The Pennsylvania State University

The Graduate School

College of Engineering

LESSONS IN SCALING A LARGE DIGITAL LIBRARY: A CASE STUDY FOR CITESEERX

A Thesis in

Computer Science and Engineering

by

Douglas Jordan

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

May 2016

The thesis of Douglas Jordan was reviewed and approved* by the following:

John Hannan

Associate Professor of Computer Science and Engineering

Thesis Advisor

C. Lee Giles

David Reese Professor of Information Sciences and Technology

Interim Associate Dean of Research at the College of Information Sciences and Technology

Mahmut Taylan Kandemir

Professor of Computer Science and Engineering

Head of Graduate Program of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

The current generation of CiteSeer^x is incredibly popular. On an average day, the website has over 2 million hits, and 100,000 downloads. CiteSeer^x currently has indexed over 7 million papers, and is ingesting new papers at a rate of 5,000 per day. While the service has been able to scale to this point, over the past year it has been approaching the limit of the current architecture. The page load times have been increasing and the ingestion of new papers has slowed. It is believed that both of these issues are largely due to the GFS file system and MySQL database. In this thesis, we investigate re-architecting the back-end of CiteSeer^x .

Table of Contents

List of Figures vii

List of Tables viii

Acknowledgments ix

Chapter 1
Introduction 1
1.1 Background . . . 1
1.2 Current Architecture of CiteSeer^x . . . 2
1.2.1 Front End Web Tier . . . 2
1.2.2 Data Layer . . . 3
1.2.3 Crawler, Extraction and Ingestion . . . 4
1.3 Goals . . . 4
1.3.1 Software Engineering Principles . . . 4
1.3.2 Horizontal Scalability . . . 5
1.3.3 Improving the Front End Performance: Making CiteSeer^x faster . . . 5
1.3.4 Growing the Collection: Making CiteSeer^x bigger . . . 6
1.4 Our Approach . . . 6

Chapter 2
Repository API 7
2.1 Introduction . . . 7
2.2 Current Issues . . . 8
2.3 Proposed Solution . . . 9
2.4 Implementation . . . 9
2.4.1 Repository API Server . . . 10
2.4.2 Java HTTP Client . . . 10
2.5 Testing . . . 12

2.6 Deploying . . . 12
2.7 Caching . . . 13
2.8 Issues . . . 14
2.8.1 Connection Closed . . . 14
2.8.2 Permissions Errors . . . 15
2.9 Results . . . 15
2.9.1 Response times . . . 16
2.9.2 Caching . . . 16
2.9.3 Cache Hits vs Cache Misses . . . 18
2.9.4 Multiple servers . . . 19

Chapter 3
Evaluation of Various NoSQL Databases **21**
3.1 Introduction . . . 21
3.2 CAP Theorem . . . 22
3.3 Evaluation . . . 22
3.3.1 Solr . . . 22
3.3.2 ElasticSearch . . . 23
3.3.3 MongoDB . . . 23
3.3.4 Redis . . . 24
3.3.5 DynamoDB . . . 24

Chapter 4
Methodology and Design Decisions **25**
4.1 Introduction . . . 25
4.2 Proposed Solution . . . 26
4.3 Architecture . . . 27
4.4 MySQL Schema -> Solr Schema . . . 27
4.5 Data Import Handler . . . 28
4.6 Refactoring . . . 29
4.7 Scalability . . . 30
4.8 Issues . . . 31
4.8.1 Eventually Consistent . . . 31
4.8.2 Modification Time . . . 31
4.8.3 Slow Solr Startup . . . 32
4.8.4 MySQL Transaction Pointcut . . . 32

Chapter 5
Results **34**
5.1 Ingestion . . . 34

5.2 Front End Performance 35

Chapter 6
Conclusions and Further Work **39**
6.1 Conclusions 39
6.2 Further Work 40
6.3 Eventual Consistency 40

Appendix A
Code **41**
A.1 GitHub 41
A.2 Repository API 41
A.3 NoSQL Migration 56
A.4 SolrConfiguration 68

Bibliography **77**

List of Figures

1.1 Architecture of the VM CiteSeer^x . Arrows indicate the major data flow directions. Dashed lines represent the internal communications inside the sub-clusters. Labels on the arrows are the ways through which the data are transferred. 2

2.1 Distribution of paper downloads for January 2016. 8
2.2 Architecture for repository API server with caching layer. 14
2.3 Histogram of response times for the old GFS based repository. . . . 17
2.4 Histogram of response times for the new REST based repository. . . 18
2.5 Histogram of response times for repository cache misses. 19
2.6 Histogram of response times for repository cache hits 20

4.1 Architecture of the NoSQL system. 27

5.1 Response times for 99% of viewdoc summary requests for January 2016 using the NoSQL system. 36
5.2 Response times for 99% of viewdoc summary requests for January 2016 using the MySQL system. 37
5.3 CPU Utilization of Solr. 37
5.4 CPU Utilization of MySQL. 38

List of Tables

2.1 Cache hit ratios vs size of cache 17
2.2 Cache hit ratios vs size of cache for paper downloads only 18

4.1 The mapping of datatypes between MySQL and Solr 28

5.1 Ingestion times into Solr from MySQL using Data-Import-Handler . 34

Acknowledgments

I would like to thank my fellow lab members including Jian Wu, Kyle Williams, Madian Khabsa, Po-Yu Chuang, Stephen Carman, and Pradeep Teregowda for all of their help with my research. To my friends and family who were incredibly patient through the entire process, thank you. To Dr. Hannan and Dr. Giles, you both were critical in helping me both with this thesis, and with everything from before I even started at Penn State - Thank you.

Dedication

This thesis is dedicated to my family for their unconditional love and support.

Chapter 1

Introduction

1.1 Background

CiteSeer^{x} is a scientific literature digital library and search engine. It is continuously evolving and is one of the largest digital libraries of its type in the world. As of the time of writing, it contains more than 7 million papers. In addition to the papers themselves, CiteSeer^{x} automatically indexes metadata about the papers including the citations, which it inserts into a citation graph that contains over 120 million edges and 48 million vertices (as of 2/7/15). CiteSeer^{x} is most similar to Google Scholar (Scholar) [1] and Microsoft Academic Search (MAS) [2]. The biggest difference between CiteSeer^{x} and Scholar / MAS is that CiteSeer^{x} hosts cached copies of the PDFs that are crawled.

1.2 Current Architecture of CiteSeer^x

CiteSeer^x contains three major components: the front end web tier, the data layer, and the crawler and ingestion pipeline. This architecture is hosted on virtual machines on a private cloud running VMWare ESXi [3]. In Figure 1.1, we present the architecture of the VM CiteSeer^x .

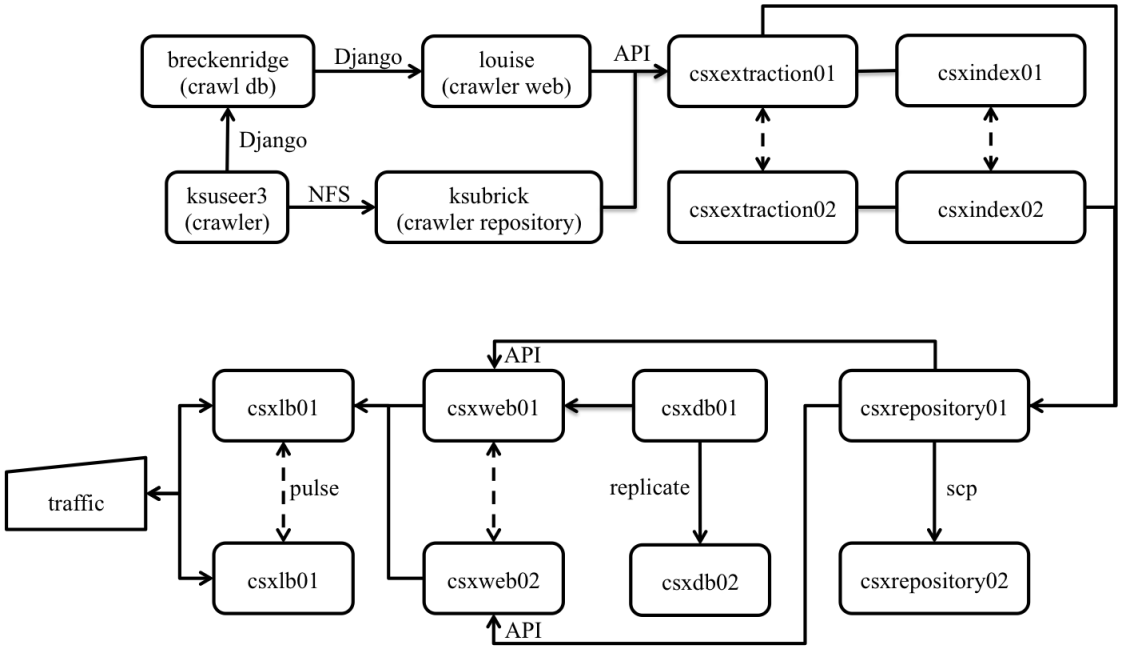

Figure 1.1. Architecture of the VM CiteSeer^x . Arrows indicate the major data flow directions. Dashed lines represent the internal communications inside the sub-clusters. Labels on the arrows are the ways through which the data are transferred.

1.2.1 Front End Web Tier

The front end consists of two load-balancers and three web servers. The Linux Virtual Servers (LVS) package is used along with ldirectord for load-balancing.

The second load balancer is a hot standby, and the two periodically ping each other in order to detect failure. If the active one goes down, the standby acquires the main CiteSeer^x IP address, and becomes the active node. In the future, round robin DNS could be used, but the current solution has worked well so far. The web servers run Apache Tomcat and the main CiteSeer^x front-end code. They are responsible for serving the dynamic content and returning papers.

1.2.2 Data Layer

There are currently three pieces of the data layer. The repository is a large file system organized by document ID (doi). CiteSeer^x assigns documents with unique identifiers (DOIs) that are composed of 5 parts. An example DOI is 10.1.1.81.5750. The file system contains a directory per segment, so to find 10.1.1.81.5750 one would traverse 10, 1, 1, 81, then 5750. The PDF as well as all metadata extracted from the paper exist in the lowest level directory. This includes the full text (.txt), citations (.cite), and XML metadata (.xml). This XML metadata includes things such as the title, authors, abstract, conference or journal and year of publication. CiteSeer^x currently use MySQL [4] as our relational database to store all of the metadata about papers. Lastly, we use Apache Solr/Lucene to index all of this data and provide the ability to search against it. It has been shown that approximately 31% of CiteSeer^x traffic is to the search page [5]

1.2.3 Crawler, Extraction and Ingestion

The crawler is responsible for scraping the web looking for academic papers. The crawler dumps its contents to a file system as well as stores metadata about the crawl in a database. The extraction system takes crawled documents and automatically extracts metadata from them. Lastly, the ingestion system inserts the data into the various data stores.

1.3 Goals

1.3.1 Software Engineering Principles

From a software engineering point of view, refactoring the repository will allow the specific implementation details to be abstracted away from the main CiteSeer^x application. This will allow us to implement the repository in another language or framework as long as it communicates using HTTP requests. This also makes CiteSeer^x much more modular so that if a better way to write the repository comes along in the future, we only need to rewrite the repository, and don't need to change any code in the main CiteSeer^x web service.

1.3.2 Horizontal Scalability

To get to the current size of CiteSeer^x ($>$ 7 million papers), we often have resorted to vertical scaling. When the database server got slower, it was replaced with a bigger VM, then eventually an even bigger physical machine. Unfortunately, we are starting to see the limits of vertical scaling, and instead want to explore ways of scaling horizontally. The front end web tier of CiteSeer^x is stateless, so it can be replicated on as many hosts as needed behind the load balancer. The front-end load balancing doesn't help scale the data layer however, and actually can put more pressure on the whole system and make it slower. When we refer to scalability, there are actually two types of scalability. The first is getting faster, the second is getting bigger.

1.3.3 Improving the Front End Performance: Making CiteSeer^x faster

As CiteSeer^x has grown, the front end performance has decreased. Page load times have increased, and at points the entire system becomes unavailable. The goal of this thesis is to improve page load times as well as make the system more available. When the system goes down entirely, it is often due to being crawled too quickly by bots from other search engines and the GFS repository cannot keep up. This can cause it to become detached from the web servers, requiring us to restart all three

web servers as well as the repository. When it happens it is extremely inconvenient for both us and our users, and as we grow in popularity we need to stay available.

1.3.4 Growing the Collection: Making CiteSeer^x bigger

The other kind of scaling is size. While we are growing, CiteSeer^x currently cannot ingest new documents into the system as fast as it can download them from the internet. This is not a problem with extraction, as we can run the extractor in parallel [6]. This bottleneck comes from MySQL and the goal of moving the data to Solr is for the frontends to query Solr, thus leaving MySQL purely for ingestion which would speed up the ingestion rate.

1.4 Our Approach

Our approach to making CiteSeer^x scale is broken down into two parts. First we will refactor the repository and replace it with a RESTful API, then we will migrate the main MySQL database to a NoSQL (not only SQL) datastore and evaluate the performance of the new system.

