

CiteSeerX Manual (CXM)

Jian Wu, Douglas Jordan, Kyle Williams, Pradeep Teregowda, and
Stephen Carman

© *Draft date June 17, 2013*

June 17, 2013

Contents

Contents	i
Preface	1
1 Overview	3
2 Architecture	7
2.1 The Crawler	8
2.2 Load Balancers	15
2.3 Web Servers	17
2.4 CiteSeerX Repository	18
2.5 CiteSeerX Database	21
2.6 Index	23
2.7 DOI	23
2.8 Text Extractor	27
2.9 Ingester	28
3 Installation	31
3.1 Install CiteSeerX on A Single Machine	31
3.1.1 Prerequisites	31
3.1.2 Getting CiteSeerX	35
3.1.3 Database Configuration	35
3.1.4 Solr Configuration	36

3.1.5	Repository	36
3.1.6	Configure CiteSeerX	36
3.1.7	Building the CiteSeerX .war File	36
3.1.8	Install the DOI Server	37
3.1.9	Installing the CiteSeerX Web Application	37
3.1.10	Test the Installation	37
3.2	Installation on the Distributed Cluster	38
3.2.1	Load Balancers and Web Servers	38
3.2.2	Repository Servers	49
3.2.3	Database Servers	51
3.2.4	Text Extraction Server	52
3.2.5	Ingestion Machine	55
3.2.6	Web Crawler Machine	58
3.2.7	Web Crawler Web Server	58
4	Running CiteSeerX	61
4.1	Running the Crawler	61
4.2	Running the Extraction	61
4.3	Running the Ingestion	63
4.4	Backup and Restoration	64
4.4.1	Database Backup	64
A	Configuration Examples	65
A.1	Databases	65
A.1.1	my.cnf for MySQL Database Server	65

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 developement
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.

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 Boalacker. 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 Council 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 legacy 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, metadata, 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 research 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://louise.ist.psu.edu/submit`

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 indices, 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 Explorer¹, 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-Harvester² (perl), OAIHarvester³ (Java), .NET OAI Harvester⁴ (.NET dll), UIUC OAI⁵.

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 comprised of a focused web crawler, a document extractor, 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 depicted in Figure 2.1.

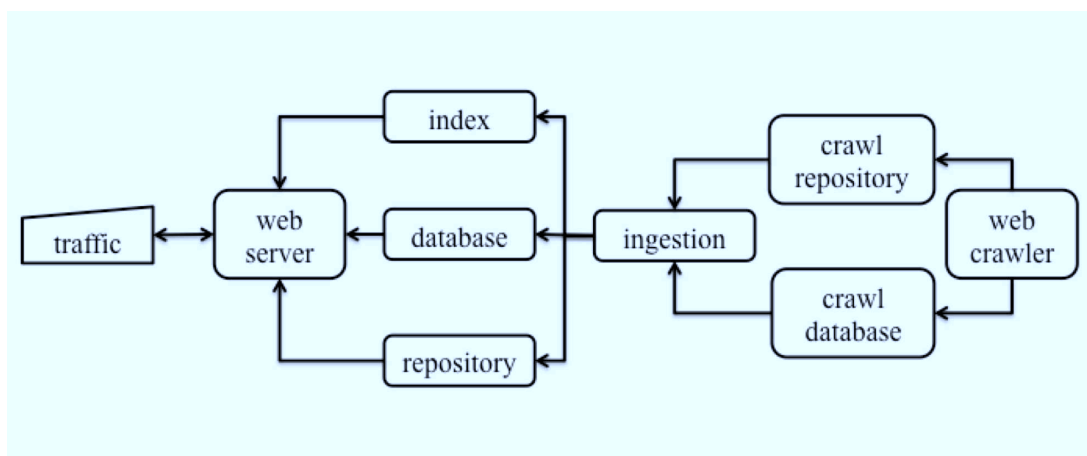


Figure 2.1 The big architecture of the CiteSeerX system.

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 the SYZ crawler to perform our incremental crawl. The architecture of the SYZ crawler is shown in Figure 2.3. Most of the seed URLs are adopted from the whitelist but we also accept user submitted URLs from the crawler website. At 8 pm each Friday evening, the crawler scheduler selects all the seed URLs from

