CiteSeerX Manual (CXM)

Jian Wu, Douglas Jordan, Kyle Williams, Pradeep Teregowda,
Stephen Carman, and Po-Yu Chuang

August 19, 2015

Contents

Preface

The purpose of this manual is to provide an instruction to CiteSeerX users on how to install and run CiteSeerX.

The manual has four main sections. In the first section, we give an overview of the CiteSeerX project and the whole system. In the second section, we provides the step-by-step instruction to install the software, including a single workstation installation and a cluster installation. The third section provides instructions on how to run the citeseerx, and maintain it. The fourth section provides solutions to trouble shoot we have already experienced or expected.

The information on system maintenance in this manual is basically a compilation of years of experiences from the CiteSeerX project leader – Dr. C. Lee Giles and technical directors, from Issac G.Councill to Pradeep B. Teregowda.

All CiteSeerX group members should be given credits to this documentation, including the founders: Steve Lawrence, Lee Giles and Kurt Bollacker; the early contributors: Bernie Nan Zang, Min-Yen Kan, Ying Liu, and Isaac G. Councill. the second generation operators Pradeep Teregowda, Juan Pablo Fernández Ramírez, Shuyi Zheng, Puck Treeratpituk; the third generation operators Madian Khabsa, Jian Wu, Douglas Jordan, Stephen Carman, Kyle Williams etc. Professor Prasenjit Mitra also gives us constructional comments and suggestions when building the CiteSeerX. I really appreciate their contribution to this document.

List of people who have been working for the CiteSeerX and their main contributions.

Table 1. List of people who have or are working for the CiteSeerX.

Name	Period	Contribution
C. Lee Giles	since 1997	One of the founders of CiteSeer. He continues to lead the CiteSeer into the new generation
Steve Lawrence	-	The creator of the original CiteSeer architecture and system.
Kurt Bollacker	-	Responsible for many innovative features in CiteSeer.
Pradeep B. Teregowda	since 2002	Leader of technical aspects of the CiteSeerX development
Prasenjit Mitra	-	Leader of the CiteSeerX table search feature.
Jian Wu	since 2011	Improving the CiteSeerX web crawler
Puck Treeratpituk	2007–2012	Leader of the author disambiguation project
Juan Pablo Fernández Ramírez	2006–2011	Software engineering
Shuyi Zheng	2005-2010	Populating the CiteSeerX document collection by web crawling
Isaac G. Councill	2003-2010	Principle engineer of technical aspects of the CiteSeerX development, designing and coding.
Ying Liu	2003-2010	Developed the table search system
Min-Yen Kan	2003-2010	Contributed the random field model and build the ParsCit application with Isaac.
Douglas W. Jordan	2003-2010	Transferring CiteSeerX data
Stephen Carman	since 2012	Transferring CiteSeerX data
Bernie Nan Zang	-	Responsible for some interfaces of CiteSeerX.
Po-Yu Chuang	since 2014	Upgrade Solr from v1.3 to v4.9

Chapter 1

Overview

Great ideas don't just happen. They evolve.

The history of the CiteSeerX project can be traced back to 1997 when its progenitor – CiteSeer was developed at the NEC Research Institute, Princeton, NJ, by Steve Lawrence, Lee Giles and Kurt Boolacker. The service transitioned to the College of Information Sciences and Technology in 2003 in the Pennsylvania State University. Since then, the project has been directed by Lee Giles. The technical and administrative aspects of this project have been directed by Isaac Councill and then Pradeep Teregowda.

CiteSeer was the first digital library and search engine to provide automated citation indexing and citation linking using the method of autonomous citation indexing. After serving as a public search engine for nearly ten years, CiteSeer, originally intended as a prototype only, began to scale beyond the capabilities of its original architecture. Since its inception, the original CiteSeer grew to index over 750,000 documents and served over 1.5 million requests daily, pushing the limits of the system capability. Based on an analysis of problems encountered by the original system and the needs of the research community, a new architecture and data model was developed for the CiteSeerX which is the "Next Generation CiteSeerX" in order to continue the CiteSeer legancy into the foreseeable future.

The CiteSeerX is an evolving scientific literature digital library and research engine that focuses primarily on the literature in computer and information sciences. CiteSeerX aims to improve the dissemination of scientific literature and to provide improvements in functionality, usability, availability, cost, comprehensiveness, efficiency, and timeliness in the access of scientific and scholarly knowledge. The CiteSeerX is also planning to expand its service to other document topics such as economy and medical sciences.

Rather than just creating a digital library, CiteSeerX attempts to provide resources such as algorithms, data, medadata, services, techniques, and software that can be used to promote other digital libraries. CiteSeerX has developed new methods and algorithms to index postscript and PDF reserach articles on the web.

You can access the CiteSeerX search engine from

`http://citeseerx.ist.psu.edu`

You can submit your document URL from

`http://csxcrawlweb01.ist.psu.edu/submit_pub/`

CiteSeerX provides the following features.

- **Autonomous citation Indexing (ACI)** – CiteSeerX uses ACI to automatically create a citation index that can be used for literature search and evaluation. Compared to traditional citation indexes, ACI provides improvements in cost, availability, comprehensiveness, efficiency, and timeliness.
- **Citation statistics** – CiteSeerX computes citation statistics and related documents for all articles cited in the database, not just the indexed articles.
- **Reference linking** – As with many online publishers, CiteSeerX allows browsing the database using citation links. However, CiteSeerX performs this automatically.
- **Citation context** – CiteSeerX can show the context of citations to a given paper, allowing a researcher to quickly and easily see what other researchers have to say about an articles of interest.
- **Awareness and tracking** – CiteSeerX provides automatic notification of new citations to give papers, and new papers matching a user profile.
- **Related documents** – CiteSeerX locates related documents using citation and word based measures and displays an active and continuously updated bibliography for each document.
- **Full-text indexing** – CiteSeerX indexes the full-text of the entire articles and citations. Full Boolean, phrase and proximity search is supported.

- **Query-sensitive summaries** – CiteSeerX provides the context of how query terms are used in articles instead of a generic summary, improving the efficiency of search.
- **Up-to-date** – CiteSeerX is regularly updated based on the user submission and regular crawls.
- **Powerful search** – CiteSeerX uses fielded search to all complex queries over content, and allows the use of author initials to provide more flexible name search.
- **Harvesting of articles** – CiteSeerX automatically harvests research papers from the web using a dedicated focused crawler *citeseerxbot*. This crawl strictly obeys the exclusion rule specified in `robots.txt`.
- **Metadata of articles** – CiteSeerX automatically extracts and provides metadata from all indexed articles.
- **Personal Content Portal** – CiteSeerX features a personal content portal – *MyCiteSeerX*, which features personal collections, RSS-like notification, social bookmarking, social network facilities, personalized search settings, institutional data tracking possible and transparent document submission system.

CiteSeerX data and metadata are available for others to use. These data include CiteSeerX metadata, databases, datasets of PDF files and text of PDF files. Currently, data is only available through rsync transfers and by downloads from Amazon S3. CiteSeerX is compliant with the Open Archive Initiative Protocol for Metadata Harvesting, which is a standard proposed by the Open Archive Initiative (OAI). The link to browse or download records programmatically from the CiteSeerX OAI collection is

`http://citeseerx.ist.psu.edu/oai2`

The archive may also be browsed from an interface via an OAI Repository Explore[1], either by using the CiteSeerX archive identifier or by directly entering the harvest URL. A list of toolkits that can be used for OAI metadata harvesting includes OAI-Harvester[2] (perl), OAIHarvester2[3] (Java), .NET OAI Harvester[4] (.NET dll), UIUC OAI[5].

[1] `http://re.cs.uct.ac.za/`
[2] `http://search.cpan.org/~thb/OAI-Harvester-1.13/`
[3] `http://www.oclc.org/research/software/oai/harvester2.htm`
[4] `http://sourceforge.net/projects/netoaihvster`
[5] `http://sourceforge.net/projects/uilib-oai/`

Chapter 2

Architecture

CiteSeerX is compised of a focused web crawler, a document extracter, an ingester, an indexer, and a search interface. The crawler stores the harvested contents into its own database and repository; through the ingestion process, text information is first extracted from the crawled documents and is parsed; useful documents and the associated metadata are saved to the search engine database and repository; these documents and metadata are all indexed and ready for search. The web server accepts online web requests and returns results in an organized manner.The big architecture describe above is dipicted in Figure 2.1.

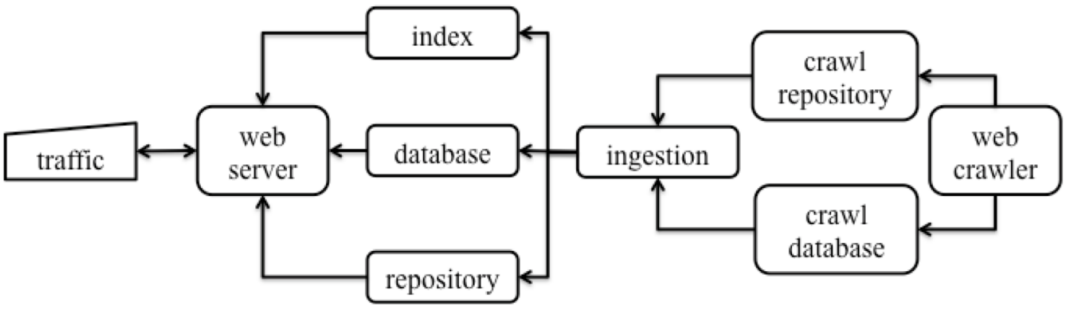

Figure 2.1 The big architecture of the CiteSeerX system.

Each process in Figure 2.1 can be broken down to a number of subprocesses and implemented by multiple servers. The current CiteSeerX production cluster includes more than fifteen machines.

The basic jobs for each server and their connections are illustrated in Figure 2.2. The focused web crawler downloads PDF and postscript files from the internet. The downloaded files are saved to the crawler repository and the associated metadata are written into the crawl database. We have a crawler web server from which we can monitor the crawling and ingestion progress. The crawler also has an API which can be used to select a number of un-ingested documents from the crawler database and writes it in XML format. This XML file is used by the text extractor to transfer these files from the crawler repository and do text extraction. Files whose text can be extracted are passed to the ingester, which saves the parsed text files and the original PDF files to the CiteSeerX repository. The ingester also writes an XML file, which is usually referred to as the CiteSeerX "metadata" file and saves it to the CiteSeerX repository. The title, authors, venues, publication information etc. are written into the CiteSeerX master database. There is a slave database which replicates the master. CiteSeerX performs a full text indexing to all the text information from the database and the repository. The online traffic accesses the CiteSeerX website which is associated with a virtual IP. The traffic first passes a Linux director (or load balancer) and is re-directed to one of the web servers. The CiteSeerX repository is mounted to the web servers via global network block device (GNBD). Searching results are returned by the index server, documents can be downloaded from the repository and metadata can be requested from the database. Note that the user is also allowed to make corrections to the metadata information which is written into the CiteSeerX repository. In the following context, when we talk about CiteSeerX, it means the search engine installed on the web servers. This is our main code written in Java. When we talk about the CiteSeerX system, we mean the whole system depicted in Figure 2.2 In the following subsections, we break down the architecture shown in Figure 2.2 and describe each major component in detail.

2.1 The Crawler

In the CiteSeer era, Heritrix was the main crawler. But that time, Heritrix was a single thread crawler, so Dr. Shuyi Zheng developed a new crawler in Python (hereafter the SYZ crawler). The Heritrix 1.14+ and 3.0+ supports multi-thread crawling, so the current CiteSeerX uses both crawlers to harvest academic documents for the CiteSeerX search engine.

We use Heritrix to perform our incremental crawl. Most of the seed URLs are adopted form the whitelist but we also accept user submitted URLs from the crawler website. Basically, the crawler scheduler selects all the seed URLs from the *whitelist2* table in the *citeseerx_crawl* database, and submits them to crawler. Each

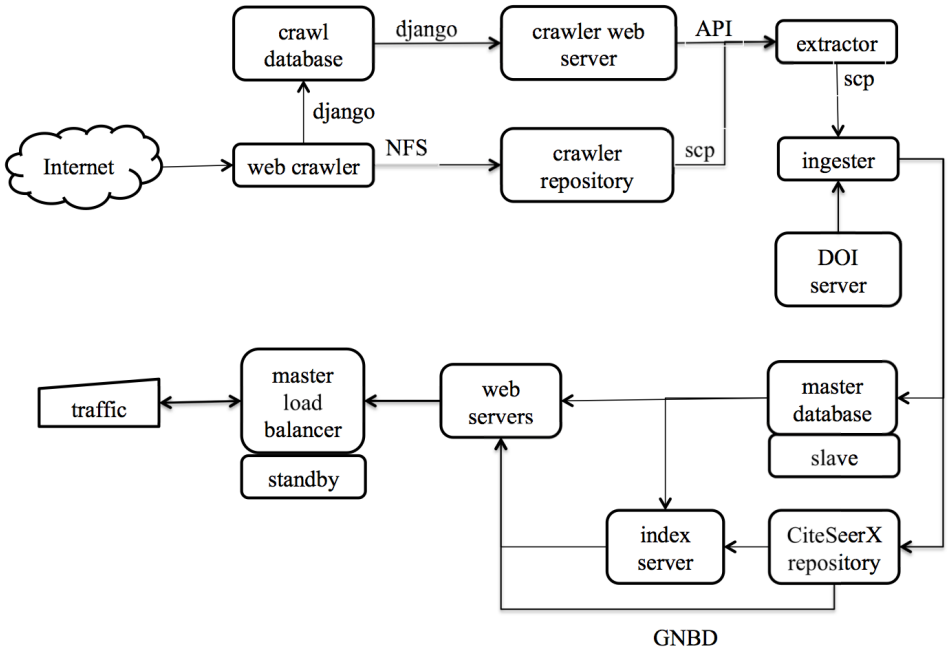

Figure 2.2 The architecture of the CiteSeerX system and their main jobs. Note that the system has two load balancers and two web servers to form a high availability cluster. It also has a slave database server which replicate the master database.