Chapter 2

Repository API

2.1 Introduction

The repository of CiteSeer^x is the filesystem containing all of the PDFs as well as extracted (and corrected) metadata about each paper. Currently, the repository occupies 9.7TB and is growing rapidly. The web servers which run the front end of CiteSeer^x must be able to both read and write to it. Over a month of production traffic it was observed that the repository received 99.99% reads and just 0.01% writes (excluding the ingestion process). In addition, as seen in Figure 2.1, the distribution of downloads per paper is not uniform but instead follows a power law distribution, and it was observed that approximately 1% of the papers accounted for 11% of the total downloads and 5% of the papers accounted for 25% of the total downloads [7].

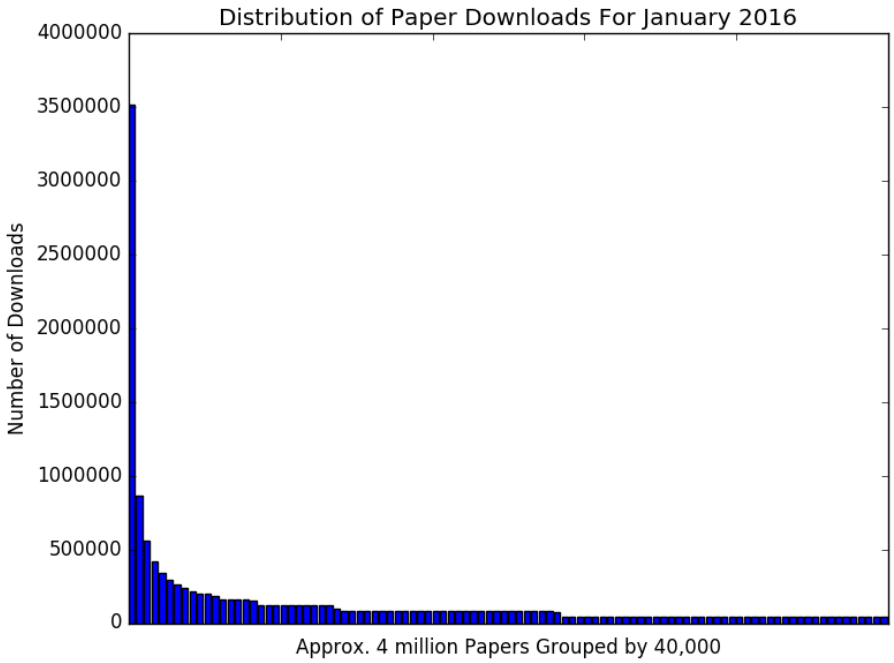

Figure 2.1. Distribution of paper downloads for January 2016.

2.2 Current Issues

One of the most pressing issues in scaling CiteSeer^x is the repository. The system currently uses the Red Hat Global File System (GFS) file system [8] shared across the repository server and all 3 web servers. When the system is being crawled at a fast rate (> about 5 papers/sec) the repository server will detach itself from the web servers and require all 4 servers to be restarted. This is a tedious process and incurs a lot of down time. It is believed that the issue occurs due to the nature of the GFS filesystem and the stateful connections it uses.

2.3 Proposed Solution

Pradeep B. Teregowda proposed the idea of a RESTful repository API [9] which would allow us to use stateless HTTP connections and abstract out the backend tier. The advantage here is that we can employ many techniques used in distributed systems behind the repository API abstraction that would seamlessly integrate into the current system. One of the goals of this project is abstraction, and with the repository RESTful API we can evaluate different solutions behind a simple interface. Since the distribution of downloads follows a power law, one idea that came to mind is to employ a cache in front of the repository so that the most frequently accessed papers can be served from memory and don't have to hit disk. We could use Memcached [10] or Varnish [11] in order to do this. In 2016, 64 GB of RAM can be purchased for under $500. With 50 GB of RAM, we can serve 25% of the requests from RAM which would reduce disk IOPS.

2.4 Implementation

To implement the repository API requires two key components. First a repository API server that can replicate the CRUD functionality of the current (POSIX) file system implementation is needed. The second component is the Java code in the frontend that will make HTTP requests and stream the documents to the clients.

2.4.1 Repository API Server

The repository API server was written in Python using the web.py framework and is deployed using mod-wsgi on the Apache web server. There are 3 major methods that needed to be implemented. First is the get_document method. This simply gets a document (in either PDF or PS format) and streams the response to the client. The next method is the get_file_types method. This method goes to the appropriate directory and returns a comma separated list of extensions contained in that directory. This is primarily used on the summary page to display which file types are available for download. The final method is the PUT file method which takes a file posted by the client and saves it to the file system. This method is used during user corrections to save a copy of the latest metadata in case we need to roll back at any point in time. Only the most recent metadata is saved in the database, which keeps its size down.

2.4.2 Java HTTP Client

Due to the tightly coupled nature of the database and repository layer in the DAO (Data Access Object) with Spring framework, implementing the repository client is a major refactoring. The first step was to create a new package called repository which contains all of the repository code. An interface called RepositoryService was created in order to abstract the specific repository implementation away from

the user. It contains 7 methods: storeDocument, writeVersion, writeXML, getDocument, fileTypes, getDocFromXML and getDocumentContent. A new class called FileSytemRepository which implements RepositoryService was created, and the current code for accessing the file system repository was moved to this class. Another class called RESTRepository was created which also implements RepositoryService. In this class new methods were written to make HTTP requests to the new repository server to store and retrieve documents. Keeping with the common goal of abstraction, the specific code to make REST requests was abstracted out to a utility class called HttpRestUtils. This project uses Java beans to construct objects when starting the front end, so the file used to construct these beans had to be modified to support the new RepositoryService class. In addition, the decoupling of the database and filesystem required a major refactoring of many classes in the front end. One example is the viewDocController. This class is responsible for generating the summary page when a user searches for a paper. In the old version, the getFileTypes method was in the DAO, so the old code was model.put("fileTypes", csxdao.getFileTypes(doi, repID));. However after the refactoring, the repository code is now in the repository package, so one must call model.put("fileTypes", RepositoryUtilities.getFileTypes(repositoryService, doi, rep));.

2.5 Testing

The repository API was first tested for major errors on a MacBook Pro laptop to ensure everything was working. One major drawback of the CiteSeer^x code base is that there are no unit tests. It is tricky to write tests when there are external dependencies (of which CiteSeer^x has many). After the proof of concept was tested, it was deployed to a staging server where more smoke tests were run against it. After seeing no major errors we moved on to deploying it.

2.6 Deploying

In the current configuration, CiteSeer^x is deployed in production on three web servers using the open source Tomcat application server. The Linux Virtual Servers (LVS) package is used as a loadbalancer to split traffic across these three hosts. To deploy the repository API, we first installed Apache and mod_wsgi on csxrepo01, which is the backup repository server. csxrepo02 is the production repository since it has a GFS file system whereas csxrepo01 has an EXT4 file system. The contents of the repository are rsync'ed weekly between the machines. After csxrepo01 was setup, the repository server was installed using the the following configuration:

Listing 2.1.

```
LoadModule wsgi_module modules/mod_wsgi.so

WSGIScriptAlias / /var/www/repo/server.py
```

```
AddType text/html .py
<Directory /var/www/repo/>
        Order deny,allow
        Allow from all
</Directory>
```

2.7 Caching

As the distribution of paper downloads follows a long tail (zipf) distribution we implemented a simple cache in order to avoid hitting the disk for common paper downloads. As the new repository utilizes HTTP, we simply installed an HTTP cache in front of the RESTful repository server. Varnish was chosen because some members of the group had prior experience with it, it is simple to install and configure, and is fast enough for our needs. The architecture of the new system with a cache is as shown in figure 2.2. In this model, HTTP requests are made for papers from the web servers, and go to csxrepo01 on port 80. Varnish is configured to run on port 80, and apache is reconfigured to run on port 8080. Varnish is configured by default to cache only GET requests which is what we need. The repository server uses POST requests for paper updates, which we want to go all the way through to the web server. If Varnish doesn't have the URL in its cache, it forwards the request to the apache server, then adds that entry to the cache in a least recently used (LRU) fashion. We initially configured it to use 4

GB of memory although that only accounts for about 0.05% of the papers so we will experiment with various cache sizes in the future. We do not cache at the application level but rather at the repository level because we still need certain features of the webapp such as the download limiter and session identifiers.

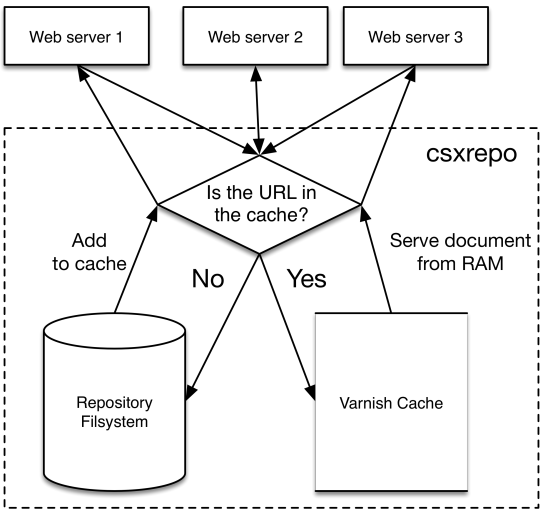

Figure 2.2. Architecture for repository API server with caching layer.

2.8 Issues

2.8.1 Connection Closed

When first deployed, it appeared that about 50% of the requests were causing an error but it was due to an error in the logging configuration. However, a small percentage of requests (<1%) cause an error, which appears to be due to the client closing the connection before the response is done being written. This was discovered to be a bug in the HttpClient library we were using, and upgrading it

to the latest version fixed the problem.

2.8.2 Permissions Errors

The repository API server runs under the apache web server, which by default runs as the *apache* user and *apache* group. This was fine as the repository is currently owned by the citeseerx user and group with everyone having read permissions. But when a user of CiteSeer^x attempts to correct the metadata of a paper, the latest version of the metadata xml is saved to the repository in order to keep a complete history. When testing in production, it was noticed that metadata updates were not succeeding as the *apache* users did not have permission to write in the repository directory. To fix it, the following settings were set in the */etc/httpd/conf/httpd.conf* file.

Listing 2.2.

```
User citeseerx
Group citeseerx
```

2.9 Results

After being deployed for about a week, no major performance changes were noticed. The CPU utilization for both csxweb01 (using the file system repository) and csxweb02 (using the REST repository) are almost identical, and the IOPS of both seem fairly similar.

2.9.1 Response times

While the performance of the new repository API seemed fine, we looked at the response times for the download page over a two week period. The load balancer distributes connections across our two web servers evenly, so if one connection happens to download more papers the distribution may not be the same. Between February 14 and February 27th csxweb01 saw 3,142,974 requests and csxweb02 saw 2,599,980 requests for paper downloads. Those numbers were obtained from the Tomcat access logs, and all download-exceeded requests were excluded. After parsing the logs, we looked at histograms of the response times for both of the web servers. In Figure 2.3 we can see that the tail of the distribution is extremely long, and a significant number of requests took over 10 seconds to process. When looking at the new REST-based repository API in figure 2.4 we can see that 95% of the requests are processed in under 7 seconds and the majority of papers are served in under a second.

2.9.2 Caching

In order to optimize cache performance, we adjusted the amount of memory allocated to varnish and replayed one day of production traffic to the repository cache. We then recorded the number of cache hits, cache misses, and computed the cache hit ratio. The results can be seen in Table 2.9.2. Clearly, using an 8GB cache

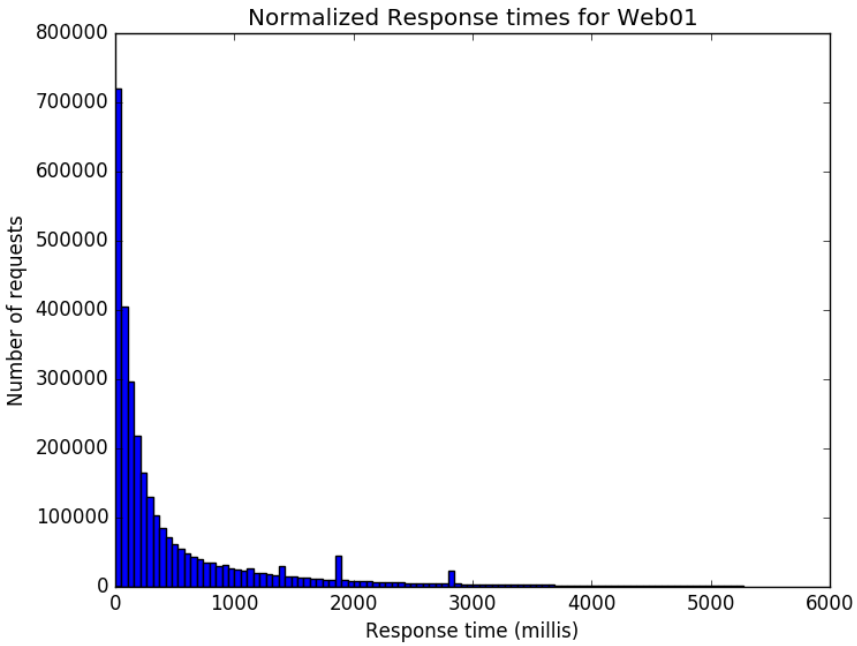

Figure 2.3. Histogram of response times for the old GFS based repository.

had the best cache hit ratio, but the results are not increasing linearly. Even a 1 GB cache gets a 30% hit rate. In addition, we ran the experiment with only the paper download requests (as compared to downloads and file type queries) and in table 2.9.2 we can see that we achieve a slightly worse cache hit ratio by approximately 1%.