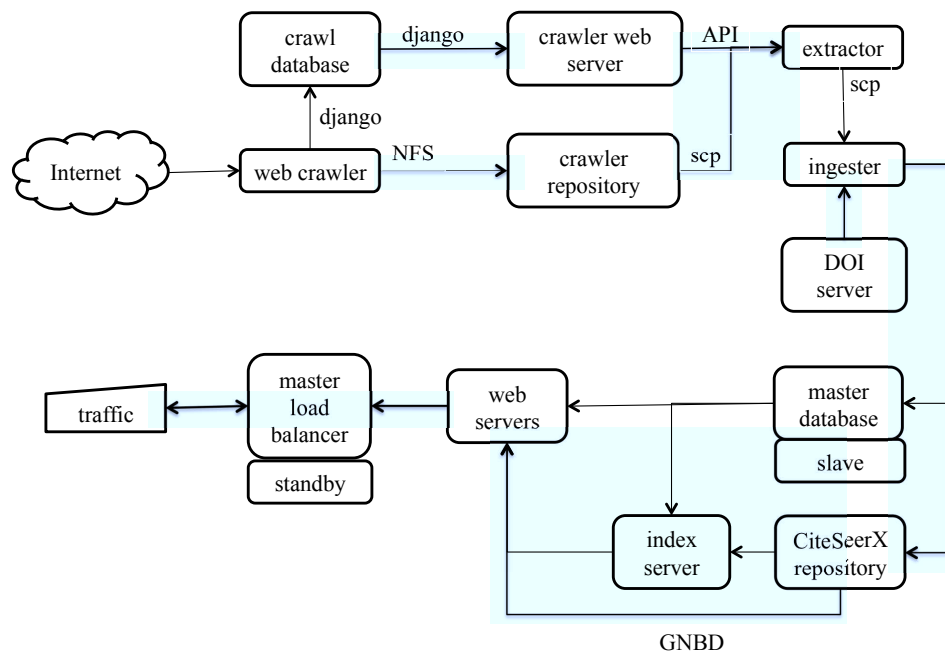


Figure 2.2 The architecture of the CiteSeerX system and their main jobs. Note that the system has two load balancers and two web servers to form a high availability cluster. It also has a slave database server which replicate the master database.

the *whitelist2* table in the *citeseerx_crawl* database, and submits them to the SYZ queue. Each 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. These URLs are filtered based on some rules. First, the length of the URLs must be no more than 255 characters. Second, we filter out URLs that match any of the hosts in the blacklist. Duplicate URLs are eliminated by the DUE module. The remaining URLs are submitted to the crawler frontier. 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. Besides `html/text`, the acceptable content types include PDF, postscripts and their zipped formats. The size of each downloaded document is limited to 100-megabytes. 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 2.4. 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 SYZ 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 in NFS4 format. 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 text extractor. 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 docu-

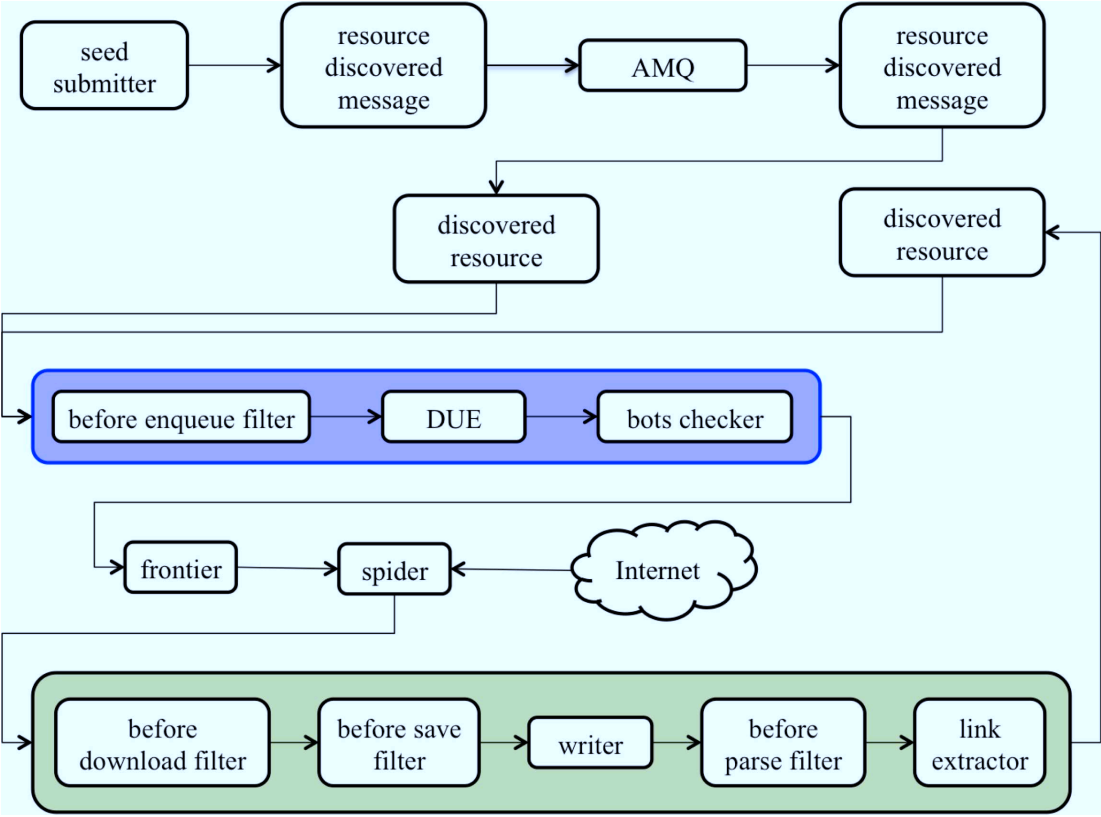


Figure 2.3 The architecture of the SYZ crawler.

