way and uses consistent hashing to partition the domains to crawl across the crawling servers. Lee *et al.* [16] describe the architecture and main data structures of IRLBot, a crawler which implements DRUM (Disk Repository with Update Management) for checking whether a URL has been seen previously. The use of DRUM allows IRLBot to maintain a high crawling rate, even after crawling billions of Web pages.

As the Web evolves, and Web pages are created, modified, or deleted [17,18], effective crawling approaches are needed to handle these changes. Cho and Garcia Molina [19] describe an incremental crawler for optimizing the average freshness of crawled Web data. Olston and Pandey [20] describe re-crawling strategies to optimize freshness based on the longevity of information on Web pages. Pandey and Olston [21] also introduce a parameterized algorithm for monitoring Web resources for updates and optimizing timeliness or completeness depending on application-specific requirements.

2.2. Focused and Deep-Web Crawling

Focused or topical crawlers [22] provide an effective way to balance the cost, coverage, and quality aspects of data collection from the Web [23], by selectively crawling pages that are relevant to a set of topics, defined as a set of keywords [24], by example documents mapped to a taxonomy of topics [25], or by ontologies [26,27]. Recent approaches also address the crawling of information for specific geographical locations [28,29].

The main challenges in focused crawling relate to the prioritization of URLs not yet visited, which may be based on similarity measures [24,26], hyperlink distance-based limits [30,31], or combinations of text and hyperlink analysis with Latent Semantic Indexing (LSI) [32]. Machine learning approaches, including naïve Bayes classifiers [25,33], Hidden Markov Models [34], reinforcement learning [35], genetic algorithms [36], and neural networks [37], have also been applied to prioritize the unvisited URLs.

Focused crawlers and crawlers in general can harvest data from the publicly indexable Web by following hyperlinks between Web pages. However, there is a very large part of the Web that is hidden behind HTML forms [38]. Such forms are easy to complete by human users. Automatic deep-Web crawlers, however, need to complete HTML forms and retrieve results from the underlying databases. Barbosa and Freire [39] develop mechanisms for generating simple keyword queries that cover the underlying database through unstructured simple search forms. Madhavan *et al.* [40] handle structured forms by automatically completing subsets of fields, aiming to obtain small coverage over many hidden Web databases.

2.3. Web Archiving

Web archiving refers to the collection and long-term preservation of data available on the Web [3]. Since archiving the whole Web is a very challenging task due to its size and dynamics, there have been several national initiatives for preserving the Web of a country, based on full crawls in Sweden [41] and on a selective collection of Web pages in the United Kingdom [42] and Australia [43]. The former

approach aims at providing complete snapshots of a domain taken at regular intervals. A drawback of this approach is the lack of knowledge about changes of Web pages between crawls and the consistency of the collected data [44]. The latter approach results in higher quality collections restricted only to selected Web sites. Denev *et al.* [45] introduce a framework for assessing the quality of archives and tune the crawling strategies to optimize quality with given resources. Gomes *et al.* [46] provide a survey of Web archiving initiatives.

Focused crawlers, as described above, can be used for creating focused Web archives, by relying on a selective content acquisition approach. The crawling process in the ARCOMEM architecture changes the paradigm in how content is collected technically via Web crawling, by performing selective crawls and also leveraging information found in online social media.

3. Crawling Architecture

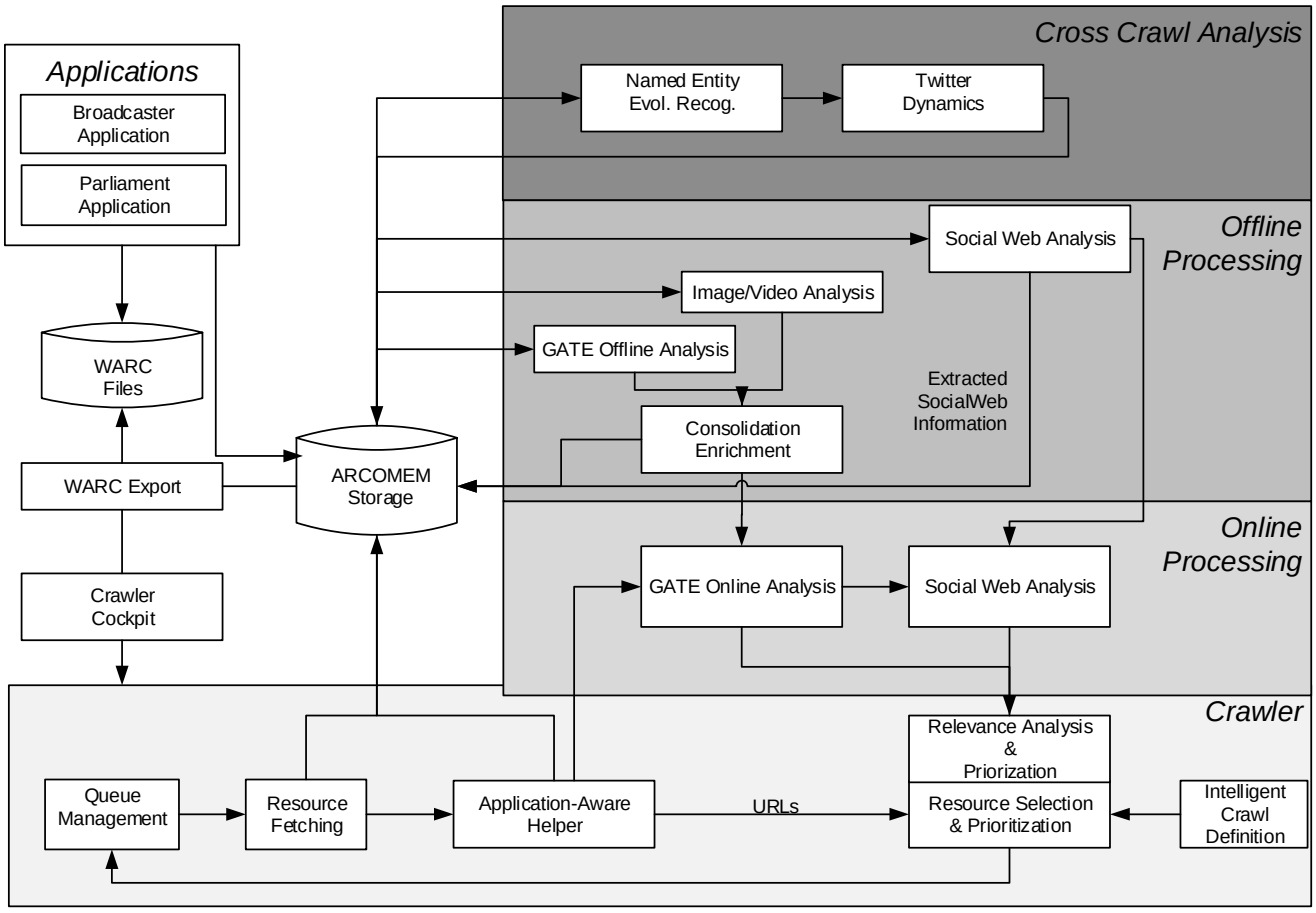
The goal for the development of the ARCOMEM crawler architecture was to implement a socially aware and semantic-driven preservation model [6]. This requires thorough analysis of the crawled Web page and its components. These components of a Web page are called *Web objects* and can be the title, a paragraph, an image, or a video. Since a thorough analysis of all Web objects is time-consuming, the traditional way of Web crawling and archiving is no longer functioning. Therefore the ARCOMEM crawl principle is to start with a *semantically enhanced crawl specification* that extends traditional URL-based seed lists with semantic information about entities, topics or events. This crawl specification is complemented by a small reference crawl to learn more about the crawl topic and intention of the archivist. The combination of the original crawl specification with the extracted information from the reference crawl is called the *Intelligent Crawl Definition* (ICD). This specification, together with relatively simple semantic and social signals, is used to guide a broad crawl that is followed by a thorough analysis of the crawled content. Based on this analysis a semi-automatic selection of the content for the final archive is carried out.