Amount of memory allocated	Cache hits	Cache misses	Cache hit ratio
1 GB	41744	140874	29.6%
2 GB	44213	138405	31.9%
4 GB	47048	135570	34.7%
8 GB	50708	131910	38.4%

Table 2.1. Cache hit ratios vs size of cache

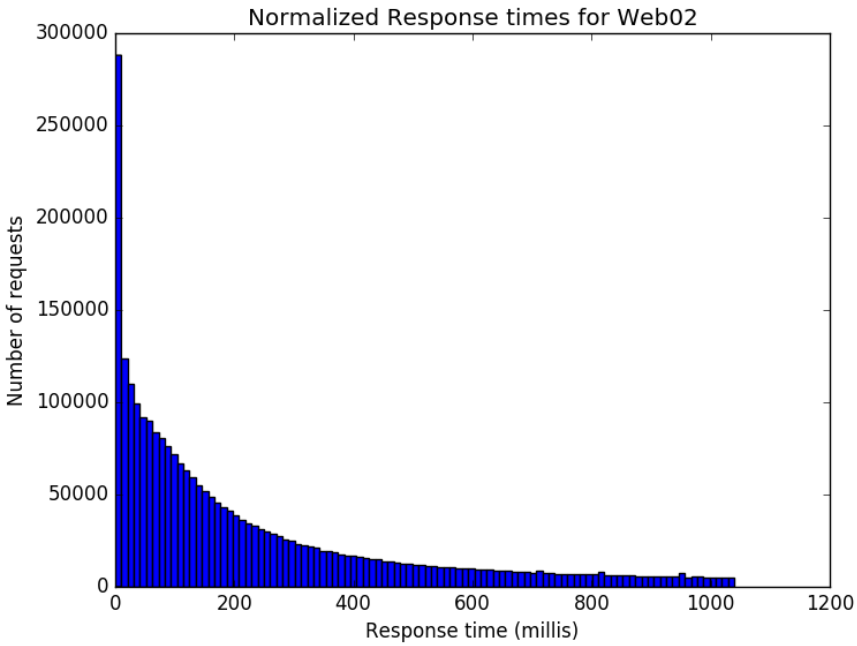

Figure 2.4. Histogram of response times for the new REST based repository.

Amount of memory allocated	Cache hits	Cache misses	Cache hit ratio
1 GB	27178	95825	28.4%
2 GB	28788	94215	30.6%
4 GB	30923	92080	33.6%
8 GB	33858	89145	38.0%

Table 2.2. Cache hit ratios vs size of cache for paper downloads only

2.9.3 Cache Hits vs Cache Misses

While hitting the cache will always be faster due to the relative speed of RAM vs disk, we looked at how much faster it is. After the varnish cache was deployed, we saw a decrease in page load times which is to be expected. We compare histograms of response times normalized to download size for cache misses and hits in Figure 2.5 and Figure 2.6 respectively. We see that the scale for misses ranges

from 0 to 35 (normalized by download size) while for hits all of the requests were served under 0.1 milliseconds (normalized by download size).

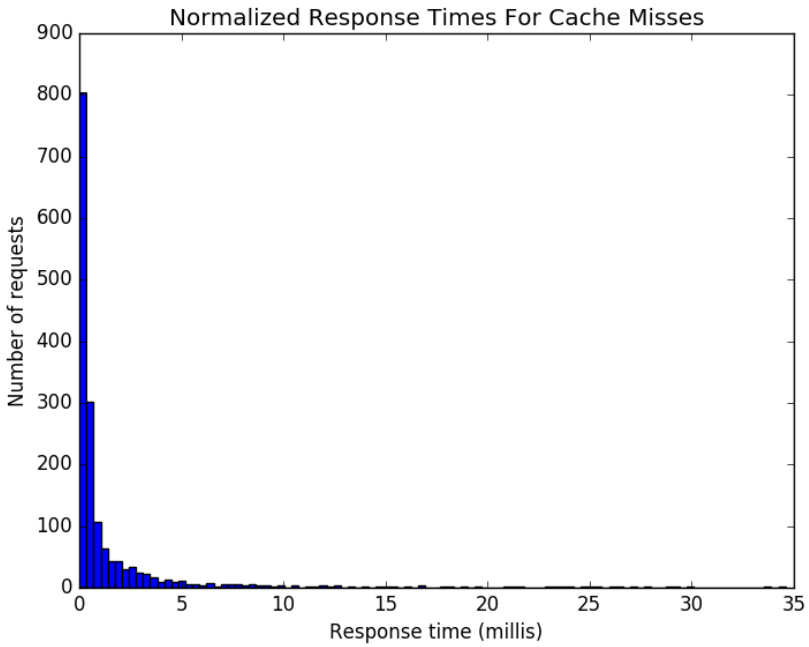

Figure 2.5. Histogram of response times for repository cache misses.

2.9.4 Multiple servers

After the repository API was deployed on csxweb02 for a few weeks, we deployed it on all three frontend webservers. After being deployed for over a month we have yet to see a locking issue and the overall system has been more available. We have had one system down issue, but it wasn't due to the repository but rather a high crawl rate let to a rapid creation of too many threads. We will investigate a combination of frontend tuning and load balancing strategies to address those issues.

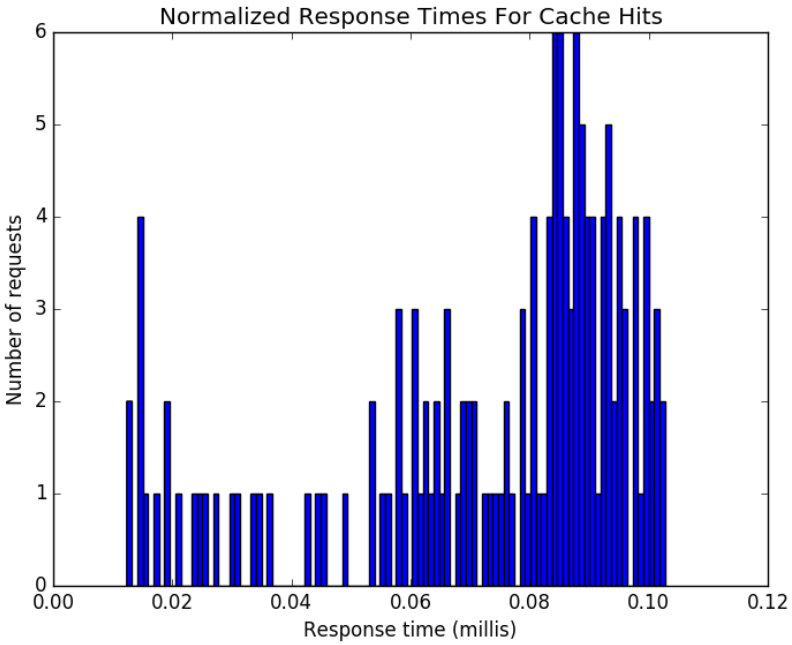

Figure 2.6. Histogram of response times for repository cache hits

Chapter 3

Evaluation of Various NoSQL Databases

3.1 Introduction

In this chapter, we evaluate five common NoSQL datastores: Solr, ElasticSearch, MongoDB, Redis, and DynamoDB. The first 4 are all open source, while DynamoDB is a proprietary service offered by Amazon Web Services (AWS). While DynamoDB is closed source, it is based on Dynamo [12], an internal project at Amazon that powers features such as the shopping cart. Each one of these datastores have pros and cons, but they all share a distributed nature and lack full ACID compliant properties offered by traditional relational database management systems (RDBMS) like MySQL.

3.2 CAP Theorem

The CAP theorem [13] states that that it is impossible for any distributed system to simultaneously provide all three of the following guarantees:

- Consistency
- Availability
- Partition Tolerance.

As the NoSQL datastores we evaluate are distributed systems, it is important to look at which two guarantees each system provides. Relational databases such as MySQL sacrifice partition tolerance to be consistent. For the rest of the NoSQL solutions, they guarantee partition tolerance but have to choose between consistency and availability. DynamoDB allows the user to choose if they want availability or consistency, although the original Dynamo [12] system was highly available but eventually consistent.

3.3 Evaluation

3.3.1 Solr

Apache Solr [14] is an open source search server based on Lucene. It was created in 2004 and the latest version, 5.2, is extremely stable. SolrCloud is the cluster sup-

port which uses ZooKeeper [15] for service discovery and leader election. SolrCloud has support to automatically shard and replicate indices across multiple hosts and can even move these cores (indices) around while continuously serving requests. By utilizing read replicas, SolrCloud can automatically load-balance queries across multiple servers.

3.3.2 ElasticSearch

ElasticSearch [16] is very similar to Solr although it is 6 years newer. It is based on Lucene but was designed for scalability from the start. Like Solr it is written in Java, but unlike Solr, which uses ZooKeeper for service discovery, ElasticSearch uses a custom module. The main reason we did not use ElasticSearch was because of previous experiments that showed that it was neither consistent nor available in certain failure modes [17].

3.3.3 MongoDB

MongoDB [18] was the next best choice after Solr. It is a document database which works well for CiteSeer^{x} since the biggest objects are the papers, however the automatic sharding of SolrCloud was more attractive for the future of CiteSeer^{x} as we scale well past 10 million papers.

3.3.4 Redis

Redis [19] is an in-memory data structure store that can be used for a variety of purposes included caches, databases, and message brokers. It supports multiple levels of on-disk persistence and has support for replication as well. Redis is extremely fast for simple key-value queries, but doesn't support some of the more advanced queries needed for CiteSeer^{x} .

3.3.5 DynamoDB

DynamoDB [20] is a cloud based NoSQL key-value store hosted by Amazon Web Services [21]. DynamoDB follows a unique pricing model where the user pays for a certain number of provisioned reads and writes per second, and also pays based on how much data is stored. At our current scale, it would cost over $200 a month to use DynamoDB, not to mention the latency occurred would be orders of magnitude more than it currently is. If our entire infrastructure were running on Amazon Web Services it might make sense to investigate DynamoDB more seriously but due to cost and latency we did not choose it.

Chapter 4

Methodology and Design Decisions

4.1 Introduction

CiteSeer^x is an incredibly large software package. Currently there are almost 45,000 lines of code, and to rewrite it from the ground up could take upwards of 11 years [22]. The current system is clearly hitting a bottleneck at the database layer, and there are two causes for it. First, the large volume of traffic, combined with expensive queries coming from the front end web servers, causes the database to become slow. Second, the rapid rate of ingestion into the same production database server that serves traffic can cause it to become slow. In some cases, we have even seen the server completely hang. Since the scope of the entire backend is too big, we instead focus on improving the experience for the users of the system.

By reducing the load on MySQL from the web servers, in turn we should see a speedup in ingestion into MySQL as there will be less load on it.

4.2 Proposed Solution

The plan to improve the performance of CiteSeer^x for its users is to index the data that is currently stored in MySQL into Solr, then refactor the DAO (data access object) of CiteSeer^x to query Solr instead of MySQL. After the data is in Solr, we can utilize the features of SolrCloud to automatically shard indices when needed and distribute the various cores (indices) across multiple hosts. ZooKeeper is used for service discovery and leader election. In this model all reads will go directly to Solr, so we can benefit from a pseudo read-only mode in Solr. When we need to do updates, they will go to the database as it will be the source of truth. Finally, we can asynchronously update Solr from MySQL on a daily or weekly batch update. The batch updates are significantly faster since they can perform large sequential scans of the disk instead of random reads. In addition, we can strategically schedule the batch import from MySQL to Solr at a time where traffic is low, such as Friday night, which reduces the duration as well.

4.3 Architecture

The architecture for the new system is rather straightforward. We will add a layer of servers between the web servers and the database. These servers will run Solr (specifically SolrCloud) and contain the data that is currently stored in MySQL. From the frontend, reads will go to Solr while writes will still go to the database as they always have.

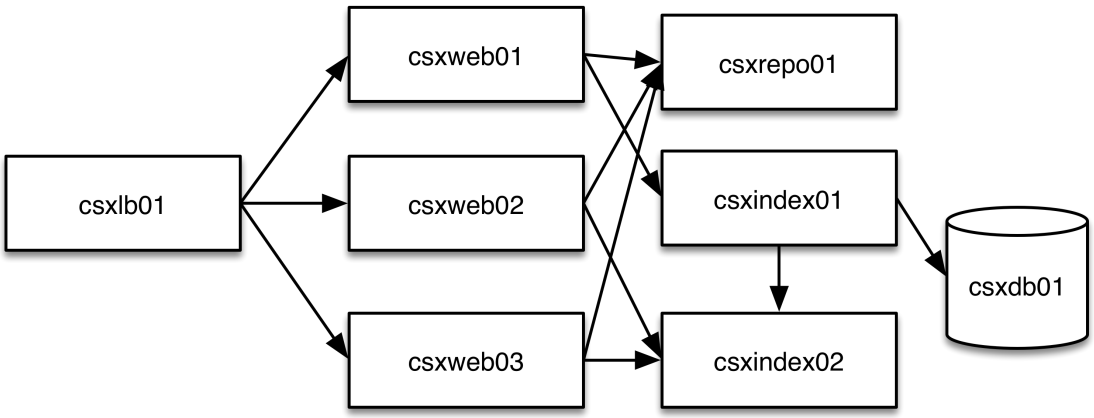

Figure 4.1. Architecture of the NoSQL system.

