Building an Accessible, Usable, Scalable, and Sustainable Service for Scholarly Big Data

Jian Wu*, Shaurya Rohatgi†, Sai Raghav Reddy Keesara‡, Jason Chhay‡, Kevin Kuo‡, Arjun Manoj Menon‡, Sean Parsons§, Bhuvan Urgaonkar‡, C. Lee Giles†‡
*Computer Science, Old Dominion University, Norfolk, VA, United States
†Information Sciences and Technology, Pennsylvania State University, University Park, PA, United States
‡Computer Science and Engineering, Pennsylvania State University, University Park, PA, United States
§Microsoft Inc., Redmond, WA, United States
j1wu@odu.edu, {szr207,giles}@ist.psu.edu, bhuvan@cse.psu.edu

2021 IEEE International Conference on Big Data (Big Data) | 978-1-6654-3902-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/BigData52589.2021.9671612

***Abstract*—Since the emergence of scholarly big data, there have been several efforts for web-based services such as digital library search engines (DLSEs). However, much of the design and specifications of an accessible, usable, scalable, and sustainable DLSE have not been well represented and discussed in the literature. We argue that these four characteristics are essential to providing a high-quality service for scholarly big data from both the user and developer's perspectives. This paper reviews the design, implementation, and operation experiences, and lessons of CiteSeerX, a real-world digital library search engine. We analyze the strengths and weaknesses of the current design, and proposed a new design with a revised architecture, enhanced hardware, and software infrastructure. The Alpha version of the new design has been implemented and tested. The new system replaces MySQL and Apache Solr with a single instance of Elasticsearch, which plays a dual role of data storage and search. Another major improvement is the integration of extraction and ingestion, which significantly boosts document ingestion speed. The web application is re-engineered to enhance the user experience by applying a learning-to-rank model and offering more refined search tools. The system is also improved in many other aspects. We believe the design considerations and experience can benefit researchers and engineers who plan, design, and upgrade future systems with comparable scales and functionalities.**

***Index Terms*—scholarly big data, big data infrastructure, big data search, big data service**

I. Introduction

A digital library is an information system that usually indexes a large collection of digitized documents and provides web-based or API services for users to search, retrieve, browse, and download documents that are maintained locally or remotely. Like many other big data systems [8], such a system is usually comprised of various modules, each responsible for different tasks. To provide high-quality services, such a system needs to be accessible, usable, scalable, and sustainable. Being accessible requires that all or part of the services are still available upon unexpected failures, e.g., a web server is down. Being usable requires the system to provide multiple interfaces for users to access data, such as a user-friendly web interface, a search API, and an OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) service [9]. Being scalable requires the system to be capable of ingesting a large number of documents and still responding to requests from high volumes of user traffic, both fulfilled within a reasonable time. Being sustainable requires the system to be functional with relatively low cost on hardware, software, and maintenance.

Many institutional digital libraries employ off-the-shelf open-source frameworks such as *DSpace*, *Fedora Commons*, and *Blacklight*. DSpace or Ferora Commons provides an underlying architecture for digital *repositories*, but lacks complete management, indexing, discovery, and delivery applications. Blacklight is a Ruby on Rails engine for creating search interfaces on top of Apache Solr. Although it is straightforward to deploy a web application out-of-box, designing the ingestion pipeline and customizing the discovery interface (such as finding similar papers) is nontrivial. Commercial digital library services such as Google Scholar (GS), Microsoft Academic (MA), and Semantic Scholar (S2) rely on the parent search engines and/or contractual data services with publishers and therefore unlikely to be available as a general public framework. The operating cost of these digital library services is generally high and not affordable for institutions with limited budgets.

In this paper, we share our experience and lessons of operating CiteSeerX, a real-world DLSE, launched 20 years ago and has continuously evolved and improved since then. Different from commercial digital library services, CiteSeerX has been built and maintained under an academic setting, with a limited budget and human resources. The source code is open-source[1]. The current system is relatively stable with both pros and cons. We analysed the architecture and infrastructure and identified challenges it faces to provide a reliable service for a much larger volume of scholarly big data. To overcome these challenges, we have designed a new architecture to fulfill the above requirements. The components of the new architecture include open-source software frameworks and customized workflows. We would like to emphasize that this is an experience paper. We are not reporting any new research. Rather we are describing the unique perspective of our academic team in the engineering choices we made over

We thank partial support from the National Science Foundation and the Sloan Foundation.

[1]https://github.com/SeerLabs/CiteSeerX

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

a period of more than a decade to sustain and evolve a system catering to real users.

II. Related Works

The inception of DLSEs can be traced back to the end of the 20th century. CiteSeer, originally launched in 1998 and renamed "CiteSeerX" in 2008, was one of the pioneers that implemented the automated indexing technique to connect research papers as a network, which made it possible for users to find related papers using citation graphs [14]. Google launched GS in 2004 [35], with its index derived from a crawl of full-text journal content provided by both commercial and open access publishers and author homepages. Being a digital library featuring completeness, GS has been regarded as one of the major data sources for scientists and evaluators to evaluate academic contributions using citation counts and h-index. Microsoft launched the Windows Live Academic Search around 2006, which was the progenitor of MA [30], [37]. MA once claimed 248 million publications[2] indexed. In 2015, the Allen Institute for Artificial Intelligence (AllenAI) launched S2, which claims 196 million paper records indexed at the time of writing. S2 integrates many AI techniques in machine learning (ML), deep learning (DL), and natural language processing (NLP). Another example that features modern NLP techniques is IBM's Science Summarizer [11]. There are also numerous digital library infrastructures built for specific communities, such as arXiv [15] and its sister websites (e.g., bioXiv, AgriRXiv, and MedRxiv), NASA's ADS, NCBI's PubMed [16], and DOE's OSTI system. Another type of DLSEs works as a mixture of digital libraries and social networking websites, some soliciting documents from users and offering various social networking features. Examples of this type include ResearchGate [28] and AMiner [31].

Although there are numerous DLSEs online, to the best of our knowledge most papers published on case studies and comparative studies have been focused on data services, e.g., [12]. There are few publications about advanced architectures of DLSEs and the development and operation of these systems. Entlich et al. (1997) described the architecture and content of the CORE (Chemical Online Retrieval Experiment) project, a library of primary journal articles in chemistry [10]. Lim & Lu (1999) described Harp, a distributed query system for legacy public libraries, which was based only on relational databases [25]. Lagoze et al. (2002) reported the technical and organizational infrastructure of the National Science, Mathematics, Engineering, and Technology Education Digital Library (NSDL) [24]. NSDL supported mainly the metadata discovery of digital items through an OAI-PMH portal. The content was also accessed using open network protocols such as HTTP or FTP. The architecture and infrastructure used by CiteSeerX are described in [33], [44]. Wu et al. (2014) introduced a big data platform that provided various services for harvesting scholarly information and enabling efficient scholarly applications in a private cloud, which outlined the

[2]https://academic.microsoft.com/home

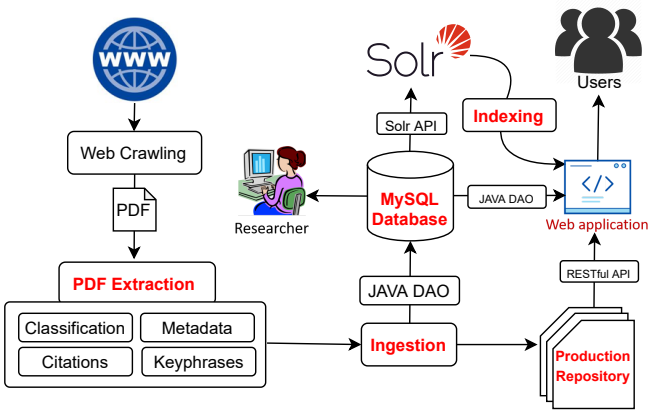

Fig. 1. The top level architecture of the current system. Components with red labels are subject to significant improvement in the new system.