The translation of these steps into the ARCOMEM crawling architecture foresees the following processing levels: the *crawler level*, the *online processing level*, the *offline processing level*, and the *cross-crawl analysis* that revolve around the ARCOMEM database as depicted in Figure 2. Since the focus of this article is the crawling and the online analysis we will focus on these levels in the rest of the article and give only a brief overview on the other levels. More details about the other processing levels and the whole architecture can be found in [6].

3.1. Crawling Level

At this level, the system decides and fetches the relevant Web objects as these are initially defined by the archivists, and are later refined by both the archivists and the online processing modules. The crawling level includes, besides the traditional crawler and its decision modules, some important data cleaning, annotation, and extraction steps (we explain this in more detail in Section 4). The Web objects (*i.e.*, the important data objects existing in a page, excluding ads, code, *etc.*) are stored in the ARCOMEM database together with the raw downloaded content.

Figure 2. Overall architecture.



3.2. Online Processing Level

The online processing is tightly connected with the crawling level. At this level a number of semantic and social signals such as information about persons, locations, or social structure taken from the intelligent crawl specification are used to prioritize the crawler processing queue. Due to the near-real-time requirements, only time-efficient analysis can be performed, while complex analysis tasks are moved to the offline phase. The logical separation between the online processing level and the crawler level will allow the extension of existing crawlers at least with some functionalities of the ARCOMEM technology. More details about the online analysis can be found in Section 4.2.

3.3. Offline Processing Level

At this level, most of the basic processing over the data takes place. The offline, fully-featured, versions of the entity, topics, opinions, and events analysis (ETOE analysis) and the analysis of the social contents operate over the cleansed data from the crawl that are stored in the ARCOMEM database. These processing tools perform linguistic, machine learning and NLP methods in order to provide a rich set of metadata annotations that are interlinked with the original data. In addition to processing of textual content, multimedia content can also be analyzed and enriched with meta-information. The respective annotations are stored back in the ARCOMEM database and are available for further processing and

information mining. After all the relevant processing has taken place, the Web pages to be archived and preserved are selected in a semi-automatic way and transferred to the Web archive (in the form of WARC files).

3.4. Cross-Crawl Analysis Level

Finally, a more advanced processing step takes places. It operates on collections of Web objects that have been collected over time and can cover several crawls. Analysis implemented exemplary on this level within the ARCOMEM system is used to recognize Named Entity Evolutions [47] and to analyze the evolutions of associations between interesting terms and tweets (Twitter Dynamics) [48].

3.5. Applications

For the interaction with the crawler and exploration of the content a number of applications are used around the ARCOMEM core system. The crawler cockpit is used to create the crawl specification, to monitor the crawl activities, and to initiate the final export of crawled content to WARC files.

The end-user applications allow users to search archives by domain, time, and keywords. Furthermore, browsing the archives via different facets such as topics, events, and entities, and visualizing the sentiments of Social Web postings complement the end-user application. However, the applications are not limited to the described examples. The ARCOMEM system is open to any kind of application that wants to use it.

4. Module Description

The modular crawling architecture introduced in Section 3 enables the integration of a wide range of functionalities and technologies in the same system. In this section, we describe in detail the Application-Aware Helper, the online analysis module, as well as the two crawlers that can be used to acquire content.

4.1. Application-Aware Helper

The goal of the application-aware helper (AAH) is to make the crawler aware of the particular kind of Web application it is crawling, in order to adapt the crawling strategy accordingly. The presence of the AAH in the crawling processing chain ensures Web content is crawled in an intelligent and adaptive manner.

The AAH applies different crawling strategies for different types of social Web sites (Web forums, blogs, social networks, photo networks, music networks, video networks, *etc.*), or for specific content management system (vBulletin, WordPress, *etc.*). The AAH detects the Web application and Web page type within the Web application before deciding which crawling strategy is best for the given Web application.

More precisely, this module performs the following steps:

(1) it detects the Web application type (general type, content management system, *etc.*);

(2) it detects the Web page type (e.g., in a Web forum, if we are at the level of a list of forums, a list of threads, or a list of messages);
(3) it executes the relevant crawling actions; extracting Web objects (e.g., comments, posts) on the one hand, and adding only relevant URLs to the crawlers' queue on the other hand.

The AAH is assisted by a *knowledge base*, which specifies how to detect a specific Web application and which crawling actions should be applied. The knowledge base is written in a custom XML format, so as to be easily shared, updated, and hopefully managed by non-programmers. The knowledge base ensures that an appropriate crawling strategy is applied for a detected Web application. For instance, the vBulletin Web forum CMS can be identified by searching for a reference to a specific script with the detection pattern: script [contains(@src,"vbulletin_core.js")]. The AAH distinguishes two main kinds of Web application levels: intermediate pages, such as lists of forums, lists of threads, can only be associated with navigation actions; terminal pages, such as the individual posts in a forum thread, can be associated with both navigation and extraction actions. For intelligent crawling, our AAH needs not only to distinguish among Web application types, but among the different kinds of Web pages that can be produced by a given Web application type. For example, the expression //h2[@class="forumtitle"]/a/@href detects intermediate pages in vBulletin, whereas the expression //table[@class="post"] identifies terminal pages. Once the application type and level is detected, the system executes the relevant crawling actions. The crawling actions are of two types: extraction actions point to the individual Web objects to be extracted from a given Web page (e.g., comments, blog post); navigation actions point to the URLs to be added in a crawling queue. For instance, the extraction action //div[contains(@id,"post_message")] extracts the post message.

To exploit the AAH in Web-scale crawling, a Web application adaptation module has been integrated. The AAH deals with both Web content changes and Web structure changes. When Web content changes, the AAH simply updates recently crawled versions with the new one. However, the Web structure changes (*i.e.*, changes in Web site template) are more complicated to identify and adapt to. The AAH deals with this challenge. Structural changes with respect to the knowledge base come from varying versions of the CMS, or alternative templates proposed by CMSs or developed for a specific Web application. Here, we assume that Web application detection patterns never fail. In our experiments, we did not see any instance where a Web application was not successfully detected.

The AAH deals with two different cases of structure adaptation: first, when (part of) a Web application has been crawled before, but recrawl of Web application fails after template changes; second, when a new Web application has been detected successfully, but (some of) the existing actions are inapplicable. The adaptation module for recrawls of Web application relearns appropriate crawling actions for each failed crawlable object. In ARCOMEM, crawled Web pages with their Web objects and metadata are stored in the form of RDF triples in the ARCOMEM database. Therefore, we have proposed an algorithm which utilizes the ARCOMEM database and first detects structural changes for already crawled Web applications by looking for the crawled content in the Web pages with crawling actions used during a previous crawl. If the system fails to extract the content, then it means that the structure of the Web application has changed. In the presence of structural changes, the algorithm detects the inappropriate crawling actions and performs updates by aligning them according to structural changes.

In the case of a new Web application whose template is slightly different from the one present in the knowledge base, the adaptation module cannot be applied on previously crawled content. Here, the adaption is applied for two different scenarios: Web application level detected and Web application not detected. We consider two classes of Web application levels: intermediate and terminal. The navigation actions are applicable for the intermediate level (e.g., list of blog posts), whereas the terminal level may require both navigation and extraction actions (e.g., individual post). When a Web application level is detected, but (some) crawling actions fail, a set of relaxed expressions are generated by relaxing predicates, and tag names. The candidate tag names are selected from the knowledge base as well as from the existing page DOM tree. For example, if an expression div[contains(@class,"post")]//h2[@class="posttitle"] fails to extract the post title, and div[contains(@class,"post")] is the detection pattern that worked, then we will try several relaxations of the second half of the expression, for instance, replacing @class with @id, "posttitle" with "posthead", h2 with div, *etc*. We favor relaxations that use parts from crawling actions in the knowledge base for other Web application types of the same general category (e.g., bulletin boards). Any successful relaxed expression will be still tested with a few more pages of the same Web application level.