4.4 MySQL Schema -> Solr Schema

To index the data into Solr, we need to tell it how our data looks. Solr does support dynamic field types, but since our data comes from a relational database, we created a new Solr core which is basically a distributed index. Then, we added a field in Solr's schema.xml for each field in MySQL. In table 4.4 we show the mapping from MySQL's data types to Solr's.

MySQL Field Type	Solr Field Type
int	tint
double	tdouble
varchar	string
float	float
datetime	date

Table 4.1. The mapping of datatypes between MySQL and Solr

4.5 Data Import Handler

In order to actually get the data into Solr we had a few options. One was to write a batch importer from the ground up, but since we are simply importing existing relational data, there is a built-in tool of Solr called the Data Import Handler. Once configured, one can simply make a GET request to `http://SOLR_URL:PORT/solr/CORE_NAME/dataimport?command=full-import`, and Solr do a full import, which is equivalent to executing the SQL query: SELECT * FROM table_name. To issue a delta import, which is equivalent to the SQL query: SELECT * from table_name WHERE update > last_import_time, make a GET requests to `http://SOLR_URL:PORT/solr/CORE_NAME/dataimport?command=delta-import`. To import the data initially, we will issue a full import, then after that we can simply run delta imports. To configure the data import handler, we create a data-config.xml file which contains the SQL statements necessary for importing the data as well as a mapping of fields from the SQL results set to the corresponding document. An example of the configuration is here:

Listing 4.1. code/ngrams_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://csxdb:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="ngram"
      pk="id"
      query="SELECT id, paper_id, ngram, count from
          paper_keywords_noun"
      >
      <field column="id" name="id"/>
      <field column="paper_id" name="paper_id"/>
      <field column="ngram" name="ngram"/>
      <field column="count" name="count"/>
     </entity>
  </document>
</dataConfig>
```

4.6 Refactoring

Refactoring the backend of CiteSeer^x was one of the most time-consuming aspects of this project. The goal is to take a page that makes queries to MySQL and

convert them to make calls to Solr. The process I used was fairly straightforward. First, I enabled the MySQL query log and configured it so that it logged all queries. Then, I installed CiteSeer^x locally, and continuously tailed the query log. From there, I knew which queries to target, and could search the code in order to find them. There was roughly one DAO (data access object) class for each database table. In order not to duplicate the update / insert logic, for each of the DAO's with SELECT statements, I subclassed it and re-wrote any of the methods that performed get requests. We use SolrJ, a Java client library for interacting with Solr which made the calls to Solr extremely straightforward. Lastly, I updated the Java bean configuration (applicationContext-csx-jdbc.xml) to instantiate my subclasses of the DAO instead of the SQL version.

4.7 Scalability

One of the primary reasons that we chose Solr over the other NoSQL data stores was because of SolrCloud. Since we use SolrJ, moving to SolrCloud from the client side is extremely easy. Within the SolrJ library, the main class is SolrServer. However, this is actually an implementation of the interface SolrServerInterface, and there is a SolrCloud implementation as well. When we migrate to solr cloud, all we have to change is how we instantiate the client object, which is one line per DAO. Instead of constructing the object with the url of the index as a parameter, we instead call it with the name of the index and location of our zookeeper nodes(s).

This way, SolrJ will query ZooKeeper for the service discovery of the indices. From then on, all of the existing code will work. The benefit that using SolrCloud provides is effectively instant scalability. Solr can automatically shard the indices when one of them gets particularly busy, or simply if the machine is running out of resources.

4.8 Issues

4.8.1 Eventually Consistent

One of the largest downsides of this solution is that the system will follow an eventually consistent model. This is because writes will still go to the MySQL database but all reads are coming from Solr, which only updates in batch on a daily or weekly basis. The benefit of this is that the performance from both the webapp and ingestion are improved. The downside is obviously the delay in updates, however the performance improvement outweighs a slight delay in user corrections. A possible solution to this problem is outlined in Chapter 6.

4.8.2 Modification Time

In order to use Solr's data handle for incremental updates, Solr needs a way to find out which records have changed. To do this, it uses a timestamp field that stores the last time a record was modified. Then, when doing a incremental import, one

can simply retrieve all records that have been modified since last batchimport into Solr. Most of the tables in CiteSeer^x didn't have a time created or time updated field, so we had to add that in order to make incremental importing work.

4.8.3 Slow Solr Startup

When deploying Solr we found that on startup Solr would use 100% CPU utilization and be completely unresponsive. Upon investigation, this was due to the SuggestComponent being loaded by default. This component has fuzzy lookup support which we do not use, and requires loading all of the documents into the field cache which is detremental to performance. Upon removing the Suggestion Component from the solrconfig.xml, we found that Solr started immediately with minimal CPU utilization.

4.8.4 MySQL Transaction Pointcut

In the current codebase, Aspect Oriented Programming (AOP) is used to wrap all methods in the Data Access Layer in a transaction. It does this with a pointcut, which begins a transaction, calls the underlying method, and upon completion ends the method. Unfortunately wrapping each read in a transaction is not necessary and causes the database to act in serial rather than in parallel. In addition, since all reads now go to Solr instead of MySQL we do not need the transactions. However, we still want the ingestion system to use transactions on insert. To fix this we

changed the regular expression to not match any class with Solr in the name, which is all of my custom subclasses that perform reads to Solr.

Chapter 5

Results

5.1 Ingestion

The first thing to look at in the new system is how the fast the system can ingest new documents. Since the new architecture is indexing the data from MySQL using Solr's data import handler, it can batch the requests to MySQL. In table 5.1 we can see the results of doing a full import of the CiteSeer^{x} collection into Solr running on a 2.8 Ghz Intel Core i7 machine with 8 GB memory using the data import handler.

Index Name	Num documents	Duration	Doc / sec
authors	21,595,531	11m 59s	30,036
citations	158,483,836	4h 21s	10,990
papers	7,101,306	11m 08s	10,631
keywords	7,379,041	2m 40s	46,119
ngrams	82,735,728	31m 49s	43,439
urls	9,608,090	4m 24s	36,394

Table 5.1. Ingestion times into Solr from MySQL using Data-Import-Handler

These times are to import the entire CiteSeer^x collection. The reason the papers index has a lower rate of import (10k/s) is due to the much larger size of each document, mostly from the abstract. Similarly, the ngrams index was the fastest at around 44k/s due to the very small size of each document.

5.2 Front End Performance

While the ingestion performance is important, equally important is the performance of the new system on the front end web servers. To evaluate the NoSQL data store (Solr) vs MySQL, we replayed a month of production traffic for the viewDoc summary page to an idle webserver configured with NoSQL, then re-ran the experiment using the older MySQL based system. When replaying the logs 8 threads were used to increase the parallelism. In total approximately 5 million requests were made in about 20 hours. The same system was used for running Solr and MySQL in order to fairly evaluate them. In this case a Dell PowerEdge R610 server was used with 48 GB of RAM and 2 Intel Xeon X5650 processors clocked at 2.67 GHz with 6 cores each. In figure 5.1 we see the distribution of the response times for 99% of the requests. We then compare this to the same chart for the NoSQL system in figure 5.2. For the SQL based system, the average response time was 96 milliseconds, and for the Solr based system it was 58 milliseconds which is a reduction of 39%. We also looked at the CPU utilization of the machine running Solr and MySQL for the two experiments. In figure 5.3 we see that Solr uses about

the same amount of CPU as MySQL as seen in figure 5.4. The average % utilized for MySQL was 12.16% and Solr used an average of 12.83%.

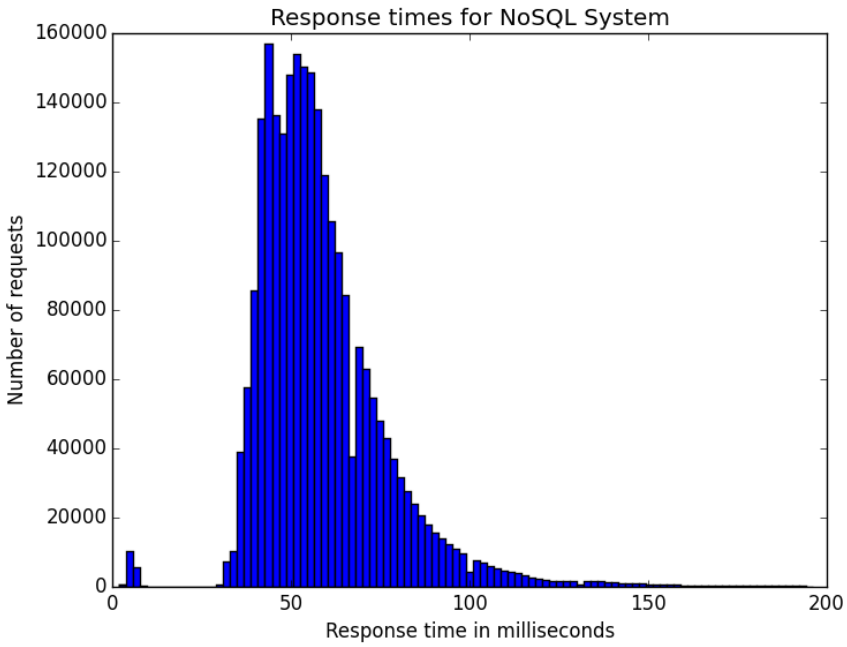

Figure 5.1. Response times for 99% of viewdoc summary requests for January 2016 using the NoSQL system.

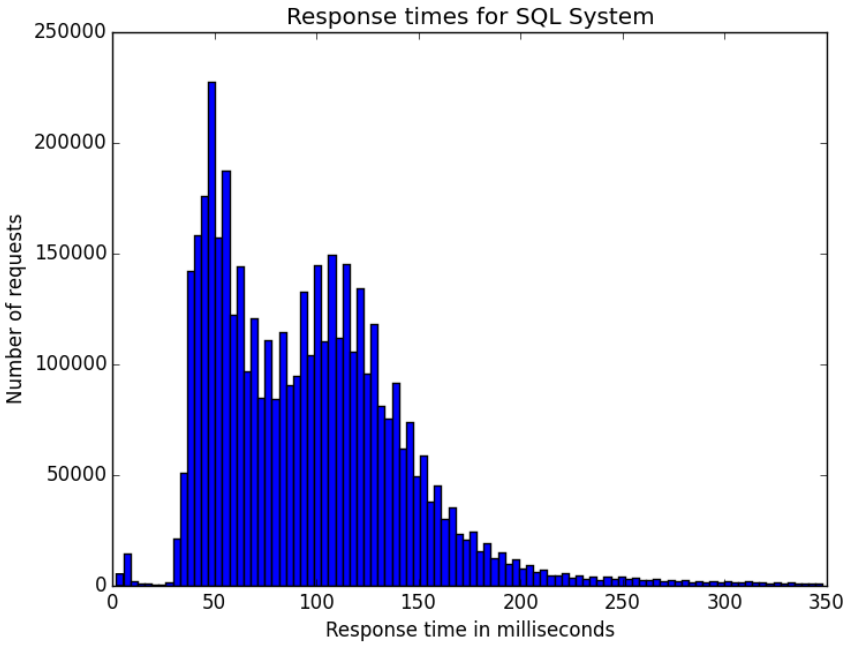

Figure 5.2. Response times for 99% of viewdoc summary requests for January 2016 using the MySQL system.

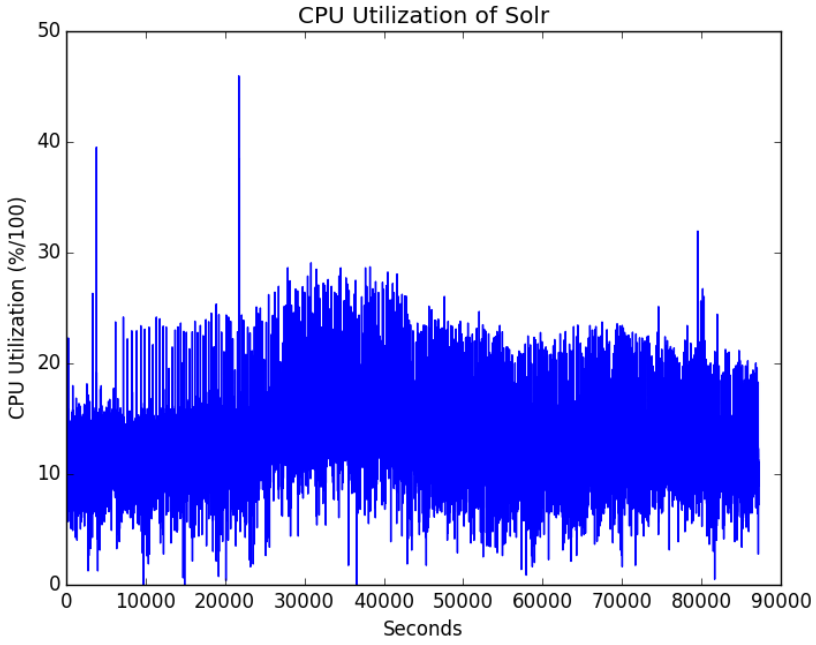

Figure 5.3. CPU Utilization of Solr.

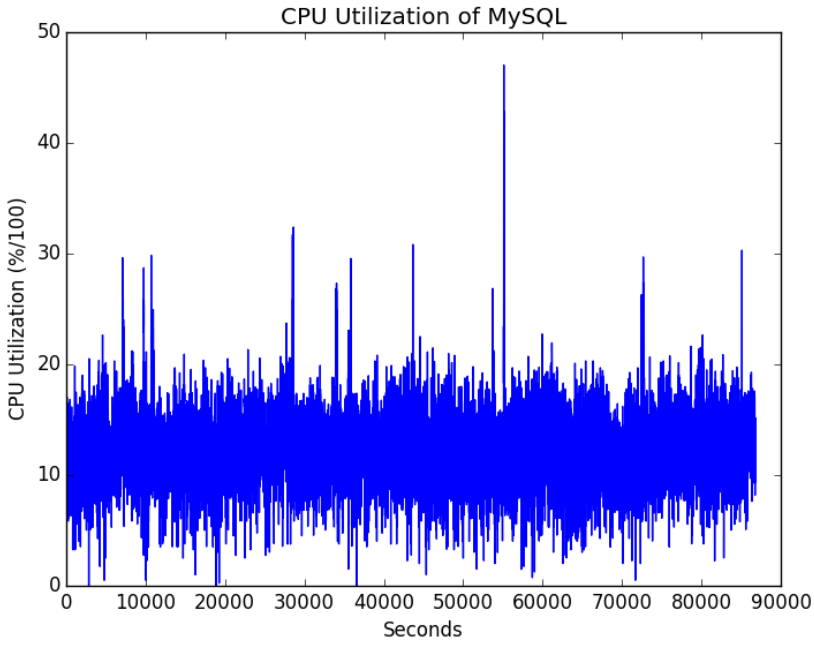

Figure 5.4. CPU Utilization of MySQL.

Chapter 6

Conclusions and Further Work

6.1 Conclusions

In this thesis we investigated scaling CiteSeer^{x} , both bigger and faster. We replaced the old GFS file system with a modular RESTful API that abstracts the specific implementation away from the user, and then added a caching layer on top of it. We found that due to the distribution of paper downloads, we can achieve a 38% cache hit rate with just 8 GB of RAM. We then evaluated various NoSQL datastores and chose Solr for its robust clustering support with SolrCloud as well as ease of use. We refactored the Data Access Object (DAO) of CiteSeer^{x} to support reading from Solr, and populated the data from MySQL into Solr. We found a 15% reduction of average response time with just one host, and found ingestion to be orders of magnitude faster as well.

6.2 Further Work

The most logical extension of the work presented here is to completely remove MySQL from the system. As it stands now, the system is a hybrid SQL and NoSQL system. In the future, one could replace MySQL completely with just Solr, or even a combination of Solr and another data store specializing in storing graphs like Neo4J. Another area that is worth exploring is migrating the system to the cloud. The price is the current barrier to enter although with cloud prices dropping extremely rapidly that may change in the future. If we were to host CiteSeerX in the cloud, we could explore using many of the cloud-only datastores such as Amazon's DynamoDB, Google's BigTable [23], or Microsoft's DocumentDB [24].

6.3 Eventual Consistency

One possible architectural solution to the eventual consistency problem is to create a write buffer that would cache any writes since the last batch update. If we do that, then we can check in that cache and apply those writes to the stale version so that from the perspective of the user writes are sequentially consistent. Then, when we bulk update every day / week, we can just apply those writes to Solr then flush the cache.

Appendix A

Code

A.1 GitHub

The code for this thesis can also be found on GitHub at `https://github.com/dwj300/citeseerx`.

A.2 Repository API

Listing A.1. code/server.py

