OWLER:
A DISTRIBUTED OPEN WEB CRAWLER

M. Dinzinger*, S. Zerhoudi,
J. Mitrović, M. Granitzer,
University of Passau, Passau, Germany

Abstract

The public availability of web data has become a main driver for innovation in the domains of Artificial Intelligence and Web Search. In this course, we have proposed the Open Web Index (OWI), a publicly funded service for providing enriched and indexed web documents to foster the development of new search applications and data products. This mission requires, among others, a comprehensive, continuous crawling effort, based on the cornerstone principles of openness and legal compliance. In order to overcome the posed technical challenge, we have further presented the Open Web Crawler (OWLer) [4], an open-source software framework driving the large-scale collection of web documents.[1]

OWLer constitutes the backbone of a fully integrated processing pipeline for indexing and sharing large amounts of curated web resources. Due to the ambitious vision of OWI as well as the geographic dispersion of the therefore available compute resources, the crawling system requires to be highly distributed and collaborative. Technically, OWLer bases on open-source projects such as StormCrawler, OpenSearch and the URLFrontier framework. Our work is an aggregation and refinement to existing efforts in open-source web crawling in order to accommodate the combined requirements of scalability, efficiency and transparency. In this paper, we present the conceptual and technical background, discuss design decisions and overview the architecture of the OWLer crawling system.

INTRODUCTION

The dominance of a few commercial search engines has led to a closed web search ecosystem where publishers must optimize their content for these gatekeepers, potentially sacrificing quality and hindering innovation [5]. To counter this, we have proposed the development of an Open Web Index (OWI) as publicly funded infrastructure, guided by the core principles of open data, legal compliance and collaborative technology. The *Open Web Crawler (OWLer)*, implemented as distributed, incremental crawling system, takes a central position in this joint development effort [6]. The intuition of OWLer resembles the motivation behind the non-profit organization Common Crawl.[2]

Whereas Common Crawl regularly publishes large-scale collections of crawled web documents as well as complementary data products such as clean-text corpora and aggregated web graphs, OWLer is crawling continuously on a federated infrastructure. The collaboration of different European institutions and infrastructure providers is crucial for the success of OWI and the encompassing crawling effort. The raw web data collected by OWLer is integrated with the technical pipeline for collaboratively building a rich web index. In order to collect enough resources given the available structural setup, the system architecture is oriented towards two major objectives:

- Modularity
  The compute resources available for the project are highly heterogeneous and dispersed over multiple datacenters in Europe. In order to maintain such as a system with manageable effort, it requires a compact and modular architecture employing well-defined interfaces for the communication between remote services.

- Scalability
  The ambitious vision of OWI necessites comprehensive crawling covering a significant part of the text-based surface web. No few-node cluster and no single European institution can take up this task on its own. The required performance is rather grounded in the system's ability to scale horizontally, integrating more nodes located in collaborating infrastructure providers.

This paper overviews OWLer by providing general and technical background information on the design of the current system. After taking a look into the related work, Section 3 describes the distributed architecture in more detail by presenting each of its three core software components. We further share preliminary results of its current live deployment (Section 4) and conclude with a resumée of the main challenges in the previous development as well as an outlook on the future development of OWLer.

RELATED WORK

The domain of web crawling dates back to the origins of the world wide web in early 1990s. Throughout these years, search engine operators and reseachers worked on software tools for the efficient traversal of the web. Due to a wide range of research endeavors, crawling systems have continously improved on its four main quality criteria: coverage and freshness, politeness and robustness [11]. Along the way, numerous technical challenges have been studied and overcome, e.g., near-duplicate detection of web documents [3, 10] and focused crawling [2].

* michael.dinzinger@uni-passau.de

[1] Link to open repositories:
`https://opencode.it4i.eu/openwebsearcheu-public`

[2] `https://commoncrawl.org`

Several notable software tools have been developed over the years. *Mercator*, described in Najork et al. [13], was one of the first commercial open-source crawlers that targeted high-performance. Along with Mercator, Najork et al. introduced the *URL frontier* (or URL manager). This software component is implemented as multi-level queue-based data structure and schedules URLs depending on some priority criteria (e.g. web page quality), while ensuring politeness towards web servers through request delays. *Heritrix* and the open-source crawler *Apache Nutch* were further early web crawlers that have been extensively used in academia and industry [8, 12]. The *IRLbot*, published in 2008 [9], was a pioneering effort in scaling open-source web crawling to handle billions of web pages on a single-machine setup. Similarly, *UbiCrawler* and its successor *BUbiNG* were developed by Boldi et al. [1] to achieve maximal throughput on a single powerful machine. The crawling tool is able to process several thousand pages per second, achieving an optimized utilization of the hardware while respecting politeness constraints.