basic components of the current CiteSeerX system [46]. The architecture of ArnetMiner (now called AMiner) is shown in [32], which consists of 5 main components: extraction, integration, storage and access, modeling, and search. The system featured an author profile generator and used MySQL for storage and indexing. Xia et al. (2017) reviewed big scholarly data management systems and depicted a general architecture of a digital library, consisting of the following modules: web crawling, document extracting, information extraction, filtering and categorizing, database, repositories, data discovery, and sharing followed by data analysis [47]. There are no papers reporting the architecture and infrastructure of many well-known DLSEs such as GS, MA, and S2. There are several reasons for this. First, many DLSE designs are proprietary and not built for general purposes. Second, the cost of maintaining the hardware and software infrastructure is relatively high. However, understanding and investigating the architecture of an information retrieval system will help people to understand the usability of the output data by examining how the data are manipulated throughout the pipeline. The experiences and lessons will also benefit future efforts in building more robust and reliable systems.

The paper is written in a comparative style. We first describe the development and usage of the current system (Section III) and then summarize the lessons learned in Section IV. Then we describe our work on improving the current system (Section V). We discuss the limitations of the new system in Section VI.

III. Design and Usage of the Current System

A. Architecture Overview

The current system (Fig. 1) is a typical mid-size crawl-based DLSE [47]. The raw documents are obtained by actively crawling open access (OA) PDFs from the Web. These PDFs are then converted to text and classified, and only academic documents are retained. A pipeline of learning-based extractors is then applied for extracting full text, metadata, citations, keyphrases, acknowledgments, and non-textual information, e.g., figures and tables. Metadata is populated into a relational

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

database (MySQL) and various types of files are saved into the production repository. The metadata in MySQL and the full text from the repository are indexed by Apache Solr, which powers the searching function. The system provides a web-based user interface, an OAI-PMH API, and a database dump on Google drive for research purposes.

B. Development History

The history of our system can be divided into three phases [42]. In the first phase, a prototype was developed around 1998. The major components, including a web crawler, the indexer, and the search API were written in Perl or C++, with custom indexing and storage routines that implemented the automatic citation indexing technique [14]. The seed URLs were manually curated homepages of computer scientists. The entire system was hosted on a single machine. The overall traffic was relatively low but was trending.

In the second phase, the search engine was expanded to a multi-server system. Specifically, multiple web servers with a front-end load balancer were added to make the system more available and scalable to higher incoming traffic. Apache *Lucene* and later Apache Solr was introduced as the main indexer, making the searching more scalable and stable. The backbone software was rewritten using Java. The web application was developed using a model-view-controller (MVC) architecture. The frontend used a mixture of Java server pages and JavaScript to generate user interfaces. The web application was composed of servlets that interacted with the index and database for keyword search and used the Data Access Objects (DAO) to interface with databases and the repository. The metadata extraction was built in Perl, working in batch mode. The ingestion system fed data into a MySQL database and global network block device (GNBD) shared by web servers. The GNBD uses the Global File System (GFS) as its file system. The documents were acquired using an incremental web crawler developed using Django.

The system was running on a cluster of loosely coupled physical servers, hosting nearly a million documents, and was able to handle all incoming traffic during that period. After 4–5 years, the hardware became stale and was prone to fail, typically hard drives and RAID controllers. A single hard drive failure could be restored because we configured the disk arrays with RAID 5. However, multiple hard drives and RAID controller failures could still happen and were potentially disastrous. At the same time, the GFS system plus the GNBD often unexpectedly fenced out web servers and/or repository servers and brought the whole system down. The physical servers were also rigid to scale up on commodity hardware.

In the third phase, which is the current phase, the search engine was migrated into a private cloud [44]. The software was inherited from the second phase. Servers hosted inside the private cloud were monitored by ESXi, a virtual environment hypervisor developed by VMware for deploying and managing virtual computers. This major upgrade of the infrastructure made the whole system more elastic and easier to monitor.

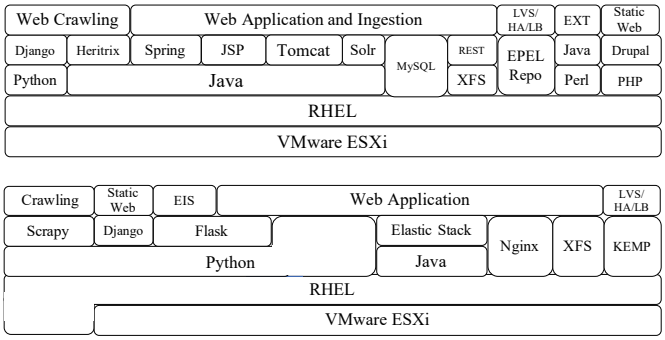

Fig. 2. Software stack of the current (top) and new system (bottom). Meanings of synonyms are: LVS – Linux virtual service; HA – high availability; LB – load balancer; EXT – extraction; RHEL – Red Hat Enterprise Linux.

We elaborate upon the infrastructure, operation, usage patterns, and research uses in the subsections below.

C. Hardware and Software Infrastructure

The current infrastructure consists of three layers: a storage layer, a processing layer, and an application layer. The *storage layer* is composed of two physical servers whose sole purpose is to store all types of data used by virtual machines (VMs). Each server has 12 cores, 32GB RAM, and 30TB SATA hard drives (HDDs). The *processing layer* consists of five high-end servers connected to the storage layer, each having 12 cores, 96GB RAM, and 1TB SATA HDDs. The *application layer* consists of various VMs overviewed by VMware ESXi 6. By using a template-based workflow in a virtualized architecture, setup time for new VMs has been reduced from a day to a matter of minutes. Currently, all frontend and backend servers are VMs except the web crawler server. This is because empirically, the time it takes for our web crawler to finish a batch crawl is much shorter on a physical server than on a VM. That is, the overheads of virtualization are particularly hurtful to crawler performance and, therefore, we host our crawler on a physical machine.

The load balancer uses *heartbeat-ldirectord*[3] to provide the Linux Virtual Service (LVS) that carries the virtual IP of the landing page. Our system employs Heritrix for web crawling, Tomcat for web service, MySQL for database management, Solr for searching, Apache PDFBox for text extraction, and a number of AI-powered software packages for classification [4], metadata extraction [18], near-duplicate detection [39], and author name disambiguation [34]. The system also provides APIs based on SOAP/WSDL, which exposes all the functionalities for programmatical access. The system provides a search API, which allows software programs to access search results using Atom or RSS feeds through an OAI interface. We use the XFS file system and implement a RESTful API that retrieves files from the repository server to the web server. The users can access these data and perform searching, browsing, downloading through a web-based interface. Django and Drupal frameworks were used for building websites that

[3]https://www.suse.com/c/load-balancing-howto-lvs-ldirectord-heartbeat-2

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

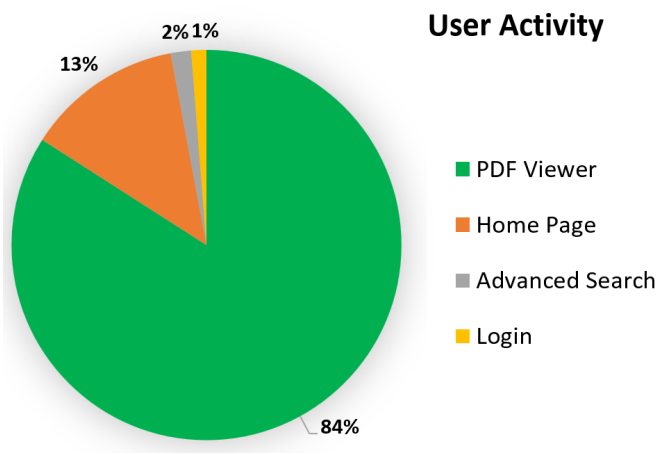

Fig. 3. Usage based on page views from July 2017 to December 2020. Most users are directed to the PDF viewer page for an article by search engines, which index the text from the article hosted by our system.