```
# Reposity API interface
# Douglas Jordan
import web
import os
import errno
import os.path

repositorypath = '/data/repository/'
api_key = "c1t3s33r"

urls = ('/*', 'index', '/document', 'document')
```

```
def mkdir_p(path):
    try:
        os.makedirs(path)
    except OSError, e:
        if e.errno == errno.EEXIST:
            pass
        else:
            raise


def fbuffer(f, chunk_size=262144):
    while True:
        chunk = f.read(chunk_size)
        if not chunk:
            break
        yield chunk


class index:
    def GET(self):
        return "Repository RESTful API"


class document:
    def GET(self):
        try:
            repid = str(web.input().get('repid'))
            doi = str(web.input().get('doi'))
            filetype = str(web.input().get('type'))
            query = str(web.input().get('q'))

            # If not all the keys are specified return 404
            if None in (repid, doi):
                return web.webapi.NotFound()

            # Main getDocument logic
            filename = repositorypath + repid
            filealias = doi + '.' + filetype

            for x in doi.split('.'):
                filename = os.path.join(filename, x)

            if (query == 'filetypes'):
                files = os.listdir(filename)
                types = set()
                for f in files:
                    extension = os.path.splitext(f)[1]
                    types.add(extension.replace('.', ''))
                return ",".join(types)

            filename = os.path.join(filename, filealias)
```

```
            if os.path.isfile(filename):
                try:
                    getFile = file(filename, 'r')
                    val_disp = 'inline; filename={0}'.format(
                        filealias)
                    val_type = 'application/{0}'.format(filetype)
                    size = os.stat(filename).st_size
                    web.webapi.header('Content-Disposition',
                        val_disp)
                    web.webapi.header('Content-Type', val_type)
                    web.webapi.header('Content-length', size)

                    # Return the file data
                    return getFile.read()
                except IOError:
                    return web.webapi.NotFound('File Not Found')

            else:
                return web.webapi.NotFound('File Not Found')
        except:
            print(web.ctx.home + web.ctx.homepath + web.ctx.
                fullpath)

    def POST(self):
        doi = web.input().get('doi')
        repid = web.input().get('repid')
        File = web.input().get('file')
        Type = web.input().get('type').lower()
        version = web.input().get('version')

        if None in (doi, repid, File, Type):
            return web.webapi.NotFound()

        print "we got post + {0}".format(doi)
        allowedTypes = ['pdf', 'ps', 'xml', 'header', 'met', '
            parscit', 'txt']
        if Type in allowedTypes:
            if version is None:
                return web.input().keys()
            else:
                filealias = doi + 'v%s.%s' % (version, Type)
                filename = repositorypath + repid
                for x in doi.split('.'):
                    filename = os.path.join(filename, x)
                # create directory if it doesn't already exist
                mkdir_p(filename)
                filename = os.path.join(filename, filealias)
                # no overwrites possible
                if os.path.isfile(filename):
                    return web.webapi.BadRequest("File already
                        exists")
```

```
                try:
                    f = open(filename, 'w')
                    f.write(File)
                    f.close()
                    return web.webapi.OK()
                except IOError, e:
                    print "File write error %s" % e.msg
                    return web.webapi.NotAcceptable()

app = web.application(urls, globals())
application = app.wsgifunc()

if __name__ == "__main__":
    app.run()
```

Listing A.2. code/RepositoryService.java

```
package edu.psu.citeseerx.domain;

import java.io.IOException;
import java.io.InputStream;
import java.util.Map;

import edu.psu.citeseerx.repository.DocumentUnavailableException;

/**     RepositoryService
        Interface for storage of files and content.
        History: forked from FileSysDAO
        Reason: Separation of responsibility
*/
public interface RepositoryService {

        // Constants used in most implementations
        public static final String FILETYPE ="type";
        public static final String TEXTFILE = "TXT";
        public static final String PDFFILE = "PDF";
        public static final String PSFILE = "PS";
        public static final String RICHTEXT = "RTF";
        public static final String MSWORD = "DOC";
        public static final String XMLFILE = "XML";
        public static final String REPOSITORYID = "repid";
        public static final String BODYFILE = "BODY";
        public static final String QUERY = "q";
        public static final String FILEQUERY = "file";
        public static final String VERSIONKEY = "version";

        /* storeDocument
         * Basic method for storing a document in the repository
         * @param p  HashMap containing more information about the
             file being stored (should include doi, repid, type
            etc)
```

```
     * @param fpath  path to file being stored
     * @return nothing
     *
    */
    public void storeDocument(Map<String,String> p, String
       fpath) throws IOException;

    /* writeVersion
     * Basic method for writing a versioned document in the
        repository
     * @param doc  Document to be written
     *
     */
    public void writeVersion(Document doc) throws IOException;

    /* writeXML
     * Basic method for writing a Documen in xml format
     * @param doc  Document to be written
     */
    public void writeXML(Document doc) throws IOException;

    /* getDocument
     * Basic method for obtaining file stream for documents
     * @param  p HashMap containing more information about the
         file being stored (should include doi, repid, type
        etc)
     * @return InputStream for the document
     */
    public InputStream getDocument(Map<String,String> p)
       throws IOException, DocumentUnavailableException;

    /* fileTypes
     *  List the filetypes available
     *  @param p HashMap containing more information about the
         document, (doi and repid are required)
     */
    public String[] fileTypes(Map<String,String> p) throws
       IOException;

    /* getDocFromXML
     *      Get CSX document by providing the doi, the
        repository service fetches the xml and extracts the
        document from it
     *  @param repID - String identifying the repository
     *  @param relpath - relative path to the document
     */
    public Document getDocFromXML(String repID, String relpath
       ) throws IOException, DocumentUnavailableException;

    /* getDocumentContent
     * @param p HashMap containing more information about the
```

```
            document, (doi and repid are required)
         * @return String contents of the document being requested
         */
        public String getDocumentContent(Map<String,String> p)
           throws IOException, DocumentUnavailableException;
}
```

Listing A.3. code/RESTRepository.java

```
package edu.psu.citeseerx.repository;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import edu.psu.citeseerx.domain.Document;
import edu.psu.citeseerx.domain.DocumentFileInfo;
import edu.psu.citeseerx.domain.RepositoryEndPoint;
import edu.psu.citeseerx.domain.RepositoryService;
import edu.psu.citeseerx.utility.HttpRESTUtils;

public class RESTRepository implements RepositoryService {

    private RepositoryEndPoint repositoryEndPoint;


    public RepositoryEndPoint getRepositoryEndPoint() {
        return repositoryEndPoint;
    }

    public void setRepositoryEndPoint(RepositoryEndPoint
       repositoryEndPoint) {
        this.repositoryEndPoint = repositoryEndPoint;
    }

    @Override
    public void storeDocument(Map<String, String> p, String file)
             throws IOException {
        if(!(p.containsKey(Document.DOI_KEY) || p.containsKey(
           RepositoryService.FILETYPE) || p.containsKey(
           RepositoryService.REPOSITORYID)))
            return;
        // Excepted by the repository - doi, type
```

```
        HttpRESTUtils.uploadFile(repositoryEndPoint, p,
            RepositoryService.FILEQUERY, file);
    }

    @Override
    public InputStream getDocument(Map<String, String> p) throws
        IOException {
        if(!(p.containsKey(Document.DOI_KEY) || p.containsKey(
            RepositoryService.FILETYPE) || p.containsKey(
            RepositoryService.REPOSITORYID)))
            return null;
        return HttpRESTUtils.getStreamFromHost(repositoryEndPoint,
            p);
    }

    @Override
    public String[] fileTypes(Map<String, String> p) throws
        IOException {
        if(!(p.containsKey(Document.DOI_KEY) || p.containsKey(
            RepositoryService.REPOSITORYID)))
            return null;
        p.put(RepositoryService.QUERY, "filetypes");
        String c = HttpRESTUtils.getFromHost(repositoryEndPoint, p
            );
        String []types = c.trim().split(",");

       List<String> validTypes = new ArrayList<String>();

        for (String type : types) {
            if (typeLookup.contains(type.toUpperCase())) {
                validTypes.add(type);
            }
        }
        Collections.sort(validTypes);
        String list[] = new String[validTypes.size()];
        validTypes.toArray(list);

        return list;
    }

    public static final String[] supportedTypes = new String[] {
            "PDF", "PS", "DOC", "RTF"
    };

    public static final Set<String> typeLookup = new HashSet<
        String>();

    static {
        for (String str : supportedTypes) {
            typeLookup.add(str);
        }
```

```
    }

    @Override
    public void writeVersion(Document doc) throws IOException {
        String repID = doc.getVersionRepID();
        String doi = doc.getDatum(Document.DOI_KEY, Document.
            UNENCODED);
        String fileType = RepositoryService.XMLFILE;
        HashMap<String,String> p = new HashMap<String,String>();
        p.put(REPOSITORYID, repID);
        p.put(Document.DOI_KEY, doi);
        p.put(VERSIONKEY, Integer.toString(doc.getVersion()));
        p.put(RepositoryService.FILETYPE, RepositoryService.
            XMLFILE);
        File tXMLFile = File.createTempFile("citeseerx", "xml");
        FileOutputStream out = new FileOutputStream(tXMLFile);
        doc.toXML(out, Document.INCLUDE_SYS_DATA);
        out.close();
        HttpRESTUtils.uploadFile(repositoryEndPoint, p,
            RepositoryService.FILEQUERY, tXMLFile.getAbsolutePath()
            );
    }

    @Override
    public void writeXML(Document doc) throws IOException {
        HashMap<String,String> p = new HashMap<String,String>();
        p.put(RepositoryService.FILETYPE, RepositoryService.
            XMLFILE);
        p.put(Document.DOI_KEY, doc.getDatum(Document.DOI_KEY,
            Document.UNENCODED));
        DocumentFileInfo finfo = doc.getFileInfo();
        String repID = finfo.getDatum(DocumentFileInfo.REP_ID_KEY,
                DocumentFileInfo.UNENCODED);
        p.put(RepositoryService.REPOSITORYID, repID);
        File tXMLFile = File.createTempFile("citeseerx", "xml");
        FileOutputStream out = new FileOutputStream(tXMLFile);
        doc.toXML(out, Document.INCLUDE_SYS_DATA);
        out.close();
        HttpRESTUtils.uploadFile(repositoryEndPoint, p,
            RepositoryService.FILEQUERY, tXMLFile.getAbsolutePath()
            );
    }

