An Architecture for Selective Web Harvesting: The Use Case of Heritrix[⋆]

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

[1] IMIS, ATHENA Research Center, Athens, Greece
[2] Internet Memory Foundation, 93100 Montreuil, France
[3] L3S Research Center, University of Hannover, Germany
[4] Institut Mines-Télécom; Télécom ParisTech; CNRS LTCI, Paris, France

Abstract. In this paper we provide a brief overview of the crawling architecture of ARCOMEM and how it addresses the challenges arising in the context of selective web harvesting. We describe some of the main technologies developed to perform selective harvesting and we focus on a modified version of the open source crawler Heritrix, which we have adapted to fit in ACROMEM's crawling architecture. The simulation experiments we have performed show that the proposed architecture is effective in a focused crawling setting.

Keywords: Web harvesting, focused crawling, web application crawling

1 Introduction

The objectives of the ARCOMEM project are to select and harvest the web content that best preserves community memories for the future by exploiting social media, as opposed to exhaustive web harvesting approaches. To achieve these objectives, there are several challenges that need to be addressed. First, it is necessary to integrate in the crawling process advanced and computationally-intensive content analysis without reducing the efficiency of the crawler. Second, it is important to guide the web harvesting with accuracy to discover and collect relevant resources. Last, the above functionalities must be scalable in order to process large amounts of data.

In this paper, we provide a brief description of the ARCOMEM's crawling architecture and some of the main technologies. One of its main features that addresses the challenges mentioned above is the decoupling of the fetching from the prioritization of resources. As a result, we can scale up both processes independently, according to requirements and available resources.

In addition to using a large-scale distributed crawler, we have also investigated to what extent Heritrix [5] can be adapted to the crawling architecture of ARCOMEM. Heritrix implements a typical crawling process, and hence, we had to modify it to support the features required by ARCOMEM.

[⋆] This work was funded by the European Commission under grant agreement n. 270239 (ARCOMEM).

2 Architecture for large-scale selective Web harvesting

The workflow of the crawling architecture is the following. The crawler fetches web pages that are written to a document repository based on HBase. The prioritization of web pages consists of two phases. The online phase runs as a map-reduce job and processes newly-crawled web pages. During the online phase, the Application-Aware Helper (Section 2.1) identifies the structure of a web page. Then for each identified element and the whole document, online analysis modules perform light-weight analysis of web pages and compute scores, which are aggregated in a priority score for each URL (Section 2.2). The offline analysis performs more computationally demanding processing of Web pages to extract named entities and to refine the computed relevance scores, which can also be integrated in the priority score of URLs. The scored URLs are finally sent back to the crawler (for example the large-scale crawler described in Section 2.3) to be fetched. Note that different runs of the online phase can send the same URL with different priorities to the crawler. Hence, the crawler must support the updating of the priorities for already scheduled URLs.

Additional links can be discovered through a social network API crawler [3]. The structured content and the computed scores are also stored in a large-scale distributed triple store [7] for preservation and later use. The configuration of the whole process, including the crawl's scope, weights, relevant keywords and entities, are given in the *Intelligent Crawl Specification* (ICS).

2.1 Application-Aware Helper

The goal of the Application-Aware Helper (AAH) is to make the crawler aware of the particular kind of web application being crawled, in terms of general classification of web sites (wiki, social network, blog, web forum, etc.), technical implementation (Mediawiki, Wordpress, etc.), and their specific instances (Twitter, CNN, etc.).

For each crawled document, the AAH first tries to identify the corresponding web application. If a matching application is found in a knowledge base of known web applications, the system attempts to identify the kind of the web page given the matched web application. Once the kind of a document has been established, structured content in the form of web objects is extracted and passed on to the online analysis modules for scoring. For example, if the AAH is processing a document crawled from a forum, it can identify the discussion thread structure even if posts span over several different web pages.

The AAH can also deal with two different cases of adaptation. First, when a web application, or part of it, has been crawled before a template change and a recrawl is carried out after that. Second, when crawling a new web application that matches the Web application type detection patterns, but for which some or all the actions are inapplicable. In addition to running the AAH as part of the ARCOMEM architecture, it can also be employed as a link extractor for Heritrix [5]. A detailed description of the AAH can be found in [2].

2.2 Online analysis

The prioritization module is used for focusing the crawler with respect to the crawl specification. Its main task is the aggregation of scores provided by different online and offline analysis modules to give URLs of relevant web pages a higher score, so that these pages are crawled with a higher priority. The scoring is also used to select the most relevant pages for creating the final archive.

The online phase of URL prioritization employs a range of modules to calculate a priority for every URL found in the crawled web pages. There are modules for updating the score of a URL when it matches user-specified regular expressions, if the URL corresponds to a known web page type identified by AAH, if the content of the page is written in a given language, or if the context of the URL contains keywords from the ICS. During the online phase, the strict time constraints do not allow us to run all modules for analyzing the content. We instead rely on the final relevance filtering after the end of the offline phase to eliminate web pages of ambiguous relevance, as we have more information and fewer time constraints at this point.