When a Web application level is not detected then an appropriate crawling strategy cannot be initiated, therefore the system first adapts the detection patterns. The idea here is the same as above: the system first collects all the candidate attributes, tag names and values; and then creates all possible combinations of relaxed expressions. For example, assume that the candidate set of attributes and values are: @class="post", @id=forumlist, @class="bloglist" with candidate set of tag names article, div, *etc*. The set of relaxed expression will be generated by trying out each possible combination:

- //div[contains(@class,"post")]
- //div[contains(@id,"forumlist")]
- //div[contains(@class,"bloglist")]

and similarly for other tag names. If the system detects the Web application with any relaxed expression then the system will apply the crawling actions adaptation as described above.

The AAH has reduced bandwidth, time, and storage (by requiring fewer HTTP requests for known Web applications, avoiding duplicates) using limited computational resources in the process. Application-aware crawling also helps adding semantic information to the ARCOMEM database. More details about the functioning and independent evaluation of the AAH are provided in [49,50].

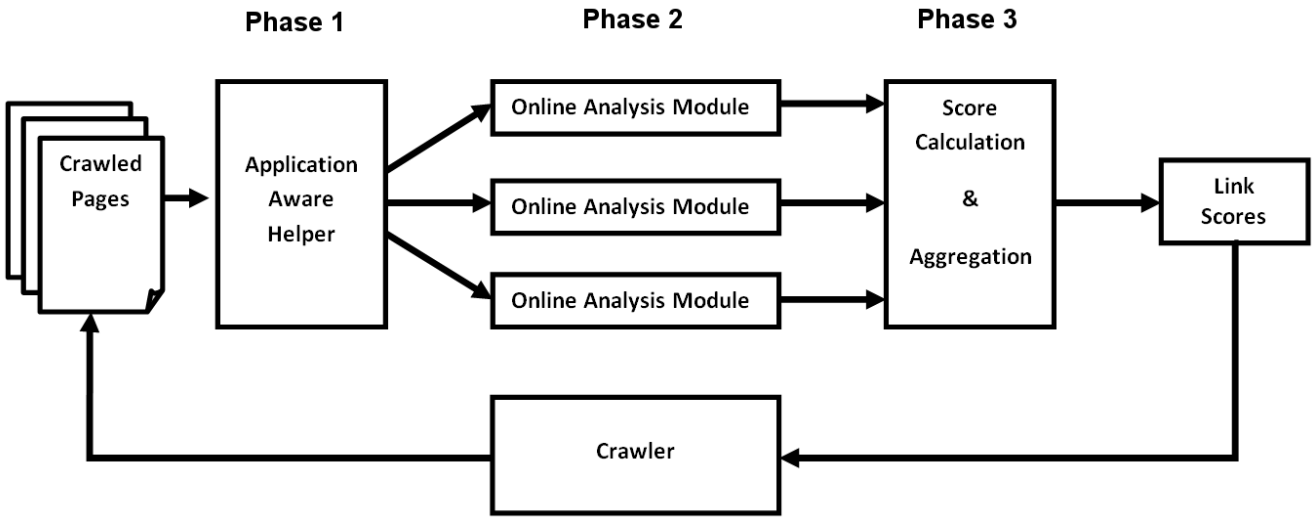
4.2. Online Analysis

Within the online analysis, several modules analyze crawled Web objects in order to guide the crawler. The purpose of this process is to provide scores for detected URLs. These scores are used for guiding the crawler in order to obtain a focused crawl with respect to the intelligent crawl definition (ICD). The main modules used within the online analysis are the AAH, the GATE platform for text analysis [51], and a prioritization module. The actual online processing consists of three phases which are displayed in Figure 3; within Figure 2 these phases are related to the connections between the online processing and the crawler.

(1) The AAH performs preprocessing steps on the crawled Web pages;

(2) Online analysis modules run on relevant document parts;
(3) The output of online analysis modules is aggregated and a score for each URL is provided.

Figure 3. Interaction of online analysis modules.



A more detailed description of the these steps is provided in the remainder of this section.

In the first phase we run the AAH on the Web page to detect regions of interest in the document and discard irrelevant parts. A detailed description of the functions provided by the AAH is given in Section 4.1. The input document is split into one or more document parts. Each document part is processed separately from now on.

In the second phase the online analysis modules are run on the content of the document part. Currently we use textual analysis modules using GATE, a URL scoring module using URL patterns and a simple spam link filter using a black list. Additional modules can be added easily. The textual analysis module performs basic NLP pre-processing on the text and allows the extraction of relevant entities.

The version of GATE used within the Online Analysis is a lightweight version of GATE since the performance and the processing time needs to be as fast as possible. The tasks carried out by the GATE component comprise basic linguistic processing steps, language identification and Named Entity Recognition (NER). In contrast to the use of the basic GATE functions which are needed to create word vectors describing the crawled objects, the use of the NER module is optional. With the NER module disabled the processing time of a Web document was reduced by about 70%. Based on these observations we disabled the NER module in the online phase for most crawls and moved this part to the offline analysis, where performance aspects are not as critical as during the online phase. Using the extracted keywords and the given crawl specification we calculate a score based on the cosine similarity of the term vectors. The matching is run at several granularities: whole document, paragraph around anchor, and only anchor text. This allows us to boost link anchors that are closer to keyword or entity matches.

Each analysis module can produce a score for the current document and one for each out-link. Some analysis modules (e.g., the URL analyzers) omit the document score, while others can only provide document scores (e.g., the text analysis). In the latter case the document score is propagated to each out-link contained in the analyzed document.

The final phase of the online analysis is the priority aggregation: The scores provided by the individual analysis modules are aggregated into one final score for each out-link. Here we use a weighted average over the individual scores using weights provided by the users.

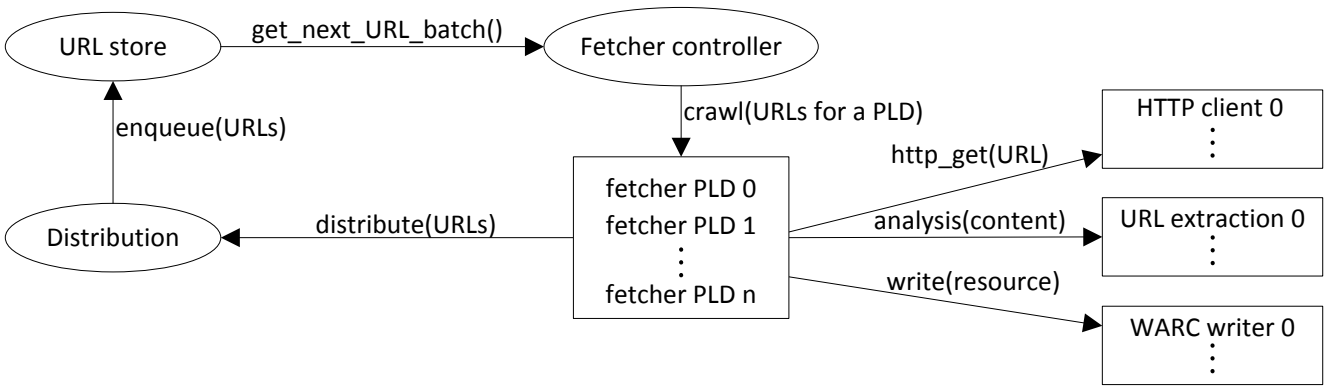
4.3. Large-Scale Crawler

The large-scale crawler is a distributed crawler, implemented by the Internet Memory Foundation (IMF). It retrieves content from the Web and stores it in WARC files, that can optionally be loaded into an HBase repository. Its main initial aim was scalability: crawling at a fast rate from the start and slowing down as little as possible as the amount of visited URLs grows to billions, all while observing politeness conventions (rate regulation, robots.txt compliance, *etc.*). This objective is achieved by incorporating recent developments in data structures and design options for crawlers [15,16]. It ran a crawl on 19 virtual machines with 8 cores and 32 GiB of RAM each for three weeks. The rate was kept over 2000 HTTP requests per second for the whole duration of the crawl for a total of close to four billion URLs crawled.