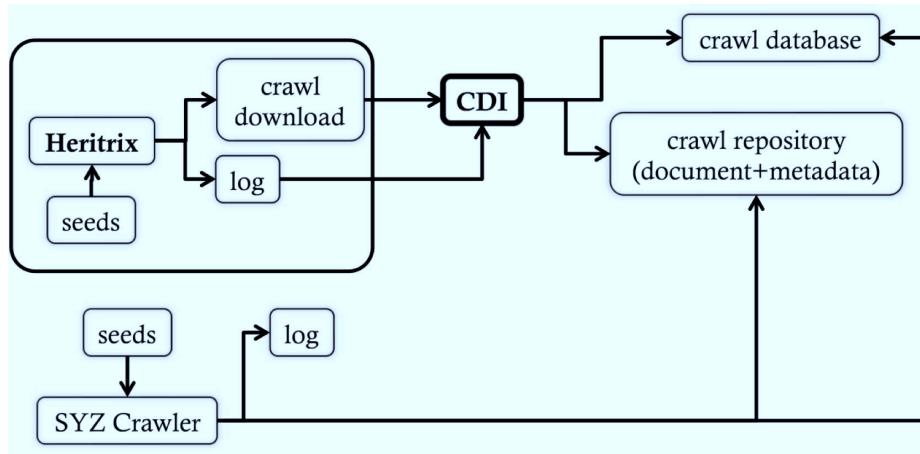


Figure 2.4 The role of the CDI middleware with the crawler, the repository and the database. The CDI plays a “bridge” between the crawler and the crawler repository/database. In this figure, Heritrix can be replaced with another crawler or FTP downloads. The SYZ crawler was designed specifically for the CiteSeerX project, so it does not require the CDI middleware.

ments. 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 encryption (`SHA1`). The crawl date and the URL information will be recorded in the `citeseerx.urls` table.

The repository is a directory to save filtered crawled documents. Subfolders 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 and saved to the same folder. If an updated version of a document is identified from an existing URL, the old document files in the repository is updated to ensure freshness.

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.

Table 2.1. The `main_crawl_document` table schema.

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

- *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 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, its 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.

Table 2.2. The `main_crawl_parenturl` table schema.

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

Table 2.3. The `main_crawl_submission` table schema.

Field	Type	Description
<code>id</code>	<code>int(11)</code>	Submission ID
<code>url</code>	<code>varchar(255)</code>	Submitted URL
<code>email</code>	<code>varchar(255)</code>	Submitter's email
<code>add_time</code>	<code>datetime</code>	Submission timestamp

Table 2.4. The `main_crawl_whitelist` table schema.

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

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 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* project¹. 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-protocol, 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 2.5.

¹<http://www.ultramonkey.org>

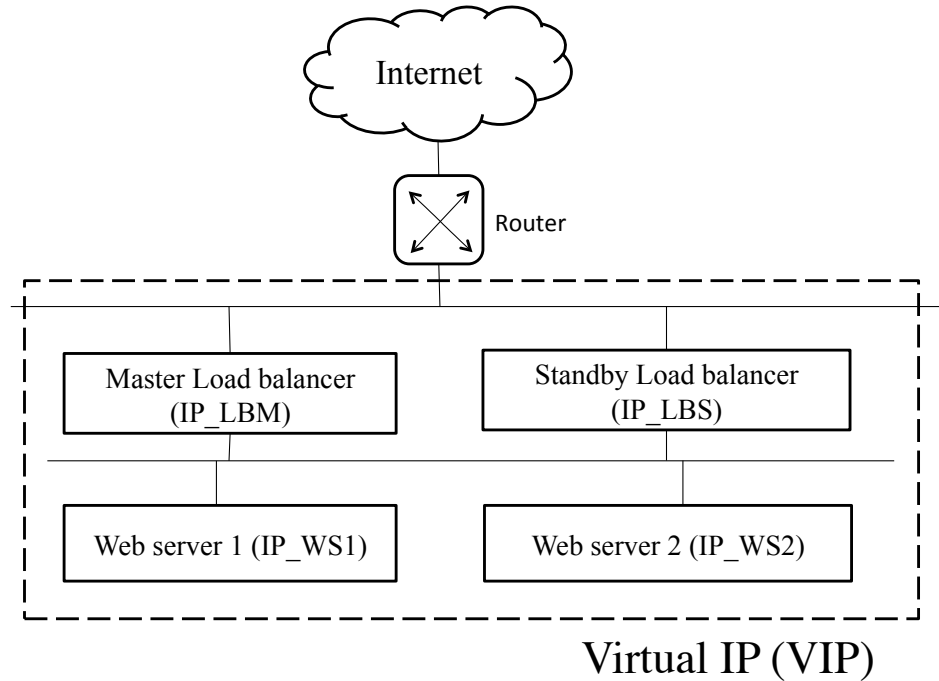


Figure 2.5 The configuration of the high availability load balancers.

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 uses port 80 and Tomcat uses port 8080 by default.