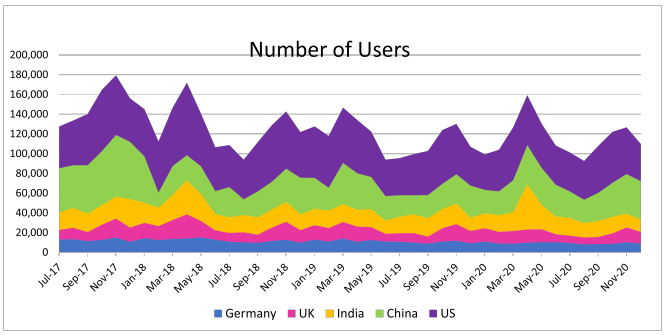

Fig. 4. Number of users from the top-5 countries based on number of visits from June 2017 to December 2020. Spikes are seen during April and November of every year, which are aligned with the end of semesters in most US universities.

display web crawling statistics. The software stack of the current system is shown in Fig. 2 (top).

D. Usage Patterns

The current system has served more than 23 million page views between 2017 and 2020, out of which 18 million pages views were unique (Fig. 3). The system served more than 9 million unique users, with more than 1 million returning users. Fig. 4 shows the timeline of usage in terms of the number of visits for top-5 countries. The number of US users peaked at 73,271 in April 2018. The US generally has the highest number of visitors, followed by China and India.

The current system has enabled much research and education by making research documents, associated metadata, and frameworks publicly available. The system receives weekly requests for metadata and other datasets, most of which are from research groups. The OAI is accessed approximately 100,000 times daily. The data has been used for a variety of research topics such as document type classification [4], keyphrase extraction [13], citation recommendation [20], collaborator recommendation [6], and citation context recommendation [5].

E. Availability and Security

High availability and security are crucial for us to provide a 24/7 online service. We adopted several strategies below.

a) Load Balancing: The entry point contains a pair of twin load balancers, one active and the other standby. If the active load balancer fails, the standby load balancer automatically takes over the virtual IP and the traffic. All servers are behind an institutional firewall so physical IPs or MAC addresses are not exposed to the outside traffic. The system contains three web servers with the same hardware specifications and web application deployment. The load balancer distributes incoming traffic across the three web servers using the weighted least connection algorithm.

b) Data Redundancy: The system contains three database servers, two slaves replicating a master. The master database handles all metadata lookups. One of its replicas supports the OAI service. The other stands by in case the master fails. The system also contains two Solr servers in the master-slave setting. The system has a backup repository that syncs to the master repository periodically. Once the master fails, the backup is used in the read-only mode. All disks are configured with RAID 5, which allows one hard drive failure in a disk array. RAID 5's distributed parity evens out the stress of a dedicated parity disk among all RAID members. Write performance is also increased since all RAID members participate in the serving of write requests.

c) Traffic Control: Iptables is a command-line firewall utility that uses policy chains to allow or block traffic. When a connection tries to establish itself on the system, iptables looks for a rule in its list to match it to. If it does not find one, it resorts to the default action. We apply a chain of sophisticated Iptables rules on the load balancer and web servers. One usage is to block certain IP addresses that attempted to send brute-force requests without respecting robots.txt, which is a plain text file telling search engine crawlers which pages or files a crawler can or can not request from a website. Because of the large number of download requests, by default, the system applies a threshold of up to 3000 downloads from a single IP address, which we consider sufficient for individual users. The web application controller maintains an allow-list that gives unlimited access to certain user-agents, such as *googlebot*. We also specify the "Crawl-delay" in the robots.txt file to curb the request cadences from web crawlers. The web application also uses a custom module to filter out any tags embedded in the queries to prevent malicious code injected into the system, such as cross-site scripting (XSS).

d) Activity Monitor: A dedicated VM is setup to periodically send direct requests to the web, database, index, and repository servers. If the monitor does not receive expected responses within a certain time (which varies depending on the type of servers), the monitor will email the administrators to resolve the issues. This method was proven effective at alarming the administrators at high traffic volume, Linux kernel panic, or hardware failures.

The strategies above proved to be useful to achieve high availability and security. However, some of them needed to

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

be adjusted to accommodate changes in the architecture of the new system, as we will describe later. The load balancer uses *heartbeat-ldirectord* installed on a VM. Although it has been working well, installation and configuration has a steep learning curve and upgrading it with the OS is non-trivial. It also depends on the server status, which may not be reliable. Because of these reasons, the software-based load balancers will be replaced by a hardware load balancer, which is more reliable. Because of the adoption of Elasticsearch, data are shared and distributed across multiple servers, which tolerates one or multiple servers (depending on the cluster size and the number of shards) to fail without bringing the system down. RAID is still necessary for most servers except for the index servers that already provide data redundancy by sharding data. RAID 10 will be adopted for certain mission-critical servers with many HDDs in the array, such as the extraction server, allowing a maximum of two drives to fail.

IV. Pros and Cons of the Current System

A. Strengths

The current architecture has the following strengths. First, the system is based on the LAMP (Linux-Apache-MySQL-PHP) architecture, except that Apache was replaced by Tomcat and PHP was replaced by JavaScript and Java. The MVC design pattern has been widely adopted and has proven reliable for various sizes of web applications. The major tools are open-source and have a large user population. Therefore, it is relatively easy to find solutions to technical problems from online forums such as Stackoverflow.com.

On the backend, the metadata and citation information is first ingested into MySQL and then into Solr, which was a natural extension of the standard LAMP architecture. The role of a search platform like Solr was crucial for a DLSE because MySQL could not effectively handle full-text queries. Separating data storage and search was a reasonable approach when the data size was not very large (below 10 million documents). As seen in Section III-E, the current system adopted different ways to increase the redundancy to avoid the single-point-of-failure.

In the web crawling cluster, document metadata are standardized, which allows the system to import data harvested with different formats and schema of output. Saving the original downloading URLs was necessary for both re-crawling and legal purposes. This helps users to file the Digital Millennium Copyright Act (DMCA) claims.

B. Weaknesses

However, the current architecture also suffers from several weaknesses. Although some of them are temporarily bearable, ignoring them in the long term has the potential to cause catastrophic damage such as permanent data loss or unacceptable user experience. The major challenges are caused by the increasing size and growth rate of academic documents. Table I shows the current and projected numbers of files.

First, the ingestion module is a key bottleneck to increasing document collection. The throughput of the web crawler, the extraction, and the ingestion modules are approximately 500k, 200k, and 50k documents in a day, respectively. The ingestion system is slow because (1) it is single-threaded and (2) the near-duplication detection module needs to first read from the MySQL database, whose hit rate (the proportion of keys that are being read from the key cache in memory instead of from disk) decreases gradually as the data size increases.

TABLE I
The current and projected data size. M=million.