SYSTEM COMPONENTS

The OWLer crawling system is designed to handle the challenges of a highly heterogeneous and distributed infrastructure with machines of different sizes and locations. The modular architecture consists of three loosely-coupled but collaborating tiers, as shown in Figure 1. Each tier is a self-contained software project and fulfills distinct, complementary tasks. The implementation of crawlers and the distributed database system can be interchanged due to well-defined interfaces in the URL Frontier layer.

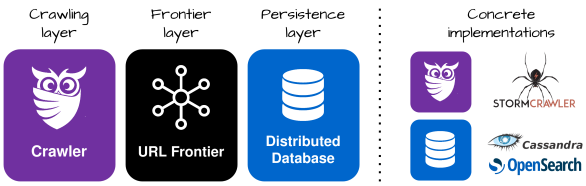

Figure 1: Three layers in the system architecture

Crawling

The first layer consists of software tools for continuously fetching web pages without managing the set of discovered links (crawl space), which is the task of the other two tiers. A crawler instance retrieves URLs to be fetched through a remote call to a URL Frontier instance and adds them to its internal task queue (see Figure 2). The web pages are downloaded, parsed, and supplemented with meta information corresponding to the page content. Finally, the URL and discovered outlinks are uploaded back to the URL Frontier instance, which updates the status of the crawl. The crawlers are lightweight and can run on commodity-sized machines in any computing center with sufficient external network bandwidth.

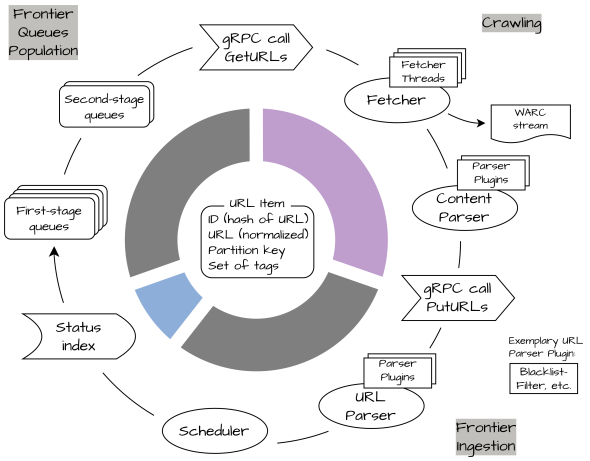

Figure 2: Technical crawling pipeline

URL Frontier

A crawling node communicates only with the URL Frontier instance, from which it retrieves and to which it uploads web links (*URL items*). The communication between the crawler and frontier nodes is implemented as remote calls over a gRPC connection,[3] resulting in a loose coupling between them. They rely on a well-defined Protocol Buffers [4] API in the URL Frontier project. The URL items exchanged consist of the normalized plain-text URL, a unique identifier (based on the URL hash), the partition key, and a set of tags. The tags are extracted by the Content Parser or the URL Parser, which internally call Parser Plugins. Tags can also be provided by users through a social tagging system, potentially contributing to a higher quality crawl by integrating user-curated meta information during data collection.

The frontier tier comprises one or more instances (also called services), each connected to one or more crawler instances and one storage backend for persisting URL items. Each URL Frontier service is assigned to a section of the *crawl space*, defined as the set of discovered web links that expands over time as the crawl continues.[5] A single service is responsible for its own partition of the crawl space, providing its clients with the next URLs to be fetched while ensuring a sufficient time interval between subsequent fetches of the same resource.

As shown in Figure 2, URL items uploaded by the crawlers are ingested and persisted. The first part of the frontier ingestion pipeline filters and parses the URL, including the execution of Parser Plugins. One exemplary Parser Plugin with a significant positive impact on the system's robustness is the `BlacklistFilter`, which checks web links against

[3] `https://grpc.io`
[4] `https://protobuf.dev`
[5] The border between the crawl space and the undiscovered web is also called *frontier*, which is the source of the name URL Frontier.

a number of public spam databases. The ingestion pipeline also includes the Scheduler component, which determines the next planned fetch date of the web resource based on the quality and change frequency of the page content.