The crawler does not require distributing a static node list to all cluster instances nor does it require external utilities to copy lists of URLs as they get discovered. It also detects nodes joining or leaving the cluster and changes the URL distribution mapping to account changes without any manual intervention.

Figure 4 depicts the main processes on each cluster node. The rectangles depict many processes with the same function. The ovals represent individual processes or subsystems made of many processes.

Figure 4. Crawler architecture.



The fetcher controller is in charge of spawning fetchers, limited to spawning as many as is allowed by the configuration (mainly to respect memory constraints) and by the availability of URLs in the URL store. It asks the URL store for a batch of URLs all belonging to the same pay level domain (PLD, approximated by searching for the longest applicable public suffix and adding one more level), resolves the domain name to an IP address and ensures no other fetcher in the entire cluster is crawling this IP address. It then spawns a fetcher and passes it the URL batch. The fetcher gets the robots.txt file and starts crawling all the allowed URLs, respecting the required delay between each fetch.

For each resource, three main steps are performed:

- Fetching (HTTP request);
- Analyzing the document according to its type in search of new URLs. It may also run other analyses which may be useful at run time; for example, language identification;

- Writing the content plus extracted or derived information into a WARC file (depending on the configuration and filtering settings);
- Filtering according to the scope configuration before sending to the distribution module.

When a fetcher has processed all of its URLs, it exits and the fetcher controller will try to replace it with a new fetcher and a fresh batch of URLs.

The distribution module maintains a consistent hashing ring that reflects the current cluster topology. It forwards URLs to the appropriate node for them to be queued in the local URL store.

The URL distribution being based on the pay level domain, it is easy to guarantee that no more than one fetcher in the whole cluster will be crawling a specific host at any time. However, there is no guarantee that different pay level domains are not mapped to the same IP address. To ensure rate control for IP addresses, we use a global IP address registry.

The WARC files get copied asynchronously to a specific directory in a Hadoop file system (HDFS). A periodic import task will insert the content from the HDFS into HBase. This makes the crawler quite independent from the storage system. In particular, the crawler can continue to work without HBase for as long as it has available disk space.

To follow the numerous events occurring inside the crawler as tens of thousands of concurrent processes run, a flexible system is necessary. We have implemented node-local filtering of events by subsystem and severity, and centralized storage in a full text index that allows complex queries and advanced graphical representations.

The IMF crawler can perform multiple crawls concurrently, supporting one URL store and a configuration (scope functions, archival functions, *etc.*) for each concurrent crawl, while having a single fetcher pool. This feature guarantees that politeness is respected across all crawls while allowing to crawl concurrently as many domains as possible.

The latest developments are geared towards greater flexibility to add ease-of-use and "archive quality" to the crawler's scalability.

The large-scale crawler supports HTTPS, can stream large files, retries on server failures, detects the real MIME type and language of documents, extracts many metadata from HTML pages (such as outlinks with type, anchor text, *etc.*). It has a fast C implementation of a comprehensive and configurable URL canonicalization. It provides advanced scoping functions that can be combined at will, allowing, for instance, to make decisions based on the language of a page or the whole path that led to it. It also employs a fully-fledged and extensible per-domain configuration framework with parameters including budget, minimum and maximum delay between two fetches. Crawler fetchers subscribe to updates of parameter values and use the new configuration immediately. It detects traps by analyzing URLs and checking for similar content.

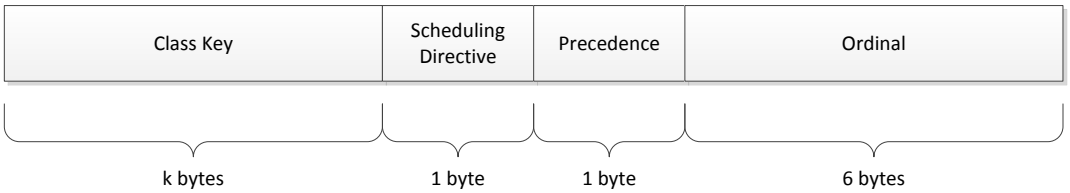
4.4. Adaptive Heritrix

In addition to the large-scale crawler developed by IMF, we have also investigated to what extent Heritrix, a widely-used open source crawler [14], can be adapted to ARCOMEM's crawling architecture. Heritrix implements a typical centralized crawling process, where a URL is prioritized only when it is added to the frontier. In order to adapt Heritrix to the needs of ARCOMEM, we have implemented

a frontier that supports updating the priorities of already scheduled URLs and receiving scored URLs from external processes, possibly running on different servers. As a result, Heritrix can be used as a fetching service for selective Web harvesting. Overall, we have extended Heritrix with a range of functionalities, regarding the storing of crawled content, the extraction of anchor text with links, *etc*. All the modifications are available as open source software in the releases of the ARCOMEM project. In this article we focus on the two main features mentioned above.

The default frontier of Heritrix employs a Berkeley DB backed hash table for storing URLs, typically grouped according to the domain or host they belong to. The key of a URL's record in the frontier is computed based on its domain, a flag indicating whether the URL should be crawled immediately, its priority (or precedence in Heritrix terminology), and a counter, which increases for every URL that is inserted in the frontier (Figure 5). The frontier implementation provides a next method for obtaining the next URL to crawl from a given domain or host, but there is no method to update the priority of an already scheduled URL.

Figure 5. The structure of the key corresponding to an entry of a URL scheduled for crawling in the default frontier of Heritrix.



To overcome this limitation, we have implemented a frontier that extends the default frontier of Heritrix, adding a hash table that maps a URL already scheduled to the key with which it was scheduled. When we need to update the priority of a URL already scheduled, we use this hash table to locate the corresponding record from the frontier, update its priority, recalculate a key and insert it in the frontier data structure at a new position. The fact that Heritrix employs an increasing counter to calculate the key for each URL ensures that there are no collisions.

Figure 6 shows as an example the state of the frontier during a crawl for a domain example.com, where there are three URLs queued for crawling. The first one is flagged for immediate downloading and has a priority of 64, while the other two URLs have equal priority of 48. Upon an update of the priority of the URL http://example.com/copyright.html from 48 to 32, the order of the two URLs is reversed, as shown in Figure 7.

Figure 6. Example of Adaptive Heritrix frontier data structures and URL index.

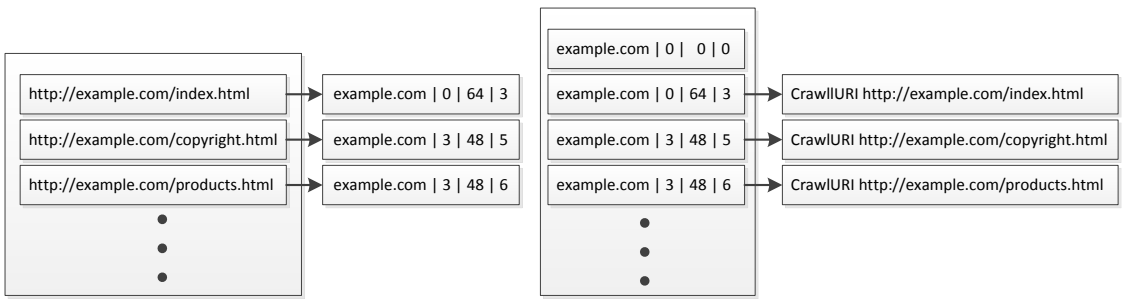
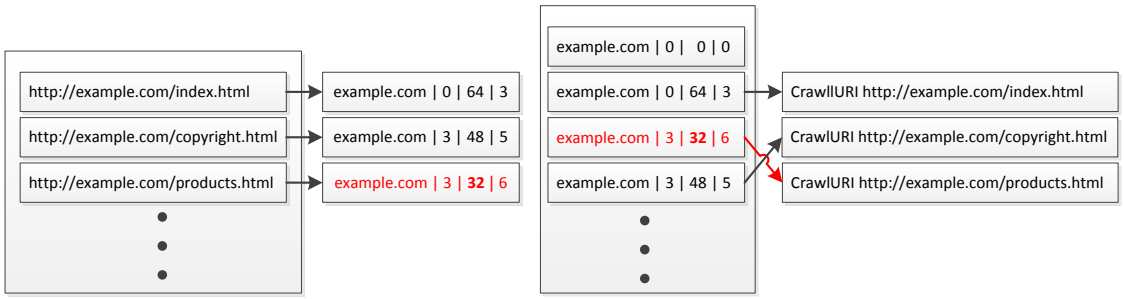


Figure 7. Example of updating the priority of a URL in the frontier of Adaptive Heritrix.