    @Override
    public Document getDocFromXML(String repID, String relpath)
       throws IOException {
        return null;
    }

    @Override
    public String getDocumentContent(Map<String, String> p) throws
```

```
        IOException {
        if(!(p.containsKey(Document.DOI_KEY) || p.containsKey(
            RepositoryService.REPOSITORYID)))
             return null;
        return HttpRESTUtils.getFromHost(repositoryEndPoint, p);
    }
}
```

Listing A.4. code/FileSystemRepository.java

```
package edu.psu.citeseerx.repository;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.StringWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.commons.io.IOUtils;

import edu.psu.citeseerx.dao2.logic.CSXDAO;
import edu.psu.citeseerx.domain.Document;
import edu.psu.citeseerx.domain.DocumentFileInfo;
import edu.psu.citeseerx.domain.RepositoryService;
import edu.psu.citeseerx.utility.FileNamingUtils;
import edu.psu.citeseerx.utility.FileUtils;
import edu.psu.citeseerx.utility.SafeText;

public class FileSystemRepository implements RepositoryService {

    private RepositoryMap repMap;
    private CSXDAO csxdao;

    public RepositoryMap getRepMap() {
        return repMap;
    }

    public void setRepMap(RepositoryMap repMap) {
        this.repMap = repMap;
    }

    public CSXDAO getCsxdao() {
```

```
    return csxdao;
}

public void setCsxdao(CSXDAO csxdao) {
    this.csxdao = csxdao;
}

/*
 * (non-javadoc)
 * @see edu.psu.citeseerx.dao2.FileSysDAO#getDocFromXML(java.
    lang.String, java.lang.String)
 */
public Document getDocFromXML(String repID, String relPath)
throws IOException {

    String path = repMap.buildFilePath(repID, relPath);
    FileInputStream in = new FileInputStream(path);

    Document doc = new Document();
    doc.fromXML(in);

    in.close();
    return doc;

}  //- getDocFromXML


/*
 * (non-javadoc)
 * @see edu.psu.citeseerx.dao2.FileSysDAO#getDocVersion(java.
    lang.String, int)
 */
public Document getDocVersion(String doi, int version) throws
   IOException {

    String repID, path;
    String name = null;
    boolean deprecated = false;
    boolean spam = false;

    if (version > 0) {

        Document holder = csxdao.getDocVersion(doi, version);
        if (holder == null) return null;

        repID = holder.getVersionRepID();
        path = holder.getVersionPath();
        name = holder.getVersionName();
        deprecated = holder.isDeprecatedVersion();
        spam = holder.isSpamVersion();
```

```
        } else {
            version = 0;
            repID = csxdao.getRepositoryID(doi);
            path = FileNamingUtils.buildXMLPath(doi);
        }

        Document doc = getDocFromXML(repID, path);

        doc.setVersion(version);
        doc.setVersionName(name);
        doc.setVersionRepID(repID);
        doc.setVersionPath(path);
        doc.setVersionDeprecated(deprecated);
        doc.setVersionSpam(spam);

        return doc;

    }  //- getVersion


    /*
     * (non-javadoc)
     * @see edu.psu.citeseerx.dao2.FileSysDAO#getDocVersion(java.
        lang.String, java.lang.String)
     */
    public Document getDocVersion(String doi, String name) throws
       IOException {
        Document holder = csxdao.getDocVersion(doi, name);
        if (holder == null) return null;

        Document doc = getDocFromXML(holder.getVersionRepID(),
                holder.getVersionPath());

        doc.setVersion(holder.getVersion());
        doc.setVersionName(holder.getVersionName());
        doc.setVersionRepID(holder.getVersionRepID());
        doc.setVersionPath(holder.getVersionPath());
        doc.setVersionDeprecated(holder.isDeprecatedVersion());
        doc.setVersionSpam(holder.isSpamVersion());

        return doc;

    }  //- getVersion


    /*
     * (non-javadoc)
     * @see edu.psu.citeseerx.dao2.FileSysDAO#writeXML(edu.psu.
        citeseerx.domain.Document)
     */
    public void writeXML(Document doc) throws IOException {
```

```
        String doi = doc.getDatum(Document.DOI_KEY, Document.
            UNENCODED);

        DocumentFileInfo finfo = doc.getFileInfo();
        String repID = finfo.getDatum(DocumentFileInfo.REP_ID_KEY,
                DocumentFileInfo.UNENCODED);
        String relPath = FileNamingUtils.buildXMLPath(doi);
        String path = repMap.buildFilePath(repID, relPath);

        FileOutputStream out = new FileOutputStream(path);
        doc.toXML(out, Document.INCLUDE_SYS_DATA);
        out.close();

    }  //- writeXML


    /*
     * (non-javadoc)
     * @see edu.psu.citeseerx.dao2.FileSysDAO#writeVersion(edu.psu
         .citeseerx.domain.Document)
     */
    public void writeVersion(Document doc) throws IOException {

        String repID = doc.getVersionRepID();
        String relPath = doc.getVersionPath();
        String path = repMap.buildFilePath(repID, relPath);

        FileOutputStream out = new FileOutputStream(path);
        doc.toXML(out, Document.INCLUDE_SYS_DATA);
        out.close();

    }  //- writeVersion


    /*
     * (non-javadoc)
     * @see edu.psu.citeseerx.dao2.FileSysDAO#getFileInputStream(
         java.lang.String, java.lang.String, java.lang.String)
     */
    public FileInputStream getFileInputStream(String doi, String
        repID,
             String type) throws IOException {

        String dir = FileNamingUtils.getDirectoryFromDOI(doi);
        String fn = doi+"."+type;
        String relPath = dir+System.getProperty("file.separator")+
            fn;
        String path = repMap.buildFilePath(repID, relPath);

        return new FileInputStream(path);
```

```
    }  //- getFileInputStream



    @Override
    public void storeDocument(Map<String, String> p, String
        filePath)
            throws IOException {
        if(!(p.containsKey(Document.DOI_KEY) || p.containsKey(
            RepositoryService.FILETYPE) || p.containsKey(
            RepositoryService.REPOSITORYID)))
                return;
        String doi = p.get(Document.DOI_KEY);
        String type = p.get(RepositoryService.FILETYPE);
        String repID = p.get(RepositoryService.REPOSITORYID);
        String fn = doi+"."+ type;
        String dir = FileNamingUtils.getDirectoryFromDOI(doi);
        String relPath = dir+System.getProperty("file.separator")+
            fn;
        String path = repMap.buildFilePath(repID, relPath);
        File fileObj = new File(relPath);
        File fromObjc = new File(filePath);
        try {
            IOUtils.copy(new FileInputStream(filePath), new
                FileOutputStream(fileObj));
        } catch(IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public InputStream getDocument(Map<String, String> p) throws
        IOException, DocumentUnavailableException {
        if(!(p.containsKey(Document.DOI_KEY) || p.containsKey(
            RepositoryService.FILETYPE) || p.containsKey(
            RepositoryService.REPOSITORYID)))
            return null;
        String doi = p.get(Document.DOI_KEY);
        String type = p.get(RepositoryService.FILETYPE);
        String repID = p.get(RepositoryService.REPOSITORYID);

        String dir = FileNamingUtils.getDirectoryFromDOI(doi);
        String fn = doi+"."+type;
        String relPath = dir+System.getProperty("file.separator")+
            fn;
        String path = repMap.buildFilePath(repID, relPath);
        if(new File(path).exists()) {
            return new FileInputStream(path);
        }
        else {
```

```
            throw new DocumentUnavailableException();
        }
    }

    @Override
    public String[] fileTypes(Map<String, String> p) throws
        IOException {
        if(!(p.containsKey(Document.DOI_KEY) || p.containsKey(
            RepositoryService.REPOSITORYID)))
            return null;
        String doi = p.get(Document.DOI_KEY);
        String repID = p.get(RepositoryService.REPOSITORYID);

        String dir = FileNamingUtils.getDirectoryFromDOI(doi);
        String path = repMap.buildFilePath(repID, dir);

        String[] files = new File(path).list();
        List<String> types = new ArrayList<String>();

        if (files == null) {
            String list[] = new String[types.size()];
            types.toArray(list);
            return list;
        }

        for (String file : files) {
            String ext = FileUtils.getExtension(file);
            ext = ext.substring(1);
            if (typeLookup.contains(ext.toUpperCase())) {
                types.add(ext);
            }
        }
        Collections.sort(types);
        String list[] = new String[types.size()];
        types.toArray(list);
        return list;
    }

    public static final String[] supportedTypes = new String[] {
            "PDF", "PS", "DOC", "RTF"
    };

    public static final Set<String> typeLookup = new HashSet<
        String>();

    static {
        for (String str : supportedTypes) {
            typeLookup.add(str);
        }
    }
```

```
/*
 * (non-javadoc)
 * @see edu.psu.citeseerx.dao2.FileSysDAO#getFileTypes(java.
    lang.String, java.lang.String)
 */
public List<String> getFileTypes(String doi, String repID)
throws IOException {

    String dir = FileNamingUtils.getDirectoryFromDOI(doi);
    String path = repMap.buildFilePath(repID, dir);

    String[] files = new File(path).list();
    List<String> types = new ArrayList<String>();

    if (files == null) {
        return types;
    }

    for (String file : files) {
        String ext = FileUtils.getExtension(file);
        ext = ext.substring(1);
        if (typeLookup.contains(ext.toUpperCase())) {
            types.add(ext);
        }
    }
    Collections.sort(types);
    return types;
}  //- getFileTypes


@Override
public String getDocumentContent(Map<String, String> p)
        throws IOException {
    if(!(p.containsKey(Document.DOI_KEY) || p.containsKey(
       RepositoryService.REPOSITORYID)))
        return null;

    String doi = p.get(Document.DOI_KEY);
    if (doi == null) {
        return null;
    }
    FileInputStream ins = null;
    BufferedReader reader = null;
    try {
        ins = (FileInputStream)getDocument(p);

    } catch (DocumentUnavailableException e) { // No body file
        p.put(RepositoryService.FILETYPE, RepositoryService.
           TEXTFILE);
        try {
            ins = (FileInputStream)getDocument(p);
```

```
            }
            catch(DocumentUnavailableException f) {} // No text
                file either - we give up
        }
        try {
            reader = new BufferedReader(new InputStreamReader(ins,
                "UTF-8"));
            StringWriter sw = new StringWriter();
            IOUtils.copy(reader, sw);
            String text = sw.toString();
            return text;

        } catch (IOException e) {
            throw(e);
        } finally {
            try { reader.close(); } catch (Exception e) { }
            try { ins.close(); } catch(Exception e) { }
        }
    }
}
```

A.3 NoSQL Migration

Listing A.5. code/AckSolrImpl.java

```
package edu.psu.citeseerx.dao2;

import edu.psu.citeseerx.domain.Acknowledgment;
import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.springframework.dao.DataAccessException;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by Doug on 1/27/16.
 */
public class AckSolrImpl extends AckDAOImpl {

    private String solrUrl;
    private SolrClient solr;

    public void setSolrUrl(String solrUrl) {
        this.solrUrl = solrUrl + "acknowledgments";
    }
```

```
    @Override
    public List<Acknowledgment> getAcknowledgments(String doi,
       boolean withContexts, boolean withSource) throws
       DataAccessException {
        if (solr == null) {
            solr = new HttpSolrClient(solrUrl);
        }

        SolrQuery query = new SolrQuery();
        query.setQuery("paperid:\"" + doi + "\"");
        QueryResponse resp = null;
        try {
            resp = solr.query(query);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        List<Acknowledgment> acknowledgments = new ArrayList<
           Acknowledgment>();

        for (SolrDocument doc : resp.getResults()) {
            Acknowledgment ack = new Acknowledgment();
            ack.setDatum(Acknowledgment.DOI_KEY, doc.getFieldValue
               ("id").toString());
            ack.setClusterID((Long)doc.getFieldValue("cluster"));
            ack.setDatum(Acknowledgment.NAME_KEY, doc.
               getFieldValue("name").toString());
            ack.setDatum(Acknowledgment.ENT_TYPE_KEY, doc.
               getFieldValue("entType").toString());
            ack.setDatum(Acknowledgment.ACK_TYPE_KEY, doc.
               getFieldValue("ackType").toString());
            acknowledgments.add(ack);
        }
        return acknowledgments;
    }
}
```

Listing A.6. code/AuthorSolrImpl.java

```
package edu.psu.citeseerx.dao2;

import edu.psu.citeseerx.domain.Author;
import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.springframework.dao.DataAccessException;
import org.apache.solr.common.SolrDocument;

import java.util.List;
```

```
import java.util.ArrayList;

/**
 * Created by Doug on 1/22/16.
 */
public class AuthorSolrImpl extends AuthorDAOImpl {

    private String solrUrl;
    private SolrClient solr;

    public void setSolrUrl(String solrUrl) {
        this.solrUrl = solrUrl + "authors";
    }

    @Override
    public List<Author> getDocAuthors(String doi, boolean
       getSource)
            throws DataAccessException {
        if (solr == null) {
            solr = new HttpSolrClient(solrUrl);
        }