To calculate a final score for each URL detected within an already-crawled document, we have implemented a scoring function, which generates a linear combination of the scores from the different modules based on a weight vector. By using this method, we can manually change the weight of the different modules on the final score. We are also developing a method to automatically adjust the weight vector by comparing the actual relevance of a crawled document with the scores predicted for the document's URL with the different methods. Based on this we adjust the weight of different modules to get more accurate relevance predictions for the given ICS.

2.3 Large-scale crawler

The large-scale crawler is a distributed crawler, implemented by Internet Memory Foundation (IMF), that retrieves content from the web and stores it in an HBase repository. It aims at being scalable: 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 [4, 1].

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 large-scale crawler also employs a full-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.

3 Adapting Heritrix to the ARCOMEM architecture

In addition to the large-scale crawler, we have also investigated to what extent Heritrix, an open source crawler [5], 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 in the releases of the ARCOMEM project. In this paper 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. 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.

To overcome this limitation, we have implemented a frontier that extends the default frontier of Heritrix, adding a hash table that maps an already scheduled URL to the key with which it was scheduled. When we need to update the priority of an already scheduled URL, 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. 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 the *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 in the ARCOMEM crawling architecture and receive URLs with priority scores from the map-reduce jobs of 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.

4 Evaluating adaptive and batch prioritization

Since the ARCOMEM crawling architecture and our modifications of Heritrix depart from the standard crawling architectures [6], it is important to evaluate their impact on a focused crawler's effectiveness. In this section we describe a set of simulations experiments we have performed to assess the impact of adaptive and batch prioritization on the crawler's effectiveness.

We assume a baseline crawler implementing a best-first crawling strategy, where 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 link 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 top-k highest priority pages 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 10000 web pages. The topic vector we use to compute similarities between each topic and the crawled web pages corresponds to the sum of the seed vectors. For the evaluation of the results, we employ three measures: a) harvest ratio: the number of Web pages having cosine similarity with topic vector greater than 0.333, b) average similarity of crawled pages and 3) 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.

We have also evaluated batch prioritization where the priorities of link are computed after crawling a batch of web pages. The evaluation results show that batch prioritization may reduce slightly the effectiveness of an adaptive crawler using the function AVG (harvest ratio drops from 0.3609 to 0.3556). If a crawler does not strictly fetch the web pages with the highest priority first, but crawls more pages from each domain (similar to the effect of the parameter *balance-per-queue* in Heritrix), then batch prioritization improves the harvest ratio (from 0.3200 to 0.3347 when crawling 5 web pages from each domain).

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**

5 Concluding Remarks

In this paper, we have provided an overview of ARCOMEM's crawling architecture and we have briefly described the technologies for some of its components. We have also described how we have modified Heritrix to fit the proposed crawling architecture, enabling the adaptive prioritization of URLs and the scheduling of prioritized URLs via a Web service. The results of our simulation experiments show that the adaptive prioritization improves crawling effectiveness, while batch prioritization improves effectiveness when URLs are also crawled in batches.

Overall, we have seen that ARCOMEM's crawling architecture is scalable both for fetching and processing content, while it allows the use of either a large-scale distributed crawler or an appropriately adapted version of Heritrix.

References

1. Boldi, P., Codenotti, B., Santini, M., Vigna, S.: UbiCrawler: a scalable fully distributed web crawler. Softw. Pract. Exper. **34**(8) (July 2004) 711–726
2. Faheem, M., Senellart, P.: Intelligent and Adaptive Crawling of Web Applications for Web Archiving. In: Procs. of Intl. Conf. on Web Engineering (ICWE). (2013)
3. Gouriten, G., Senellart, P.: API Blender: A Uniform Interface to Social Platform APIs. In: Procs. of WWW Conf., Developer Track. (2012)
4. Lee, H.T., Leonard, D., Wang, X., Loguinov, D.: IRLbot: Scaling to 6 billion pages and beyond. ACM Trans. Web **3**(3) (July 2009) 1–34
5. Mohr, G., Stack, M., Ranitovic, I., Avery, D., Kimpton, M.: An Introduction to Heritrix: An open source archival quality web crawler. In: Procs. of 4th Intl. Web Archiving Workshop (IWAW'04). (2004)
6. Olston, C., Najork, M.: Web Crawling. Found. Trends Inf. Retr. **4**(3) (2010) 175–246
7. Papailiou, N., Konstantinou, I., Tsoumakos, D., Koziris, N.: H2RDF: adaptive query processing on RDF data in the cloud. In: Procs. of WWW Conf., Companion. (2012)