In the frontier we have developed, when a URL u is scheduled for crawling, first we have to check whether the hash table mapping URLs to entries in the frontier contains an entry for u. If u is found, then we update the priority, otherwise we need to check whether u has already been crawled.

The second feature we discuss enables Heritrix to receive prioritized URLs from other processes. Heritrix provides an action directory, where processes having access to the same filesystem can write files with seeds or URLs to be crawled. In order to fit Heritrix into the ARCOMEM crawling architecture and receive URLs with priority scores from the online analysis phase, we have implemented a Web service which receives scored URLs in an ARCOMEM-specific JSON format. In the simplest case, the required information is an identifier for the crawl, the URL, a score in the range $[0, 1]$, and optionally a flag indicating whether this URL should be blacklisted, *i.e.*, not crawled at all. The developed Web service enables Heritrix to receive links from any external process or even from other instances of Heritrix, facilitating the distributed operation of the crawler. The URL score is transformed from the range $[0, 1]$ to an integer, as expected by Heritrix.

5. Evaluation

Since the ARCOMEM crawling architecture departs from the standard crawling architectures [8], it is important to evaluate its impact on the effectiveness of a crawler. In this section we first evaluate how adaptive and batch prioritization affects the performance of a crawler based on a set of simulation experiments. Next, we compare a crawl performed by using the ARCOMEM crawling architecture to a crawl performed by using the standard Heritrix crawler.

5.1. Adaptive and Batch Prioritization

We assume a baseline crawler implementing a best-first crawling strategy. We represent the topic of the crawl with a topic vector, which is defined as follows. For each seed Web page, we download its content and we create a term vector from it. The topic vector corresponds to the vector sum of the seed page term vectors. We assume that URLs are prioritized according to their similarity to the topic vector. More specifically, the priority of a URL u is computed as the average of: (a) the cosine similarity between the content of Web page p in which u was found and the topic vector and; (b) the cosine similarity between the anchor text of the out-link from p to u and the topic vector.

An adaptive crawler can update the score of an already scheduled Web page using a function such as MAX, SUM, AVG. For example, the function MAX updates the priority of an already scheduled

Web page if the new priority was higher than the existing one. The function LAST always updates the score to the most recently computed one and the function FIRST is equivalent to the baseline crawler.

A crawler supporting batch prioritization schedules links for crawling only after having downloaded a batch of Web pages. In such a case, a URL can be discovered in many Web pages, so the cosine similarities are computed between the topic vector and the sum of the vectors of Web pages in which the URL was found, or the sum of the anchor text vectors associated with links pointing to the URL. In this setting, we also simulate a crawler that fetches the k pages with the highest priority from each domain, instead of fetching just one Web page with the highest priority.

To evaluate the focused crawler architectures, we perform simulated crawls on datasets created with three topics of DMOZ. We create three random samples of 20 seeds for each of the topics and the results we obtain for each configuration are the average of 9 simulations. For each set of seeds, we simulate a crawl of 10,000 Web pages. The topic vector we use to compute similarities between each topic and the crawled Web pages corresponds to the vector sum of the seed term vectors. For the evaluation of the results, we employ three measures: (a) harvest ratio, which we define as the ratio of Web pages whose cosine similarity with the topic vector is greater than 0.333 over all crawled Web pages; (b) average similarity of crawled pages; and (c) fraction of DMOZ subtopics with at least one crawled page.

Table 1 shows the evaluation results for adaptive prioritization with different priority update functions. The highest harvest ratio is achieved with the AVG function, while LAST achieves the highest fraction of DMOZ subtopics.

Table 1. Results of simulated adaptive crawls.

Update Function	Harvest Ratio	Average Similarity	DMOZ Topics
FIRST	0.3317	0.2945	0.4979
AVG	**0.3609**	**0.3024**	0.5779
MAX	0.3388	0.2967	0.5270
SUM	0.2679	0.2759	0.4650
LAST	0.3404	0.2961	**0.5985**

We also consider an additional parameter related to how the simulated crawler schedules links to crawl. For each queue, which corresponds to a domain, the crawler selects the bl URLs with the highest priority to crawl. In all previous experiments, $bl = 1$, meaning that each time the crawler selects the URL with the highest priority. For efficiency reasons, crawlers use higher values for the parameter bl. In the case of Heritrix, bl corresponds to the parameter *balance per queue*.

In our experiments, we test the values $bl = 1$ and $bl = 5$. The evaluation results for batch priority updating are shown in Table 2. We can observe that when increasing the value of bl, the effectiveness of the crawler drops, both in the case of a baseline crawler and an adaptive crawler (first 4 rows in Table 2). However, the combination of a higher bl value with batch updating performs better than an adaptive crawler with $bl = 5$ (rows 4 and 6 in Table 2).

Table 2. Results from simulated crawls with and without batch priority updating.

Batch	*bl*	Update Function	Harvest Ratio	Average Similarity	DMOZ Topics
No	1	FIRST	0.3317	0.2945	0.4979
No	5	FIRST	0.2948	0.2819	0.4677
No	1	AVG	0.3609	0.3024	0.5779
No	5	AVG	0.3200	0.2897	0.5420
Yes	1	AVG	0.3556	0.3013	0.5260
Yes	5	AVG	0.3347	0.2952	0.5176

5.2. Comparison of ARCOMEM Versus Standard Crawl

For evaluating the quality of the crawls in terms of how focused the collected documents are, we conducted a series of experiments. The main task of these experiments is to give an overview of the number of the crawled documents match the keywords given by the ICD, and how similar the textual content of the crawled documents is compared to the seed documents. Since there is no ground truth for the actual relevance of a document we have investigated some alternatives on how to measure the relevance of a document with respect to the given crawl definition. The dataset used for the experiment consists of two different crawls in the financial domain. One crawl was performed using the described ARCOMEM architecture while the other crawl is a standard Heritrix crawl.

The results we are presenting in this section are based on the document score generated by the Solr scoring module [52]. This score is computed as follows. For each crawl, we create an inverted index of the crawled documents. Next, we form a query with the keywords provided by the ICD, or the most representative keywords from the textual content of the seed documents. The idea of these settings is to find documents with similar textual content to the ICD keywords or to the content of the seed documents. We assume that the documents matching the query are relevant and that their relevance is represented by their similarity to the query, or in other words, their retrieval status value (RSV).

Since the standard Solr scoring is based on TF/IDF, the whole document collection is taken into account for normalization. One drawback of this method may arise if all analyzed documents contain the relevant keywords. The IDF score which is used for normalization will be relatively low if many documents of the collection contain these words, and due to this, relevant documents may get a lower similarity score. In order to deal with this factor we checked how many of the documents contained our keywords. The results are shown in Table 3. The ARCOMEM crawl consist of 234,749 documents and the Heritrix Crawl contained 366,806 documents. In order to make both crawls comparable we analyzed the percentage of documents containing keywords from the ICD. The results show that most of the keywords do not appear inside most documents which allows us to use a TF/IDF based scoring approach. It becomes also visible that the percentage of occurrences of the keywords from the ICD in the crawled documents is in nearly all cases higher when crawling with the ARCOMEM framework than when crawling with Heritrix.

Table 3. Percentage of crawled documents containing keywords from ICD.