Type	Current	Future	Notes
Full text papers	10M	35M	
Database size	550GB	1.8TB	MySQL
Largest table (rows)	250M	875M	
Database dump	300GB	900GB	.sql
Database buffer	38GB	128GB	10% cached
Indexed records	70M	245M	Apache Solr
Index size	360GB	1.2TB	
Index heap	36GB	120GB	10% cached
Repository	15TB	53TB	File system
PDF	10TB	35TB	
TXT	900GB	3TB	
XML	400GB	1.4TB	
Figures	10TB	35TB	Estimated
Crawl	43TB	150TB	Estimated

Second, when the size of the collection reaches 35 million, estimated to be the number of OA academic documents that were available online in 2014 [23], the MySQL database will grow up to 1.8TB. The citation graph will contain at least 245 million nodes and nearly 1 billion edges. To retain a reasonable hit rate, MySQL demands the hosting server to scale up with at least 200GB memory to cache at least 10% data (the OS consumes at least 20% memory). Although a MySQL cluster could achieve extremely high availability, distributing data across multiple shards will slow down queries that join multiple tables. Dumping and importing such a big database could take days to over a week.

The scalability issue also affects Solr. Although Solr can easily scale up to 80+ million documents on a single server, simply increasing its memory heap may decrease performance due to garbage collection [19]. The current master-slave architecture is hard to scale out. SolrCloud requires ZooKeeper and installation and configuration both tools impose a significant overhead because these two are not well-integrated.

The growing size also impacts data sharing and backup. Table I shows that as the data grows, the download service for all OA documents will reach at least 53TB. Adding the database (1.8TB), the index (1.2TB), and a backup for each, the total disk space needed will be about 109TB. Although this can be fulfilled using a regular server, backing up and sharing such a repository across servers is non-trivial. The time to backup the entire repository will take weeks. In addition, each document is associated with several other types of files such as TXT and XML files. The total number of files and

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

folders to move will be at least $35 \times 4 = 140$ million. In the current production repository, all types of files associated with an academic document are saved under a single directory. Although this provides a convenient way to access files, it takes an extremely long time to traverse the entire repository for exporting a specific type of file (e.g., TXT only).

The frontend also needs to be improved. The search engine uses the default ranking function of Apache Solr, which is based on term frequency-inverse document frequency (TF-IDF). Although this algorithm can effectively capture terms that match the query, it suffers from low relevancy for ambiguous queries. The web-based user interfaces only contains a search box and does not offer other options that allow users to narrow down search results based on subject categories [22], [40] and other properties.

From the developer's point of view, the system contains many complicated dependencies and several software packages are not supported anymore (e.g., Perl packages). Furthermore, although the current code was written in a highly structured manner, the use of DAO made the hierarchical structure very deep and rigid, so it is difficult to add new features or modify existing functionalities. Specifically, the ingestion module is strongly coupled with the controller, so it is difficult to disentangle numerous dependencies and parallelize the ingestion.

V. Designing a New System

A. Design Considerations

The new system was designed to meet the four requirements introduced in Section I by addressing limitations of the current system. Specifically, the system should be able to keep running at a low cost for 10 years, assuming there are no catastrophic natural disasters. The system should be usable to a growing user population and continue supporting scholarly big data to a broad research community. Finally, the system should provide a reliable service with enriched data through accessible user interfaces. To this end, the major changes are below.

1) Design a scalable web crawler and crawl the majority (if not all) of OA academic documents.
2) Transition to using a more scalable (potentially NoSQL) database.
3) Integrate and parallelize extraction and ingestion with upgraded hardware to accelerate these two processes.
4) Split the repository into subrepositories, each of which carries a specific type of files.
5) Redesign the frontend to interact with the new backend and give users more options to search and navigate. The code structure should be shallower, which is easier for future development and maintenance.
6) Balance the redundancy, cost, and complexity to achieve the overall availability and security levels.

B. Architecture

The architecture of the new system is depicted in Fig. 5. Several notable changes are summarized below. (1) A Scrapy-based breadth-first parallel crawler has already been developed to harvest PDF documents by directly following links from existing large digital repositories, e.g., Microsoft Academic Graph (OAG) and the Internet Archive Scholar. The CDI middleware [43] converts crawl metadata into a standard format, which is fed into the Extraction and Ingestion System (EIS). (2) The EIS system contains a customizable and parallel extraction module that takes PDFs as input and outputs metadata, citations, figures and tables, math formulas, and keyphrases, which are directly ingested into Elasticsearch and the production repository, with a minimal number of intermediate files to reduce disk I/O. (3) The web application interacts with Elasticsearch and processes both searching and metadata retrieval requests. The database, which is only used for research purposes, pulls data from Elasticsearch.

1) Web Crawler: We use verified sources such as S2, MAG, arXiv.org, PubMed, Internet Archive, and other .edu sources to obtain seed URLs. These sources generally provide rich and high-quality academic documents, so we can crawl less useless PDFs. We built a multi-threaded high throughput Scrapy-based crawler that can download 1 million PDF documents in 7–8 hours using up to 48 processes. The crawled metadata such as the source URLs, timestamps, and checksums (encrypted using SHA1) are stored in Elasticsearch, a NoSQL datastore. It is worth mentioning that even crawling URLs from the sources above generated a considerable amount of non-academic documents, such as presentations, books, and, resumes. We use a basic academic filter [26] to filter out these documents.

2) Extraction and Ingestion System (EIS): The EIS framework converts academic documents in PDF format into searchable formats. The framework processes extraction, clustering, and ingestion of each document into Elasticsearch. The core component of the EIS system is PDFMEF (PDF Multi-entity Extraction Framework), a customizable and scalable framework for extracting content from scholarly documents [41]. PDFMEF encapsulates various content classifiers and extractors, such as Apache PDFBox, a learning-based academic paper classifier, and pdffigures2 [7] to perform comprehensive information extraction. We employ GROBID [27] for parsing and re-structuring raw documents into structured XML/TEI encoded documents. We then pipeline the XML into extraction parsers to extract various types of information. The EIS system starts with the crawled metadata that contains the PDF file locations. The modules are described below.

a) Extraction: PDFMEF extracts various types of entities such as metadata, authors, citations, and figures from PDF documents. The code is containerized and additional entities like entities, math equations, and figures are extracted in parallel, i.e., multiple documents can be processed simultaneously. The code is adapted to run asynchronously in separate containers in batches so that slower extraction on relatively large documents does not decrease the overall document processing rate.

b) Clustering (aka deduplication or conflation): We define a cluster to be a distinct bibliographic unit that contains either a full-text document or a citation record or both of them. Under this definition, clustering is the process of identifying near-duplicate paper records and their citations in other papers. We implement the key-mapping algorithm [45] as an online

