A Framework for Bridging the Gap Between Open Source Search Tools

Madian Khabsa[1], Stephen Carman[2], Sagnik Ray Choudhury[2] and C. Lee Giles[1,2]
[1]Computer Science and Engineering
[2]Information Sciences and Technology
The Pennsylvania State University
University Park, PA
madian@psu.edu, shc5011@ist.psu.edu, sagnik@psu.edu, giles@ist.psu.edu

ABSTRACT

Building a search engine that can scale to billions of documents while satisfying the needs of the users presents serious challenges. Few successful stories have been reported so far [37]. Here, we report our experience in building YouSeer, a complete open source search engine tool that includes both an open source crawler and an open source indexer. Our approach takes other open source components that have been proven to scale and combines them to create a comprehensive search engine. YouSeer employs Heritrix as a web crawler, and Apache Lucene/Solr for indexing. We describe the design and architecture, as well as additional components that need to be implemented to build such a search engine. The results of experimenting with our framework in building vertical search engines are competitive when compared against complete open source search engines. Our approach is not specific to the components we use, but instead it can be used as generic method for integrating search engine components together.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - search process; H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Design, Documentation, Performance

Keywords

Search Engines, Software Architecture, Open Source

1. INTRODUCTION

In the past fifteen years, many search engines have emerged out of both industry and academia. However, very few have been successful [37]. Those are a number of challenges [28]. Firstly, the documents need to be collected before they are searchable. These documents may be found on the local intranet, a local machine, or on the Web. In addition, these documents range in format and type from textual files to multimedia files that incorporate video and audio. Expediently ranking the millions of results found for a query in a way that satisfies the end-user need is still an unresolved problem. Patterson [37] provides a detailed discussion of these hurdles.

As such, researchers and developers have spent much time and effort designing separate pieces of the search engine system. This lead to the introduction of many popular search engine tools including crawlers, ingestion systems, and indexers. Examples of such popular tools include: Apache Lucene/Solr, Indri, Terrier, Sphinx, Nutch, Lemur, Heritrix, and many others. The creators of each tool have introduced software with multiple features and advantages, but each one of them has its own limitations. Since the problems faced by designing a crawler are quite different from what an indexer developer would face, researchers have dedicated projects to tackling certain parts of the search engine. Furthermore, the same unit within the search engine (such as indexer) may introduce different challenges based on the application need. Working on scaling an indexer to support billions of documents is different than creating a real time indexer, therefore each use case has lead to a respective solution.

Few projects aim at building a complete open source search engine that includes a web crawler, ingestion module, indexer, and search interface. While complete search engine tools provide all the different pieces to run a search engine, these pieces tend to be outperformed by task specific open source search tools when compared against each other based on the specific task only. For example, while Apache Nutch provides an entire search engine solution, Heritrix, which is just a web crawler, is more powerful and versatile when compared solely against the Nutch crawler. This observation has lead us to consider building a unified framework where search engine components can be plugged in and out to form a complete search engine.

New projects are started everyday to solve a specific problem for search engines, or to to introduce new features. Likewise, many projects have been out of support and development after being abandoned by the community. The level of support and the richness of the features are what usually determines how prevalent an open source project is. We propose building a search engine framework that is modular and component agnostic where different crawlers or indices can be interchanged as long as they conform to a set of standards. This modularity facilitates plugging components that satisfies the users need with minimal to no changes of the framework's code base. Under such framework, powerful components can be included as they mature, and the community can focus on advancing specific parts of the search engine without worrying about building a complete search engine.

In this paper, we demonstrate how to exploit these freely available tools to build a comprehensive search framework for building vertical and enterprise search engines. We outline the architecture of the framework, and describe how each component contributes to building a search engine, and the standards and communication mechanisms between the different components. We rely on agreed upon standards to control the communication mechanisms between the different parts of the search engine in order to allow maximum flexibility and reduce coupling between the components. For the crawler, we assume that crawlers will save the crawled documents in WARC format, which became the standard for web archiving in 2009 [1]. Warc files are compressed files that contain records about the metadata of the crawled document along with the documents itself. We implement the middleware which is responsible for ingesting the crawled files, and pass them to the indexer over a REST API [26]. REST has become the defacto standard for accessing web services, and in this case we assume that indices are providing a REST API to communicate with the ingestion module and with the query interface. This assumption is rational as most indices end up providing some access mechanism over HTTP to support distribution.