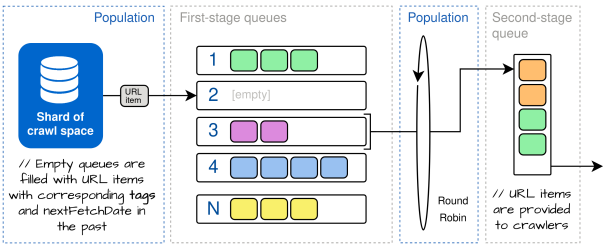

Figure 3: Population of frontier data structure

The URL Frontier service constantly queries the storage backend to populate its internal frontier data structure (see Figure 3), while simultaneously ingesting incoming URLs. A service can define an "interest" using a signature of tags, which limits the set of web resources it fetches from the backend. This allows for focused crawling within the overall general crawling system. For example, some crawlers may only be interested in sitemaps of family-friendly websites, so the corresponding URL Frontier service only retrieves web links tagged as `Sitemap` and `FamilyFriendly`.

To ensure strict politeness and high throughput, the frontier's internal data structure uses two stages of queues. The first-stage queue is determined by the *queue ID*, which is based on the hash of the Paid-Level Domain (PLD). Web resources with the same PLD (shown as the same color in Figure 3) are always kept in the same queue. Empty first-stage queues are refilled to ensure a constant supply of links to be fetched. In addition to matching the queue ID and the scope of interest (represented by required tags), URL items retrieved from the crawl space must have a `nextFetchDate` in the past. This simple yet effective approach enforces a time interval between two fetches of the same link.

The second-stage queue (or buffer queue) collects and sends URL items to crawlers requesting new fetch tasks. It is populated by iterating over the first-stage queues in a Round Robin manner. A delay between each traversal round prevents the buffer queue from being populated too frequently with URL items having the same queue ID (and possibly the same domain). OWLer defaults to a minimum five-minute delay between subsequent rounds of buffer queue population, and for each first-stage queue, only the first ten items are popped and added to the buffer queue. The buffer queue has a maximum capacity smaller than the number of first-stage queues, ensuring that a crawler working in a streaming manner will not visit pages with the same Paid-Level Domain more than 20 times in five minutes. Each crawler also applies a default crawl delay of 1 second (or as specified in the robots.txt file) to avoid overwhelming a web server at any given moment.

Persistence

The storage backend choice is crucial for the system's overall performance, as it is the main factor impacting the latency of GetURLs and PutURLs requests. OWLer initially relied on the open-source Search & Analytics Platform OpenSearch,[6] which had certain drawbacks. An interface was added to allow substituting the concrete backend implementation, further modularizing the URL Frontier software and minimizing dependency on a single technology.

The persistence layer stores the crawl space on long-term memory and constantly provides the previously persisted URL items to be fetched next. The database system must accommodate a high number of read operations for populating first-stage queues and read-insert operations for updating the crawl space (tens of thousands per second). Any storage backend must be optimized for two commands: data querying (DQ) and data manipulation (DM), resulting in a tight data model. This data modeling step is implemented slightly differently in every database system, but it is key to achieve low request latency and high crawling throughput. Beyond data modeling, query routing and data locality significantly impact performance in a distributed setup. The partition key, previously mentioned as part of the URL item, has proven useful for effectively spreading the crawl space over tables and machines, enabling efficient data querying. However, these aspects are not discussed in more detail as they are core concepts of any distributed system and can be successfully managed by the concrete DBMS.

OWLER IN ACTION

This section shares insights and results obtained with the current, preliminary crawling system setup. The past nine months have been a phase of iterative development, during which OWLer was set up and integrated with the downstream Open Web Indexing pipeline. During this time, the deployment was not fully permanent but interrupted by several breaks used to eliminate deficiencies in politeness, performance, and robustness discovered along the way. Nevertheless, within these nine months, OWLer discovered 10.2B web links that are persisted on the current OpenSearch-based storage backend. 1.17B of them have been visited at least once by one of the StormCrawler-based agents. The visited URLs are distributed over 37.3M hosts, dispersed over different topical and geographical domains of the web. As some web pages have been recrawled, OWLer has processed a total of 3.50B URLs, with an 88% successful fetch rate. This leads to a total of around 3.08B web documents provided to the indexing pipeline.

In addition to crawling, OWLer has ingested parts of publicly available dumps of Common Crawl to fill our crawl space with a broad range of seed URLs and provide a continuous output of crawled web documents despite the discontinuous deployment of OWLer in the early phases. The result is approximately 150 TiB of mostly HTML web documents,

[6] `https://opensearch.org`

compressed and archived in WARC file format. All data was made publicly available as compressed Parquet and CIFF [7] files encoding the Open Web Index as inverted files with complementary metadata information.