        SolrQuery query = new SolrQuery();
        query.setQuery("paperid:\"" + doi + "\"");
        QueryResponse resp = null;
        try {
            resp = solr.query(query);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        List<Author> authors = new ArrayList<Author>();

        for (SolrDocument doc : resp.getResults()) {
            Author auth = new Author();
            Object id = doc.getFieldValue("id");
            Object cluster = doc.getFieldValue("cluster");
            Object name = doc.getFieldValue("name");
            Object affil = doc.getFieldValue("affil");
            Object addr = doc.getFieldValue("address");
            Object email = doc.getFieldValue("email");
            Object ord = doc.getFieldValue("ord");

            if (id != null) {
                auth.setDatum(Author.DOI_KEY, id.toString());
            }
            if (cluster != null) {
                auth.setClusterID((long)cluster);
            }
            if (name != null) {
                auth.setDatum(Author.NAME_KEY, name.toString());
```

```
            }

            if (affil != null) {
                auth.setDatum(Author.AFFIL_KEY, affil.toString());
            }
            if (addr != null) {
                auth.setDatum(Author.ADDR_KEY, addr.toString());
            }
            if (email != null) {
                auth.setDatum(Author.EMAIL_KEY, email.toString());
            }
            if (ord != null) {
                auth.setDatum(Author.ORD_KEY, ord.toString());
            }

            authors.add(auth);
        }

        return authors;
    }
}
```

Listing A.7. code/CitationSolrImpl.java

```
package edu.psu.citeseerx.dao2;

import edu.psu.citeseerx.domain.Citation;
import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import org.springframework.dao.DataAccessException;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by Doug on 1/27/16.
 */
public class CitationSolrImpl extends CitationDAOImpl {

    private String solrUrl;
    private SolrClient solr;

    public void setSolrUrl(String solrUrl) {
        this.solrUrl = solrUrl + "citations";
    }

    @Override
    public List<Citation> getCitations(String doi, boolean
```

```
withContexts)
     throws DataAccessException {

 if (solr == null) {
     solr = new HttpSolrClient(solrUrl);
 }

 SolrQuery query = new SolrQuery();
 query.setQuery("paperid:\"" + doi + "\"");
 QueryResponse resp = null;
 try {
     resp = solr.query(query);
 } catch (Exception e) {
     e.printStackTrace();
     return null;
 }
 List<Citation> citations = new ArrayList<Citation>();

 for (SolrDocument doc : resp.getResults()) {
     Citation citation = new Citation();

     citation.setDatum(Citation.DOI_KEY, doc.getFieldValue(
        "id").toString());
     citation.setClusterID((Long)doc.getFieldValue("cluster
        "));

     String[] authors = doc.getFieldValue("authors").
        toString().split(",");
     for (int i = 0; i < authors.length; i++) {
         citation.addAuthorName(authors[i]);
     }

     String[] keys = {Citation.TITLE_KEY, Citation.
        VENUE_KEY, Citation.VEN_TYPE_KEY,
             Citation.YEAR_KEY, Citation.PAGES_KEY,
                Citation.EDITORS_KEY, Citation.
                PUBLISHER_KEY,
             Citation.PUB_ADDR_KEY, Citation.VOL_KEY,
                Citation.NUMBER_KEY, Citation.TECH_KEY,
             Citation.RAW_KEY};

     for(String key : keys) {
         Object obj = doc.getFieldValue(key);
         if (obj != null) {
             citation.setDatum(key, obj.toString());
         }
     }

     citation.setDatum(Citation.PAPERID_KEY, doc.
        getFieldValue("paperid").toString());
     citation.setSelf(((Integer)doc.getFieldValue("self"))
```

```
                == 1);
            citations.add(citation);
        }

        return citations;

    }  //- getCitations
}
```

Listing A.8. code/DocumentSolrImpl.java

```
package edu.psu.citeseerx.dao2;

import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.springframework.dao.DataAccessException;
import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;

import java.util.ArrayList;
import java.util.Date;
import edu.psu.citeseerx.domain.*;
import java.text.DateFormat;
import java.util.List;

/**
 * Created by Doug on 1/20/16.
 */
public class DocumentSolrImpl extends DocumentDAOImpl {
    /* (non-Javadoc)
     * @see edu.psu.citeseerx.dao2.DocumentDAO#getDocument(java.
        lang.String, boolean)
     */

    private String solrUrl;

    private String NGRAM_SOLR_URL;
    private SolrClient solr;
    private SolrClient ngramSolr;

    public void setSolrUrl(String solrUrl) {
        this.solrUrl = solrUrl + "papers";
        this.NGRAM_SOLR_URL = solrUrl + "ngrams";
    }

    @Override
    public Document getDocument(String doi, boolean getSource)
       throws DataAccessException {
        if (solr == null) {
            solr = new HttpSolrClient(solrUrl);
```

```
        }

        SolrQuery query = new SolrQuery();
        query.setQuery("id:\"" + doi + "\"");

        QueryResponse resp = null;
        try {
            resp = solr.query(query);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

        if (resp.getResults().getNumFound() == 0) {
            System.out.println("no results found for doi:" + doi);
            return null;
        }
        SolrDocument solrDoc = resp.getResults().get(0);

        Document doc = new Document();

        String[] keys = {Document.TITLE_KEY, Document.ABSTRACT_KEY
            , Document.YEAR_KEY, Document.VENUE_KEY,
                        Document.VEN_TYPE_KEY, Document.PAGES_KEY
                            , Document.VOL_KEY, Document.
                            PUBLISHER_KEY,
                        Document.PUBADDR_KEY, Document.TECH_KEY
                            };

        for (String key : keys) {
            Object obj = solrDoc.getFieldValue(key);
            if (obj != null) {
                doc.setDatum(key, obj.toString());
            }
        }

        Object id = solrDoc.getFieldValue("id");
        Object version = solrDoc.getFieldValue("version");
        Object cluster = solrDoc.getFieldValue("cluster");
        Object ncites = solrDoc.getFieldValue("ncites");
        Object selfCites = solrDoc.getFieldValue("selfCites");
        Object versionName = solrDoc.getFieldValue("versionName");
        Object state = solrDoc.getFieldValue("public");
        Object versionTime = solrDoc.getFieldValue("versionTime");

        if (id != null) {
            doc.setDatum(Document.DOI_KEY, id.toString());
        }
        if (version != null) {
            doc.setVersion((int)version);
        }
```

```
        if (cluster != null) {
            doc.setClusterID((long)cluster);
        }
        if (ncites != null) {
            doc.setNcites((int)ncites);
        }
        if (selfCites != null) {
            doc.setSelfCites((int)selfCites);
        }
        if (versionName != null) {
            doc.setVersionName(versionName.toString());
        }
        if (state != null) {
            doc.setState((int)state);
        }
        if (versionTime != null) {
            doc.setVersionTime((Date)versionTime);
        }

        DocumentFileInfo finfo = new DocumentFileInfo();
        finfo.setDatum(DocumentFileInfo.CRAWL_DATE_KEY, DateFormat
            .getDateInstance().format((Date)solrDoc.getFieldValue("
            crawlDate")));
        finfo.setDatum(DocumentFileInfo.REP_ID_KEY, solrDoc.
            getFieldValue("repositoryID").toString());
        finfo.setDatum(DocumentFileInfo.CONV_TRACE_KEY, solrDoc.
            getFieldValue("conversionTrace").toString());
        doc.setFileInfo(finfo);

        return doc;

    }  //- getDocument

    @Override
    public List<String> getKeyphrase(String doi) throws
        DataAccessException {
        if (ngramSolr == null) {
            ngramSolr = new HttpSolrClient(NGRAM_SOLR_URL);
        }

        // TOOD: deal with sort
        SolrQuery query = new SolrQuery();
        query.setQuery("paper_id:\"" + doi + "\"");

        QueryResponse resp = null;
        try {
            resp = ngramSolr.query(query);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
```

```
        List<String> ngrams = new ArrayList<String>();

        for (SolrDocument doc : resp.getResults()) {
            ngrams.add(doc.getFieldValue("ngram").toString());
        }
        return ngrams;

    } //- getKeyphrase
}
```

Listing A.9. code/HubSolrImpl.java

```
package edu.psu.citeseerx.dao2;

import edu.psu.citeseerx.domain.Author;
import edu.psu.citeseerx.domain.Hub;
import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import org.springframework.context.ApplicationContextException;
import org.springframework.dao.DataAccessException;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by Doug on 1/27/16.
 */
public class HubSolrImpl extends HubDAOImpl {

    private String solrUrl;
    private SolrClient solr;

    public void setSolrUrl(String solrUrl) {
        this.solrUrl = solrUrl + "urls";
    }

    @Override
    public List<String> getUrls(String doi) throws
       DataAccessException {
        if (solr == null) {
            solr = new HttpSolrClient(solrUrl);
        }

        SolrQuery query = new SolrQuery();
        query.setQuery("paperid:\"" + doi + "\"");
        QueryResponse resp = null;
        try {
```

```
            resp = solr.query(query);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        List<String> urls = new ArrayList<String>();

        for (SolrDocument doc : resp.getResults()) {
            urls.add(doc.getFieldValue("url").toString());
        }

        return urls;
    }

    @Override
    public List<Hub> getHubs(String doi) throws
       DataAccessException {
        if (solr == null) {
            solr = new HttpSolrClient(solrUrl);
        }

        SolrQuery query = new SolrQuery();
        query.setQuery("paperid:\"" + doi + "\"");
        QueryResponse resp = null;
        try {
            resp = solr.query(query);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        List<Hub> hubs = new ArrayList<Hub>();

        for (SolrDocument doc : resp.getResults()) {
            Hub hub = new Hub();
            hub.setUrl(doc.getFieldValue("url").toString());
        }

        return hubs;
    }  //- getHubs
}
```

Listing A.10. code/KeywordSolrImpl.java

```
package edu.psu.citeseerx.dao2;

import edu.psu.citeseerx.domain.Keyword;
import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
```

```
import org.springframework.dao.DataAccessException;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by Doug on 1/27/16.
 */
public class KeywordSolrImpl extends KeywordDAOImpl {

    private String solrUrl;
    private SolrClient solr;

    public void setSolrUrl(String solrUrl) {
        this.solrUrl = solrUrl + "keywords";
    }

    @Override
    public List<Keyword> getKeywords(String doi, boolean getSource
        )
            throws DataAccessException {

        if (solr == null) {
            solr = new HttpSolrClient(solrUrl);
        }

        SolrQuery query = new SolrQuery();
        query.setQuery("paperid:\"" + doi + "\"");
        QueryResponse resp = null;
        try {
            resp = solr.query(query);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        List<Keyword> keywords = new ArrayList<Keyword>();

        for (SolrDocument doc : resp.getResults()) {
            Keyword keyword = new Keyword();
            keyword.setDatum(Keyword.DOI_KEY, doc.getFieldValue("
                id").toString());
            keyword.setDatum(Keyword.KEYWORD_KEY, doc.
                getFieldValue("keyword").toString());
            keywords.add(keyword);
        }
        return keywords;
    }
}
```

Listing A.11. code/TagSolrImpl.java

```
package edu.psu.citeseerx.dao2;

import edu.psu.citeseerx.domain.Keyword;
import edu.psu.citeseerx.domain.Tag;
import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.impl.HttpSolrClient;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import org.springframework.dao.DataAccessException;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by Doug on 1/27/16.
 */
public class TagSolrImpl extends TagDAOImpl {

    private static String solrUrl;
    private SolrClient solr;

    public void setSolrUrl(String solrUrl) {
        this.solrUrl = solrUrl + "tags";
    }

    public List<Tag> getTags(String doi) throws
       DataAccessException {
        if (solr == null) {
            solr = new HttpSolrClient(solrUrl);
        }

        SolrQuery query = new SolrQuery();
        query.setQuery("paperid:\"" + doi + "\"");
        QueryResponse resp = null;
        try {
            resp = solr.query(query);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        List<Tag> tags = new ArrayList<Tag>();

        for (SolrDocument doc : resp.getResults()) {
            Tag tag = new Tag();
            tag.setTag(doc.getFieldValue("tag").toString());
            tag.setCount((Integer)doc.getFieldValue("count"));
            tags.add(tag);
        }
        return tags;
    } //- getTags
```

```
}
```

A.4 SolrConfiguration

Listing A.12. code/ack_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://csxdb:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="acknowledgment"
      pk="id"
      query="SELECT id, cluster, name, entType, ackType, paperid
         from acknowledgments"
      >
      <field column="id" name="id"/>
      <field column="cluster" name="cluster"/>
      <field column="name" name="name"/>
      <field column="entType" name="entType"/>
      <field column="ackType" name="ackType"/>
      <field column="paperid" name="paperid"/>
     </entity>
  </document>
</dataConfig>
```

Listing A.13. code/ack_schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>


<schema name="papers" version="1.5">

   <field name="_version_" type="long" indexed="true" stored="true
      "/>

   <field name="id" type="long" indexed="true" stored="true"
      required="true" multiValued="false"/>
   <field name="cluster" type="long" indexed="true" stored="true
      "/>
   <field name="name" type="string" indexed="true" stored="true"/>
   <field name="entType" type="string" indexed="true" stored="true
      "/>
   <field name="ackType" type="string" indexed="true" stored="true
      "/>
   <field name="paperid" type="string" indexed="true" stored="true
      "/>
```