The framework we are introducing here, *YouSeer*, not only can be used for building niche search engines but also for educational purposes. For the last three years YouSeer has been piloted in an advanced undergraduate/graduate class at Penn State designed to give students the ability to build high end search engines as well as properly reason around the various parts and metrics used to evaluate search engines. Students in this class were tasked with finding a customer within the Penn State community, usually a professor or graduate student, who is in need of a search engine. Students then were required to build a complete search engine from concept to delivery. This included crawling, indexing and query tuning if necessary. This class has yielded many niche based search engines of varying levels of complexity all using YouSeer as a software package. The level of technical know how in the class ranges from beginners with UNIX/Java to PhD level students in Computer Science/Engineering and Information Sciences and Technology.

The rest of this paper is organized as follows. Section 2 discusses related work and describes other open source search engines. Section 3 provides an overview of the architecture of YouSeer, while Section 4 discusses the workflow inside the framework. In Section 5 we describe the experiments. Finally, we conclude and identify areas of future work in Section 6.

2. RELATED WORK

The use of search engines to find content goes back to the days of library search, where librarians have used and developed the techniques of information retrieval to find content within books. These techniques have been carried out to the domain of web search, and enterprise search. Though web search added the concept of web crawler, or spider, to download the documents from the web, which can be used to discriminate the web search era from the previous era of librarian search.

The open source community felt the urge for an alternative to the commercial search engines that are dominating the market. Part of the need was to provide transparent solutions where users control the ranking of results and made sure they have not been manipulated. In addition, licensing the search services from these commercial search engines can be expensive.

ht://Dig [7] is one of the early open source search tools which was created back in 1995 at San Diego State University. The project is not designed to scale for the needs of entire web indexing. Instead it's designed to index content of few websites, or intranet. *ht://Dig* supported boolean and fuzzy queries, and had the ability to strip text out of HTML tags. Currently the project is out of support, as the latest release was back in 2004.

Apache Luecene [3] is considered one of the very popular search libraries which has been ported to multiple languages. It was originally written in Java by Doug Cutting back in 2000, and later it became part of the Apache Software Foundation. Though Lucene is not a search engine itself, but it's an indexing library which can be used to perform all the indexing operations on text files. It can be plugged into any search application to provide the indexing functionality.

Numerous search engines were developed on top of the Lucene library, including commercial and open source solutions. Nutch [29, 24] was among the early search engines developed on top of lucene. It added a web crawler and a search interface and used the lucene indexing library to build a comprehensive search engine. Nutch was later added to the Apache Software Foundation as a sub-project of Lucene. In developing Nutch, the developers have aimed at creating a scalable tool that can index the entire web. However, the largest crawl ever reported on Nutch was 100 million documents [29, 34] despite supporting parallel crawling on different machines. In addition, Nutch added link analysis algorithms to its ranking function to take into account the importance of the pages along with the relevancy.

Although Nutch provides rich set of crawling features, many other open source crawlers, i.e. Heritrix [5], provide far more complex and advanced features. For examples, the deciding rules for accepting a file or rejecting it in the crawling process are much powerful in Heritrix than Nutch. The ability to take check points, pause and resume crawling, and restore the crawling process in case of failure are also advantages of Heritrix that Nutch lacks. In addition, Nutch obeys the robots exclusion protocol, *Robots.txt*, and force the users to obey it without giving them the option ignore it totally or partially. Along with forcing minimum waiting time between fetching attempts for files on the same domain, the crawling process using Nutch may end up being slow.

Another popular distribution of Lucene is Apache Solr [4] which provides enterprise search solution on top of Lucene. Solr provides a RESTful like API on top of Lucene, so that all communication with the index is done over HTTP requests which makes Solr embed-able into any application without worrying about the implementation. On top of that, Solr provides distributed searching, query caching, spell corrections, and faceted search capabilities. But as mentioned, Solr is another search framework and not a complete search engine solution. The main missing component is a web crawler. Though it can be plugged into Nutch as the background indexer instead of Lucene. A query interface and results page are also missing, as the shipped interface is for testing purposes only. This is attributed to the fact that Solr is not a standalone application, rather it's a library or framework which get plugged into another application.

NutchWAX [13] attempts to merge Nutch with the web archive extensions such that the solution will search the web archive. Currently it only supports ARC files, though WARC [1] files (standard format for archiving web content) are easily converted to ARC. NutchWAX requires Hadoop platform [2] to run the jobs on it, as the tasks are implemented using the Map/Reduce paradigm [25].