The firewall is implemented using *iptables*. The iptables configuration should allow external access to port 80. Another important 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

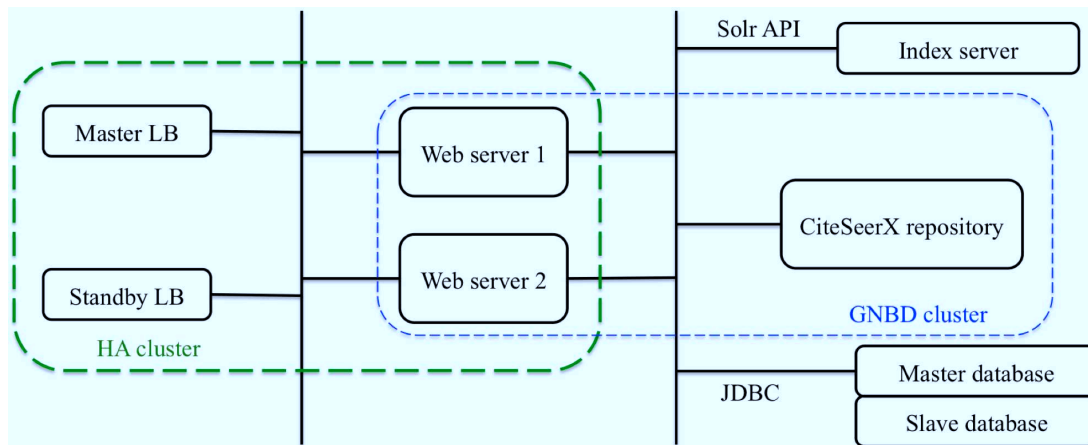


Figure 2.6 The connections between the web servers, the load balancers, the repository server, the index server and the database servers.

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.

<http://www.ultramoney.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 depicted in Figure 2.6.

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 2.7. 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

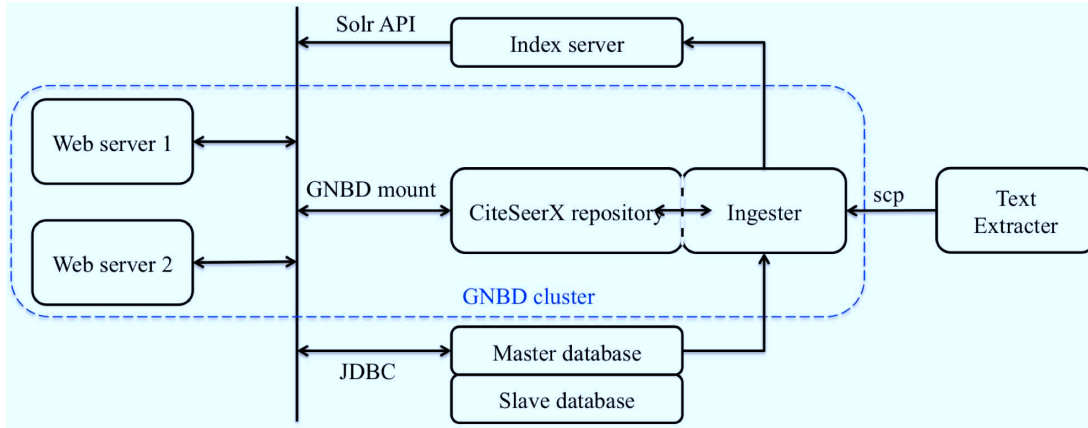


Figure 2.7 The CiteSeerX repository server and other servers that connect to it. The dashed line between the ingester and the repository means that they are on the same server.

server (we use the former). On one hand, the ingester retrieves a copy of the text files extracted from PDF from the text extractor 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 meta-data 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 be covered in Section 2.6. The repository drive is formatted 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 directory can be in any directory on the server, but it must have the following hierarchical structure (Figure 2.8). 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 usually 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 correspond 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.