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

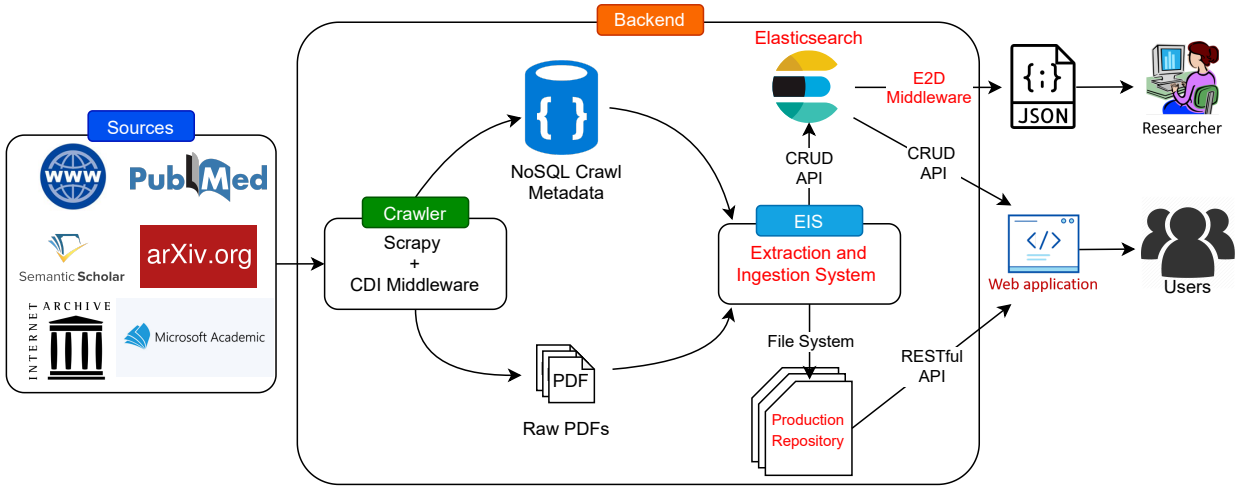

Fig. 5. The architecture of the new digital library system.

parallel process to make ingestion near real-time. The idea is to index the keys generated by concatenating title snippets and author names and use these keys to match documents. When ingesting a paper its keys are compared with existing keys to retrieve candidate clusters using Elasticsearch's GET-by-id API. Doing so enables parallelization of the algorithm while also significantly reducing the search space during matching. This gives an enormous boost in extraction and ingestion throughput of about one million documents a day. The candidate clusters are further matched by using similarity metrics from [26] before merging the entity into a cluster. The similarity metric S_{title} is obtained as the harmonic mean of Jaccard index (J) and containment measure (C):

$$S_{\text{title}} = \frac{2JC}{J+C}, J = \left|\frac{N_1 \cup N_2}{N_1 \cap N_2}\right|, C = \frac{|N_1 \cap N_2|}{\min(|\ N_1\ |, |\ N_2\ |)}$$

where N_1, N_2 are normalized n-grams ($n = 3$) of the current title and matched cluster title respectively. The similarity threshold is empirically set to 0.6.

c) Index Schema: In the current database, papers, citations, and authors are stored in separate tables. Although it is intuitive to make separate indices when migrating data from MySQL to Elasticsearch, it is not the most efficient way because the system still needs to jointly query different indices to obtain certain results. To take advantage of the fast searching performance of Elasticsearch, we organize all data into a single index, a short version of which is depicted in the Fig. 6. The advantage of this design enables us to store both full-text papers and citations together in form of clusters. Since the search is performed on clusters, the search engine result page (SERP) will not include near-duplicates. The citation relation information is stored in the `cited_by` field that contains a list of paper ids cited by the current cluster. To retrieve citations for a given cluster we retrieve all clusters that contain the current `paper_id` in its `cited_by` field.

d) Ingestion: The paper data and citation data are converted into entities consumable by the Elasticsearch

```
{
 "paper_id": ["paper_id1", "paper_id2"]
 "title": "Title goes here",
 "abstract": "Abstract goes here",
 "text": "Full Text goes here, will be
    empty for citations"
 "has_pdf": true
 "source_urls": ["source_url1", "
    source_url2"]
 "cited_by": ["paper_id3", "paper_id4"]
 "authors": [
     {"name": "author1", "affiliation": "
        affiliation1"},
     {"name": "author2", "affiliation": "
        affiliation2"},
 ]
 "source_urls": ["source_url1", "
    source_url2"],
 "venue": "venue goes here",
 "year": 2015
}
```

Fig. 6. Major fields in the single index schema for Elasticsearch.

API. We use the single index solution, where each indexed document is a document cluster that holds both metadata and citation relationships, so we can easily find out its citing and cited clusters. The PDFs are then renamed by their SHA1 values. We store them in the repository server by nesting them by name. For example, a PDF with SHA1 `a7b98ea0f94b920920524cdeee142232d7ccc488` is stored at location `/data/a7/b9/8e/a0/f9/4b/92/a7b98ea0f94b920920524cdeee142232d7ccc488/a7b98ea0f94b920920524cdeee142232d7ccc488.pdf`. These PDF resources are accessible through a REST API that is served for end users via the search interface.

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

e) Scalability: The EIS framework scales well to our ingestion throughput requirement. Extraction is a CPU-intensive operation with a small memory footprint, which will thus benefit from using a multi-core server. Ingestion to Elasticsearch is a network-intensive operation. The system is designed to extract and ingest a batch of documents using all available cores. The batch processing time depends on the batch size, the number of extraction processes, and the number of threads in the ingestion thread pool. The system benefits from more cores and higher batch sizes before it saturates. The best throughput was observed with 128 processes, a 1000 thread pool, 1000 documents per batch with an ingestion rate of about 1 million PDFs per day on a machine with dual AMD EPYC 7702P 64-Core hyperthreaded processors.

3) The Choice of Elasticsearch: We compared three NoSQL datastores that support indexing and retrieval: Elasticsearch, Apache Solr, and MongoDB. Graph databases like Neo4j [38] were not considered because they were optimized specifically for graph data. Although the citation graph can fit into Neo4j, most data queries are still relational. Although all of them support a JSON-based key-value pair data structure, they act differently in large production systems [17].

One scalability metric is how fast the datastore ingests new documents. To perform an even comparison, we designed an experiment to ingest metadata of 1 million research articles. These articles are randomly chosen from the Open Research Corpus [1]. The metadata includes text content such as title and abstract, with metadata such as venue, year, and authors, which resembles a similar schema for the data we will index into our system. For simplicity, default configurations of all the tools were applied and we ran them on the same machine. We use bulk indexing capabilities for all these frameworks. It took 123 seconds, 205 seconds, and 227 seconds for Apache Solr, MongoDB, and Elasticsearch to finish indexing, respectively. Although Elasticsearch is the slowest its speed is on the same order of magnitude as Solr. We choose it because of the following reasons. First, MongoDB lacks a good tokenizer, text analyzer, and strong support of full-text search. It is primarily used for data storage, not search. Second, Elasticsearch offers a much better horizontal scalability and documentation. It also integrates well with powerful visualization tools like Kibana and log analysis tools like Logstash.

4) Repository: We developed a RESTful API that retrieves documents from an XFS repository to the web servers. We keep using *varnish* to automatically cache frequently downloaded documents. This REST+XFS model is stable and easy to deploy.

To solve the scalability problem, our solution is to split the repository into sub-repositories, each of which carries a certain type of file. These sub-repositories will be saved in a storage-area network (SAN) hosted in our institution's data center. The web servers access the SAN storage via iSCSI, which acts as a virtual SCSI cable. In this way, the repository is hosted in a block storage array rather than as a partition in a virtual server, which ensures that it is scalable. The multipathing configuration between iSCSI clients (web servers) and the SAN eliminates the single-point-of-failure in the current system.

In the current system, user corrections are inserted into the database *and* written into a new XML file into the repository. In the new design, users do not have permission to modify data. However, they can make modification requests if they see any wrong metadata. Metadata updates are performed on the backend. The repository is read-only, which enables an incremental backup by just keeping track of new documents.

5) Web Application: The backend of the web application utilizes the FastAPI framework for ease of development and maintenance. For the frontend, Vue.js and Nuxt.js are chosen because of the flexibility in the server-side rendering feature. The utilization of a JavaScript framework, in general, allows for more efficient development and maintainability through a component-based design, strong documentation, and better performance. The web application exists in a decoupled structure, allowing for separation of responsibilities and for the backend and frontend to be developed relatively independently.