URL is crawled within a depth of two in a breath-first policy. We strictly obey the `robots.txt` exclusion rule for each site. The page parser then parses all the URLs in a web page. As one of crawling policies, we require the crawling be performed within the host domain of its parent URL. For example, if the host domain of a seed URL is `cmu.edu`, all subsequential URLs fetched from this seed URL are restricted to be within `cmu.edu`. Only files with certain content types are downloaded. Currently, only `application/pdf` is accepted. There is not limit on the size of each document. If a URL contains links to more than one acceptable document, it is regarded as a "good parent URL and saved to the parent url table in our database for future crawl. The document URL is saved to the `main_crawl_document` table. The documents themselves and their associated metadata files are saved to the crawl repository.

We use Heritrix (version 3.1+ and version 1.14+) to perform bulk crawls, in which the seed URLs are specified as an input file. The user should read Heritrix manual to configure the crawler. The documents harvested by Heritrix can be imported to the crawl database and repository through a middleware called crawl document importer (CDI). This middleware is designed to provide a universal interface for multiple crawlers and input file mime types to minimize the extra coding when a new crawler is used. The rule of this middleware in the crawler architecture is illustrated in Figure 2.3. This importer checks all successful downloads from a Heritrix crawl and performs a content based filtering. Optionally, the user can choose to check whether this document is an academic paper or not. It then saves the URL information to the crawl database. The documents themselves, the associated metadata files and the extracted text files are saved to the crawl repository.

The connections between the crawler and the other servers are illustrated in Figure 2.2. The web crawler gets seeds from the crawl database and downloads documents from the internet, it writes the crawled documents to the crawler repository, which is mounted to the crawler server as NFS. We also have a crawler web server, which is used to present the crawling history and progress. It is also a port from where the users can submit URLs of documents which they want to index to our search engine. An API is running on the crawler web server, which provides an interface to the extraction server. This API selects uningested documents from the crawler database and generates an XML file. This XML file contains the relative directories of these documents, so the text extractor can retrieve these documents. The metadata is written into a `.pdf.met` file and saved to the same directory of the original document. The `.pdf.met` contains the following attributes: crawl date (`crawlDate`), last modified date (`lastModified`), URL (`url`), parent URL (`parentURL`), and encription (`SHA1`). The crawl date and the URL information will be recorded in the *citeseerx.urls* database table.

The repository is a directory to save filtered crawled documents. Subfolders

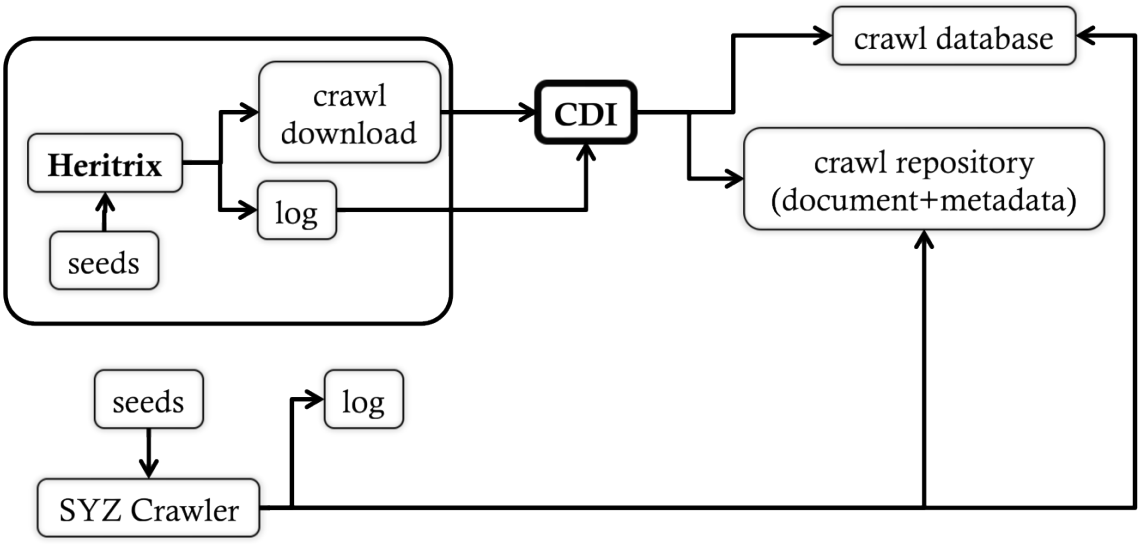

Figure 2.3 The role of the CDI middleware with the crawler, the repository and the database. The CDI plays a "bridge" between the crawler and the crawler repository/database. In this figure, Heritrix can be replaced with another crawler or FTP downloads. The SYZ crawler was designed specifcally for the CiteSeerX project, so it does not require the CDI middleware.

are automatically generated in a hierarchical order. Each subfolder is named using three digital numbers corresponding to a piece of document ID. For example, a PDF document with ID 1234567 is saved to `001/234/567/001.234.567.pdf`. The associated metadata are named as `001.234.567.pdf.met` and `001.234.567.txt`, respectively under the same folder. If an updated version of a document is identified from an existing URL, the XML document in the repository is updated.

The crawler database is named *citeseerx_crawl* and is run on a separate server. There are four main tables in this database (the others are generated and used by Django):

- *main_crawl_document*. This table stores the entire collection of crawled documents.
- *main_crawl_parenturl*. This table stores the collection of parent URLs.
- *main_crawl_submission*. This table stores the user submitted URLs.
- *main_crawl_whitelist*. This table stores the whitelist which contains the high quality seed URLs. There could be multiple versions of the whitelist.

The schemas of these four tables are tabulated in Tables 2.1, 2.2, 2.3 and 2.4. The encryption algorithms MD5 and SHA1 are used for duplicate detection. When a

Table 2.1. The `main_crawl_document` table schema.

Field	Type	Description
`id`	int(11)	Document ID
`url`	varchar(255)	Document URL
`md5`	varchar(32)	MD5 value of document URL
`host`	varchar(255)	host name of document URL
`content_sha1`	varchar(40)	SHA1 value of crawled document
`discover_date`	datetime	First time to crawl this document
`update_date`	datetime	Last time to crawl this document
`parent_id`	int(11)	parent URL ID
`state`	int(11)	Ingested or not

document is saved , its parent URL MD5 value is calculated. The document itself is downloaded to the cache and its SHA1 value is also calculated. If this document URL is new (its MD5 is new), a new database record is created. Otherwise, the SHA1 and the update date are both refreshed. If a document is saved into the crawl database, its parent URL must be saved or the last crawl datetime of the existing parent URL is updated. Similar to the document table, the parent URL MD5 value is also calculated to ensure that there is no URL duplication. The state field in the *mai_crawl_document* table indicates whether a document is ingested or not. This field is reset to the default value if an updated document version is identified. Ingestion is a post-crawl process to examine, parse, classify, and index the documents and make them searchable. There are four possible states for a given document: retrieved (2), ingested (1), failed(-1), or not ingested (0). As long as the document is selected by the API, it state is set to 2. If the extractor fails to extract text from the document or the document is not classified as academic related, its state is set to -1. If the document is written in to the *citeseerx.papers* and *citeseerx.urls* tables, its state is set to 1.

2.2 Load Balancers

When an end user attempts to visit this address, the http request is directed by a linux director (or load balancer, here after LB). The purpose of this LB is reduce workload to a single web server by distribute the traffic volume among multiple web servers. In order to build a high availability (HA) website, we setup two LBs. The

Table 2.2. The `main_crawl_parenturl` table schema.

Field	Type	Description
`id`	int(11)	parent URL ID
`url`	varchar(255)	parent URL
`md5`	varchar(32)	MD5 value of parent URL
`first_crawl_date`	datetime	Time of the first visit
`last_crawl_date`	datetime	Time of the last visit

Table 2.3. The `main_crawl_submission` table schema.

Field	Type	Description
`id`	int(11)	Submission ID
`url`	varchar(255)	Submitted URL
`email`	varchar(255l)	Submiter's email
`add_time`	datetime	Submission timestamp

Table 2.4. The `main_crawl_whitelist` table schema.

Field	Type	Description
`id`	int(11)	Whitelist URL ID
`ndocs`	varchar(255)	Number of ingested documents
`hosts`	varchar(255l)	URL host, e.g., `www.psu.edu`
`url`	varchar(384)	Whitelist URL

master LB takes all the traffic, but once it fails, the standby LB will There is a master LB and a standby LB.

The load balancing is implemented by *Ldirectord* and *heartbeat* as developed in the *Ultra Monkey* project[1]. Ultra Monkey is a project to create load balanced and highly available network services. For example a cluster of web servers that appear as a single web server to end-users. The service may be for end-users across the world connected via the internet, or for enterprise users connected via an intranet.

Ldirectord runs on linux-directors to monitor the health of the real-servers by periodically making a request and checking for an expected response. For HTTP servers this means requesting a known URL and checking that the response contains an expected string. This is called a "negotiate" check. If a real-servers fails then the server is made quiescent and will be reinserted once it comes back on line. If all the real-servers are down then a fall-back server is inserted into the pool, which will be made quiescent one of the real web servers comes back on line. Typically, the fall-back server is localhost. If an HTTP virtual service is being provided, it is useful to run an Apache HTTP server that returns a page indicating that the service is temporarily inaccessible.

Heartbeat is the core component of the Linux-HA project, which is aimed at providng flexible high availability network. Heartbeat implements a heartbeat-protocl, in which messages are sent at regular intervals between machines. If aa message is not received from a particular machine, the machine is regarded as failed. When heartbeat is configured, a master node s selected. When heartbeat starts up, this node sets up an interface for a virtual IP address, which will be accessed by external end users. If this node fails, the other node in the heartbeat cluster starts up an interface for this IP address and use gratuitous ARP to ensure that all traffic bound for this address is received by this machine. This method of fail-over is called "IP Address Takeover". Once the master node becomes available again, resources will failover again and owned by the master node. At a certain time, one of these two servers is holding the IP address. Ultra Monkey makes use of heartbeat on Linux directors to manage IP addresses and monitor real servers using the IPaddr2 resource and ldirectord resource, respectively.

The configuration of the CiteSeerX HA cluster is illustrated in Figure 2.4. The front page corresponds to the virtual IP. LB1 is the master load balancer and LB2 is the standby. The traffic is directed to either RS1 or RS2 depending on the scheduling algorithm. All of the IPs are accessible. The apache http servers are installed and deployed on load balancers, which basically gives fallback pages which are displayed when both of the real servers are down. The CiteSeerX are installed on each real servers, which are deployed using Tomcat. Note that Apache HTTP

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

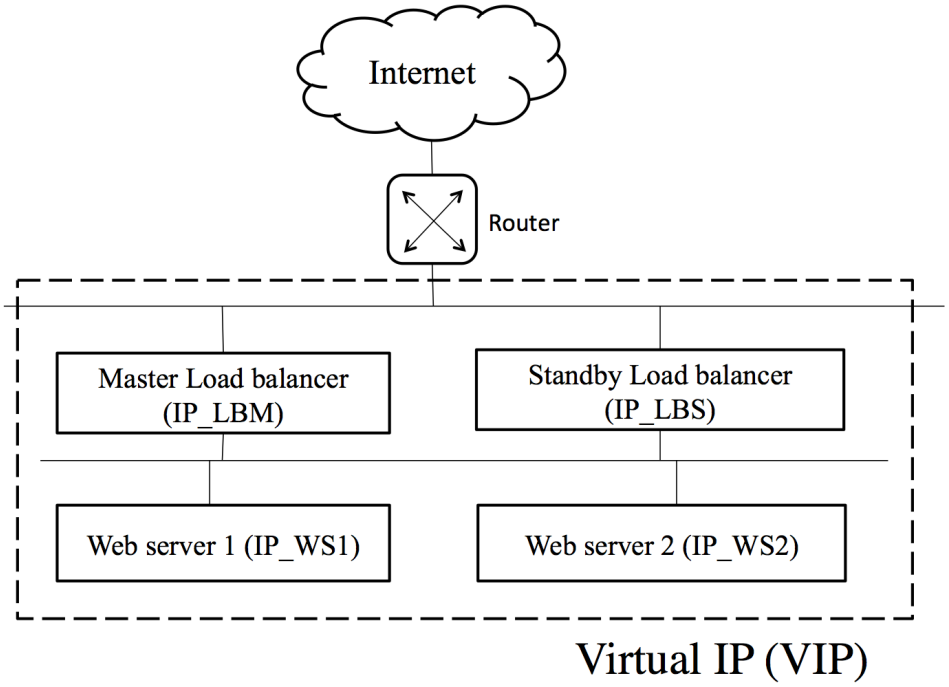

Figure 2.4 The configuration of the high availability load balancers.

uses port 80 and Tomcat uses prot 8080 by default.