Since Lucene proved to be a scalable indexing framework, many open source search engine adopted it and built solutions on top of it. For example, Hounder [6] not only capitalize on Lucene, but also on some modules of Nutch to provide large scale distributed indexing and crawling services. The crawling and the indexing processes are easily configured, launched and monitored via GUI or shell script. However, the options that can passed to the crawler are limited compared to large scale crawlers like Heritrix, as most of the configurations are regular expressions only. These regular expressions are either entered into a simple java GUI, or appended to the numerous configuration files. The extendibility of the system is not an easy task as well.

Another popular search framework is Xapian [21], which is built with C++ and distributed under GPL license. The framework is just an indexing library, but it ships with web site search application called Omega [14] that can index files of multiple formats. Though the Xapian framework doesn't contain a crawler, Omega can crawl files on the local files system only.

Researchers at Carnegie Mellon University and University of Massachusetts Amherst have developed Indri [40, 8] as a part of the Lemur project [33, 10]. Indri is a language model based search engine that can scale for 50 million documents on a single machine and 500 million documents on a cluster of machines. It supports indexing documents from different languages, and multiple formats including PDF, Word, PPT. In addition to traditional IR models, Indri combines inference networks along with language models which makes it a unique solution when compared to other frameworks. However, Indri doesn't contain a web crawler, and has to be used along with a 3rd party crawler. Nevertheless, Indri can ingest documents from the file system, TREC collections, or archived files in a WARC format. So, the choices for a crawler to use with Indri are large.

Besides indexing web content and intranet documents, some search engines were developed to index SQL databases. Providing full text search for DBMS content is important for enterprises with large databases, especially when the built in full text search is not fast enough. Sphinx [15] provides a full text search solution for SQL databases, as it comes with connectors to many commercial databases. Besides connecting to databases, Sphinx may be configured to index content from XML files which were written in specific format. But, since Sphinx is aimed at indexing SQL databases, it can't be considered as a complete search engine, and rather a SQL indexing framework.

At RMIT university, researchers have developed Zettair [22], an open source search engine which is written in C. Zettair's main feature is the ability to index large amounts of text files [31]. It's been used to index 426GB of TREC terabyte track collection, according to the official documentation page. On the flip side, Zettair can only deal with HTML and plain text files, thus lacking the feature of indexing rich media files. Besides, it assumes the user has already crawled the files, and doesn't provide any crawler out of the box.

Swish-e [16] is another open source search engine which is suitable for indexing small content, less than 1 million documents. It can be used to crawl the web via a provided perl script, and index files from various data-types: PDF, PPT ...etc. But since it can only support up to one million documents, scalability is not a feature of this search engine.

As academia continued to contribute to open source search engines, researchers at the University of Glasgow have introduced Terrier [17, 36, 35] as the "first serious answer in Europe to the dominance of the United States on research and technological solutions in IR" [36]. Terrier provides a flexible and scalable indexing system with implementations of many state-of-the-art information retrieval algorithms and models. Terrier has proven to compete with many other academic open source search engines, like Indri and Zettair, in TREC workshops [36]. To support large scale indexing, Terrier uses MapReduce on Hadoop clusters to parallelize the process. Interacting with Terrier is made easy for almost all users by providing both desktop and web interface. Nevertheless, as the case with many other open source search solutions, Terrier doesn't ship with a web crawler, but it can be integrated with a crawler that was also developed at the University of Glasgow: labrador [9].

MG4J *(Managing Gigabytes for Java)* [23, 11] is a search engine released under GNU lesser general public license. It's being developed at the University of Milan, where researchers are plugging state-of-the-art algorithms and ranking functions into it. The package lacks a fully-fledges web crawler, and relies on the user to provide the files, but it can crawl files on the file system. Despite the numerous advantages of the system, ease of use seems to elude this search engine.

WebGlimpse [20] is yet another search engine, though it has a different licensing model as it is free for students and open source projects, but needs to be licensed for any other use. It's built on top of Glimpse indexer which generate indices of very small size compared to the original text (2-4% of the original text) [30]. WebGlimpse can index files on the local filesystem, remote websites, or even crawl webpages from the web and index them. But the crawler has limited options which makes it incompetent to do a large scale web crawl.

mnoGoSearch [12] is another open source search engine which have versions for different platforms including Linux/Unix and Windows. It's a databases back ended search engine, which implements its own crawler and indexer. Since it's dependent on the databases in the back end, the database connectivity may become the bottleneck of the process in case the number of running threads passed the limit of concurrent open connections to the database. The configuration options of the crawler are also limited compared to Heritrix and Nutch. Besides, indexing rich media files like PDF and Doc is not supported internally, though external plugins can be used to convert these files.