In addition to the key features such as searching and querying paper data utilizing the built-in Elasticsearch API, many features from the current system were inherited. For example, co-citation and bibliographic coupling algorithms are implemented to recommend similar papers. To collect user data, feedback, and for users to request updates to incorrect metadata, the account management, and authentication features are adopted. When implementing the current features, the Elasticsearch DSL library is used to reduce the code needed in the service layer of the backend.

Several new features were implemented that do not exist in the current system. The new system allows for faceted search. When a user makes a query, that query is passed along to the Elasticsearch instance and retrieves aggregations of the authors and journals with the most results. These aggregations are displayed in the user interface as a list of checkboxes, which users can toggle to filter search results by including only documents by certain authors. A filter is also added to constrain the publication year to a new range. Autocomplete is an added feature, by which the search bar will display a list of titles in real-time that match the user's query strings. Elasticsearch's "More Like This Query" API based on TF-IDF is used for similar paper recommendations. A "like" button is added to gather user feedback on high-quality papers.

C. Capacity Planning & Load Testing

The development Elasticsearch cluster is configured with 2 nodes each with 8 cores Intel Xeon Gold 5118 CPU @ 2.30GHz. To determine the number of primary shards required for a scale of about 100 concurrent users each making multiple requests searching about one million full-text documents while navigating the user interface, a load test was performed to test the limits of a single primary shard until the latencies increased beyond the desired experience. We used *Locust*, an open-source load testing tool to test the Web API. The environment is set up locally on a development server, without a load balancer, and with an initial cold cache. In the experiments,

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

TABLE II
LOAD TESTING ON SINGLE SHARD WITH 100 SIMULATED CONCURRENT USERS ON SINGLE SHARD WITH NO REPLICAS.

API Endpoint	T_{Med}	T_{Avg}	T_{90}	RPS
/paper	5	9	13	10.5
/search	20	71	260	9.4
/show_citing	10	31	80	10
/similar_papers	6	7	9	8.7
/search_suggestion	7	34	100	9.4
/citations	39	69	160	9.3
/document_download	32	42	75	9.2
Total Aggregated	**9**	**38**	**91**	**65.4**

T_{Med}, T_{Avg}, and T_{90} are the median, average, and 90% percentile response time in milliseconds, respectively. RPS stands for Request Per Second.

we configured an Elasticsearch cluster with about one million full-text documents and their citation clusters on an index configured with one primary shard. Simulated users uniformly choose one of the API endpoints to make HTTP requests every 1 to 2 seconds. Depending on the type of endpoint, that request will randomly choose from either of the 5000 preselected unique `paper_id` or the 5000 preselected `cluster_id` to construct the query. For search and suggestion endpoints, a random word generator is used to construct search terms. Two tests were conducted. The first test was setup with 100 concurrent users (the current system has 40–60 concurrent active users on average). Table II shows the result of the first test. The new web API has a median response time of 9 ms with 65.4 requests per second. This is an extreme test where there is only one primary shard and no replica shards, the results show that approximately we require a shard for every million documents and a sufficient number of replicas to improve the latencies even further.

The second test probed the limit of the new web API and our single index schema design. This test spawns 10 users every second until all requests start to fail. The results are shown in Fig. 7. At around 700 concurrent users with about 400 requests per sec, a fraction of requests started to fail. At around 2,000 concurrent users with 400 to 900 requests per sec, all requests have failed. The current system usage is well below this limit. The experiments demonstrated that the new system should be able to handle a normal amount of incoming traffic in the foreseeable future. Moreover, we configure the number of primary shards and replicas by extrapolating the results from the above experiments and could scale horizontally to meet the required performance goals.

To improve the relevance of our search results, we use s2search reranker[4], which uses a LightGBM ranker trained on query-document pairs. This ranker is built using real-world usage data of S2, which we believe will serve as a good baseline for our system. This model considers features such as date, venue, abstract, and query overlap, title, and query overlap to rerank the search results from Elasticsearch. We will fine-tune and improve this ranker after the new production system collects enough user clickthrough data.

[4]https://github.com/allenai/s2search

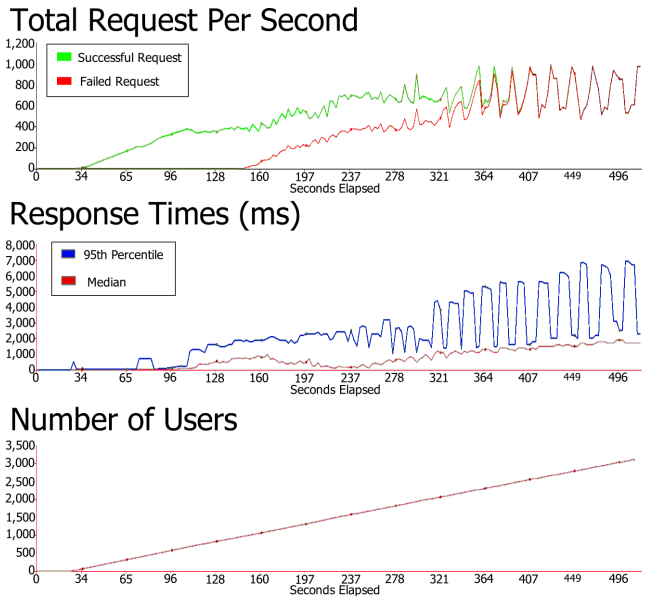

Fig. 7. Stress testing the limit of new web API and single index schema design on single shard with one replica.

TABLE III
HARDWARE SPECIFICATIONS OF SERVERS WITH VARIOUS FUNCTIONALITIES. VHDD STANDARDS FOR VIRTUAL HARD DRIVES. ONLY PRODUCTION SERVERS ARE LISTED. ALL VHDDS USES RAID 5.

Function	Type (#)	Main Specifications			
		#Core	RAM	Storage	Disk type
Web	VM (x3)	2	5GB	1TB	vHDD
Database	VM (x2)	4	16GB	3TB	vHDD
Index	PH (x2)	16	96GB	4TB	SSD[1]
	PH (x2)	16	64GB	8TB	HDD[1]
	VM (x3)	4	16GB	200GB	vHDD
Extraction	PH (x1)	40	196GB	4TB	SSD[2]
Repository	SAN	–	–	50TB	vHDD

[1] No RAID. [2]After RAID 10.

D. Hardware and Software Infrastructures

Table III shows the new hardware. Major upgrades and rationale are summarized below.

1) Private Cloud: The private cloud is retained and will be used for deploying lightweight servers, including but not limited to the Kibana server, Logstash server, staging server, static web server, and monitoring server. Kibana is used for visualizing Elasticsearch data. With the assistance of Logstash, users can create interactive data dashboards using real-time data. The staging server is used for feature development and experiments. The master servers of the Elastic cluster (described below) are also hosted in the cloud.

2) Elastic Cluster: A cluster consisting of 4 physical servers and 3 VMs is setup to host Elasticsearch. There are two main reasons to use data nodes as physical servers. First,

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

these servers will be equipped with solid-state drives (SSDs), instead of HDDs as in the private cloud. Although HDDs are more affordable, SSDs are much faster, durable, and reliable for a mission-critical unit. Many commercial search engines have replaced HDDs with SSDs in their infrastructure [36]. Second, because Elasticsearch will play roles as a datastore and a search engine, a dedicated TCP/IP connection will ensure smooth communication from/to other servers.