The firewall is implemented using *iptables*. The iptables configuration on should allow external access to port 80. Another import configuration is to set the firewall mark using

```
$ iptables -t mangle -A PREROUTING -d [VIP] -p tcp -j MARK --set-mark 0x1
```

in the `mangle` section. Incoming packets are marked with a 32-bit unsigned value using iptables. The Linux Virtual Service (LVS) is able to match use this mark to designate packets destined for a virtual service and route them accordingly. This is particularly useful if a large number of contiguous IP based virtual services are required with the same real servers. Or to group persisitency between different ports. For instance, to ensure that a given end user is sent to the same real server for both HTTP and HTTPS. The specific configuration varies, but should follow the instructions described on the the Ultra Monkey page.

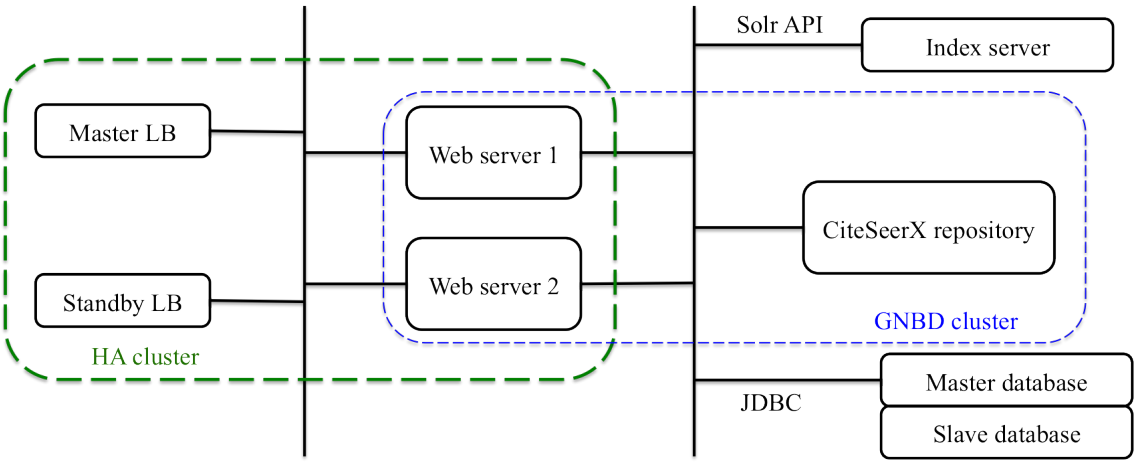

Figure 2.5 The connections between the web servers, the load balancers, the repository server, the index server and the database servers.

`http://www.ultramonkey.org/3/topologies/ha-lb-eg-fwmark.html`

This section is intended to give an overview of the architecture. For configuration details, please refer to the Installation section.

2.3 Web Servers

The connections between the web servers, the load balancers, the repository server, the index server and the database servers are dipicted in Figure 2.5.

The two web servers (WBs) and the two LBs form a HA cluster. In case one of the webservers is down, the other will take over all the traffic. Because CiteSeerX is deployed using Tomcat, the ports opened for the web servers are 8080. The port 80 is also opened so that the server can be directly accessed without specifying the port. Requests to the 80 ports are forwarded to 8080. This is called *port forwarding* and is specified by the following statement in the "nat" chain of iptables:

```
$ iptables -t nat -A PREROUTING -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 8080
```

The two WBs form a GNBD cluster with the repository server. The repository directory is mounted to each of the web servers via a GNBD device. Documents can be directly retrieved from the mounted directory just as if they are saved in local disk. The index is retrieved from the index server via a Solr API. The metadata is retrieved from the database via JDBC.

The CiteSeerX search engine is deployed by Tomcat. We have tested that it works for Tomcat 6 and Tomcat 7. We will cover the installation procedures in the next chapter, but the basic idea is to build a `citeseerx.war` file and copy it under the `tomcat/webapps/` directory along with the compiled code. You can deploy multiple Tomcat instances simultaneously. For example, you can deploy CiteSeerX and Solr index service under one Tomcat. This is how the staging machine is implemented.

2.4 CiteSeerX Repository

By default, when we use "repository" only, it means the CiteSeerX repository, which contained the ingested PDF/postscript documents and the associated metadata files. The crawler has a separated repository called "crawl repository", which contains all the PDF/postscript harvested. The production repository is located in a separate server. Because the repository is very large, and data are not just appended (user corrections may occur for any existing document), it is difficult to "replicate" the repository. Thus, multiple backups are necessary in case the production repository server fails. We keep at least two copied of the repository.

The repository server is mounted on several machines, forming a cluster. This "shared storage" model is implemented by global network block device (GNBD). In order to use this device, we must format the drives to the global file system (GFS) or GFS2. The advantage of GNBD is that it supports fencing, which is the process of isolating a node of a computer cluster or protecting shared resources when a node appears to be malfunctioning. The failed node may have control over shared resources that need to be reclaimed and if the node is acting erratically, the rest of the system needs to be protected. Fencing may thus either disable the node, or disallow shared storage access, thus ensuring data integrity. The disadvantage of GNBD is that it is not supported any more. Particularly, it is not supported by RHEL 6 (or CentOS 6), so we must seek other method for the shared storage model.

The repository server and other servers that connect to it are shown in Figure 2.6. Based on our current repository model, the ingester must be on the same server as the repository, or the repository server must be mounted to the ingest server (we use the former). On one hand, the ingester retrieves a copy of the text files extracted from PDF from the text extracter and builds an XML file. This XML, the original PDF file along with the extracted text files are imported to the CiteSeerX repository. On the other hand, in order to build the index, the ingester retrieves the text and XML files from the repository. It then combines the metadata information obtained from the CiteSeerX database and send them to the index server. There are more than one index servers to hold different indexes, which will

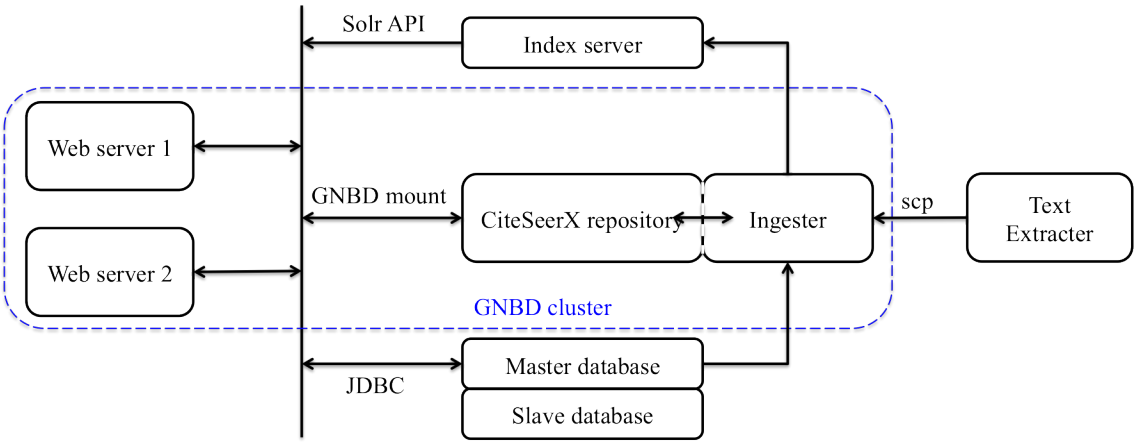

Figure 2.6 The CiteSeerX repository server and other servers that connect to it. The dashed line between the ingester and the repositoy means that they are on the same server.

be covered in Section 2.6. The repository drive is formated in GFS and is mounted to both of the web servers as a GNBD device. The web servers can process the download request by retrieving documents from the repository server. The users can also correct the papers' metadata and body and save them to the database (metadata) and repository (text body).

The repository directary can be in any directory on the server, but it must have the following hiarachical structure (Figure 2.7). The files inside the deepest repository path are named corresponding to the numerical order in the path, e.g., 10.1.1.1.1483.*[extension]*. There are usualy six types of files inside each deepest repository path (see Table 2.5). In some cases, there could be multiple versions of files with an `.xml` extension.

The files in the repository correpond to the *citeseerx.paper* table in the CiteSeerX database. By June 13, 2012, the repository contains about 2.4 million academic documents which takes about 4.8 tera-bytes of space. We recommend to backup the repository every other week.

2.5 CiteSeerX Database

Like the repository, the "database" means the CiteSeerX database while the crawler keeps a separate database called the "crawl database". The main databases are *citeseerx* and *myciteseex*. There are another two databases named *csx_citegraph* and *csx_external_metadata*. Besides, the *mysql* database contains the credential

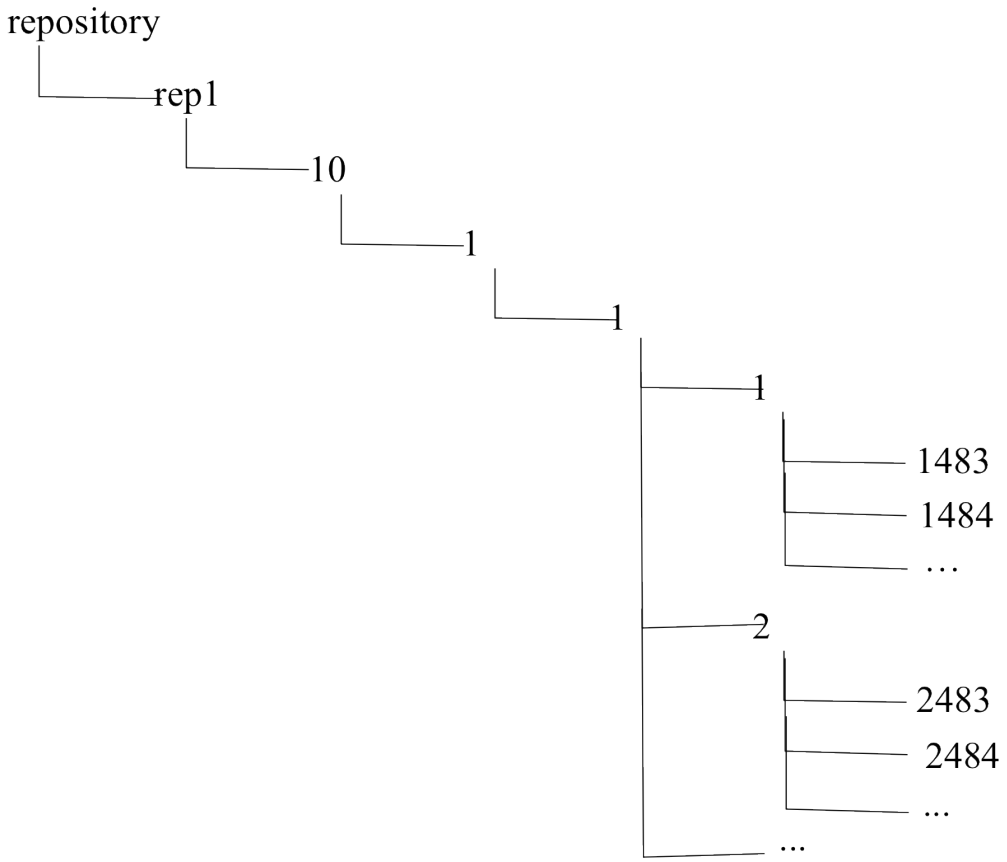

Figure 2.7 The hierachical structure of the CiteSeerX repository. The deepest directory in this diagram, e.g., 1483, contains document files, such as 10.1.1.1.1483.xml.

Table 2.5. File types in the repository.

Extension	Mimetype[1]	Description
pdf	application/pdf	Original PDF file
ps	application/postscript	Original postscript file
body	text/plain	Body section of document text (no citation)
txt	text/plain	Text extracted from the PDF file
xml	text/plain	Labeled text in XML format
v1.xml	text/plain	The first modified version of text in XML format[2]
tetml	text/plain	Labeled text in TET Markup Language[3]
cite	text/plain	Citation section of document text

[1]Mime type results returned by the `file --mime` command.

[2]There could be multiple versions such as v2.xml co-existing with the orignial .xml file.

[3]TET is the software used for text extraction from PDF files.

information, which should be confidential to the public, i.e., only the production and its replication can have this database. The staging machine should have a different set of user credentials. Only database on the staging machine is open to authorized users from certain machines. The *citeseerx* database contains 28 tables (see Table 2.6) and the dumped `.sql` file of the CiteSeerX databases is about 50GB (in December, 2012). The other databases are much smaller. After importing the database on the database server, it takes about 150GB disk space. Therefore, a database server should have at least 200GB.

The connections between the master database server and the other servers are shown in Figure 2.7. The slave database server performs a replication to the master database to keep them synchronized. The metadata information displayed in the paper profile page is retrieved from the database. If the user makes a correction, these corrections will be written into the database after they are submitted. Everytime a paper is ingested, the metadata of this paper is writen into master database. The indexing process also needs to read the metadata from the database and send it over to the index server.

The most frequently used tables in the *citeseerx* database are *urls*, *papers*, *citations*, *authors* and *paper_listing*. The fields and their meanings are tabulated in Tables 2.7, 2.8, 2.9, 2.10 and 2.11, respectively. The *myciteseerx* database contains 13 tables. The most frequently used table is the *users* (Table 2.12).

