Design Considerations for a Sustainable Scholarly Big Data Service

Jian Wu  
Computer Science  
Old Dominion University  
Norfolk, Virginia, USA  
j1wu@odu.edu

Shaurya Rohatgi  
Information Sciences & Technology  
Pennsylvania State University  
University Park, Pennsylvania, USA  
szr207@ist.psu.edu

Manoj K. Angadi  
Computer Science & Engineering  
Pennsylvania State University  
University Park, Pennsylvania, USA  
mxa5887@psu.edu

Kavya S. Puranik  
Computer Science & Engineering  
Pennsylvania State University  
University Park, Pennsylvania, USA  
kzp5555@psu.edu

C. Lee Giles  
Information Sciences & Technology  
Pennsylvania State University  
University Park, Pennsylvania, USA  
clg20@psu.edu

ABSTRACT

The advancement of web programming techniques, such as Ajax and jQuery, and datastores, such as Apache Solr and Elasticsearch, have made it much easier to deploy small to medium scale web-based search engines. However, developing a sustainable search engine that supports scholarly big data services is still challenging often because of limited human resources and financial support. Such scenarios are typical in academic settings or small businesses. Here, we showcase how four key design decisions were made by trading-off competing factors such as performance, cost, and efficiency, when developing the Next Generation CiteSeerX (NGX), the successor of CiteSeerX, which was a pioneering digital library search engine that has been serving academic communities for more than two decades. This work extends our previous work in Wu et al. (2021) and discusses design considerations of infrastructure, web applications, indexing, and document filtering. These design considerations can be generalized to other web-based search engines with a similar scale that are deployed in small business or academic settings with limited resources.

CCS CONCEPTS

• Information systems → Search engine indexing; Document filtering; Digital libraries and archives.

KEYWORDS

digital library search engine, citeeseerx, infrastructure, architecture

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

FIRE '22, December 9–13, 2022, Kolkata, India  
© 2022 Copyright held by the owner/author(s).  
https://doi.org/10.1145/3574318.3574340

1 INTRODUCTION

Google has dominated the search engine market for more than 20 years. However, a general search engine like Google may not satisfy all search needs from users. Vertical search engines have played vital roles in searching focused document collections. Digital library search engines (DLSEs) are a type of vertical search engine that serves digital academic documents. Examples of DLSEs included CiteSeer [10], Microsoft Academic ([21]; retired in 2021), Google Scholar [12], and Semantic Scholar [9]. Google Scholar obtains its documents from Google with a large portion of metadata directly from publishers or automatically extracted using a proprietary extractor. Google Scholar does not cache documents. Instead, the search engine result page (SERP) redirects users to URLs where the documents could be downloaded, such as CiteSeerX. Semantic Scholar maintains a copy of open-access papers but the search architecture is proprietary and unlikely to be deployed for building institutional DLSEs, most likely due to the highly customized design and running cost. CiteSeerX released a framework called SeerSuite [20], which could be used for deploying a DLSE for an institutional document collection. With the updates of software, such as operating systems (OSs) and web development frameworks, many components that were used for building SeerSuite exhibit security vulnerabilities and/or no longer supported. The infrastructure that was used to support relatively medium-size document collection has shown scalability bottlenecks, and is no longer suitable for a sustained service for scholarly big data. Therefore, it is necessary to design a new framework to provide an accessible, usable, scalable, and sustainable scholarly big data service. The new framework, called the Next Generation CiteSeerX (NGX), is designed as a general framework that can be deployed for other digital documents.

Previously we [23] analyzed the strengths and weaknesses of the CiteSeerX design, and proposed a new design for the NGX with a revised architecture, enhanced hardware, and software framework. In this paper, we extend [23] and focus on several key design considerations for the frontend and backend. We compare different design options and justify our choices based on analytical or experimental results. Our goal is to present these design decisions and articulate the factors to consider when making trade-offs so as to justify our decisions. Although these decisions were made for
Table 1: Major hardware used for deploying NGX. ES: Extraction-Ingestion System. Repository is on a storage array network (NAS) mounted to other servers.