During a four-week period of consistently high performance, a setup of six crawlers, two URL Frontier services, and one OpenSearch node achieved around 36M to 42M visits per day (equivalent to over 1 TB of WARC files per day). This means between 6M and 7M visits per crawling node and around 75 URLs per second per node. Ongoing experiments indicate the potential for further significant performance increases. Software and infrastructure engineering efforts in enhancing the processing pipeline, improving the backend data model, and extending the OpenSearch cluster suggest an increase in throughput by a factor of two to three in a similar but more robust setup. More advanced crawling tools manage up to 1000 URLs per second, and according to our tests, a URL Frontier instance with one OpenSearch node can consistently provide enough URLs to supply it. As a next step in order to meet OWI's aspiration, the system needs to scale horizontally by employing a multi-node database cluster.

CONCLUSION

This report introduces the Open Web Crawler (OWLer), a crucial component of the Open Web Index (OWI) development project. The OWI aims to promote open access to web data and encourage innovation in web search technologies. By providing a publicly funded, legally compliant, and open-source alternative to web indices of commercial search engines, the OWI challenges their current dominance and strengthens the community-driven collection and processing of web data.

OWLer's modular and scalable architecture is designed to handle the diversity and geographic distribution of European compute resources effectively. This design enables the system to manage the large-scale, continuous crawling required to capture a comprehensive snapshot of the web. By integrating open-source technologies such as StormCrawler, OpenSearch, and URLFrontier, OWLer demonstrates a commitment to utilizing and contributing to the open-source community. The tier architecture consisting of crawler, frontier and persistence layer modularizes the software project. The URL Frontier services take the central position within this architecture and define interfaces towards crawlers and the distributed storage system.

Throughout the project, significant technical challenges, primarily related to achieving efficient distribution and robustness in data handling, have been addressed. These efforts have already resulted in the collection of billions of URLs, demonstrating the system's ability to handle web-scale data. Future development of OWLer will focus on improving performance and expanding crawling capabilities. By continuously refining the system, the project aims to make even more substantial contributions to the open web ecosystem, facilitating the creation of innovative applications and services that leverage the vast amounts of data processed by OWLer.

[7] CIFF denotes the Common Index File Format; for more details, see [7]

ACKNOWLEDGEMENT

This work is part of OpenWebSearch.eu, funded by the EU under GA 101070014, and part of CAROLL, funded by the German Federal Ministry of Education and Research (BMBF) under 01|S20049.

REFERENCES

[1] P. Boldi, A. Marino, M. Santini, S. Vigna, *BUbiNG: Massive Crawling for the Masses*, ACM Trans. Web 12 (2), May 2018.

[2] S. Chakrabarti, M. van den Berg, B. Dom, *Focused crawling: a new approach to topic-specific Web resource discovery*, Computer Networks, Volume 31, Issues 11–16, 1999, pp. 1389–1640.

[3] M. Charikar, *Similarity Estimation Techniques from Rounding Algorithms*, Proc. of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 380–388.

[4] M. Dinzinger et al, *OWLer: Preliminary results for building a Collaborative Open Web Crawler*, Proc. of the 5th International Open Search Symposium (OSSYM), Oct. 2023.

[5] M. Granitzer et al, *Impact and development of an Open Web Index for open web search*, Journal of the Association for Information Science and Technology, Aug. 2023.

[6] G. Hendriksen et al, *The Open Web Index: Crawling and Indexing the Web for Public Use*, Advances in Information Retrieval (ECIR 2024), Mar. 2024.

[7] D. Hiemstra et al, *Challenges of Index Exchange for Search Engine Interoperability*, Proc. of the 5th International Open Search Symposium (OSSYM), Oct. 2023.

[8] R. Khare, D. Cutting, K. Sitaker, A. Rifkin, *Nutch: A Flexible and Scalable Open-Source Web Search Engine*, 2005.

[9] H. Lee, D. Leonard, X. Wang, D. Loguinov, *IRLbot: Scaling to 6 Billion Pages and Beyond*, Proc. of the 17th International Conference on World Wide Web, 2008, pp. 427–436.

[10] G. S. Manku, A. Jain, A. Das Sarma, *Detecting Near-Duplicates for Web Crawling*, Proc. of the 16th International Conference on World Wide Web, 2007, pp. 141–150.

[11] C. Manning, P. Raghavan, H. Schütze, *Web crawling and indexes* (Chapter 20), In: *Introduction to Information Retrieval*, Cambridge University Press, 2008.

[12] G. Mohr, M. Kimpton, M. Stack, I. Ranitovic, *Introduction to Heritrix, an archival quality web crawler*, Proc. of the 4th International Web Archiving Workshop (IWAW'04), Jul. 2004.

[13] M. Najork, A. Heydon, *High-Performance Web Crawling*, Handbook of Massive Data Sets. Massive Computing (4), Springer, 2002.