Table 2.6. List of tables in the *citeseerx* database.

Table	Rows[1]	Description
acknowledgmentContexts	0	
acknowledgments	0	
acknowledgments_versionShadow	0	
authors	6,052,516	Author information
authors_versionShadow	6,052,496	
cannames	308,116	
checksum	2,499,476	
citationContexts	39,906,123	
citations	41,505,622	Citation information
citecharts	264,357	
eTables	880,849	
elinks	435,417	
hubMap	2,596,443	
hubUrls	385,458	
keywords	2,109,405	Paper keywords
keywords_versionShadow	2,109,405	
legacyIDMap	734,756	
link_types	2	
paperVersions	2,032,628	
paper_listing	264,721[2]	Paper removal/re-list request and actions
papers	2,132,204	Paper information
papers_versionShadow	2,132,150	
pdfTables	880,849	
redirectpdf	132,34	
redirecttemplates	1	
tags	2,932	
textSources	1	
urls	2,538,157	Document URLs where papers were downloaded[3]
userCorrections	781,455	

[1]Number of rows are update to June 13, 2012.

[2]This table was added in September, 2012. The number reflects the records in 2012-12-21.

[3]From the crawler metadata file.

Table 2.7. The schema of the *urls* table in the *citeseerx* database.

Field	Type	Description
`id`	`bigint(20) unsigned`	URL ID, auto increment
`url`	`varchar(255)`	Document URL where it was crawled from
`paperid`	`varchar(100)`	DOI, corresponding to the `id` in the `paper` table

Table 2.8. The schema of the *papers* table in the *citeseerx* database.

Field	Type	Description
`id`	`varchar(100)`	DOI, corresponding to the `paperid` in the `url` table
`version`	`int(10) unsigned`	Document version
`cluster`	`bigint(20) unsigned`	Cluster number[1]
`title`	`varchar(255)`	Paper title
`abstract`	`text`	Paper abstract
`venue`	`varchar(255)`	Paper venue, e.g., JCDL
`venueType`	`varchar(20)`	Venue type, e.g., conference, journal
`pages`	`varchar(20)`	Page range, e.g., 163–176
`volume`	`int(11)`	Volume number
`number`	`int(11)`	Number
`publisher`	`varchar(100)`	Publisher name, e.g., Springer
`pubAddress`	`varchar(100)`	Publisher address
`tech`	`varchar(100)`	Technical report information
`public`	`tinyint(4)`	Paper is searchable (1) or not (0)
`ncites`	`int(10) unsigned`	Number of citation/references
`versionName`	`varchar(20)`	Usually equals to `USER` or `INFERENCE`
`crawlDate`	`timestamp`	Time when the document URL was crawled
`repositoryID`	`varchar(15)`	Always equals to `rep1`
`conversionTrace`	`varchar(255)`	How text were extracted, e.g., PDFLib TET
`selfCites`	`int(10) unsigned`	How many self citations the paper contains
`versionTime`	`timestamp`	When this paper/version was ignested

[1]Papers are clustered based on their similarity. If they are identified to contain the same content, they are put into the same cluster.

Table 2.9. The schema of the *citations* table in the *citeseerx* database.

Field	Type	Description
id	bigint(20) unsigned	Citation ID, auto increment
cluster	bigint(20) unsigned	Cluster number[1]
authors	text	Author list in citations
title	varchar(255)	Citation title
venue	varchar(255)	Citation venue, e.g., JCDL
venueType	varchar(20)	Venue type, e.g., conference, journal
year	int(11)	Year of publication
pages	varchar(20)	Page range, e.g., 163–176
editors	text	Editor list
publisher	varchar(100)	Publisher name, e.g., Springer
pubAddress	varchar(100)	Publisher address
volume	int(11)	Volume number
number	int(11)	Number
tech	varchar(100)	Technical report information
raw	text	The whole raw citation text
crawlDate	timestamp	Time when the document URL was crawled
paperid	varchar(100)	DOI, corresponding to id in the papers table
self	tinyint(4)	Whether it is a self-citation (1) or not (0)

[1]Like papers, citations can also be clustered. If two or more citations are identified to be the same, they are put into the same cluster. **(need to verify this from Pradeep)**

Table 2.10. The schema of the *authors* table in the *citeseerx* database.

Field	Type	Description
id	bigint(20) unsigned	Author ID, auto increment
cluster	bigint(20) unsigned	Cluster number[1]
name	varchar(100)	Author full name
affil	varchar(255)	Author affiliation
address	varchar(255)	Author address
email	varchar(100)	Author email address
ord	int(11)	**check from Pradeep,Puck**
paperid	varchar(100)	DOI, corresponding to id in the papers table

[1]Like papers, authors can also be clustered. If two or more author names are identified to be the same, they are put into the same cluster. **(need to verify this from Pradeep)**

Table 2.11. The schema of the *paper_listing* table in the *citeseerx* database.

Field	Type	Description
id	int(11)	Paper listing/removal ID, auto increment
paperid	varchar(100)	DOI, corresponding to id in the papers table
dates	date	Date of operation, e.g., 2012-09-13
operation	enum('REMOVE','LIST')	Operations: 'REMOVE'/'LIST'
requester	varchar(100)	Name of requester, e.g., 'James Bond'
email	varchar(256)	Email of requester, e.g., 'bond@gmail.com'
reason	varchar(256)	Reason of operation, e.g., 'DMCA', 'REQUEST'
operator	varchar(100)	Person taking the operation, e.g., 'Jian Wu'

Note. — paperid and dates forms a composite key, i.e., you can have only one operation to a paper on the same date.

Table 2.12. The schema of the *users* table in the *myciteseerx* database.

Field	Type	Description
userid	varchar(100)	User registered ID, e.g., jxw394
password	varchar(255)	Encrypted user password
firstName	varchar(100)	User first name
middleName	varchar(100)	User middle name
lastName	varchar(100)	User last name
email	varchar(100)	User registered email
affil1	varchar(255)	User primary affiliation
affil2	varchar(255)	User secondary affiliation
enabled	tinyint(4)	User account is enabled (1) or not (0)
country	varchar(100)	User country
province	varchar(100)	User provine/state
webPage	varchar(255)	User personal/home web page
updated	timestamp	Last time when user information is updated
internalid	bigint(20) unsigned	Internal user ID, never used
appid	varchar(255)	Encrypted user ID **(Ask pradeep)**

2.6 Index

The index server runs on a separate machine, but it can be implemented on the same server as the web server, such as on the staging machine. We use Solr for indexing. The connections between the indexing server and the other servers are shown in Figure 2.2. The ingester combines the metadata and body text and pushes to the indexing server. The search engine on the web server communicates with the index server and searches the input text from the index.

The CiteSeerX does a full text index to its documents, including the body text, the title, the abstract etc. We keep separate indexes for author, table, figures (coming soon) and algorithm (coming soon) searches. The Solr is deployed as an instance of Tomcat, so the Solr administration page can be accessed from `http://your.index.server/solr`.

2.7 DOI

The DOI server runs on a separate machine, but it can be implemented on the same machine as the web server, such as on the staging machine. The DOI server communicates with the database server and returns an unused document ID to the ingester. The CiteSeerX uses a five section DOI for each document. For historical reasons, the first three sections are always `10.1.1`. The other two sections range from `1` to `9999`.

It should be noted that this DOI is the same as the *id* field in the *citeseerx.papers* table and the *paperid* field in the *citeseerx.urls* table but **different** from the *id* field in the *citeseerx_crawl.main_crawl_document* table. The latter is a three section ID number, e.g., `007.654.321`.

2.8 Text Extractor

The text extractor runs on a separate machine, but it can run on the same machine as the other parts, such as on the staging machine. The connections between the text extractor and the other servers are shown in Figure 2.2. The PDF and postscript files cannot be indexed directly. The goal of the text extractor is to extract the text information from these files and make them indexable.

The workflow of the text extractor is shown in Figure 2.8. The current extraction applies a batch processing model. The process starts by calling the crawler API running on the crawler web server, which gathers metadata information from

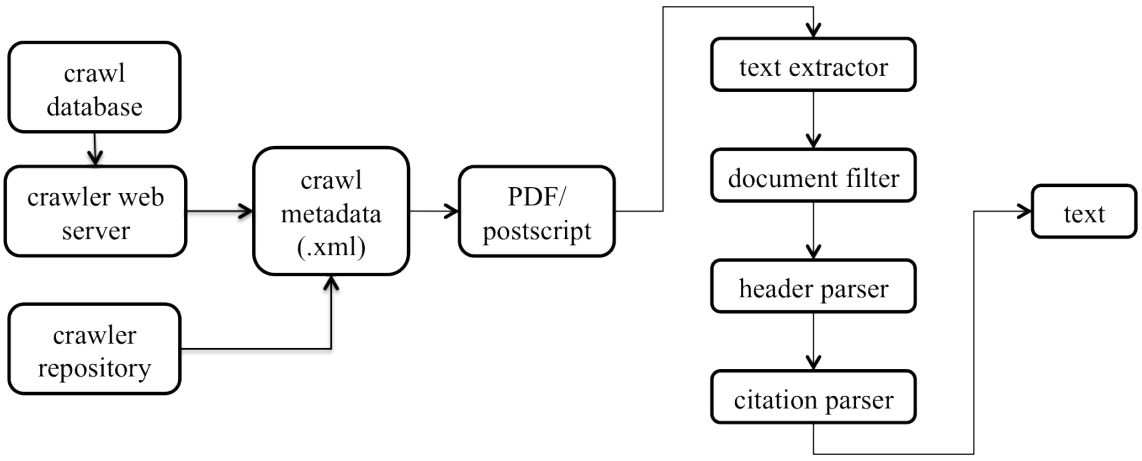

Figure 2.8 The workflow of the text extractor. The "text extractor" box in this figure represents a specific text extraction tool such as PDFBox or PDFLib TET.

the *main_crawl_document* in the *citeseerx_crawl* database, and generate an XML file, which includes path information of a certain number of documents that are not ingested (indicated by the *state* flag). Once a document metadata is written into the XML file, its state is updated and set to 2[2].

The process then retreives the documents from the crawler repository and saves them to a local directory. A text extractor is then used. CiteSeerX uses PDFlib TET PDF IFilter[3] to extract text out of the PDF files. A small portion of downloaded documents are in postscript format, which can be extracted by the *ps2text* built-in command with the Linux OS. Documents whose text cannot be extracted are dropped off. The text documents are examined by a regular expression based filter to identify whether it is an academic related paper. documents that pass this filter are then processed by multiple parses to extract metadata including titles, authors, citations and the text bodies. These contents are saved as separate files in the same folder as the PDF documents. Table 2.13 lists all the files contained in the document folder just before imported to the repository. After extraction, the `state` field in the `citeseerx_crawl.main_crawl_document` table is set to `1`.

[2]This allows multiple instances of the text extractor running so that the documents they selected do not overlap.

[3]TET is not a free software, see `http://www.pdflib.com/`.

Table 2.13 Files types in a temporary document folder before imported to the repository.

Generated_by	Suffix	Description
Web crawler	`.pdf`	Original PDF file
Web crawler	`.met`	XML file containing crawled date, original URL, parent URL and SHA1
Text extractor	`.txt`	Text extracted from the PDF file
Text extractor	`.body`	Text body only (no bibliography)
Text extractor	`.cite`	Citation (bibliography) text
Text extractor	`.file`	XML file containing text converter (e.g., PDF Lib, TET), file type (e.g., PDF) and SHA1
Text extractor	`.header`	XML file containing header parser, title, author names, addresses and the abstract
Text extractor	`.parscit`	XML file containing parsed information of the citation, output of ParsCit
Ingester	`.xml`	XML file containing parsed text

2.9 Ingester

Currently, the ingester runs on the repository server. The connections between the ingester and the other components are illustrated in Figure 2.2. The ingester does three tasks: creating XML files, importing documents to the CiteSeerX repository and creating search index.

The ingesting process first copies the document folder from the extraction server from the repository server (but not to the repository directory). It then build the XML files. This XML file contains all the parsed text information and is considered as the most valuable metadata file. This is the file that is sent to the index server. The ingester then requests a document ID from the DOI server. It then runs a batch import process, in which some associated documents are copied to the repository server. The associated metadata are also writen to the database. Files contained in the repository folders are listed in Table 2.5. However, the document is not indexed yet. The ingester finally create a full text index by sending the XML file to the indexer (**how about database fields?**).

Chapter 3

Installation

In this chapter, we describe the procedures to install CiteSeerX on a single workstation, such as the staging machine, and on a cluster. The configuration of the cluster can vary a lot. In this document, we just focus on the production cluster.

3.1 Install CiteSeerX on A Single Machine

The CiteSeerX has been tested to run on both RHEL 5.8 or RHEL 6.3 on a single machine. The difference between these two are subtle. Here, we assume the OS is RHEL 6.3.

3.1.1 Prerequisites

The hardware requirement depends on the database and repository sizes. As a guideline, in Table 3.1, we list the specifications of our staging servers.

Table 3.1 Hardware specifications of staging machines.