```
</schema>
```

Listing A.14. code/authors_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://csxdb:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="author"
      pk="id"
      query="SELECT id, cluster, name, affil, address, email, ord,
          paperid from authors"
      >
      <field column="id" name="id"/>
      <field column="cluster" name="cluster"/>
      <field column="name" name="name"/>
      <field column="affil" name="affil"/>
      <field column="address" name="address"/>
      <field column="email" name="email"/>
      <field column="ord" name="ord"/>
      <field column="paperid" name="paperid"/>
     </entity>
  </document>
</dataConfig>
```

Listing A.15. code/authors_schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>


<schema name="papers" version="1.5">

   <field name="_version_" type="long" indexed="true" stored="true
      "/>

   <field name="id" type="long" indexed="true" stored="true"
      required="true" multiValued="false"/>
   <field name="cluster" type="long" indexed="true" stored="true
      "/>
   <field name="name" type="string" indexed="true" stored="true"/>
   <field name="affil" type="string" indexed="true" stored="true
      "/>
   <field name="address" type="string" indexed="true" stored="true
      "/>
   <field name="email" type="string" indexed="true" stored="true
      "/>
   <field name="ord" type="int" indexed="true" stored="true"/>
```

```
   <field name="paperid" type="string" indexed="true" stored="true
      "/>

</schema>
```

Listing A.16. code/citations_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://csxdb:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="citation"
      pk="id"
      query="SELECT id, cluster, authors, title, venue, venueType,
          year, pages, editors, publisher, pubAddress, volume,
         number, tech, raw, paperid, self from citations"
      >
      <field column="id" name="id"/>
      <field column="cluster" name="cluster"/>
      <field column="authors" name="authors"/>
      <field column="title" name="title"/>
      <field column="venue" name="venue"/>
      <field column="venueType" name="venueType"/>
      <field column="year" name="year"/>
      <field column="pages" name="pages"/>
      <field column="editors" name="editors"/>
      <field column="publisher" name="publisher"/>
      <field column="pubAddress" name="pubAddress"/>
      <field column="volume" name="volume"/>
      <field column="number" name="number"/>
      <field column="tech" name="tech"/>
      <field column="raw" name="raw"/>
      <field column="paperid" name="paperid"/>
      <field column="self" name="self"/>
     </entity>
  </document>
</dataConfig>
```

Listing A.17. code/citations_schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>


<schema name="papers" version="1.5">

   <field name="_version_" type="long" indexed="true" stored="true
      "/>
```

```
   <field name="id" type="long" indexed="true" stored="true"
      required="true" multiValued="false"/>
   <field name="cluster" type="long" indexed="true" stored="true
      "/>
   <field name="authors" type="string" indexed="true" stored="true
      "/>
   <field name="title" type="string" indexed="true" stored="true
      "/>
   <field name="venue" type="string" indexed="true" stored="true
      "/>
   <field name="venueType" type="string" indexed="true" stored="
      true"/>
   <field name="year" type="int" indexed="true" stored="true"/>
   <field name="pages" type="string" indexed="true" stored="true
      "/>
   <field name="editors" type="string" indexed="true" stored="true
      "/>
   <field name="publisher" type="string" indexed="true" stored="
      true"/>
   <field name="pubAddress" type="string" indexed="true" stored="
      true"/>
   <field name="volume" type="int" indexed="true" stored="true"/>
   <field name="number" type="int" indexed="true" stored="true"/>
   <field name="tech" type="string" indexed="true" stored="true"/>
   <field name="raw" type="string" indexed="true" stored="true"/>
   <field name="paperid" type="string" indexed="true" stored="true
      "/>
   <field name="self" type="int" indexed="true" stored="true"/>

</schema>
```

Listing A.18. code/keywords_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://csxdb:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="keyword"
      pk="id"
      query="SELECT id, keyword, paperid from keywords"
      >
      <field column="id" name="id"/>
      <field column="keyword" name="keyword"/>
      <field column="paperid" name="paperid"/>
     </entity>
  </document>
</dataConfig>
```

Listing A.19. code/keywords_schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>


<schema name="papers" version="1.5">

   <field name="_version_" type="long" indexed="true" stored="true
      "/>

   <field name="id" type="long" indexed="true" stored="true"
      required="true" multiValued="false"/>
   <field name="keyword" type="string" indexed="true" stored="true
      "/>
   <field name="paperid" type="string" indexed="true" stored="true
      "/>

</schema>
```

Listing A.20. code/ngrams_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://csxdb:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="ngram"
      pk="id"
      query="SELECT id, paper_id, ngram, count from
         paper_keywords_noun"
      >
      <field column="id" name="id"/>
      <field column="paper_id" name="paper_id"/>
      <field column="ngram" name="ngram"/>
      <field column="count" name="count"/>
     </entity>
  </document>
</dataConfig>
```

Listing A.21. code/ngrams_schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>


<schema name="papers" version="1.5">

   <field name="_version_" type="long" indexed="true" stored="true
      "/>

   <field name="id" type="long" indexed="true" stored="true"
```

```
      required="true" multiValued="false"/>
   <field name="paper_id" type="string" indexed="true" stored="
      true"/>
   <field name="ngram" type="string" indexed="true" stored="true
      "/>
   <field name="count" type="int" indexed="true" stored="true"/>

</schema>
```

Listing A.22. code/papers_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://csxdb:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="paper"
      pk="id"
      query="select id, version, cluster, title, abstract, year,
         venue, venueType, pages, volume, number, publisher,
         pubAddress, tech, public, ncites, versionName, crawlDate,
          repositoryID, conversionTrace, selfCites, versionTime
         from papers"
      deltaImportQuery="SELECT id, version, cluster, title,
         abstract, year, venue, venueType, pages, volume, number,
         publisher, pubAddress, tech, public, ncites, versionName,
          crawlDate, repositoryID, conversionTrace, selfCites,
         versionTime from papers WHERE id='${dih.delta.id}'"
      deltaQuery="SELECT id FROM papers  WHERE versionTime > '${
         dih.last_index_time}'"
      >
      <field column="id" name="id"/>
      <field column="version" name="version"/>
      <field column="cluster" name="cluster"/>
      <field column="title" name="title"/>
      <field column="abstract" name="abstract"/>
      <field column="year" name="year"/>
      <field column="venue" name="venue"/>
      <field column="venueType" name="venueType"/>
      <field column="pages" name="pages"/>
      <field column="volume" name="volume"/>
      <field column="number" name="number"/>
      <field column="publisher" name="publisher"/>
      <field column="pubAddress" name="pubAddress"/>
      <field column="tech" name="tech"/>
      <field column="public" name="public"/>
      <field column="ncites" name="ncites"/>
      <field column="versionName" name="versionName"/>
      <field column="crawlDate" name="crawlDate"/>
```

```
      <field column="repositoryID" name="repositoryID"/>
      <field column="conversionTrace" name="conversionTrace"/>
      <field column="selfCites" name="selfCites"/>
      <field column="versionTime" name="versionTime"/>
    </entity>
  </document>
</dataConfig>
```

Listing A.23. code/papers_schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>


<schema name="papers" version="1.5">

   <field name="_version_" type="long" indexed="true" stored="true
      "/>

   <field name="id" type="string" indexed="true" stored="true"
      required="true" multiValued="false"/>
   <field name="version" type="int" indexed="true" stored="true"/>
   <field name="cluster" type="long" indexed="true" stored="true
      "/>
   <field name="title" type="string" indexed="true" stored="true
      "/>
   <field name="abstract" type="string" indexed="true" stored="
      true"/>
   <field name="year" type="int" indexed="true" stored="true"/>
   <field name="venue" type="string" indexed="true" stored="true
      "/>
   <field name="venueType" type="string" indexed="true" stored="
      true"/>
   <field name="pages" type="string" indexed="true" stored="true
      "/>
   <field name="volume" type="int" indexed="true" stored="true"/>
   <field name="number" type="int" indexed="true" stored="true"/>
   <field name="publisher" type="string" indexed="true" stored="
      true"/>
   <field name="pubAddress" type="string" indexed="true" stored="
      true"/>
   <field name="tech" type="string" indexed="true" stored="true"/>
   <field name="public" type="int" indexed="true" stored="true"/>
   <field name="ncites" type="int" indexed="true" stored="true"/>
   <field name="versionName" type="string" indexed="true" stored="
      true"/>
   <field name="crawlDate" type="date" indexed="true" stored="true
      "/>
   <field name="repositoryID" type="string" indexed="true" stored
      ="true"/>
   <field name="conversionTrace" type="string" indexed="true"
      stored="true"/>
```

```
   <field name="selfCites" type="int" indexed="true" stored="true
      "/>
   <field name="versionTime" type="date" indexed="true" stored="
      true"/>

</schema>
```

Listing A.24. code/tags_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="tag"
      pk="id"
      query="SELECT id, paperid, tag, count from tags"
      >
      <field column="id" name="id"/>
      <field column="paperid" name="paperid"/>
      <field column="tag" name="tag"/>
      <field column="count" name="count"/>
     </entity>
  </document>
</dataConfig>
```

Listing A.25. code/tags_schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<schema name="papers" version="1.5">

   <field name="_version_" type="long" indexed="true" stored="true
      "/>

   <field name="id" type="long" indexed="true" stored="true"
      required="true" multiValued="false"/>
   <field name="paperid" type="string" indexed="true" stored="true
      "/>
   <field name="tag" type="string" indexed="true" stored="true"/>
   <field name="count" type="int" indexed="true" stored="true"/>

</schema>
```

Listing A.26. code/urls_data–config.xml

```
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
```

```
              url="jdbc:mysql://csxdb:3306/citeseerx"
              user="root"
              password=""/>
  <document>
    <entity name="url"
      pk="id"
      query="SELECT id, url, paperid from urls"
      >
      <field column="id" name="id"/>
      <field column="url" name="url"/>
      <field column="paperid" name="paperid"/>
     </entity>
  </document>
</dataConfig>
```

Listing A.27. code/urls_schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>


<schema name="papers" version="1.5">

   <field name="_version_" type="long" indexed="true" stored="true
      "/>

   <field name="id" type="long" indexed="true" stored="true"
      required="true" multiValued="false"/>
   <field name="url" type="string" indexed="true" stored="true"/>
   <field name="paperid" type="string" indexed="true" stored="true
      "/>

</schema>
```

Bibliography

[1] Google, "Google Scholar," .
URL http://scholar.google.com

[2] Microsoft, "Microsoft Academic Search," .
URL http://academic.microsoft.com

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

[4] MySQL, "MySQL," .
URL https://www.mysql.com/

[5] Wu, J., P. B. Teregowda, M. Khabsa, E. Treece, D. Jordan, S. Carman, P. Mitra, and C. L. Giles (2013) "Scalability bottlenecks of the citeseerx digial library search engine," in *LSDS-IR 2013*.

[6] Wu, J., J. Killian, H. Yang, K. Williams, S. R. Choudhury, S. Tuarob, C. Caragea, and C. L. Giles (2015) "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, Palisades, NY, USA, October 7-10, 2015* (K. Barker and J. M. Gómez-Pérez, eds.), ACM, pp. 13:1–13:8.
URL http://doi.acm.org/10.1145/2815833.2815834

[7] Khabsa, M. (2015) *Towards Better Accessibility of Scholarly Data*, Ph.D. thesis, Penn State University.

[8] Hat, R., "Global File System," .
URL http://www.redhat.com/whitepapers/rha/gfs/GFS_INS0032US.pdf

[9] Teregowda, P. (2012) *Computational Issues in Digital Library Search Engines*, Ph.D. thesis, Penn State University.

[10] Memcached, "Memcached," .
URL http://memcached.org/

[11] Software, V., "Varnish Cache," .
URL https://www.varnish-cache.org/

[12] DeCandia, G., D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels (2007) "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS Operating Systems Review*, vol. 41, ACM, pp. 205–220.

[13] Brewer, E. (2012) "Pushing the CAP: Strategies for Consistency and Availability," *Computer*, **45**(2), pp. 23–29.
URL http://dx.doi.org/10.1109/MC.2012.37

[14] Solr, "Solr," .
URL http://lucene.apache.org/solr/

[15] ZooKeeper, "ZooKeeper," .
URL https://zookeeper.apache.org/

[16] Elastic, "ElasticSearch," .
URL https://www.elastic.co/products/elasticsearch

[17] Kingsbury, K. (2014), "Call Me Maybe Elasticsearch," https://aphyr.com/posts/317-jepsen-elasticsearch.

[18] MongoDB, "MongoDB," .
URL https://www.mongodb.org/

[19] Redis, "Redis," .
URL http://redis.io/

[20] Services, A. W., "DynamoDB," .
URL https://aws.amazon.com/dynamodb/

[21] Amazon, "Amazon Web Services," .
URL https://aws.amazon.com/

[22] Wu, J., K. Williams, H.-H. Chen, M. Khabsa, C. Caragea, A. Ororbia, D. Jordan, and C. L. Giles (2014) "CiteSeerX: AI in a Digital Library Search Engine." in *AAAI*, pp. 2930–2937.

[23] Chang, F., J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber (2008) "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, **26**(2), pp. 4:1–4:26.
URL http://doi.acm.org/10.1145/1365815.1365816

[24] Microsoft, "DocumentDB," .
URL https://azure.microsoft.com/en-us/documentation/services/documentdb/