Keyword	ARCOMEM	Heritrix
Antonis Samaras	0.0175	0.0052
European Union	0.6011	0.4346
EU	1.7870	1.8986
financial crisis	0.8443	0.3645
Eurozone	0.3157	0.1238
bailouts	0.0826	0.0393
austerity	0.3838	0.1638
measures	1.4880	0.7993
strategy	1.9123	0.9362
growth	3.9319	1.9752
public	6.7357	3.5990
investments	1.3406	0.5545

The results of the experiments to assess the quality of the ARCOMEM and Heritrix crawls are shown in Tables 4 and 5, respectively. We compared four different settings for describing what makes a document relevant. These settings differ in how the terms are selected and the number of terms that are used for calculating the similarity score of crawled documents to the topic of the crawl. For the experiment in the first row (ICD), we use the terms from the ICD. For the experiments in the last three rows, we choose the terms based on their TF/IDF value for the seed documents; maxQt = 10 indicates that the 10 most representative keywords are taken into account, which gives a very narrow definition of relevance. Additionally we used 50 and 100 words giving us a broader definition of relevance, since more words are considered to be important.

Table 4. Statistics on ARCOMEM Crawl.

Ground Truth	Relevant Documents (%)	Average Similarity	Maximum Similarity	Standard Deviation
ICD	14.29	0.0178	0.8440	0.0426
Seeds maxQt = 10	10.81	0.0179	0.4830	0.0263
Seeds maxQt = 50	44.06	0.0049	0.4299	0.0149
Seeds maxQt = 100	48.63	0.0060	0.5279	0.0172

Table 5. Statistics on Heritrix Crawl.

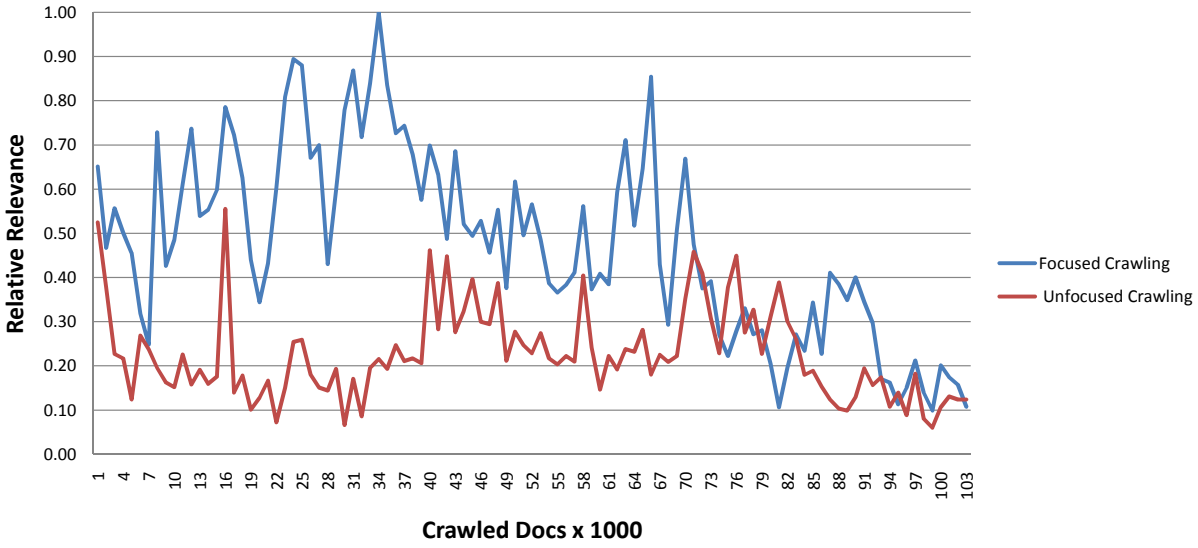
Ground Truth	Relevant Documents (%)	Average Similarity	Maximum Similarity	Standard Deviation
ICD	7.61	0.0196	0.7580	0.0440
Seeds maxQt = 10	6.14	0.0175	0.2957	0.0264
Seeds maxQt = 50	56.26	0.0026	0.3401	0.0114
Seeds maxQt = 100	59.13	0.0021	0.3122	0.0099

When comparing the results of the two crawls we see some obvious differences. The maximum similarity score per document was always the highest inside the ARCOMEM crawl, no matter which

ground truth was chosen. For the first two setups, with a limited number of relevant keywords, the percentage of relevant documents was much higher inside the ARCOMEM crawl; when many different keywords are taken into account this changes. In contrast to that, the average similarity of the documents was higher for the Heritrix crawl when only few keywords are considered. This can be explained by looking at the absolute number of relevant documents for the different setups. The focused crawler found many more documents related to the small set of keywords. As a result, the average relevance has dropped, since this is calculated based only on the relevant documents.

Beside the results describing the relevance of all the crawled documents, we also analyzed how the relevance evolves over time. Figure 8 shows how the relevance of the crawled documents evolves over time. The relative relevance is calculated using the average similarity over 1000 crawled documents and dividing this value by the maximum of the averages.

Figure 8. Focused (ARCOMEM) and unfocused (Heritrix) crawling over time.



We see that the relevance of the crawled documents exhibits large fluctuations during the crawl. While the Heritrix crawl does not show a certain tendency over the crawl, we see that the relative relevance of the documents of the ARCOMEM crawl increases up to the maximum after around 34,000 documents and then drops. Overall the relevance of the ARCOMEM based crawl was higher for most of the time.

Since one of the ideas of the ARCOMEM architecture is to use social networks to get additional content for the crawl we also used the described methods for analyzing the quality of links posted within Twitter. For this experiment we collected a total of 14,703 tweets related to our topic, out of which 7677 contained at least one URL. The content of these URLs was crawled and indexed in the same way as we did it with the standard Heritrix and ARCOMEM crawl. Overall we performed this experiment on a set of 2.2 million crawled documents. The overall number of documents was much larger, but for this experiment we only took documents with textual content into account. Table 6 shows some basic statistics on the crawl.

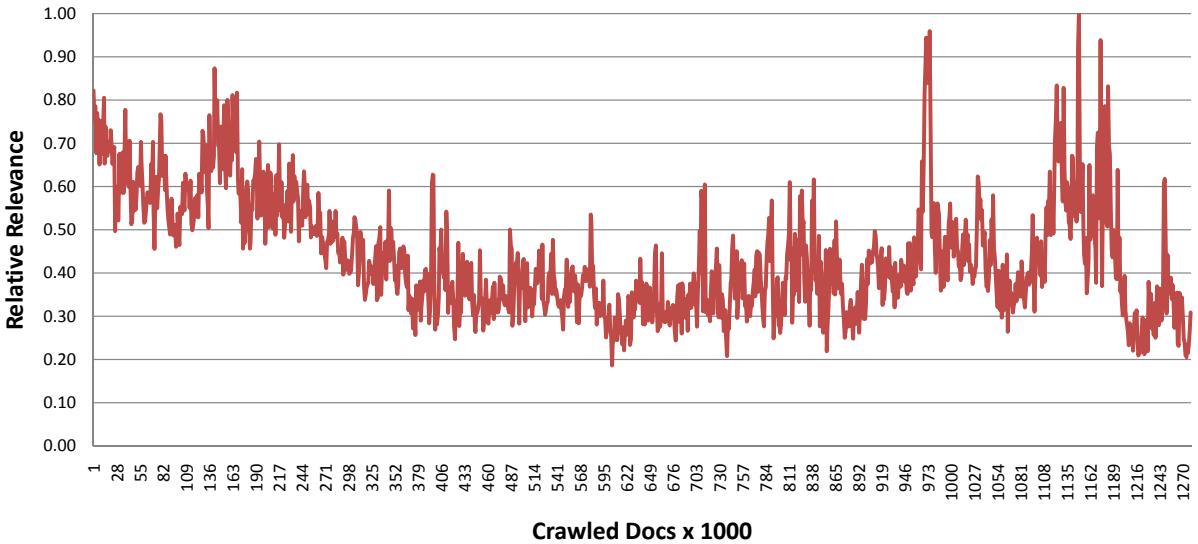
Table 6. Statistics on Twitter based crawl.