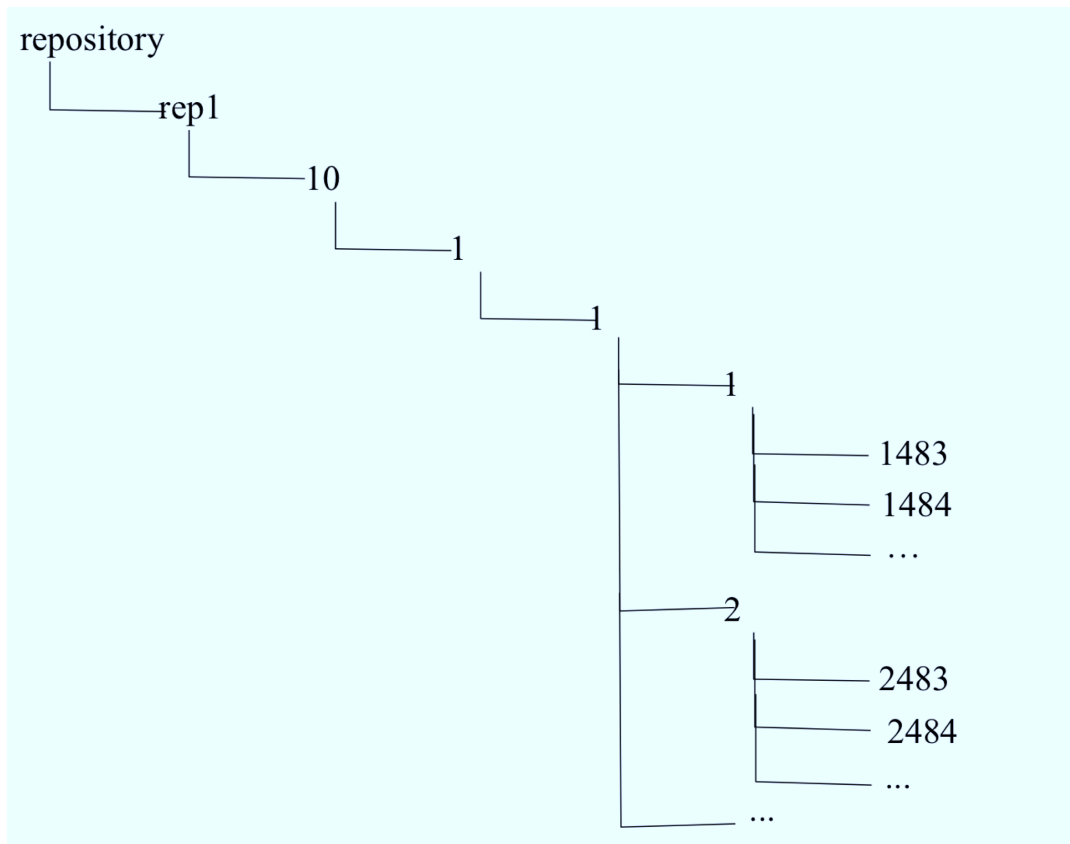


Figure 2.8 The hierarchical structure of the CiteSeerX repository. The deepest directory in this diagram, e.g., 1483, contains document files, such as 10.1.1.1.1483.xml.

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

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

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

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

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 *myciteseerx*. There are another two databases named *csx_citegraph* and *csx_external_metadata*. Besides, the *mysql* database contains the credential 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 2.8. 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. Every-time a paper is ingested, the metadata of this paper is written into master database. The indexing process also needs to read the metadata from the database and send it over to the index server.

Table 2.6. List of tables in the *citeseerr* 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	

¹Number of rows are update to June 13, 2012.²This table was added in September, 2012. The number reflects the records in 2012-12-21.³From the crawler metadata file.

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

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

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).

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 *cite-*

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

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

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

¹Like 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
<code>id</code>	<code>bigint(20) unsigned</code>	Author ID, auto increment
<code>cluster</code>	<code>bigint(20) unsigned</code>	Cluster number ¹
<code>name</code>	<code>varchar(100)</code>	Author full name
<code>affil</code>	<code>varchar(255)</code>	Author affiliation
<code>address</code>	<code>varchar(255)</code>	Author address
<code>email</code>	<code>varchar(100)</code>	Author email address
<code>ord</code>	<code>int(11)</code>	check from Pradeep,Puck
<code>paperid</code>	<code>varchar(100)</code>	DOI, corresponding to <code>id</code> in the <code>papers</code> table

¹Like 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
<code>id</code>	<code>int(11)</code>	Paper listing/removal ID, auto increment
<code>paperid</code>	<code>varchar(100)</code>	DOI, corresponding to <code>id</code> in the <code>papers</code> table
<code>dates</code>	<code>date</code>	Date of operation, e.g., 2012-09-13
<code>operation</code>	<code>enum('REMOVE','LIST')</code>	Operations: 'REMOVE'/'LIST'
<code>requester</code>	<code>varchar(100)</code>	Name of requester, e.g., 'James Bond'
<code>email</code>	<code>varchar(256)</code>	Email of requester, e.g., 'bond@gmail.com'
<code>reason</code>	<code>varchar(256)</code>	Reason of operation, e.g., 'DMCA', 'REQUEST'
<code>operator</code>	<code>varchar(100)</code>	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
<code>userid</code>	<code>varchar(100)</code>	User registered ID, e.g., jxw394
<code>password</code>	<code>varchar(255)</code>	Encrypted user password
<code>firstName</code>	<code>varchar(100)</code>	User first name
<code>middleName</code>	<code>varchar(100)</code>	User middle name
<code>lastName</code>	<code>varchar(100)</code>	User last name
<code>email</code>	<code>varchar(100)</code>	User registered email
<code>affil1</code>	<code>varchar(255)</code>	User primary affiliation
<code>affil2</code>	<code>varchar(255)</code>	User secondary affiliation
<code>enabled</code>	<code>tinyint(4)</code>	User account is enabled (1) or not (0)
<code>country</code>	<code>varchar(100)</code>	User country
<code>province</code>	<code>varchar(100)</code>	User province/state
<code>webPage</code>	<code>varchar(255)</code>	User personal/home web page
<code>updated</code>	<code>timestamp</code>	Last time when user information is updated
<code>internalid</code>	<code>bigint(20) unsigned</code>	Internal user ID, never used
<code>appid</code>	<code>varchar(255)</code>	Encrypted user ID (Ask pradeep)