When comparing open source search engines, many aspects are taken into consideration. These aspects include: completeness of the solution in terms of components (example: some libraries don't have crawlers), the scalability of the solution, the extendibility of the search engine, the supported file types, license restrictions, support of stemming, stop words removal, fuzzy search, index language, character encoding, providing snippets for the results, ranking functions, index size compared to the corpus size, and query response time.

Many researchers had done work on comparing the performance of multiple open source search engines. Middleton and Baeza-Yates compared 29 popular open source search engines in [31]. Their comparison considered many of the aspects mentioned before, along with precision and recall performance results on TREC [18] dataset. However, their analysis is more focused on the indexing section of the search engine without considering the crawling process at all. In fact, many of the libraries that they compare are only indexing libraries, and not complete search engines (i.e. Lucene). Another experiment on indexing with open source search libraries was performed in 2009 on data from Twitter [39]. This experiment was conducted on small data which doesn't test the scalability of the indexer. Similar to the study by Middleton and Baeza-Yates [31], [39] doesn ot take into consideration the crawling task of the search engine.

3. ARCHITECTURE AND IMPLEMENTATION

Our design must include the most important parts of a search engine, a crawler and the index engine. YouSeer's architecture is presented in Figure 1. While most of YouSeer's components can be substituted with equivalent open source components, we describe the architecture and the implementation using the components we deploy, without loss of generality of the approach.

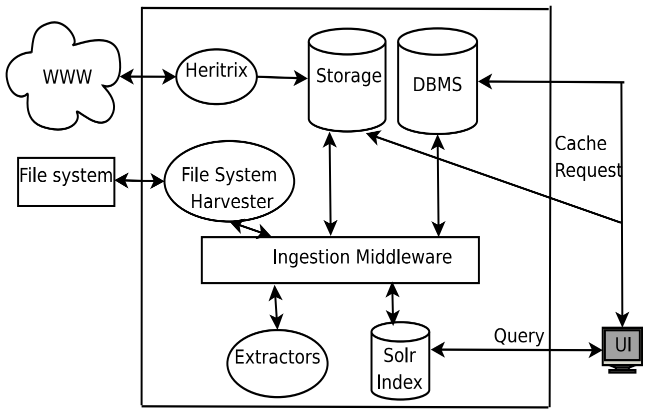

Figure 1: YouSeer Architecture.

The framework is implemented in Java and the interfaces are in JSP.

3.1 Crawler

A web crawler is a software that downloads documents from the web and stores them locally. The process of downloading is sequential where the crawler will extract the outgoing links from every downloaded document and schedule these links to be fetched later according to the crawling policy.

The Internet Archive's crawler, named Heritrix [32], was chosen as a web crawler for YouSeer. Heritrix serves as good example of embedding any web crawler into a search engine since it dumps the downloaded documents to the hard disk in the Warc format, which in 2009 became the standard for archiving web content[1]. By default, Heritrix writes the downloaded documents into compressed ARC files, where each file aggregates thousands of files. Compressing and aggregating the files is essential to keeping the number of files in the file system manageable, and sustaining lower access time.

Heritrix expects the seed list of the crawl job to be entered as a text file along with another file that defines the crawling policy. Then the crawler will proceed by fetching the URLs in the seed list and write them to ARC/Warc files. This process can be assumed to be the standard workflow of any web crawler, thus the integration of Heritrix can be used as example on how to integrate almost any web crawler into a search engine.

Heritrix provides flexible and powerful crawling options that make it ideal for multiple focused crawling jobs. These features include the ability to filter documents based on many deciding rules such as regular expressions, file types, file size, and override the policies per domain. The ability to tune the parameters of connection delay, and control the max wait time along with number of concurrent connections are advantageous when crawling for a vertical search engine. Despite the lack of support for parallel crawling on multiple instances, Heritrix is continuously being used at the Internet Archive to crawl and archive the web which can be argued to be the largest crawl ever to be conducted using an open source crawler. While teaching a search engine class, students have preferred using heritrix over other open source crawlers such as Nutch because Heritrix provides an easy to use web interface to run crawling jobs, and for the richness of the features and the detailed control of the parameters which the students can specify. This helped the students grasp the challenges of crawling the web, while at the same time gave them the chance to monitor how the crawl job is progressing and what parameters they need to change.