Ground Truth	Relevant Documents (%)	Average Similarity	Maximum Similarity	Standard Deviation
ICD	8.16	0.2435	2.1465	0.2299
Seeds maxQt = 10	19.87	0.0348	1.7057	0.0805
Seeds maxQt = 50	58.51	0.0180	1.5738	0.0489
Seeds maxQt = 100	66.67	0.0193	1.6462	0.0493

We can see that the numbers for the percentage of relevant documents are comparable to the Heritrix and ARCOMEM crawls, except the number of relevant documents for the top 10 words from the seeds are much higher with 19%. The numbers for the average relevance and maximum relevance are not directly comparable with the results from the two other crawls because this crawl was stored in a different index and due to that the TF/IDF values are calculated based on different corpora. Nevertheless we can still see some parallels between the crawls: the average relevance is still the highest for documents related to the ICD and relative order of the other values is also comparable to the previous crawls.

Figure 9 shows the evolution of the relative relevance of the crawled documents based on the 50 most representative terms from the seed set. Compared to the two previous crawls we cannot see the strong tendency of a dropping relevance over time.

Figure 9. Twitter based crawl over time.



6. Conclusions

The scale of the Web, the volatility of information found in it, as well as the emergence of social media, require a shift in the way Web archiving is performed. Towards this goal, the ARCOMEM project has developed a scalable and effective framework that allows archivists to leverage social media and guide crawlers to collect both relevant and important information for preservation and future reference.

In this article, we have presented ARCOMEM's crawling architecture, providing a detailed description of its main modules for extracting structured information from Web applications and prioritizing the URLs to be crawled. We have also outlined the main features of a large-scale distributed

crawler, which can collect the content from billions of URLs while maintaining a high download rate. The architecture we have described enables us to use either the large-scale crawler or an enhanced version of Heritrix, for which we have described the required modifications we have implemented to support the adaptive prioritization of URLs and the scheduling of URLs from remote processes.

Our experimental results show that the adaptive and batch prioritization, which are employed in the proposed crawling architecture, are effective in acquiring relevant content. When comparing the quality of a crawl performed with the ARCOMEM architecture against a crawl performed with Heritrix, we have seen that the ARCOMEM crawler has downloaded earlier more relevant content. Overall, the proposed crawling architecture is both extensible, by adding new modules to expand the analysis, and scalable, offering a new approach to crawling content for Web archiving.

Acknowledgments

This work was funded by the European Commission under grant agreement No. 270239 (ARCOMEM).

Author Contributions

P.Se. has contributed to Section 1. V.P. has contributed to Section 2. T.R. has contributed to Section 3. M.F. and P.Se. have contributed to Section 4.1. P.Si. has contributed to Section 4.2. J.M. and F.C. have contributed to Section 4.3. Y.S. and V.P. have contributed to Section 4.4. Finally, P.Si. and V.P. have contributed to Section 5.

Conflicts of Interest

Thomas Risse is co-editor of the special issue on Archiving Community Memories.

References

1. Koehler, W. A longitudinal study of Web pages continued: A consideration of document persistence. *Inf. Res.* **2004**, 9, 174. Available online: http://www.informationr.net/ir/9-2/paper174.html (accessed on 10 April 2014).
2. Historical Data Not Working. Available online: https://dev.twitter.com/discussions/2483 (accessed on 10 April 2014).
3. Masanès, J. *Web Archiving*; Springer-Verlag: Secaucus, NJ, USA, 2006.
4. Sigurðsson, K. Incremental crawling with Heritrix. In Proceedings of the 5th International Web Archiving Workshop (IWAW'05), Vienna, Austria, 22–23 September 2005.
5. ARCOMEM: Archiving Communities Memories. Available online: http://www.arcomem.eu/ (accessed on 10 April 2014).
6. Risse, T.; Dietze, S.; Peters, W.; Doka, K.; Stavrakas, Y.; Senellart, P. Exploiting the Social and Semantic Web for Guided Web Archiving. In *Theory and Practice of Digital Libraries*; Zaphiris, P., Buchanan, G., Rasmussen, E., Loizides, F., Eds.; Springer: Berlin/Heidelberg, Germany 2012; Volume 7489, pp. 426–432.

7. Plachouras, V.; Carpentier, F.; Masanés, J.; Risse, T.; Senellart, P.; Siehndel, P.; Stavrakas, Y. An Architecture for Selective Web Harvesting: The Use Case of Heritrix. In Proceedings of the 1st International Workshop on Archiving Community Memories, Lisbon, Portugal, 6 September 2013.
8. Olston, C.; Najork, M. Web Crawling. *Found. Trends Inf. Retr.* **2010**, *4*, 175–246.
9. Brin, S.; Page, L. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In Proceedings of the 7th International Conference on World Wide Web, Brisbane, Australia, 14–18 April 1998; Elsevier Science Publishers: Amsterdam, The Netherlands, 1998; pp. 107–117.
10. Burner, M. Crawling towards eternity: Building an archive of the World Wide Web. Available online: http://people.apache.org/ jim/NewArchitect/webtech/1997/05/burner/index.html (accessed on 10 April 2014).
11. Heydon, A.; Najork, M. Mercator: A Scalable, Extensible Web Crawler. *World Wide Web* **1999**, *2*, 219–229.
12. Najork, M.; Heydon, A. High-Performance Web Crawling. In *Handbook of Massive Data Sets*; Kluwer Academic Publishers: Norwell, MA, USA, 2002; pp. 25–45.
13. Shkapenyuk, V.; Suel, T. Design and implementation of a high-performance distributed Web crawler. In Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, 26 February–1 March 2002; pp. 357–368.
14. Mohr, G.; Kimpton, M.; Stack, M.; Ranitovic, I. Introduction to heritrix, an archival quality web crawler. In Proceedings of the 4th International Web Archiving Workshop (IWAW'04), Bath, UK, 16 September 2004.
15. Boldi, P.; Codenotti, B.; Santini, M.; Vigna, S. UbiCrawler: A Scalable Fully Distributed Web Crawler. *Softw. Pract. Exp.* **2004**, *34*, 711–726.
16. Lee, H.T.; Leonard, D.; Wang, X.; Loguinov, D. IRLbot: Scaling to 6 Billion Pages and Beyond. *ACM Trans. Web* **2009**, *3*, 8:1–8:34.
17. Ntoulas, A.; Cho, J.; Olston, C. What's New on the Web?: The Evolution of the Web from a Search Engine Perspective. In Proceedings of the 13th International Conference on World Wide Web (WWW '04), New York, NY, USA, 17–22 May 2004; pp. 1–12.
18. Fetterly, D.; Manasse, M.; Najork, M.; Wiener, J. A Large-scale Study of the Evolution of Web Pages. In Proceedings of the 12th International Conference on World Wide Web (WWW '03), Budapest, Hungary, 20–24 May 2003; pp. 669–678.
19. Cho, J.; Garcia-Molina, H. The Evolution of the Web and Implications for an Incremental Crawler. In Proceedings of the 26th International Conference on Very Large Data Bases (VLDB '00), Cairo, Egypt, 10–14 September 2000; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2000; pp. 200–209.
20. Olston, C.; Pandey, S. Recrawl Scheduling Based on Information Longevity. In Proceedings of the 17th International Conference on World Wide Web (WWW '08), Beijing, China, 21–25 April 2008; ACM: New York, NY, USA, 2008; pp. 437–446.
21. Pandey, S.; Dhamdhere, K.; Olston, C. WIC: A General-purpose Algorithm for Monitoring Web Information Sources. In Proceedings of the 30th International Conference on Very Large Data Bases, (VLDB '04), Toronto, Canada, 29 August–3 September 2004; pp. 360–371.