Item	Machine A	Machine B
CPU	Intel Xeon E5405	Intel Xeon E5649
Cores	8	8
Memory	16GB	16GB
Storage	7TB	6TB

CiteSeerXrelies on many other applications to work. The following software packages must be installed before beginning the CiteSeerXinstallation. We recom-

mend to use the installation procedures introduced for each software because this minimizes the possible problems and thus make the installation smooth.

Java

We recommend to install Java SDK 5.0 or higher. Although RHEL 6.3 comes with *openjdk* and *java-gcj*, we strongly recommend to install the standard JDK downloaded from Oracle website. One should check the architecture of your system (our machine is x86_64). One can either download the current distribution of the binary file, or use the one in our software library.

After installation, we create a symbolic link of java
`$ ln -s /usr/local/sbin/jdk1.7.0_09/bin/java /usr/sbin/`
The `JAVA_HOME` environment parameter should also be specified in `.bashrc`
`export JAVA_HOME=/usr/local/sbin/jdk1.7.0_09`. Do not forget the compile the `.bashrc` after updating it.

Perl

We recommend to install Perl 5.8 or higher. This usually automatically installed with RHEL 6.3.

MySQL

We recommend to install MySQL 5.1 or higher. This version usually comes in the yum repository with RHEL 6.3, but we strongly recommend MySQL 5.5 or MySQL 5.6 for enhanced scalability and performance.

```
$ yum install mysql-server.x86_64
$ yum install mysql-devel.x86_64
```

Note that `mysql.x86_64` will be automatically installed as a dependency of `mysql-server`. To upgrade to MySQL 5.5, download the `mysql-community.repo` and it to `/etc/yum.repos.d/`, edit that file to enable MySQL 5.5 and run `yum update`.

After installation, change the MySQL configuration file at `/etc/my.cnf` before starting the server. A sample version is below, assuming replication may be done on another server.

```
[mysqld]
datadir = /data/mysql
socket = /var/lib/mysql/mysql.sock
```

```
innodb_file_per_table
innodb_flush_log_at_trx_commit =1
log-bin = /data/mysql/mysql-bin
server-id = 1
innodb_data_home_dir = /data/mysql/
innodb_data_file_path = ibdata1:50M:autoextend
innodb_log_group_home_dir = /data/mysql/
sync_binlog = 1
innodb_buffer_pool_size = 6G
innodb_additional_mem_pool_size = 20M
innodb_log_file_size = 256M
innodb_log_buffer_size = 4M
[mysql.server]
user = mysql
basedir = /var/lib
[mysqld_safe]
log-error = /var/log/mysqld.log
pid-file = /var/run/mysqld/mysqld.pid
```

You can start MySQL as

```
$ service mysqld start
```

You should be able to see two processes running by using

```
$ ps aux |grep mysql
```

One is `/usr/bin/mysqld_safe` and the other is `/usr/libexec/mysqld`.

Next, you should set the root password, by running

```
$ /usr/bin/mysql_secure_installation
```

The root password in this instance does not need to be complicated, because we will soon import the production database, including the user credential information, which will overrides the current root password.

Now you can try to login to MySQL as root *using the right socket file* (consistent with `my.cnf`:

```
$ mysql -u root -p --socket=/var/lib/mysql/mysql.sock
```

You should be able to see two databases: `mysql` and `information_schema`.

You can import the database dumped from the production machine using the script in `script/dbimport.sh`. This script can calculate the importing time.

After importing, the first login still requires you to use the *previous* root password. After logging in, you need to run

```
mysql> FLUSH PRIVILEGES;
```

command to make the new user credentials to be effective. After logging out, you must use the new user names and password to login again.

Tomcat

We recommend Apache Tomcat 6 or higher, Tomcat 7.0 is currrently used in production. We install Tomcat under `/usr/local/tomcat`. All we need to do is to copy the tomcat folder under `/usr/local/`. But we need to setup environment variables, e.g.,

```
JAVA_HOME=/usr/local/sbin/jdk1.7.0_10
CATALINA_HOME=/usr/local/tomcat
```

To start Tomcat:

```
$CATALINA_HOME/bin/startup.sh
```

Note that in this case, the user must have full permission to access to `/usr/local/tomcat`[1]. You should see something like this:

```
Using CATALINA_BASE: /usr/local/tomcat
Using CATALINA_HOME: /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME: /usr/local/sbin/jdk1.7.0_10
Using CLASSPATH: /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
```

To check if Tomcat is running, use

```
$ ps aux |grep tomcat
```

To stop Tomcat, use

[1]We recommend to set the permission to `775` so that group members can still have full access.

```
$ $CATALINA_HOME/bin/shutdown.sh
```

Sometimes, the `shutdown.sh` does not kill the Tomcat instances. Even if you add the `-force` option. You will still see the process using `ps aux|grep tomcat`. In order to completely shutdown Tomcat, you must manually kill the process:

$ kill [tomcat pid]

or, follow the workaround below:

1. Add the following statement in red in the `catalina.sh` to define the path of Catalina PID file.

   ```
   # Copy CATALINA_BASE from CATALINA_HOME if not already set
   [ -z " $CATALINA_BASE" ] & & CATALINA_BASE="$CATALINA_HOME"

   CATALINA_PID="$CATALINA_HOME"/catalina.pid
   ```

   This makes Tomcat to generate a `catalina.pid` file under `$CATALINA_HOME`.

2. Shutdown Tomcat using the `-force` option:

   ```
   $ $CATALINA_HOME/bin/shutdown.sh -force
   ```

   You should see the following output saying that Tomcat has completely shutdown:

   ```
   Using CATALINA_BASE: /usr/local/tomcat-csx
   Using CATALINA_HOME: /usr/local/tomcat-csx
   Using CATALINA_TMPDIR: /usr/local/tomcat-csx/temp
   Using JRE_HOME: /usr/local/java/jdk1.7.0_05
   Using CLASSPATH: /usr/local/tomcat-csx/bin/bootstrap.jar:/usr/local/tomcat-csx/bin/tomcat-juli.jar
   Using CATALINA_PID: /usr/local/tomcat-csx/catalina.pid
   Killing Tomcat with the PID: 2615
   ```

Solr

We recommend Apache Solr 4.9. To install Solr 4.9, first install Tomcat:

1. Create an account "solr"

2. Unpack Tomcat to `/usr/local/tomcat-solr`

   ```
   $ tar xvzf apache-tomcat-8.0.11.tar.gz
   $ sudo mv apache-tomcat-8.0.11 /usr/local/tomcat-solr
   $ sudo chown -R solr:solr /usr/local/tomcat-solr
   ```

Then install Solr,

1. Get solr package

```
$ tar xvzf solr-4.9.0.tgz
```

2. Install Solr web application and libraries to Tomcat

```
$ sudo -u solr cp solr-4.9.0/dist/solr-4.9.0.war \
  /usr/local/tomcat-solr/webapps/solr.war
$ sudo -u solr cp solr-4.9.0/example/lib/ext/*.jar \
  /usr/local/tomcat-solr/lib/
```

3. Copy the example Solr home directory as a template, assuming `Solr` data was put under `/data`:

```
$ sudo cp -r solr-4.9.0/example/solr /data/
$ sudo chown -R solr:solr /data/solr
```

4. Rename collection name to `citeseerx`

```
$ sudo mv /data/solr/collection1 /data/solr/citeseerx
```

   And edit `/data/solr/citeseerx/core.properties`:

```
name=citeseerx
```

   Solr will automatically discover Solr core directories which contain `core.properties`. The name of a Solr core is determined by `name` option in `core.properties` and the directory name doesn't matter.

5. Edit `/usr/local/tomcat-solr/webapps/solr/WEB-INF/web.xml`

```
<env-entry>
   <env-entry-name>solr/home</env-entry-name>
   <env-entry-value>/data/solr</env-entry-value>
   <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

6. Restart tomcat

```
$ /usr/local/tomcat-solr/bin/shutdown.sh
$ /usr/local/tomcat-solr/bin/startup.sh
```

7. Open `http://localhost:8080/solr` in the browser.

Ant

We recommend Apache Ant 1.7.1 and follow the instructions in

```
http://ant.apache.org/manual/install.html
```

To install Ant. Basically, after making sure Java is installed and setting the `JAVA_HOME` environment variable, just download the binary edition and uncompress the downloaded file into the directory, e.g., `/usr/local/sbin`. For convenience, you can create a symbolic link:

```
$ ln -s /usr/local/sbin/apache-ant-1.7.0/bin/ant /usr/bin/ant
```

It is also recommended to add `/usr/local/sbin/apache-ant-1.7.0/bin` to the `PATH` environment variable.

Axis2

We recommend Apache Axis2 (for DOI server). At least Axis2-1.6.2 is tested. To install Axis2, first download the `war.zip` file, unzip it and copy it under `tomcat/webapps/axis2.war`. An example URL is

```
$ wget http://apache.mirrors.tds.net//axis/axis2/java/core/1.6.2/axis2-1.6.2-war.zip
```

Restart Tomcat and we should see a folder named `axis2` in the `tomcat/webapps` folder. You shold be able to verify this by visiting

```
http://localhost:8080/axis2
```

However, there is no service at this time.

3.1.2 EPEL repository

The EPEL repository contains many useful packages. It can be installed from the rpm package, which can be downloaded from the URL below

```
http://ftp.osuosl.org/pub/fedora-epel/
```

You must choose the folder based on your Redhat release (typically 5 or 6) and CPU architecture (i386 or x86_64). The rpm file can be found from the file list, such as `epel-release-5-4.noarch.rpm` or `epel-release-6-8.noarch.rpm`.

Installation uses rpm command

```
$ sudo rpm -ivh epel-release-5-4.noarch.rpm
```

Table 3.2. Parameters in database configuration.

Parameter	Default	Functionality
Which databases	`ALL`	Databases you would like to install
Host	`localhost`	The database server
Root password	`[passwd]`	The root mysql password
CSX User	`csx-devel`	The username of the CiteSeerX user
CSX Password	`csx-devel`	The password for the CiteSeerX user
CSX Domain	`ist.psu.edu`	The domain on which CiteSeerX will run
Site ID	`10`	The id number for the website
Deployment ID	`1`	The id of the deployment

3.1.3 rpmforge repository

The rpmforge repository can be installed in a similar way as EPEL.

```
$ wget http://packages.sw.be/rpmforge-release/rpmforge-release-0.5.2-2.el5.rf.x86_64.rpm
$ sudo rpm --import http://apt.sw.be/RPM-GPG-KEY.dag.txt
$ sudo rpm -K rpmforge-release-0.5.2-2.el5.rf.*.rpm
$ sudo rpm -ivh rpmforge-release-0.5.2-2.el5.rf.*.rpm
```

3.1.4 Getting CiteSeerX

We recommend to download the latest CiteSeerX version from sourceforge

```
https://github.com/SeerLabs/CiteSeerX
```

3.1.5 Database Configuration

The CiteSeerXsource contains a script called `installdb.pl` located in the install directory. Run the following command to execute the script:

```
$ perl installdb.pl
```

When the script runs, it is going to ask for a couple parameters. Here are the parameters:

3.1.6 Repository

Create a directory on your system. The default is `/repositories/rep1`. Also, make sure to give the user who will be running CiteSeerX the ownership and full permission of the repository.

3.1.7 Configure CiteSeerX

The basic premise behind the configuration is to edit a few files which contain parameters in the source directory without editing the code. First, copy the

```
csx.config.properties.template

csx.config.properties
```

in the `conf` directory. Then assign the `jdbc.csx.username` to be the MySQL username. Do the same thing for the `jdbc.csx.password` and for the next three sections (Citation Graph DB, MyCiteSeer DB, CiteSeerX External Metadata database) changing the username and passwords to match your database configuration.

Next, make sure that `repository.path` matches the location of your repository.

Lastly, make sure `foul.word.list` points to the location of the fowl word list (this is a big source of error so **double check** the path.

3.1.8 Building the CiteSeerX `.war` File

Now that we have all the files configured, it is time to build the CiteSeerX application so we can deploy it with Tomcat. Navigate to the top level of the source directory and you should see the Ant build script – `build.xml`. To build the CiteSeerX app and DOI server, execute the following two commands:

```
$ ant
$ ant doiserver
```

3.1.9 Install the DOI Server

Resources ingested by CiteSeerX require a unique ID, or Digital Object Identifier (DOI) that will stay the same for lifespan of the recourse. DOIServer provides the means to obtain unique DOIs safely, such that once a particular DOI is issued, that DOI will never be issued again at a particular site or at other sites that use DOIServer. DOIs are of the following form:

```
<SITE_ID>.<DEPLOYMENT_ID>.<TYPE>.<BIN>.<RECORD>
```

All variables are integers and have the following semantics:

- SITE_ID: a label that uniquely identifies the site where CiteSeerX is hosted.
- DEPLOYMENT_ID: a label that uniquely identifies the particular DOIServer deployment within the site (allows safe replication of DOIServer application).
- TYPE: a label identifying the type of resource that a DOI was issued for.
- BIN: an ID space for a particular resource type, meant to bound the size of the DOI string.
- RECORD: an ID for a particular resource, with values ranging from 1–9999.

When a new DOI is issued for a particular site, deployment, and resource type, if incrementing the RECORD field would result in a value greater than 9999, the BIN value is incremented instead and the RECORD value is dropped to 1.

In order to run the DOI server, Apache Axis2 must be installed and configured on Tomcat 6. A running MySQL database will also be needed to store the site configuration and DOI data.

First, setup the MySQL databases for the DOI information. If you already have a dump of the csxdoi database, you can just import it. If you start with a fresh install, the database can be created with the following specs:

```
create database csxdoi;
use csxdoi;


create table configuration (deployment varchar(15) not null,
 site_id int unsigned not null,
 deployment_id int unsigned not null,
 primary key(deployment));


insert into configuration values ("DEPLOYMENT", <SITE_ID>, <DEP_ID>); (Here
SITE_ID is a unique site label for your CiteSeerX mirror and DEP_ID is a
label for this particular deployment of DOIServer.  Only one row should
ever be put in a single configuration table.)
create table doi_granted (id int unsigned not null auto_increment,
 doi_type int not null, bin int unsigned not null,
 rec int unsigned not null,
 assigned timestamp default current_timestamp,
 primary key(id), index(doi_type), index(bin), index(rec),
 index(assigned));
grant all on csxdoi.* to csxdoi@localhost identified by "doipass"
```

CiteSeerX uses a SITE_ID=10. For another mirror site, a diffent SITE_ID should be used, e.g., 20. This will generate a sequence of DOI starting with 20, e.g., 20.1.1.1.1. Doing this can resolve the conflict between a set of transferred data, and a set of self-ingested papers.

Then, edit the Tomcat tomcat/conf/server.xml, adding the following lines within the <Host> block, editing the bracketed strings (in red) to the appropriate values you specified during database installation:

```
<Context path="/axis2" docBase="axis2" debug="5"
reloadable="true" crossContext="true">
<Resource name="jdbc/DOIDB"
auth="Container"
type="javax.sql.DataSource"
maxActive="10"
maxIdle="30"
maxWait="10000"
username="csxdoi"
password="doipass"
driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://[HOST]:3306/csxdoi?autoReconnect=true"
/>
</Context>
```

Next make sure that the MySQL driver mysql-connector-java-5.0.8.jar is in the lib directory of your Tomcat 6 installation, i.e., /tomcat/lib/. This drive can be found from the delopyment of the CiteSeerX web server under tomcat/webapps/ROOT/WEB-INF/lib.

Then, build the DOI system. You need to do it under the citeseerx/trunk.

```
$ ant doiserver
```

This will generate the following file

```
dist/services/DOIServer/DOIServer.aar
```

Copy this file to the tomcat/webapps/axis2/WEB-INF/services.

You can check the DOI server is running by starting the Tomcat web server and browse to axis2 and Services.

```
$ elinks http://localhost:8080/axis2
```

Click Services and you should see DOIServer below the Version service.

To run the testing script, you need to install the SOAP::Lite perl module using the PAN tool. Make sure that perl, gcc-c++, make, expat, expat-devel, and perl-CPAN are

all installed. Although you can configure `CPAN` to install everything under your home directory, you still need sudo account to write the manual files under `/usr/share/man`, so it is recommended to install it just using `sudo` account.

```
# Enter the CPAN shell
$ sudo perl -MCPAN -e shell


# Install YAML, may be needed it later
$ install YAML

# install SOAP::Lite
$ install SOAP::Lite
```

After `SOAP::Lite` is installed, you can test the DOI server using the perl script `src/perl/DOIClient/bin/doiclient.pl`.

```
$ perl doiclient.pl http://localhost:8080/axis2/services/DOIServer
```

If everything goes smoothly, you should see the following output:

```
SOAP::Transport::HTTP::Client::send_receive:  POST http://localhost:8080/axis2/services/DOIServer
HTTP/1.1
Accept:  text/xml
Accept:  multipart/*
Accept:  application/soap
Content-Length:  460
Content-Type:  text/xml; charset=utf-8
SOAPAction:  "http://doi.citeseerx.psu.edu#getDOI"


<?xml version="1.0" encoding="UTF-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://doi.citeseerx.psu.edu"><doiType xsi:type="xsd:int">1</doiType></getDOI></soap:Body></soap:Envelope
SOAP::Transport::HTTP::Client::send_receive:  HTTP/1.1 200 OK
Connection:  close
Date:  Mon, 25 Mar 2013 19:50:56 GMT
Server:  Apache-Coyote/1.1
Content-Type:  text/xml;charset=utf-8
Client-Date:  Mon, 25 Mar 2013 19:50:56 GMT
Client-Peer:  127.0.0.1:8080
Client-Response-Num:  1
Client-Transfer-Encoding:  chunked


<?xml version='1.0' encoding='utf-8'?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><
nv:Body><ns:getDOIResponse xmlns:ns="http://doi.citeseerx.psu.edu"><ns:return>10.1.1.273.1</ns:return></ns:getDOI
e></soapenv:Body></soapenv:Envelope>
Response:  10.1.1.273.1
```

If you do not see the output above, you may just need to correct the namespace.

3.1.10 Test the Installation

To test the installation, open a browser and navigate to

```
http://localhost:8080/citeseerx
```

The main CiteSeerX homepage should be displayed (Figure 3.1). You may also display the mainpage using `elinks`, which is a terminal-based browser.

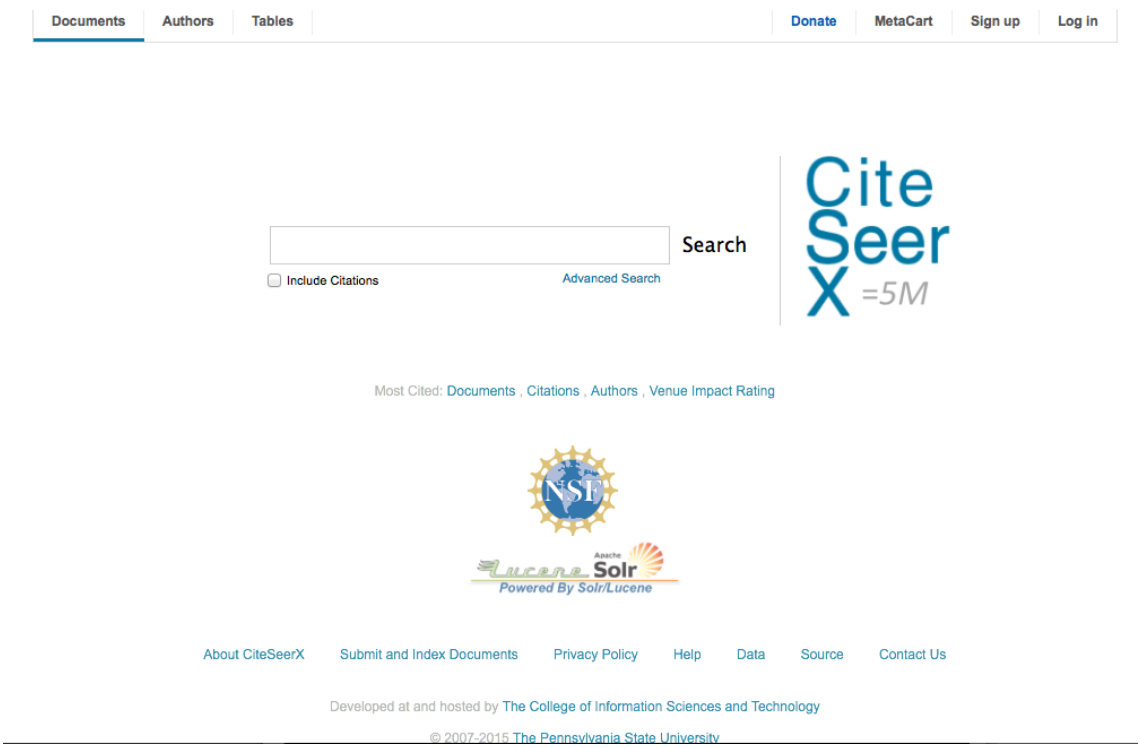

Figure 3.1 The homepage for the CiteSeerX search engine.

3.2 Installation on the Distributed Cluster

In the text below, we breakdown the installation process for each server and install them on a distributed server cluster. Follow Section 3.1 to install general software packages.

3.2.1 Load Balancers and Web Servers

The motivation to have a load balancer is that a single server cannot handle all the traffic. Therefore, if we have multiple web servers, we need a special server to distribute the workload, which is called *load balancers*. These web servers may have different public IP addresses, but the website is accessed from a virtual IP which is attached to a NIC device on the load balancer. The incoming requests are rerouted to multiple web servers. The HTTP responses are then returned either via the load balancer or directly from the web server.

There are software and hardware load balancers. Software load balancers include `heartbeat ldirectord` and `Piranha`. Hardware load balancers are most expensive but more reliable. Enterprize level services such as university department usually leverages hardware load balancers. CiteSeerX uses `heartbeat ldirectord` as the load balancer. However, this software package has not been supported for years[2]. Although it works for RHEL5, it does not support RHEL6 (or CentOS 6). Users may consult Piranha websites for configuration details.

If the digital library instance is just for testing or has a small user group or high availability is not required, a single web server can take all requests, so a load balancer is not needed.

3.2.2 Repository Servers

If the repository is too big so be hosted on a single machine, it can be hosted by a dedicated server. The storage can then be mounted to the web server. CiteSeerX uses GFS2+GNBD to create a block device and mount it to all web servers. We are upgrading this model so the repository storage is connected to web servers using iSCSI. For non-production purposes, mounting the storage as NFS should work. CiteSeerX will launch without the repository, but the download function will not work. As long as the repository is visible on the web servers, the download function should work.

3.2.3 Database Servers

Installing MySQL on a dedicated database server follows the same steps as installing all components on a single server. The steps are more complicated if a replication is required, which synchronizes data with the master server. To setpu a replication, follow the steps below.

[2] `http://www.ultramonkey.org/news_archive.shtml`

1. Determine what ID value you want to assign to each server and record it in an option file that the server reads when it starts (usually the `my.cnf` file). These IDs must be different and each should be a positive number from 1 to $2^{32} - 1$. The ID values will be needed for the `server-id` setup option used with each server. In addition, enable binary logging on the master. To accomplish this on the master and slave, respectively, use option groups with the following lines:

```
[mysqld]
server-id=master_server_id
log-bin=binlog_name


[mysqld]
server-id=slave_server_id
```

2. On the master server, setup an account that the slave server can use to connect to the master server and request updates. No other privileges are needed if the account is used only for the single limited purpose of replication. However, we recommend to grant a `SELECT` privilege just in case you plan to view the databases [3] and connect from master from the slave host "manually" for testing.

```
mysql> CREATE USER 'slave'@'%' IDENTIFIED BY 'slave_pass';
mysql> GRANT SELECT,REPLICATION SLAVE ON *.* TO 'slave'@'%';
mysql> FLUSH PRIVILEGES;
```

3. Connect to the master server and determine its current replication coordinate by executing

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

   Remember the `File` and `Position` values. You will need them later so that you can tell the slave the point from which to start reading binary log events from the master. **you must make sure that no updates occur on the master from the time that you determine its replication coordinate until you make a snapshot of its data.**

4. Perform the initial synchronization of this slave to the master server by copying the master's database to the slave. This can be done by making a backup copy on the master. See § 4.4.1 for a details explanation of the command options.

[3] If `REPLICATION SLAVE` is the only privilege granted to the account, you might not even be able to see database names on the master server with `SHOW DATABASES;`

5. Connect to the slave and use `CHANGE MASTER` to configure it with the parameters for connecting to the master server and the initial replication coordinates:

```
mysql> CHANGE MASTER TO
MASTER_ROOT='master_host',
MASTER_USER='master_user',
MASTER_PASSWORD='slave_pass',
MASTER_LOG_FILE='log_file_name',
MASTER_LOG_POS=log_file_pos;
```

6. Tell the slave to start replicating:

```
mysql> START SLAVE;
```

   The slave should connect to the master and start replicating. You can check this with the

```
mysql> SHOW SLAVE STATUS;
```

   statement on the slave. The `STOP SLAVE` and `START SLAVE` statements suspend and resume a slave server's replication related activity.

3.2.4 Text Extraction Server

In the real production, at least one server should be exclusively used for text extraction because it consumes a lot of CPU. This section talks about the procedures to install the text extraction codes on a dedicated server with RHEL6 as the OS. The structure of the extraction module is presented below:

```
build # java class files generated by compiler - directory created automatically
on build
build.xml # ant build file
/config # holds the config.properties file which contains project settings
/converters # contains binaries and needed files for pdf to text converters
/cpy # all files in here got copied to the dist directory on 'ant jar' or
'ant run' command
/crfpp # contains crf_learn and crf_test binaries as well as traindata folder.
Used by parsCit I think
/dist # where the built jar file is placed as well as working resources
during runtime - generated on 'ant jar'
/lib # contains perl libraries for parsing, jar files required by the java
program, and parseDocuments.pl script executed by jar
/logs # contains log files from each run of jar
/resources# contains resources such as dictionaries used by perl scripts
during parsing
```

```
/src # contains java source code
/svm-light # SVM classifier
/tmp # holds inconsequential files used temporarily
```

1. Install the ActivePerl Community version. This can be downloaded from

   ```
   http://www.activestate.com/activeperl/downloads
   ```

   The 5.8 version works OK, but may have some bugs that were fixed in later versions. We recommend to use version 5.16 which does not produce those error messages seen with version 5.8. After downloading the package, just copy the folder to `/opt/ActivePerl-5.16`. Note that at this point, you may have two Perls installed: a regular perl and an ActivePerl. We will use the latter one. You may use the regular perl, but it is not recommended at this time.

2. Install perl packages to ActivePerl library.

   ```
   $ /opt/ActivePerl-5.16/bin/cpan String::Approx
   $ /opt/ActivePerl-5.16/bin/cpan XML::Bare
   ```

3. Mount the crawl repository to the text extraction machine. The NFS service must be started on the crawl repository server, and the `nfs-utils` must be installed on the extraction server. The following line must added to `/etc/exports` on the crawl repository server to make the directory mountable assuming the crawled data are saved under `/data/crawldata`.

   ```
   /data/crawldata x.x.x.0/24(rw,sync,no_root_squash,insecure)
   ```

   This statement allows `/data/crawldata/` to be mounted on any machine in the `x.x.x.0` subnet with both read and write permission[4]. Do not forget to restart the NFS service after changing this file. NFS service does not need to be started on the text extraction server.

4. Install `glibc.i686` and `libstdc++.i686` on the text extraction machine if it is a 64-bit machine. This is because the extraction code was compiled in a 32-bit machine. To run it on a 64-bit machine, these two packages need to be installed. To test this, try to run `crf_test` before installing the above two packages, you will get a "bad ELF interpreter" error:

   ```
   bash: ./crf_test: /lib/ld-linux.so.2: bad ELF interpreter: No such
   file or directory
   ```

[4]If `/data/crawldata/` was set to be mountable to another subnet, the new subnet and mount options can be set by just appending the second column above to the same line separated by a space, e.g., `/data/crawldata x.x.x.0/24(rw,sync,no_root_squash,insecure) 192.168.1.0/24(rw,sync,no_root_squash,insecure)`

After installing the above two packages, the output becomes

```
tagger.cpp(57) [feature_index_->open(model.c_str(), 0)] feature_index.cpp(105)
[mmap_.open(filename1)] mmap.h(160) [(fd = open__(filename, flag | O_BINARY))
>= 0] open failed:
```

These two packages can be installed using `yum`.