Besides the web crawler, YouSeer implements its own local hard drive harvester. This allows it to function as a desktop search engine as well. The crawler runs in a breadth-first manner, starting at a certain directory or network location iterating over the the files and folders inside that folder. This would complement our assumption in the framework that crawlers should produce Warc files, as the local file harvester would enable YouSeer to index documents mirrored onto the file system by different crawlers that do not produce Warc files.

3.2 Indexer

YouSeer adopts Apache Solr for indexing, which provides a REST-like API to access the underlying Lucene index. Dealing with an indexing interface with a RESTful API [26] over HTTP gives a layer of abstraction to the underlying indexing engine, and provides YouSeer with the ability to employ any indexing engine as long as it provides a REST API [26]. Such an API may be built as a wrapper on top of the existing non-web API. Thus, the indexing engine in YouSeer is just a URL with the operations that the index provides. These operations are typically: index, search, delete, and optimize.

Besides the native features of Lucene, Solr provides additional features like faceted search, distributed search, and index replication. All these features combined with the flexibility to modify the ranking function makes a good case for adopting Solr as indexer. In addition, Solr is reported to be able to index 3 billion documents [38].

YouSeer distribution deploys two instances of Solr, one for web documents and another one for files crawled from the desktop. The separation between the instances can be achieved either by having two standalone Solr instances, or two cores deployed on the same instance. Cores are methods for running multiple indices with different configuration in the same Solr instance. By default one core is configured to index content from the web, and the other core is used to index documents on the file system. In the case of multi-core solr, users maintain a single Solr instance, while having the ability to tune each index independently. This becomes a need as field numbers for web content differs from file-system content. More importantly, the ranking of web documents may be far more complicated than ranking file-system content. Since YouSeer is only aware of the URL of the core (a core is treated just like a dedicated index), it can be easily modified to use a dedicated index instead of a core in case the number of documents scales beyond what a single core can handle. Furthermore, Solr distributed search techniques can be used to replace a core when the number of documents grows beyond the capabilities of a single machine. This seamless transition is made possible because all the different distributions of the index (cores, standalone, distributed) provide the same RESTful API, and the ingestion module along with the query interface only care about the server URL without knowing the specific implementation of the server.

3.3 Database

A search engine occasionally needs to store some information in a database. Such information can be for reporting purposes, or needed for performing certain operations. YouSeer uses a database for three reasons: (1) keep track of successfully processed Warc/Arc files in order to avoid processing them again, (2) for storing metadata about cached documents, and (3) to log errors during ingestion. YouSeer uses MySQL as DBMS server, however SQLite was proved to be suitable for small to medium level datasets. YouSeer interacts with the database through JDBC, hence it can adopt any DBMS that has a JDBC driver.

3.4 Extractor

Search engines need to handle files in multiple formats ranging from simple html pages to files with rich content like Word and PDF along with audio and video. Apache TIKA [19] empowers YouSeer with the ability to extract metadata and textual content from various file formats such as PDF, WORD, Power Point, Excel Sheets, MP3, ZIP, and multiple image formats. Tika is currently a standalone Apache project that supports standard interface for converting and extracting metadata from popular document formats. While YouSeer ships coupled with Tika, it's still fairly straightforward to replace it with other converters as needed.

3.5 Ingestion Module

The ingestion module is where the crawled documents get processed and are held to be indexed. The Warc/Arc files are processed to extract the records containing individual documents and the corresponding metadata. Documents of predefined media types are passed to multiple extraction modules such as PDFBox to extract their textual content and metadata. The user specifies the mime types she is interested in indexing by editing a configuration file that lists the accepted mime types. The extracted information is later processed and submitted to the index. This module also stores the document's metadata into the database and keeps track of where the cached copy is stored.

The ingestion module is designed in a such a way that different extractors operate on the document, after that each extractor emits extracted fields, if any, to be sent to the index. By default the *Extractor* class provides all the out of the box extraction and population for the standard fields of the index such as title, url, crawl date and others, while *CustomExtractor* is left for the end user to implement. *CustomExtractor* is called after *Extractor* giving the user the ability to override the extracted fields, and extract new fields. This approach makes it easy for the users to implement their own information extractors. For example, while building a search engine for a newspaper website, the customer asked for providing search capability based on the publication date. The publication date could be extracted from the URL as the newspaper formats its URL as follows:

www.example.com/YYYY/MM/DD/article.html

To achieve this, we implemented the *CustomExtractor* class of the ingestion module so that it would extract the information from the URL and append the extracted date to the xml file which is to be sent to the indexer.