NGX, we feel such designs will be appropriate in the development of other vertical similar scale search engines.

2 RELATED WORK

Early vertical search engines were relatively small scale due to hardware and software constraints, e.g., the vage image search engine [2]. After 2000, geographic search engines received serious attention from major search companies. In [16], the design considerations of building a geographic search engine are discussed, such as the choice between vector data model and a raster data model. In [13], design considerations are outlined when migrating from CiteSeer to CiteSeerX, where a new data model and stratified architecture is introduced. In [4], design considerations are discussed for search engine caching, although not limited to vertical search engines. Since 2010, medical search engines and large scale DLSEs architectures were introduced. For example, the iMed designers incorporated query interfaces and investigated how to make it easier for users to find answers [15]. Similarly, the designers of Yale Image Finder focused on backend designs featuring customized layout analysis over images published in academic journals [28]. In [19], comparing and making choices of AI technologies was discussed for Arnetminer, a social-oriented academic search engine. In [26], a new framework was proposed that incorporated different design factors aiming at building a scalable and maintainable cloud-based scholarly big data platform. Many ideas were adopted in the current CiteSeerX system. Recently, knowledge graphs are used for powering search engines. For example, Huang et al. discussed design considerations for building an oil gas information intelligent search engine, including the choice of ontology, the named entity recognition model, and integration between Neo4J and Elasticsearch (ES) [11].

Our work is different in the sense we focus on comparing different design considerations when migrating a large-scale production system to a new infrastructure, which is hosted in an academic setting with relatively limited resources. How to balance the usability, maintainability, scalability, and sustainability is challenging. Please note that we use NGX to identify the new system and CiteSeerX to identify the current system.

3 NGX DESIGN OVERVIEW

To support scaling up the document collection and the new design, we deploy the NGX on a new hardware infrastructure, shown in Table 1. The new architecture contains a frontend and a backend.
The system was designed to process document batches using all JavaScript to automatically generate user interfaces and has been running stably on several VMs for about 10 years. The web application is composed of servlets that correspond to user queries and has been deployed in a distributed system. The first question to ask is whether NGX should be hosted in a private cloud or a physical cluster.

The decision was to host the system in a hybrid infrastructure. Specifically, we host the web servers, three master nodes of the ES cluster, and the static web server in the cloud. These servers carry relatively low computational load and consume a relatively less IO bandwidth. The performance of these servers is unlikely to be affected by the shared disk channel. We host the EIS server, the web crawling server, and the ES data nodes in a loosely coupled physical cluster. Jobs running on these servers are usually computationally expensive and IO extensive. For example, it was shown that it took more than twice as long to finish a batch crawling job on a virtual machine (VM) versus on a physical server [25].

One trade-off we needed to make was the availability vs. performance because over a long time, disks may fail and one is now at risk of losing data. The problem is less concerning for VMs because the cloud hypervisor can dynamically allocate resources and send warnings for replacing failed disks. For physical servers, disk failure can be catastrophic and bring the system down. To make the VMs in the cloud more sustainable, we expanded the current cloud infrastructure with new physical servers. Existing hardware in the cloud has been running for about 10 years, so we expect the expanded cloud to last comparably long. To mitigate the potential risks of data loss on the ES data nodes, we employ hot-plug solid state disks (SSDs) instead of hard drives (HDDs) because SSDs have exhibited improved performance and durability [17]. ES also shards data across four data nodes to ensure horizontal scalability and high availability.

4 DESIGN CONSIDERATIONS

4.1 Infrastructure: Physical vs. Cloud

CiteSeerX has been running in a private cloud since around 2013 [25]. The cloud offers several advantages such as elasticity and maintainability. For example, cloning a new virtual machine takes only a few minutes. Note that NGX will host at least 30 million academic documents with a high throughput. Because of this, it needs to be deployed in a distributed system. The first question to ask is whether NGX should be hosted in a private cloud or a physical cluster.