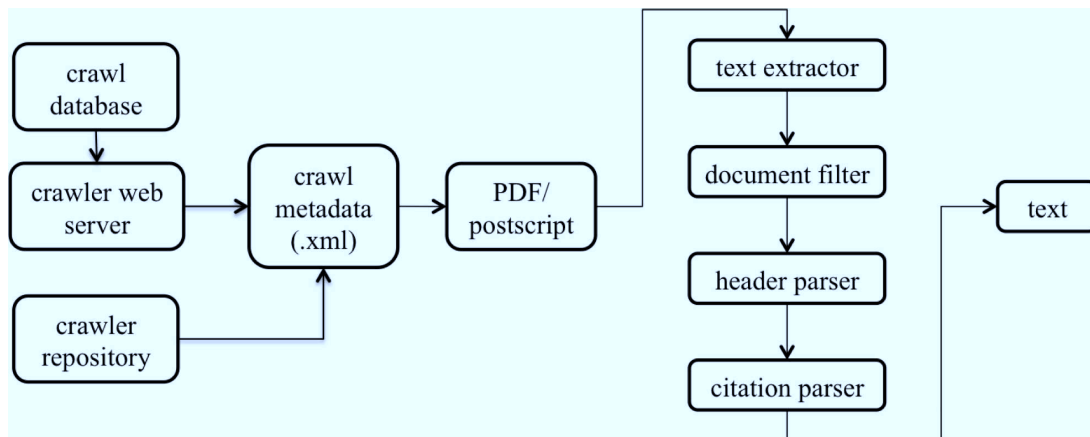


Figure 2.9 The workflow of the text extractor. The “text extractor” box in this figure represents a specific text extraction tool such as PDFBox or PDFLib TET.

seerx.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 2.9. 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 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 2².

The process then retrieves the documents from the crawler repository and saves them to a local directory. A text extractor is then used. CiteSeerX uses PDFlib TET

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

PDF IFilter³ 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.

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

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

2.9 Ingestor

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

³TET is not a free software, see <http://www.pdflib.com/>.

runs a batch import process, in which some associated documents are copied to the repository server. The associated metadata are also written 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. The VM production configuration is very similar to the physical cluster with some slight difference, which we will address in the following description.

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. However, the CiteSeerX currently cannot be installed in a cluster with RHEL 6 because the load balancer we use and the GNBD device cannot be installed on RHEL 6. Here, we assume the OS is RHEL 6.3.

3.1.1 Prerequisites

CiteSeerX does not have a high requirement to the hardware. For example, our previous (physical) and current (VM) staging machine has the following specifications:

CiteSeerX relies on many other applications to work. The following software packages must be installed before beginning the CiteSeerX installation. We recommend to use the installation procedures introduced for each software because this minimizes the possible problems and thus make the installation smooth.

Table 3.1 Hardware specifications of a physical staging machine.

Item	Physical	VM
CPU	Intel Xeon E5405	Intel Xeon E5649
Cores	8	8
Memory	16GB	16GB
Storage	7TB	6TB

Java

We recommend to install Java SDK 5.0 or higher. Although RHEL 6.3 comes with *openjdk* and *java-gcj*, we still 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/
```

and add the java directory to the system path in the `.bashrc` file

`export PATH=/usr/local/sbin/jdk1.7.0_09/bin:$PATH`. The `JAVA_HOME` environment parameter should also be setup

`export JAVA_HOME=/usr/local/sbin/jdk1.7.0_09`. Do not forget to compile the `.bashrc` after updating it.

Perl

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

MySQL

We recommend to install MySQL 5.0 or higher. This version usually comes in the yum repository with RHEL 6.3.

```
$ 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`.

After installation, change the MySQL configuration file at `/etc/my.cnf` before starting the server. You may copy a version on the production machine or in the configuration library, but the replication statements are not necessary (such as `relay-logs`). A sample version is in the `conf/my.cnf`.

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 override 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. You may want to use the `screen` tool to run the importing job at the background. It may take hours to days depending on the memory allocated to MySQL buffer (in `my.cnf` file).

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 also tested to work with CiteSeerX. In most cases, we install Tomcat to `/usr/local/tomcat`. All we need to do is to copy the tomcat folder under `/usr/local/`. But we need to setup environment variables to make sure that Tomcat knows where the java is installed, e.g., set

```
$ JAVA_HOME=/usr/local/sbin/jdk1.7.0_10
```

and

```
$ CATALINA_HOME=/usr/local/tomcat
```

in `.bashrc` (and compile it) under the home directory of the user who launches Tomcat. To start Tomcat, you first switch to that user e.g., `citeseerx`, and then launch Tomcat:

```
$ sudo su - citeseerx
```

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

Note that in this case, `citeseerx` must have full

permission to access to `/usr/local/tomcat`¹. Of course, if you switch to a `sudo` user, you do not have this problem, just do

```
$ sudo bash
```

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

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
```

To check if Tomcat is running, use

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

To stop Tomcat, use

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

Solr

We recommend Apache Solr 1.6 (not 3.x).

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

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

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

```
$ 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 should be able to verify this by visiting

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

However, there is no service at this time.