```
$ sudo yum install glibc.i686
$ sudo yum install libstdc++.i686
```

5. Download the code from bitbucket

```
https://bitbucket.org/jkillian/csx-extractor
```

6. To build and run project:

```
$ ant jar # builds jar in dist directory and copies other resources
there
$ ant run # starts program
```

These commands should be run from the application's root directory. Everything in the cpy directory gets copied to the dist folder along with the jar when `ant jar` or `ant run` command is run.

7. Configuration. The config options should be set appropriate in the `config/config.properties` file. Note that these settings are only read once at startup and changing them while the program is running won't have any effect. The various perl modules in the lib directory also have Config files where some options can be set.

8. How to stop the program. To stop extraction, modify the `dist/runtime.properties` file so the `stopProcessing` property is set to `true`.

9. Troubleshooting. If an error like this appears when trying to run TET:

```
/lib/ld-linux.so.2:  bad ELF interpreter:  No such file or directory
```

It's most likely caused by a lack of proper 32-but libraries. See

```
http://stackoverflow.com/questions/8328250/centos-64-bit-bad-elf-interpreter
```

for a solution.

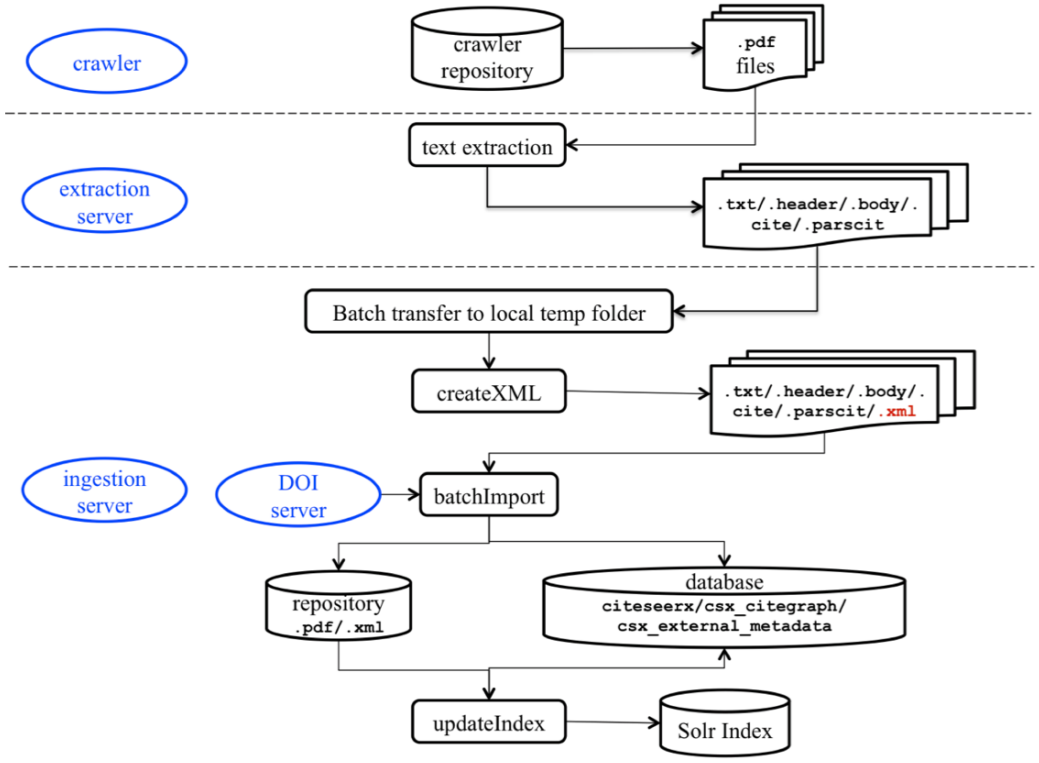

Figure 3.2 The text extraction and ingestion processes.

3.2.5 Ingestion Machine

The ingestion system can be installed on the production repository server[5] (the one mounted to web servers) or it can be installed on the backup repository server (data is pushed to the production server later when traffic is low). As illustrated in Figure 3.2, the ingestion process starts from the output folders of the text extraction machines. Each folder contains `.pdf`, `.txt`, `.body`, `.cite`, `.header` and `.parscit` files. The ingestion currently follows a batch processing model, i.e., a batch of documents in a folder is ingested. The ingestion process contains four steps

1. Transfer the data folder from the text extraction machine to the ingestion machine using `scp` or `rsync`.
2. Build the `.xml` file.
3. Import the documents to the repository and database.

[5]This is the production repository server, which is different from the crawl repository server.

4. Update the index.

The detailed procedures to run the ingestion process is described in Section 4.3. This section focuses on installation of the ingestion system.

1. Make sure that the DOI server is installed and working (see Section 3.1.9).
2. Create an account for `citeseerx` described in Section 3.2.4.
3. Copy the `application` folder under the home directory of `citeseerx`.
4. Install `perl-CPAN`. Install `Log::Log4perl` and `Log::Dispatch`.
5. Create an rsa key for `citeseerx` and copy it to the text extraction server, so that you do not need to input password for the `scp` or `rsync` command. The procedures are

   - On the ingestion server, under the `citeseerx` account

     ```
     $ ssh-keygen -t rsa
     ```

   - Copy the the content in `.ssh/id_rsa.pub` to `.ssh/authorized_keys` under the `citeseerx` home directory.
   - Change the permissions

     ```
     $ chmod 600 .ssh/authorized_keys
     $ chmod 700 .ssh
     ```

   You must create a directory for `citeseerx` to to store files transferred from the text extraction machine. This is a temporary directory before importing the documents. Make sure that `citeseerx` has the permission to write to the destination directory, e.g., `/data/ingest/`.

6. Change two configuration files under `application/conf/`:

   - `applicationContext-csx-jdbc.xml`. Only one place need to be changed in this file in the `repositoryMap` bean, changing the value of `rep1` to the directory of the main repository (just above the `10/1/1/1/1` repository), e.g., `/data/repository/rep1/`.
   - `csx.config.properties`. This is the main configuration file. Some key parameters are listed below

     ```
     ## CiteSeerX core DB
     jdbc.csx.driverClassName=com.mysql.jdbc.Driver
     jdbc.csx.url=jdbc:mysql://dbserver/citeseerx?useUnicode=true&characterEncoding=UTF-8&autoReconnect=true&jd
     jdbc.csx.username=dbuser
     jdbc.csx.password=dbpasswd
     ```

```
## Citation Graph DB
jdbc.citegraph.driverClassName=com.mysql.jdbc.Driver
jdbc.citegraph.url=jdbc:mysql://dbserver/csx_citegraph?useUnicode=true&characterEncoding=UTF-8&jdbcComp
jdbc.citegraph.username=dbuser
jdbc.citegraph.password=dbpasswd
## MyCiteSeer DB
jdbc.mcs.driverClassName=com.mysql.jdbc.Driver
jdbc.mcs.url=jdbc:mysql://dbserver/myciteseerx?useUnicode=true&characterEncoding=UTF-8&jdbcCompliantTru
jdbc.mcs.username=dbuser
jdbc.mcs.password=dbpasswd
## CiteSeerX External Metadata database
jdbc.csxemetadata.driverClassName=com.mysql.jdbc.Driver
jdbc.csxemetadata.url=jdbc:mysql://dbserver/csx_external_metadata?useUnicode=true&characterEncoding=UTF
jdbc.csxemetadata.username=dbuser
jdbc.csxemetadata.password=dbpasswd
## Solr
solr.selectUrl=http://indexserver:port/solrpaperdir/select
solr.updateUrl=http://indexserver:port/solrpaperdir/update
solr.selectTableUrl=http://indexserver:port/solrtabledir/select
solr.updateTableUrl=http://indexserver:port/solrtabledir/update
solr.selectAuthorUrl=http://indexserver:port/solrauthordir/select
solr.updateAuthorUrl=http://indexserver:port/solrauthordir/update
## DOI Server
doi.endpointAddress=http://doiserver:8080/axis2/services/DOIServer
doi.namespace=http://doi.citeseerx.psu.edu/xsd
```

The default Solr port is 8983, but if you install it under Tomcat, the port can be 8080. The namespace of the DOI server may or may not contain the suffix `/xsd`. See Section 3.1.9 for details. The index servers for paper, table and author searches are not necessarily to be the same.

- Run the `doiclient.pl` on the ingestion server to make sure that it passes the client test.

  ```
  $ perl src/perl/DOIClient/bin/doiclient.pl http://doiserver:8080/axis2/services/DOIServer
  ```

  If it has the same error as the localhost (Section 3.1.9), just change the namespace in `doiclient.pl`. But this means that you may need to change the Solr namespace in the source code (see below).

- Make sure that you can access the MySQL database using the account setup in `csx.config.properties` from the ingestion server to the database server. Note that for ingestion purposes, you need more than the `citeseerx` database. In fact, you need `SELECT` and `UPDATE` privileges to the `csx_citegraph`. database. If things do not work through, you can grant more privileges based on the error messages when running the `batchImport` script.

- Now you can run the `ingest.sh`.

3.2.6 Web Crawler

In principle, you can use any crawler to crawl, but because they use different ways to save their crawled contents and metadata, a middleware is needed to import

the crawled contents to the CiteSeerX crawl repository and database. CiteSeerX used Heritrix 3.3.0 as the main crawler. The data are saved in WARC format and imported to the crawl database and repository using Crawl Document Importer (CDI).

Configuring Heritrix 3.3.0 is not that user friendly as Heritrix 1.14.4, mainly because of the complex Spring framework and not all options are provided in the configuration template. We quote some critical blocks here.

Seeds

```
<!-- SEEDS ALTERNATE APPROACH: specifying external
seeds.txt file in the job directory, similar to the H1 approach.
Use either the above, or this, but not both.  -->
<bean id="seeds" class="org.archive.modules.seeds.TextSeedModule">
<property name="textSource">
<bean class="org.archive.spring.ConfigFile">
<property name="path" value="seeds.txt" />
</bean>
</property>
<property name='sourceTagSeeds' value='false'/>
<property name='blockAwaitingSeedLines' value='-1'/>
</bean>
```

The `seeds` bean specifies an alternative seed source: instead of reading seed URLs from the configuration file, the seeds are supplied from an external file called `seeds.txt`.

Domain Whitelist

```
<bean id="acceptSurts" class="org.archive.modules.deciderules.surt.SurtPrefixedDecideRule">
<property name="decision" value="ACCEPT"/>
<property name="surtsSourceFile" value="surts.txt" />
</bean>
```

The `acceptSurts` bean specifies URLs, or hosts, or domains of the crawl scope. There should be a `surts.txt` specifying where you would *like* to go. If the candidate URL does not match any of SURT expressions in this file, this URL is not inserted to the frontier. The content of the SURT file is like[6].