The decision was to host the system in a hybrid infrastructure. Specifically, we host the web servers, three master nodes of the ES cluster, and the static web server in the cloud. These servers carry relatively low computational load and consume a relatively less IO bandwidth. The performance of these servers is unlikely to be affected by the shared disk channel. We host the EIS server, the web crawling server, and the ES data nodes in a loosely coupled physical cluster. Jobs running on these servers are usually computationally expensive and IO extensive. For example, it was shown that it took more than twice as long to finish a batch crawling job on a virtual machine (VM) versus on a physical server [25].

One trade-off we needed to make was the availability vs. performance because over a long time, disks may fail and one is now at risk of losing data. The problem is less concerning for VMs because the cloud hypervisor can dynamically allocate resources and send warnings for replacing failed disks. For physical servers, disk failure can be catastrophic and bring the system down. To make the VMs in the cloud more sustainable, we expanded the current cloud infrastructure with new physical servers. Existing hardware in the cloud has been running for about 10 years, so we expect the expanded cloud to last comparably long. To mitigate the potential risks of data loss on the ES data nodes, we employ hot-plug solid state disks (SSDs) instead of hard drives (HDDs) because SSDs have exhibited improved performance and durability [17]. ES also shards data across four data nodes to ensure horizontal scalability and high availability.

4.2 Web Application: Modify vs. Refactor

The CiteSeerX web application was developed on top of Java and JavaScript to automatically generate user interfaces and has been running stably on several VMs for about 10 years. The web application is composed of servlets that correspond to user queries and interact with the index and database servers for keyword-based search, metadata retrieval and browsing, and file download. One consideration is whether to develop the web application by modifying the existing one or refactoring to a entirely new framework.

We decided to refactor the web application. This decision was made after consulting with a chief-technology officer of an insurance website. The decision was also made by trading-off between compatibility and complexity. Modifying the current version may be less complicated as many features have already been implemented. On the other hand, the current application contains several key compatibility issues that must be fixed. The rationales behind this decision include the following.

First, many components used to build the CiteSeerX web application are out of date or not being maintained. For example, CiteSeerX employed log4j (version 1.2.17), which has not been supported since 2015. Updating log4j will impact all Java programs that rely on this package. Another key issue is the OS version compatible with the web application. RHEL6/7 and CentOS 6, are no longer supported. To continue running the application on these operating systems would expose the application to multiple security vulnerabilities. Although CiteSeerX can be deployed on RHEL7/CentOS 7, the current application only supports 128-bit ciphers. TLS (Transport Layer Security) handshakes using all known versions of 256-bit cipher suites fail with the current version of the web application. Fixing this issue requires us to reexamine all components.

Second, the architecture change calls for significant corresponding changes at the frontend. One major change is the removal of the relational database. The CiteSeerX implementation relied heavily on the Data Access Object (DAO) pattern in Java. The DAO pattern is a structural pattern that allows web developers to isolate the application layer from the persistence layer (the relational database in our case) using an abstract API. Since the database is removed, this design pattern was unnecessary. In addition, the current search index was by Apache Solr but the NGX frontend only interacts with ES for both search and metadata retrieval. Cleaning the DAO pattern and changing the search API from Solr to ES would take a remarkably long time and errors may be accidentally introduced.

4.3 What to Index and Top $k$

Returning the most relevant documents to users is a key function of a DLSE. CiteSeerX implemented the search function using Apache Solr, which by default used a slightly modified version of BM25 [18]. The algorithm returned decent results with a sub-second time scale. Elasticsearch adopts BM25 as the default search algorithm, but as the collection increases, researchers face the challenge of finding the exact research papers from a massive amount of documents using limited text. One way to overcome this challenge is to use a two-stage ranking mechanism [27]. The first stage is usually a naive bulk retrieval using BM25, aiming at a high recall by retrieving the top $k$ results and the second stage uses a learning-to-rank model to fine-tune the ranking of documents retrieved in the first stage, [27, 29] thus increasing the precision. Therefore, determining a proper value of $k$ not only reduces the time spent in the first stage but also on the second stage.