22. Gouriten, G.; Maniu, S.; Senellart, P. Scalable, Generic, and Adaptive Systems for Focused Crawling. In Proceedings of the 25th ACM Conference on Hypertext and Social Media, Santiago, Chile, 1–4 September 2014.
23. Tang, T.T.; Hawking, D.; Craswell, N.; Griffiths, K. Focused Crawling for Both Topical Relevance and Quality of Medical Information. In Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM '05), Bremen, Germany, 31 October–5 November 2005; pp. 147–154.
24. Menczer, F.; Pant, G.; Srinivasan, P.; Ruiz, M.E. Evaluating Topic-driven Web Crawlers. In Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '01), New Orleans, LA, USA, 9–12 September 2001; pp. 241–249.
25. Chakrabarti, S.; van den Berg, M.; Dom, B. Focused crawling: A new approach to topic-specific Web resource discovery. *Comput. Netw.* **1999**, *31*, 1623–1640.
26. Halkidi, M.; Nguyen, B.; Varlamis, I.; Vazirgiannis, M. THESUS: Organizing Web document collections based on link semantics. *VLDB J.* **2003**, *12*, 320–332.
27. Ehrig, M.; Maedche, A. Ontology-focused Crawling of Web Documents. In Proceedings of the 2003 ACM Symposium on Applied Computing (SAC '03), Melbourne, FL, USA, 9–12 March 2003; pp. 1174–1178.
28. Ahlers, D.; Boll, S. Adaptive Geospatially Focused Crawling. In Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09), Hong Kong, China, 2–6 November 2009; pp. 445–454.
29. Gao, W.; Lee, H.C.; Miao, Y. Geographically Focused Collaborative Crawling. In Proceedings of the 15th International Conference on World Wide Web (WWW '06), Edinburgh, UK, 23–26 May 2006; pp. 287–296.
30. De Bra, P.M.E.; Post, R.D.J. Information Retrieval in the World-Wide Web: Making Client-based Searching Feasible. In Proceedings of the 1st Conference on World-Wide Web, Geneva, Switzerland, 25–27 May 1994; Elsevier Science Publishers B. V.: Amsterdam, The Netherlands, 1994; pp. 183–192.
31. Hersovici, M.; Jacovi, M.; Maarek, Y.S.; Pelleg, D.; Shtalhaim, M.; Ur, S. The shark-search algorithm. An application: tailored Web site mapping. *Comput. Netw. ISDN Syst.* **1998**, *30*, 317–326.
32. Almpanidis, G.; Kotropoulos, C.; Pitas, I. Combining Text and Link Analysis for Focused crawling—An Application for Vertical Search Engines. *Inf. Syst.* **2007**, *32*, 886–908.
33. Diligenti, M.; Coetzee, F.; Lawrence, S.; Giles, C.L.; Gori, M. Focused Crawling Using Context Graphs. In Proceedings of the 26th International Conference on Very Large Data Bases (VLDB '00); Cairo, Egypt, 10–14 September 2000; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2000; pp. 527–534.
34. Liu, H.; Janssen, J.; Milios, E. Using HMM to Learn User Browsing Patterns for Focused Web Crawling. *Data Knowl. Eng.* **2006**, *59*, 270–291.

35. Partalas, I.; Paliouras, G.; Vlahavas, I. Reinforcement Learning with Classifier Selection for Focused Crawling. In Proceedings of the 2008 Conference on Artificial Intelligence, Chicago, IL, USA, 13–17 July 2008; IOS Press: Amsterdam, The Netherlands, 2008; pp. 759–760.
36. Johnson, J.; Tsioutsiouliklis, K.; Giles, C.L. Evolving Strategies for Focused Web Crawling. In Proceedings of the 20th International Conference on Machine Learning, Washington, DC, USA, 21–24 August 2003; Fawcett, T., Mishra, N., Eds.; AAAI Press: 2003; pp. 298–305.
37. Zheng, H.T.; Kang, B.Y.; Kim, H.G. An ontology-based approach to learnable focused crawling. *Inf. Sci.* **2008**, *178*, 4512–4522.
38. Bergman, M.K. White paper: the Deep Web: surfacing Hidden Value. *J. Electron. Publ.* **2001**, *7(1)*. Available online: http://dx.doi.org/10.3998/3336451.0007.104 (accessed on 10 April 2014).
39. Barbosa, L.; Freire, J. Siphoning Hidden-Web Data through Keyword-Based Interfaces. In Proceedings of the 19th Brazilian Symposium on Databases, Brasilia, Brazil, 3–7 October 2004; pp. 309–321.
40. Madhavan, J.; Ko, D.; Kot, L.; Ganapathy, V.; Rasmussen, A.; Halevy, A. Google's Deep Web crawl. *Proc. VLDB Endow.* **2008**, *1*, 1241–1252.
41. Arvidson, A.; Persson, K.; Mannerheim, J. The Kulturarw3 Project—The Royal Swedish Web Archiw3e—An example of "complete" collection of web pages. In Proceedings of the 66th IFLA Council and General Conference, Jerusalem, Israel, 13–18 August 2000.
42. Bailey, S.; Thompson, D. UKWAC: Building the UK's First Public Web Archive. *D-Lib Mag.* **2006**, *12*. Available online: http://www.dlib.org/dlib/january06/thompson/01thompson.html (accessed on 10 April 2014).
43. Cathro, W.; Webb, C.; Whiting, J. Archiving the Web: The PANDORA Archive at the National Library Australia. Available online: http://www.nla.gov.au/openpublish/index.php/nlasp/article/view/1314/1600 (accessed on 10 April 2014).
44. Spaniol, M.; Denev, D.; Mazeika, A.; Weikum, G.; Senellart, P. Data Quality in Web Archiving. In Proceedings of the 3rd Workshop on Information Credibility on the Web (WICOW '09), Madrid, Spain, 20–24 April 2009; pp. 19–26.
45. Denev, D.; Mazeika, A.; Spaniol, M.; Weikum, G. SHARC: Framework for Quality-conscious Web Archiving. *Proc. VLDB Endow.* **2009**, *2*, 586–597.
46. Gomes, D.; Miranda, J.A.; Costa, M. A Survey on Web Archiving Initiatives. In Proceedings of the 15th International Conference on Theory and Practice of Digital Libraries: Research and Advanced Technology for Digital Libraries (TPDL'11), Berlin, Germany, 26–28 September 2011; Springer-Verlag: Berlin/Heidelberg, Germany, 2011; pp. 408–420.
47. Tahmasebi, N.; Gossen, G.; Kanhabua, N.; Holzmann, H.; Risse, T. NEER: An Unsupervised Method for Named Entity Evolution Recognition. In Proceedings of the 24th International Conference on Computational Linguistics (COLING' 12), Mumbai, India, 8–15 December 2012; Kay, M., Boitet, C., Eds.; Indian Institute of Technology Bombay: Mumbai, India, 2012; pp. 2553–2568.

48. Plachouras, V.; Stavrakas, Y.; Andreou, A. Assessing the Coverage of Data Collection Campaigns on Twitter: A Case Study. In *On the Move to Meaningful Internet Systems: OTM 2013 Workshops*; Demey, Y.T., Panetto, H., Eds.; Springer: Berlin/Heidelberg, Germany 2013; Volume 8186, pp. 598–607.
49. Faheem, M.; Senellart, P. Intelligent and Adaptive Crawling of Web Applications for Web Archiving. In Proceedings of the 13th International Conference on Web Engineering (ICWE), Aalborg, Denmark, 8–12 July 2013; Daniel, F., Dolog, P., Li, Q., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7977, pp. 306–322.
50. Faheem, M.; Senellart, P. Demonstrating intelligent crawling and archiving of web applications. In Proceedings of the 22nd ACM International Conference Information and Knowledge Management (CIKM '13), Burlingame, CA, USA, 27 October–1 November 2013; pp. 2481–2484.
51. Cunningham, H.; Maynard, D.; Bontcheva, K.; Tablan, V.; Aswani, N.; Roberts, I.; Gorrell, G.; Funk, A.; Roberts, A.; Damljanovic, D.; *et al*. *Text Processing with GATE (Version 6)*; Department of Computer Science, University of Sheffield: Sheffield, UK, 15 April 2011.
52. Apache Lucene Core. Available online: http://lucene.apache.org/core/ (accessed on 10 April 2014).