3) EIS Server: The server hosting the EIS frame is also outside the private cloud because this server will take heavy-duty extraction and ingestion workload.

4) SAN: As mentioned in Section V-B, the repository will be hosted in a SAN cluster located in an on-premise data center managed by the institutional IT service.

5) Software Stack: The software stack of the new system is shown in Fig. 2 (bottom). The new system uses Python as the major developing language. Python offers many popular and well-maintained packages such as Django and FastAPI. Python is also a popular programming language for data science, which makes it more convenient to integrate AI technologies. Compared with the current system, the new system software stack is simpler and thus more maintainable and extensible.

E. Challenges

Implementing the ingestion-time clustering and near-duplicate detection of an existing paper is a challenging task. The decision to mark a newly-crawled PDF as a near-duplicate of an existing PDF has to be made real-time during ingestion to not impact ingestion throughput. Since simhash and hamming distance approach [29] would be complicated and inefficient for our purpose, we designed a novel parallel version of Key Mapping algorithm [45] by storing the key to cluster mappings and further matching candidate clusters with similarity metric to avoid false positives. The parallel version avoids creating duplicates when ingested in parallel since the keys of the PDF that is clustered first will be available in real-time to the PDF clustered second as our key mapping index leverages Elasticsearch's GET-by-id API that works real-time. We will investigate other algorithms, such as Locality Sensitive Hashing [2] and Bloom's filter [3], on this task.

The backend API development faced several revisions in terms of how it should interact with Elasticsearch while still allowing the frontend to make requests through it. We considered using Django but it is more suitable for objects stored in an SQL database, as opposed to a NoSQL server like Elasticsearch. As such, we decided to go with FastAPI because it is more suitable for general purposes.

Deployment of the new system using Docker containers also posed challenges. Because users may make requests from the web UI or the API, we wanted the backend API to be accessible so that users still have access to the API service when the web UI is down. The solution was to move both the frontend and backend services into Docker containers. An Nginx proxy server is used for routing requests to either the web application or the API service based on the URL string. This configuration allows users to access data from the web UI or the API using the same port.

VI. Issues

One limitation is that there is still not a system that oversees and manages all servers. On the front end, Apache Kubernetes is a promising system for automating web application deployment and management. On the backend, Apache Airflow allows for better managing different extraction tasks. For web crawling, the documents are still collected by batch crawling URLs from other digital repositories. To increase freshness, an intelligent incremental crawler can be developed that predicts the emergence of new documents from author homepages by analyzing crawl history and then download them [21]. Currently, the extraction is synchronous with the ingestion. Future work includes making them asynchronous using message queues in Kafka. This could lighten the workflow and further improve throughput.

VII. Conclusions

Here we described the design and implementation of a new digital library system and compared it with the current system. With the goals of making the new system more accessible, usable, scalable, and sustainable, we inherited valuable features of the current system but also made several major changes to architecture, hardware, and software. In the new architecture, the searching and datastore functionalities are consolidated into Elasticsearch. The web application is re-engineered using Vue.js for the frontend stack and Nuxt.js as the universal server-side rendering (SSR), both having a smaller learning curve, robust documentation, and code transparency, which supports the continuous development and convenient deployment. The backend is refactored so information extraction and ingestion modules are integrated and parallelized, which significantly boosted the ingestion speed to at least 1 million documents per day. The Alpha version of the framework is accomplished and under vigorous testing. A prototype based on the ACL Anthology containing about 55k documents will be available, followed by deployment with the full collection. The framework software will be open-source to the public after the system is fully deployed.

We expect the deployed system to provide a sustainable service with more complete, up-to-date, and semantic access to scholarly big data. We believe this framework will benefit the building of more scalable and customizable institutional digital library systems.

Acknowledgment

We gratefully acknowledge partial support from the National Science Foundation (Award#: 1823288).

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] W. Ammar, D. Groeneveld, C. Bhagavatula, I. Beltagy, M. Crawford, D. Downey, J. Dunkelberger, A. Elgohary, S. Feldman, V. A. Ha, R. M. Kinney, S. Kohlmeier, K. Lo, T. C. Murray, H.-H. Ooi, M. E. Peters, J. L. Power, S. Skjonsberg, L. L. Wang, C. Wilhelm, Z. Yuan, M. van Zuylen, and O. Etzioni. Construction of the literature graph in semantic scholar. In *NAACL-HLT*, 2018.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan. 2008.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[4] C. Caragea, J. Wu, S. D. Gollapalli, and C. L. Giles. Document type classification in online digital libraries. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3997–4002, 2016.

[5] H. Chen, Y. Yang, W. Lu, and J. Chen. Exploring multiple diversification strategies for academic citation contexts recommendation. *Electron. Libr.*, 38(4):821–842, 2020.

[6] H.-H. Chen, L. Gou, X. Zhang, and C. L. Giles. Collabseer: A search engine for collaboration discovery. In *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*, JCDL '11, page 231–240, New York, NY, USA, 2011. Association for Computing Machinery.

[7] C. Clark and S. K. Divvala. Pdffigures 2.0: Mining figures from research papers. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries, JCDL 2016, Newark, NJ, USA, June 19 - 23, 2016*, pages 143–152, 2016.

[8] V. P. de Almeida, S. Bhowmik, G. F. Lima, M. Endler, and K. Rothermel. DSCEP: an infrastructure for decentralized semantic complex event processing. In X. Wu, C. Jermaine, L. Xiong, X. Hu, O. Kotevska, S. Lu, W. Xu, S. Aluru, C. Zhai, E. Al-Masri, Z. Chen, and J. Saltz, editors, *IEEE International Conference on Big Data, Big Data 2020, Atlanta, GA, USA, December 10-13, 2020*, pages 391–398. IEEE, 2020.

[9] H. V. de Sompel, M. L. Nelson, C. Lagoze, and S. Warner. Resource harvesting within the OAI-PMH framework. *D Lib Mag.*, 10(12), 2004.

[10] R. Entlich, J. Olsen, L. Garson, M. Lesk, L. Normore, and S. Weibel. Making a digital library: The contents of the core project. *ACM Trans. Inf. Syst.*, 15(2):103–123, Apr. 1997.

[11] S. Erera, M. Shmueli-Scheuer, G. Feigenblat, O. P. Nakash, O. Boni, H. Roitman, D. Cohen, B. Weiner, Y. Mass, O. Rivlin, G. Lev, A. Jerbi, J. Herzig, Y. Hou, C. Jochim, M. Gleize, F. Bonin, and D. Konopnicki. A summarization system for scientific documents. In S. Padó and R. Huang, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019 - System Demonstrations*, pages 211–216. Association for Computational Linguistics, 2019.

[12] M. E. Falagas, E. I. Pitsouni, G. A. Malietzis, and G. Pappas. Comparison of pubmed, scopus, web of science, and google scholar: strengths and weaknesses. *The FASEB Journal*, 22(2):338–342, 2008.

[13] C. Florescu and C. Caragea. Positionrank: An unsupervised approach to keyphrase extraction from scholarly documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1105–1115, 2017.

[14] C. L. Giles, K. D. Bollacker, and S. Lawrence. CiteSeer: An automatic citation indexing system. In *Proceedings of the 3rd ACM International Conference on Digital Libraries, June 23-26, 1998, Pittsburgh, PA, USA*, pages 89–98, 1998.

[15] P. Ginsparg. Arxiv at 20. *Nature*, 476(7359):145–147, 2011.