```
+http://(com,microsoft,
+http://(com,ibm,
```

[6]For details about the SURT expression, go to `http://crawler.archive.org/articles/user_manual/glossary.html`

Decide Rules

```
<!-- SCOPE: rules for which discovered URIs to crawl; order is very
important because last decision returned other than 'NONE' wins.  -->


 <bean id="scope" class="org.archive.modules.deciderules.DecideRuleSequence">
  <property name="rules">
   <list>
    <bean class="org.archive.modules.deciderules.RejectDecideRule" />
    <ref bean="acceptSurts" />
    <bean class="org.archive.modules.deciderules.TooManyHopsDecideRule">
          <property name="maxHops" value="1" />
    </bean>
    <bean class="org.archive.modules.deciderules.TransclusionDecideRule">
    </bean>
    <bean class="org.archive.modules.deciderules.surt.SurtPrefixedDecideRule">
          <property name="decision" value="REJECT"/>
          <property name="seedsAsSurtPrefixes" value="false"/>
          <property name="surtsDumpFile" value="${launchId}/negative-surts.dump" />
          <property name="surtsSource">
           <bean class="org.archive.spring.ConfigFile">
            <property name="path" value="negative-surts.txt" />
           </bean>
          </property>
    </bean>
    <bean class="org.archive.modules.deciderules.MatchesListRegexDecideRule">
          <property name="decision" value="REJECT"/>
          <property name="listLogicalOr" value="true" />
          <property name="regexList">
           <list>
            <value>^.*\.js(\?.*|$)</value>
           </list>
          </property>
    </bean>
    <bean class="org.archive.modules.deciderules.PathologicalPathDecideRule">
    </bean>
    <bean class="org.archive.modules.deciderules.TooManyPathSegmentsDecideRule">
    </bean>
    <bean class="org.archive.modules.deciderules.PrerequisiteAcceptDecideRule">
    </bean>
    <bean class="org.archive.modules.deciderules.SchemeNotInSetDecideRule">
    </bean>
   </list>
  </property>
 </bean>
```

The block above specifies the decide rules. These rules are summarized below. Note

tha the order is very important. Only URLs marked with `ACCEPT` after the last rule will be crawled.

1. `RejectDecideRule` Reject all URLs by default.
2. `TooManyHopsDecideRule` Accept URLs whose hops are less or equal to 1 (Maxium crawl depth is 1).
3. `TransclusionDecideRule` Accept URLs whose path-from-seed ends in at least one non-navlink hop.
4. SurtfPrefixedDecideRule Reject URLs whose prefix match any of the listed SURTs in `negative-surts.txt`. This is where the blacklist is applied.
5. `MatchesListRegexDecideRule` Reject URLs that match any of the listed regular expressions. Here, we filter out all potential javascript files.
6. `PathologicalPathDecideRule` Reject any URL that contains an excessive number of identical, consecutive path-segments.
7. `PrerequisiteAcceptDecideRule` Accept all prerequisite URLs – URLs whose hops-pathes have a "P" in the last position.
8. `SchemeNotInSetDecideRule` Reject any URL that has a URI-scheme NOT contained in the configured set.

URL Fetcher

```
<!-- now, processors are assembled into ordered FetchChain bean -->
 <bean id="fetchProcessors" class="org.archive.modules.FetchChain">
  <property name="processors">
   <list>
    <ref bean="preselector"/>
    <ref bean="preconditions"/>
    <!-- ...fetch if DNS URI... -->
    <ref bean="fetchDns"/>
    <!-- ...fetch if HTTP URI... -->
    <ref bean="fetchHttp"/>
    <!-- ...extract outlinks from HTTP headers... -->
    <ref bean="extractorHttp"/>
    <!-- ...extract outlinks from HTML content... -->
    <ref bean="extractorHtml"/>
```

This block above specifies from which types of documents the URLs are extracted.

Disposition

```
<bean id="warcWriter" class="org.archive.modules.writer.WARCWriterProcessor">
      <property name="shouldProcessRule">
       <bean class="org.archive.modules.deciderules.DecideRuleSequence">
        <property name="rules">
         <list>
          <!-- Begin by REJECTing all -->
          <bean class="org.archive.modules.deciderules.RejectDecideRule" />
          <bean class="org.archive.modules.deciderules.ContentTypeMatchesRegexDecideRule">
           <property name="decision" value="ACCEPT" />
           <property name="regex" value="^application/pdf" />
          </bean>
         </list>
        </property>
       </bean>
      </property>
 </bean>
```

This block specifies the disposition (download) criteria. Specifically, it only downloads papers whose content-type is `application/pdf`. The data are saved in WARC format.

Crawl Control

```
<!-- CRAWLCONTROLLER: Control interface, unifying context -->
<bean id="crawlController"
  class="org.archive.crawler.framework.CrawlController">
      <property name="maxToeThreads" value="50" /> </bean>
```

This block specifies the number of threads, which is `50` here.

3.2.7 Web Crawler Web Server

The website provides information on the crawler history and statistics, including historical ranking by number of crawled documents and accumulative citations as well as a user submission link. It also has a page for users to submit their PDF paper link for us to crawl or they can check what documents are crawled by providing a URL prefix.

This server can be combined with the web crawler. However, we set it up in a separate machine because it may consume a lot of resouces and affect the performance of the web crawler machine, e.g., when updating the crawler history. The following packages are prerequisites.

1. **apache**. The HTTP server, both `httpd` and `httpd-devel` should be installed.
2. **mod_wsgi 2.6**. This can be installed using `yum`.
3. **gcc-c++**. This can be installed using `yum`.
4. **redhat-lsb-graphics**. This can be installed using `yum`.
5. **python-devel, freetype-devel, libjpeg-devel, libpng-devel**. These can be installed using `yum`. The last three are required to display the captcha. You must have them installed before installing `PIL` or `python-imaging`.
6. **python-imaging**. You must install EPEL repository first and update your yum repository. See Section 3.1.2 if you need help to install EPEL.
7. **MySQL**. Install `mysql` and `mysql-devel` using `yum`.
8. **Python2.4+**. Python2.4 is the default Python with RHEL5. Installing multiple Python versions may result in system instability.
9. **django v1.1.1+**. We have tested that it works for Django 1.3.
10. **MySQLdb**. Install `MySQL-python`, and `MySQLdb-1.2.1p2` or higher.
11. **python-setuptools**.

Setting up the crawler website requires four steps. Use *sudo* account if necessary.

1. Copy the *csxbot_code_base* folder to the home directory of a user (may not necessarily be *citeseerx*) and start from here.
2. **Install the csxbot**. Bofore installing it, make sure that the database connection for django project *citeseerx_crawl* is configured in `csxbot-0.3/citeseerx_crawl`. Go to *csxbot_code_base*`/csxbot-0.3/` and run the standard python installation command. You should also make sure that you can connect to the your (local or remote) crawl database.
3. **Install captcha**. Go to `csxbot-0.3/citeseex_crawl/django-simple-captcha-0.3.0` and run the standard python installation command.
4. **Deploy web files**. Just copy the web contents to the Apache home directory.

   ```
   $ cp -r csxbot_code_base/web/* /var/www/csxcrawl_web/
   ```

   Change the ownership of the `stat` folder:

```
$ chown apache:apache /var/www/csxcrawl_web/static/stat
```

5. **Create Python egg cache folder (optional depending on systems)**

```
$ mkdir /var/www/python-eggs
$ chown -R apache.apache /var/www/python-eggs
```

6. **Config apache**. Add the following two red lines to the apache configuration file, usually located at `/etc/httpd/conf/httpd.conf`.

```
Alias /icons/ "/var/www/icons/"
Alias /static/ /var/www/csxcrawl_web/static/
WSGIScriptAlias / /var/www/csxcrawl_web/wsgi/default.wsgi
Alias /robots.txt /var/www/html/robots.txt
```

7. **Start (or restart) apache**.

```
$ /etc/init.d/httpd start
```

or

```
$ sudo service httpd start
```

To restart apache, replace the `start` in the above command with `restart`; to stop apache, replace the `start` in the above command with `stop`.

Chapter 4

Running CiteSeerX

4.1 Running the Crawler

Writing soon...

4.2 Running the Extraction

The extraction process extracts text and metadata information from the crawled documents. After logging into the extraction machine, follow the steps below:

1. Start `screen` (or resume screen with `screen -r`).

2. Change user to *citeseerx*.

```
sudo su - citeseerx
```

3. Make sure that the crawler repository is mounted to the text extraction server using the `df -h` command. For example, `web.crawler.server:/data/crawldata` is mounted to `/export/data` in local.

4. Make sure that the crawler API is available. For example, if the crawler web server is `crawl.web.server`, you can check it from the link below:

```
elinks http://crawl.web.server/api/getdocs.xml?key=yourkey&n=1
```

5. Run the extraction code.

Table 4.1. Files generated by the extraction code.

Suffix	Description
`.pdf`	Original PDF file copied from crawler repository
`.txt`	Text extracted from the PDF file
`.body`	Text body only (no bibliography)
`.cite`	Citation (bibliography) text
`.file`	XML file containing text converter (e.g., PDF Lib, TET), file type (e.g., PDF) and SHA1
`.header`	XML file containing header parser, title, author names, addresses and the abstract
`.met`	XML file containing crawled date, original URL, parent URL and SHA1 (copied from crawler repository)
`.parscit`	XML file containing parsed information of the citation, output of ParsCit

```
$ cd ~/importer/bin/
$ ./processRemotes.pl 1
```

The extrated data are saved to a temporary folder `/data/exports` in folders named as `set#`, in which the number after `set` is the numerical parameter given to the `processRemotes.pl` script. Everytime before running the process command, you should check the sequential number of the last set and use the next number as the parameter that is passed to `processRemotes.pl`. This perl script generates eight outputs files for each document if the text is successfully extracted and parsed, their meanings are listed in Table 4.1. After extraction, the `state` field in the `citeseerx_crawl.main_crawl_document` table is set to `1`. If, for any reason, the extraction process is terminated at some point and the last document is not completely processed, the `state` field remains to be `0`, so you need to re-run that document. It should be noted that the number of *.pdf* files may be smaller than the number of documents requested. For example, you may request 30,000 documents but you only have 25,000 documents in your `set#` folder. The other documents are not extractable, so they are dropped directly.

6. If using `screen` now, it is a good time to detach the screen (Ctrl+a then d).

7. Repeat Step 5 using another number.

The number of URLs consumed in each run can be specified from the `$maxCount` variable in the `processRemotes.pl` script. You should also change the API key to make sure it is consistent with the API key specified in

csxbot-0.3/citeseerx_crawl/main_crawl/config.py

4.3 Running the Ingestion

The ingestion process writes the extracted and parsed text information into the CiteSeerX database and repository and updates the indices. As mentioned before, the ingestion system is installed on the repository server.

1. Start a new screen: `$ screen`
2. Switch to `citeseerx` user account: `$ sudo su - citeseerx`
3. Go to the script directory:

   ```
   cd application/bin
   ```

4. Copy the extraction output from the text extraction server

   ```
   $ rsync -arvz citeseerx@text.extraction.server:/data/exports/set# /data/ingest/
   ```

5. Build the xml files:

   ```
   $ ./createXML.pl /data/ingest/set#
   ```

   This command will generate an `.xml` file for each document.

6. Batch import the set

   ```
   $ ./batchImport /data/ingest/set#
   ```

   When running this command, you may get a connection error. You may need to restart the service on the DOI server. For details, see Troubleshooting.

7. Update the index:

   ```
   $ ./updateIndex
   ```

   When running this command, you may get a 500 error returned by solr on the index server. You may need to delete (or transfer) the old solr snapshot on that server. For details, see Troubleshooting. There is not any parameter in the `updateIndex` process. This script will compare the import date for each cluster in the `csx_citegraph.clusters` database with the last time the update index is run and automatically index the clusters that are not indexed. Here, a cluster is either a paper or a citation. This is different from the cluster of documents, in which different versions are grouped together and share the same cluster number in `citeseerx.papers` table.

There is an bash script `ingest.sh` which combines the commands above. To run this script, all you need is to specify the set number and make sure that `citeseerx` has all the permission to the `/data/ingest` directory, which is a temporary directory.

4.4 Backup

All production systems need to be backed up in case of accidental failure. This applies to CiteSeerX too. We have learned a lot of lessons from failures of hard drives and system compromise. Backing up is more than a copy operation. We need to be careful to ensure that we can get a full restoration without lossing any data.

4.4.1 Database Backup

Although the production database is replicated, it is recommended to keep a copy of database dump on another machine. We use `mysqldump` to backup the all the databases on the production machine. The command below is the command we used to back up the databases on the production, assuming MySQL 5.1+ is installed and *mysqldump* and *mysql* are all in the root path.

```
$ sudo mysqldump --verbose --lock-all-tables -S /var/lib/mysql/mysql.sock
--databases `mysql -S /var/lib/mysql/mysql.sock --skip-column-names -u root -p
-e "SELECT GROUP_CONCAT(schema_name SEPARATOR ' ') FROM information_schema.schemata
WHERE schema_name NOT IN ('performance_schema','information_schema');"`
-u root -p > /data/dbdump.sql
```

The `SELECT` statement is used to select the databases *except* `performance_schema` and `information_schema`. The reason to exclude these two databases is that even the root account may have some table locking permission problems when dumping out these two tables. The text format of this command above is included in the `supplement/csxdbbackup.sh`. This shell script will print the time it spends in seconds when the dumping finishes. Note that the `--lock-all-tables` option is necessary to avoid any changes of the database during the dumping process. The time to dump the databases depends mainly on the buffer size assigned in `/etc/my.cnf`.

If you just need to backup the database at some point, running the command above is enough. However, if you need to do replication, you need to record the binary file coordinate *before* running `mysqldump`.

```
$ mysql>FLUSH TABLES WITH READ LOCK;
$ mysql>SHOW MASTER STATUS;
```

For a full description of replication, refer to § 3.2.3.