3.6 Query Interface

YouSeer has two search interfaces basic and advanced that provide access to the underlying Lucene index. The basic interface is similar to most search engines where users enter a simple query term before the relevant links are returned. The advanced search provides search fields for each searchable field in the index and allows the users to set the ranking criterion. Query suggestions, aka auto-complete, are displayed for the user while inputting the query terms. These suggestions are generated offline by extracting the word-level unigram, bigram, and trigram of terms in the index. When enough query logs are accumulated, they can be used for query suggestions instead of the index terms.

Furthermore, queries are checked for misspellings using the terms in the index instead of a third party dictionary. This would be suitable in the case of vertical search engines that deal with special domains terminologies.

Since the index is accessed through a REST web service, the query interface receives the query terms from the user and send an HTTP request to the index. The REST API provides a level of abstraction for the interface to communicate with multiple types of indices as long as they provide a similar API.

Along with the query interface, YouSeer provides an admin interface from which users can launch new ingesting jobs and track the progress of previously started jobs.

3.7 Documents Caching

Accessing older versions of some documents, or being able to view them while their original host is down, is considered an advantage for a search engine. The caching module keeps track of the different versions that have been crawled and indexed of a document. As documents are stored into Ward/Arc files, the relative location of the containing Warc file is stored in the index along with the documents fields. In addition to the surrogate file name, the index would contain the offset of the file within the Warc file. The offset is needed because Warc and Arc files can only be read sequentially. The Warc/Arc files are mounted on virtual directory on the web server, therefore they can be accessed over the network allowing them to be located on a different location than the server or the crawler.

When the user requests a cached version for an indexed document, the caching module locates the containing Arc/Warc file and seeks to the beginning of the document's record reading it and returning content to the user. If the requested file is not an HTML document, the module can convert DOC, PDF, PPT, XLS format and other formats into HTML.

YouSeer caching module provides integration with Google Docs preview, so that cached documents can be viewed as a Google document on the fly. The feature works on supported formats only, like PDF, Doc, and PPT. This allows users to quickly view rich media documents without the need to download them. Figure 2 shows the workflow of the caching module.

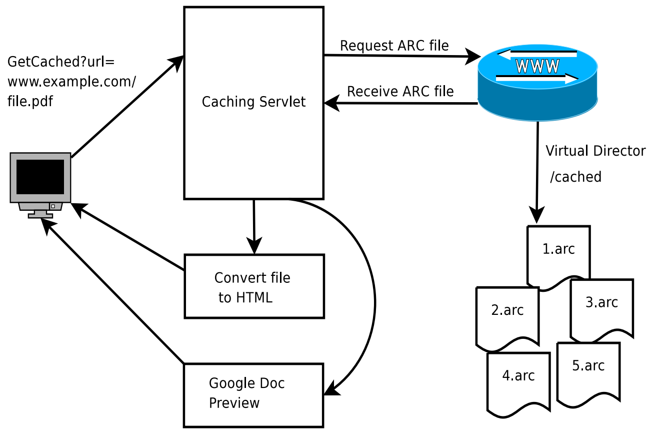

Figure 2: Cache Architecture

4. WORKFLOW

In this section we present an overview of how the whole system works. A typical job starts by scheduling a crawl task on Heritrix. First the seed URLs are provided and the rest of the parameters are defined. These parameters include the max-hop, max file size limit, max downloaded files limit, and other crawl politeness and request-delay values. The crawler proceeds by fetching the seed URLs, extracting their outgoing links, and scheduling these links for an in breadth-first crawl. As part of the parameter specification, the user chooses the format by which the crawled results are written into. The default is Arc, but other file formats such as Warc or simply mirroring the fetched documents on a hard drive are available. Converting the Arc files into Warc format can be accomplished through command line tool. Should the user keep the format as Arc, the downloaded documents are combined and then written to a single compressed ARC file [32], which is in this case limited to 100MB. Along with every document, Heritrix stores a metadata record in the compressed file.

The ingestion module, which is the middleware between the crawler and the index, waits for the ARC/WARC files to be written and then iterates on all the documents within the ARC file processing them sequentially. The ingestion process does not necessarily wait for the crawler to terminate, rather it keeps polling for new files to be written so it can process them. The middleware extracts the textual content from the HTML pages and the corresponding metadata created by Heitrix. For rich media formats such as Word, PDF, Power Point, YouSeer converts the document into text using Apache TIKA. The output of the middleware is an XML file containing the fields extracted from the documents. The URL of each document serves as the document ID within the index.