[16] G. Goeckenjan, H. Sitter, M. Thomas, D. Branscheid, M. Flentje, F. Griesinger, N. Niederle, M. Stuschke, T. Blum, K. Deppermann, et al. Pubmed results. *Pneumologie*, 65(8):e51–e75, 2011.

[17] S. Greca, A. Kosta, and S. Maxhelaku. Optimizing data retrieval by using mongodb with elasticsearch. In E. Xhina and K. Hoxha, editors, *Proceedings of the 3rd International Conference on Recent Trends and Applications in Computer Science and Information Technology, RTA-CSIT 2018, Tirana, Albania, November 23rd - 24th, 2018*, volume 2280 of *CEUR Workshop Proceedings*, pages 114–119. CEUR-WS.org, 2018.

[18] H. Han, C. L. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E. A. Fox. Automatic document metadata extraction using support vector machines. In *Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '03, pages 37–48, 2003.

[19] S. Heisey. SolrPerformanceProblems, 2020. https://cwiki.apache.org/confluence/display/SOLR/SolrPerformanceProblems.

[20] W. Huang, Z. Wu, C. Liang, P. Mitra, and C. Giles. A neural probabilistic model for context based citation recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Feb. 2015.

[21] Y. Jayawardana, A. C. Nwala, G. Jayawardena, J. Wu, S. Jayarathna, M. L. Nelson, and C. Lee Giles. Modeling updates of scholarly webpages using archived data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1868–1877, 2020.

[22] B. Kandimalla, S. Rohatgi, J. Wu, and C. L. Giles. Large scale subject category classification of scholarly papers with deep attentive neural networks. *Frontiers in Research Metrics and Analytics*, 5:31, 2021.

[23] M. Khabsa and C. L. Giles. The number of scholarly documents on the public web. *PLoS ONE*, 9(5):e93949, May 2014.

[24] C. Lagoze, W. Arms, S. Gan, D. Hillmann, C. Ingram, D. Krafft, R. Marisa, J. Phipps, J. Saylor, C. Terrizzi, W. Hoehn, D. Millman, J. Allan, S. Guzman-Lara, and T. Kalt. Core services in the architecture of the national science digital library (nsdl). In *Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '02, page 201–209, New York, NY, USA, 2002. Association for Computing Machinery.

[25] E.-P. Lim and Y. Lu. Harp: A distributed query system for legacy public libraries and structured databases. *ACM Trans. Inf. Syst.*, 17(3):294–319, July 1999.

[26] K. Lo, L. L. Wang, M. Neumann, R. Kinney, and D. Weld. S2ORC: The semantic scholar open research corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4969–4983, Online, July 2020. Association for Computational Linguistics.

[27] P. Lopez. Grobid: Combining automatic bibliographic data recognition and term extraction for scholarship publications. In *Proceedings of the 13th European Conference on Research and Advanced Technology for Digital Libraries*, ECDL'09, pages 473–474, Berlin, Heidelberg, 2009. Springer-Verlag.

[28] S. Manca. Researchgate and academia.edu as networked socio-technical systems for scholarly communication: A literature review. *Research in Learning Technology*, 26, 2018.

[29] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 141–150, New York, NY, USA, 2007. Association for Computing Machinery.

[30] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. P. Hsu, and K. Wang. An Overview of Microsoft Academic Service (MAS) and Applications. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 243–246, Republic and Canton of Geneva, Switzerland, 2015.

[31] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In Y. Li, B. Liu, and S. Sarawagi, editors, *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 990–998. ACM, 2008.

[32] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: Extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 990–998, New York, NY, USA, 2008. Association for Computing Machinery.

[33] P. B. Teregowda, B. Urgaonkar, and C. L. Giles. Citeseerx: a cloud perspective. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 9–9, Berkeley, CA, USA, 2010.

[34] P. Treeratpituk and C. L. Giles. Disambiguating authors in academic publications using random forests. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '09, pages 39–48, New York, NY, USA, 2009. ACM.

[35] R. Vine. Google scholar. *Journal of the Medical Library Association*, 94(1):97–99, 01 2006.

[36] J. Wang, E. Lo, M. L. Yiu, J. Tong, G. Wang, and X. Liu. Cache design of ssd-based search engine architectures: An experimental study. *ACM Trans. Inf. Syst.*, 32(4), Oct. 2014.

[37] K. Wang, Z. Shen, C. Huang, C.-H. Wu, D. Eide, Y. Dong, J. Qian, A. Kanakia, A. Chen, and R. Rogahn. A review of microsoft academic

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.

services for science of science studies. *Frontiers in Big Data*, 2:45, 2019.

[38] J. Webber and I. Robinson. *A Programmatic Introduction to Neo4j*. Addison-Wesley Professional, 1st edition, 2018.

[39] K. Williams and C. L. Giles. Near duplicate detection in an academic digital library. In *Proceedings of the 2013 ACM Symposium on Document Engineering*, DocEng '13, pages 91–94, New York, NY, USA, 2013. ACM.

[40] J. Wu, B. Kandimalla, S. Rohatgi, A. Sefid, J. Mao, and C. L. Giles. Citeseerx-2018: A cleansed multidisciplinary scholarly big dataset. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 5465–5467, 2018.

[41] J. Wu, J. Killian, H. Yang, K. Williams, S. R. Choudhury, S. Tuarob, C. Caragea, and C. L. Giles. PDFMEF: A Multi-Entity Knowledge Extraction Framework for Scholarly Documents and Semantic Search. In *Proceedings of the 8th International Conference on Knowledge Capture*, K-CAP 2015, New York, NY, USA, 2015. Association for Computing Machinery.

[42] J. Wu, K. Kim, and C. L. Giles. Citeseerx: 20 years of service to scholarly big data. In H. Wang and K. Webster, editors, *Proceedings of the Conference on Artificial Intelligence for Data Discovery and Reuse, AIDR 2019, Pittsburgh, PA, USA, May 13-15, 2019*, pages 1:1–1:4. ACM, 2019.

[43] J. Wu, P. Teregowda, M. Khabsa, S. Carman, D. Jordan, J. San Pedro Wandelmer, X. Lu, P. Mitra, and C. L. Giles. Web Crawler Middleware for Search Engine Digital Libraries: A Case Study for CiteSeerX. In *Proceedings of the Twelfth International Workshop on Web Information and Data Management*, WIDM '12, pages 57–64, New York, NY, USA, 2012. ACM.

[44] J. Wu, P. Teregowda, K. Williams, M. Khabsa, D. Jordan, E. Treece, Z. Wu, and C. L. Giles. Migrating a digital library to a private cloud. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 97–106, March 2014.

[45] J. Wu, K. Williams, H. Chen, M. Khabsa, C. Caragea, A. Ororbia, D. Jordan, and C. L. Giles. Citeseerx: AI in a digital library search engine. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2930–2937, 2014.

[46] Z. Wu, J. Wu, M. Khabsa, K. Williams, H. Chen, W. Huang, S. Tuarob, S. R. Choudhury, A. Ororbia, P. Mitra, and C. L. Giles. Towards building a scholarly big data platform: Challenges, lessons and opportunities. In *IEEE/ACM Joint Conference on Digital Libraries, JCDL 2014, London, United Kingdom, September 8-12, 2014*, pages 117–126, 2014.

[47] F. Xia, W. Wang, T. M. Bekele, and H. Liu. Big scholarly data: A survey. *IEEE Transactions on Big Data*, 3(1):18–35, 2017.

Authorized licensed use limited to: Old Dominion University. Downloaded on July 25,2022 at 17:46:14 UTC from IEEE Xplore. Restrictions apply.