A related design decision is whether we should index the metadata only or index the metadata and the full text. The metadata
includes titles, authors, years, venues, and abstracts. This decision is unimportant when the document collection is relatively small, but as the number of documents grows to over 30 million - our goal, it is important to consider how to make a trade-off between the users` experience (response time) and information capacity of the search engine, with limited resources.

We decided to index the metadata only and set \( k = 500 \) in the first retrieval stage, given our infrastructure (see Table 1). We demonstrate this using a proof-of-concept experiment on a simplified setting consisting of a single node server and a client hosted on a separate computer. We compared the API response time of querying against all documents (corresponding to the "match_all" filter) indexed by two ES instances, each built for the same 1 million documents. One instance indexed only the metadata and the other indexed both metadata and full text. We submitted single keyword queries and requested the API to return the top \( k \) \((k = 500 \text{ or } 1000)\) documents with their content. The results shown in Table 2 indicate that indexing metadata only with \( k = 500 \) took only 1/3 time for indexing metadata and full text. The retrieval time when \( k = 500 \) is about 57 – 66% of the time when \( k = 1000 \). In particular, the retrieval time when indexing both metadata and full text exceeded 1 second \((k = 500)\) and 2 seconds \((k = 1000)\). Therefore, we decided to index the metadata only with \( k = 500 \). Note that the exact query time depends on the number of terms, the logical operator, and the filters used. The difference of experimental results in Table 2 demonstrates the advantage of our choice under the same setting.

### 4.4 Which Academic Filter to Use?

Similar to CiteSeerX, NGX also implemented a focused web crawler that actively harvests PDF documents from the Web. However, only a fraction of PDF documents are academic papers and that proportion depends on seed URLs ranging from 30% to 50% [24]. Therefore, an efficient academic filter is crucial to build a relatively clean collection. Previously, CiteSeerX used a rule-based academic filter, which simply checked if the PDF documents contain the word "reference", "bibliography" and their variants. Due to the simplicity of this rule, the F1 measure of this method was only 0.77 and the extraction pipeline introduced many non-academic PDF documents, such as curriculum vitae or resumes [6]. To improve this, we developed a machine-learning-based method, incorporating four types of structural features, such as FileSize, PageCount, DocumentLength, etc. This method achieved a much higher performance with F1=0.90 tested on a corpus of randomly selected crawled PDFs [5]. Recently, Internet Archive developed an academic filter Trio [1, 3]. Different from [5], Trio was an ensemble model that incorporates three different classifiers, a BERT-based classifier [8], an image-based classifier, and a linear classifier. Each classifier computes a classification confidence between 0 and 1. The design question is whether we should adopt the academic filter by [5] or Trio.

To answer this question, we compared the performance of these two classifiers on two benchmark datasets [5], both consisting manually labeled PDFs crawled between 2008 and 2013. The difference is that one contains 1009 PDFs and the other contains 1000 PDFs. In the experiment, we test the academic filter by [5] and Trio on a task to classify PDF documents into academic vs. non-academic documents. Here, academic documents include conference papers, journal articles, technical reports, books, theses, dissertations, slides, and abstracts. The performance is shown in Figure 2. The result indicates that the Academic Filter by [5] outperforms Trio by at least 3% (for Sample B) and at least 15% (for Sample A). Therefore, we decide to continue using the Academic Filter [5].

### 5 CONCLUSION

We showcased four design considerations and justified their use. These include a hybrid infrastructure, a refactor of the web application, indexing the metadata only as opposed to the full text, retrieving the top \( k = 500 \) documents as candidates in a two-stage retrieval system, and employing a machine learning based academic filter based on structural features. These design considerations are not exhaustive but represent key aspects to consider for design of production systems of a similar scale.

### ACKNOWLEDGMENTS

We gratefully acknowledge NSF partial support (Award #1823288).
REFERENCES