Each ingestion plug-in contributes to building this XML file by appending its result as an XML tag. The URL of the ARC file, and the offset of the document within the ARC file are appended to the XML file to expedite retrieval.

The resulting XML file from the processing is posted to the index. After processing all the documents within a single ARC file, the middleware commits the changes to the index and marks the ARC file as processed. While indexing, the word-level n-grams are extracted and added to the query suggestion module.

5. EXPERIMENTS

We perform a number of experiments to measure the performance of our proposed framework. The experiments entail crawling the web by focusing on a set of seed URLs then processing the crawled documents in the ingestion module before they are indexed.

In the first experiment, we aim at creating a search engine for the OpenCoursWare , OCW, [1] courses. We compile a seed list of 50 English speaking universities and crawl the seeds with Heritrix 1.14.4. We set a limit of 100,000 to the maximum number of files that can be downloaded. The job finished after reaching 100,000 documents in 4 hours and 17 minutes running 50 threads. We used an out of the box configuration for Heritrix, and only modified the max number of documents to be fetched. The size of the data crawled by Heritrix was 15 GB compressed into ARC files. For comparison, we use Nutch 1.5 to crawl the same seed list. Similar to Heritrix, we used 50 threads for crawling and keep the rest of the configurations to their default values. We limited the hops to 5 and specify the *topN* value at 20,000. *topN* controls the maximum number of pages that can be downloaded at each level of the crawl. This should limit the entire crawl to roughly 100,000 pages. The job terminated after 9 hours, and downloaded 42806 documents. The size of the entire crawl files was 838 MB, including *segments, linkdb, and crawldb.* We guess that Nutch prioritized crawling small HTML documents over PDF and PPT files.

For both YouSeer and Nutch, we used Apache Solr 3.6 as an indexing engine. We start by running Nutch solr indexer, and monitor the process using *Sysstat* [27], which is a popular tool for monitoring system resources and utilization on Linux. As Nutch does not allow specifying the number of threads for ingestion, unlike YouSeer, we started the ingestion command and monitored the threads long with memory and CPU usage through Sysstat. We recorded 16 active threads ran that under Nutch process during ingestion. The entire ingestions and indexing process took 3.35 minutes, that is 199 URLs/second and around 3.8 MBs/second. On the other hand, since YouSeer allows controlling the number of ingestion threads, we used the same number of threads as reported by Sysstat. YouSeer middleware took 15 minutes to process the 100,000 documents. That is 111 URL/second and around 16 MBs/second. Table 1 summarizes the results for OCW search engine experiment. The CPU and memory usage represent the max usage as captured by Sysstat. The machine on which the experiments was ran is Dell workstations with 2 dual core processors and 4 GB of memory. The CPU usage is normalized by the total number of CPUs.

[1]`http://www.ocwconsortium.org/`

Table 1: Comparison of different parameters for ingesting and indexing OpenCourseWare content

Parameter	YouSeer	Nutch
# docs	100,000	42806
Size	15 GB	838 MB
CPU	0.81%	0.25%
Memory	37.44%	14.37%
Time in minutes	15	3.35
URL / Second	119	199
MB / Sec	16	3.8

This experiments shows how YouSeer framework can ingest larger amount of data per second. And since Heritrix crawl jobs can run faster, plugging in different components of search engines seems to yield faster turn around time and larger processing power supporting our idea of utilizing different open source components rather than building all the pieces of a search engine.

In another experiment, we crawled for 20 million documents with 50 threads, this job took less than 40 wall clock hours. One million documents of multiple formats (pdf, html, ppt, doc, etc.) were indexed in less than 3 hours. These experiments were conducted on a Dell server with 8 processors, 4 cores each and 32 GB RAM, running linux.

6. CONCLUSION AND FUTURE WORK

We described the architecture of YouSeer, a complete open source search engine. The approach used for building YouSeer can be extended to support constructing powerful search tools by leveraging other open source components in such a way that maximizes usability and minimizes redundancy. YouSeer a natural fit for vertical search engines and to the enterprise search domain. It also serves as pedagogical tool for information retrieval and search engine classes. The ease of use and flexibility of modification makes adoptable for research experiments. Our experiments shows that YouSeer can be more effective that other complete open source search engines in certain scenarios.

We enumerated the list of open source libraries that the system uses and introduced a middleware to coordinate these modules. The current version of YouSeer is hosted on SourceForge and a virtual appliance box is available for download to eliminate the installation overhead.

In the future, we plan to introduce modules to parallelize the processing and take advantage of the MapReduce paradigm. We also look forward to investigating security models that would protect the data from being access by unauthorized users. Currently we rely solely on the web server and the operating system to provide security mechanisms.

7. ACKNOWLEDGMENTS

We acknowledge partial funding from NSF and the Information Technology Services at the Pennsylvania State University. We also thank Pradeep Teregowda, Mike Halm, and Kevin Kalupson for their contribution.

8. REFERENCES

[1] http://www.iso.org/iso/news.htm?refid=Ref1255.

[2] Apache hadoop. http://hadoop.apache.org/.

[3] Apache lucene. http://lucene.apache.org/.

[4] Apache solr. http://lucene.apache.org/solr/.

[5] Heritrix. http://crawler.archive.org/.

[6] Hounder on google code. http://code.google.com/p/hounder/.

[7] ht://dig. http://www.htdig.org/.

[8] Indri homepage. http://www.lemurproject.org/indri.php.

[9] Labrador homepage. http://www.dcs.gla.ac.uk/~craigm/labrador/.

[10] Lemure homepage. http://www.lemurproject.org.

[11] Mg4j homepage. http://mg4j.dsi.unimi.it/.

[12] mnogosearch homepage. http://www.mnogosearch.org.

[13] Nutchwax. http://archive-access.sourceforge.net/projects/nutch/.

[14] Omega homepage. http://xapian.org/docs/omega/overview.html.

[15] Sphinx homepage. http://www.sphinxsearch.com.

[16] Swish-e homepage. http://swish-e.org/.

[17] Terrier homepage. http://terrier.org/.

[18] Text retrieval conference (trec). http://trec.nist.gov/.

[19] Tika homepage. http://tika.apache.org/.

[20] Webglimpse homepage. http://webglimpse.net.

[21] Xapian homepage. http://xapian.org.

[22] Zettair homepage. http://www.seg.rmit.edu.au/zettair.

[23] P. Boldi and S. Vigna. MG4J at TREC 2005. In E. M. Voorhees and L. P. Buckland, editors, *The Fourteenth Text REtrieval Conference (TREC 2005) Proceedings*, number SP 500-266 in Special Publications. NIST, 2005. http://mg4j.dsi.unimi.it/.

[24] M. Cafarella and D. Cutting. Building nutch: Open source search. *Queue*, 2(2):61, 2004.

[25] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[26] R. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[27] S. Godard. Sysstat: utilities for linux. http://sebastien.godard.pagesperso-orange.fr/.

[28] M. Henzinger, R. Motwani, and C. Silverstein. Challenges in web search engines. In *ACM SIGIR Forum*, volume 36, pages 11–22. ACM, 2002.

[29] R. Khare, D. Cutting, K. Sitaker, and A. Rifkin. Nutch: A flexible and scalable open-source web search engine. *Oregon State University*, 2004.

[30] U. Manber and S. Wu. GLIMPSE: A tool to search through entire file systems. In *Usenix Winter 1994 Technical Conference*, pages 23–32, 1994.

[31] C. Middleton and R. Baeza-Yates. A comparison of open source search engines. In *grid-computing*, volume 1, page 1.

[32] G. Mohr, M. Stack, I. Rnitovic, D. Avery, and M. Kimpton. Introduction to heritrix. In *4th International Web Archiving Workshop*, 2004.

[33] P. Ogilvie, , P. Ogilvie, and J. Callan. Experiments using the lemur toolkit. In *In Proceedings of the Tenth Text Retrieval Conference (TREC-10*, pages 103–108, 2002.

[34] C. Olston and M. Najork. Web Crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.

[35] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and D. Johnson. Terrier information retrieval platform. *Advances in Information Retrieval*, pages 517–519, 2005.

[36] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

[37] A. Patterson. Why writing your own search engine is hard. *Queue*, 2(2):48, 2004.

[38] J. Rutherglen. Scaling solr to 3 billion documents. http://2010.lucene-eurocon.org/slides/Scaling-Solr-to-3Billion-Documents_Jason-Rutherglen.pdf. Apache Lucene EuroCon 2010.

[39] V. Singh. A comparison of open source search engines. http://zooie.wordpress.com/2009/07/06/a-comparison-of-open- source-search-engines-and-indexing-twitter/, 7 2009.

[40] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: a language-model based search engine for complex queries. Technical report, University of Massachusetts Amherst, 2005.