3.1.2 Getting CiteSeerX

We recommend to download the latest CiteSeerX version from sourceforge

```
http://sourceforge.net/projects/citeseerx/
```

3.1.3 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:

3.1.4 Solr Configuration

First make sure that solr is running by going to

```
http://localhost:8080/solr
```

If you get the `Welcome to Solr!` page, solr is running and you can proceed. If not, check the logs for why solr did not start. Next, copy the files located in `resources/solr` to the `conf` directory in your local instance of Solr.

3.1.5 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.

3.1.6 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` to `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 `fowl.word.list` points to the location of the fowl word list (this is a big source of error so **double check** the path).

3.1.7 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
```

3.1.8 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 resource. DOI Server 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 DOI Server. 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 DOI Server deployment within the site (allows safe replication of DOI Server 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 DOISeer. 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 different `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 deployment 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/DOISeer/DOISeer.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/envelop
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, see Section ?? for troubleshoots.

3.1.9 Installing the CiteSeerX Web Application

In order for CiteSeerX to function properly, you need a couple of libraries. For each of the following libraries, download the .jar file and place it on the Tomcat classpath: log4j, mail, mysql-connector-java. Next, copy the citeseerx.war file from the dist/webapp directory to the webapps directory inside your tomcat instance.

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).

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 ?? 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. 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 two web servers. The HTTP responses are then returned either via the load balancer or directly from the web server.

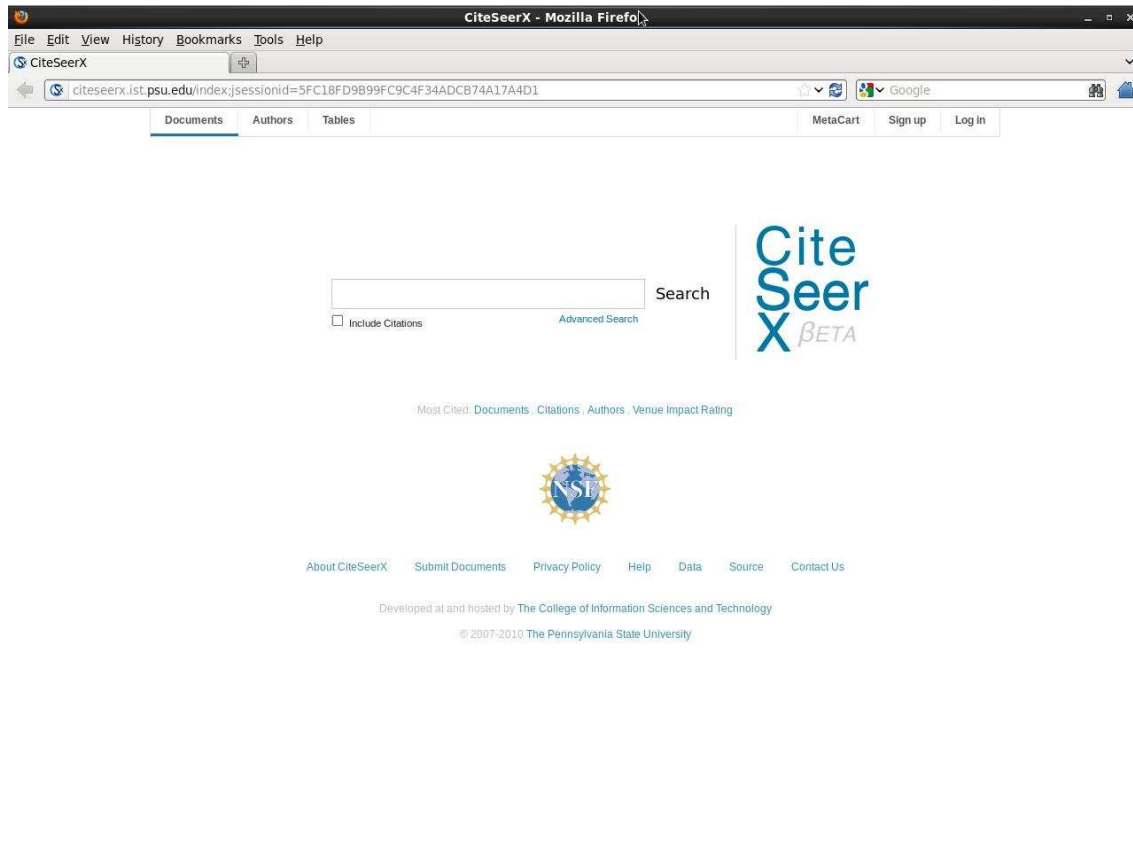


Figure 3.1 The homepage for the CiteSeerX search engine.

There are many ways to setup a load balancer. The production machines of CiteSeerX use *ldirectord* which is a daemon to monitor and administer real servers in a Linux Virtual Server (LVS) cluster of load balanced virtual servers. However, this software package has not been supported for years². Although it works for RHEL5, it does not support RHEL6 (or CentOS 6). As a result, we install the RHEL5 OS on the VM hosts.

We implemented the Linux Virtual Server direct routing (LVS-DR) with firewall-mark (FWM) on load balancers. We use the FWM because it provides a convenient way to reduce complexity by grouping services by address and/or ports. The grouping also provides a convenient way to couple persistence together. If persistence was set for the FWM in *ldirectord.cf* an end user connecting to port 80 and then following a link to port 443 would end up on the same server for both connection.

²http://www.ultramoney.org/news_archive.shtml

Table 3.3. VM servers and network preparations.

hostname	cores	memory	storage ¹	interface	IP
<u>csxlb01</u>	2	4GB	36GB/20GB	eth0	130.203.136.63
				eth1	130.203.136.80
<u>csxlb02</u>	2	4GB	36GB/20GB	eth0	130.203.136.70
				eth1	130.203.136.77
<u>csxweb01</u>	8	16GB	80GB/1TB	eth0	130.203.136.60
				lo:0	130.203.136.95 ²
<u>csxweb02</u>	8	16GB	80GB/1TB	eth0	130.203.136.69
				lo:0	130.203.136.95 ²

Note. — All servers are installed with RHEL5.8 with x86_64 architecture. The netmasks, broadcasts and gateways of both eth0 and eth1 are set to be 255.255.255.0, 130.203.136.255 and 130.203.136.1, respectively. The default gateway can be checked by the `route` command.

¹Storage on processing server and under `/data`.

²130.203.136.95 is the virtual IP. See below on how to attach it to the loopback interface: lo:0.

Server and Network Preparation

We created the VM servers with the specifications in Table 3.3. The topology of our load balancer configuration is illustrated in Figure 3.2. It should be noted that our configuration is different from the configuration on the Ultra-Monkey project website because all public IPs are in the same subnet (130.203.136.x). This should not bring much difference but it should be noted that the gateway cannot be changed to the VIP as in the case of LVS-NAS. That will cause some accessibility problem.

By default, the IPv4 policy in Red Hat Enterprise Linux kernels disables support for IP forwarding. This prevents machines that run Red Hat Enterprise Linux (RHEL) from functioning as dedicated edge routers. To enable IP forwarding, use the following command:

```
$ sysctl -w net.ipv4.ip_forward=1
```

This configuration change is only valid for the current session; it does not persist beyond a reboot or network service restart. To permanently set IP forwarding, edit the `/etc/sysctl.conf` file as follows: Locate the following line:

```
net.ipv4.ip_forward = 1
```

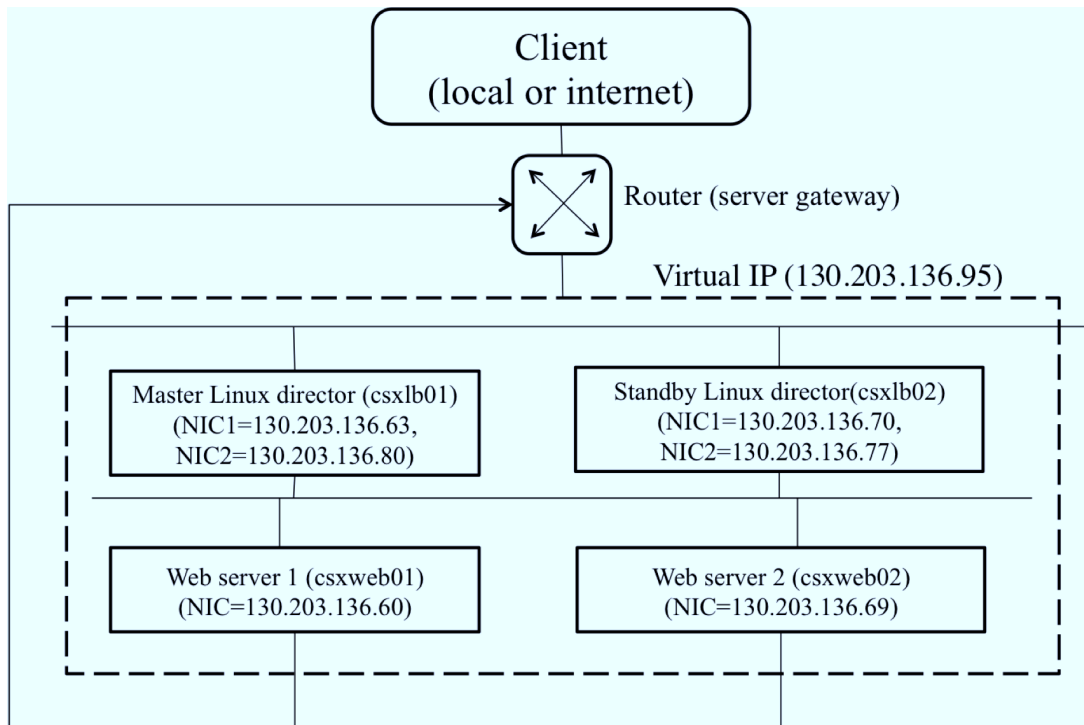


Figure 3.2 The topology of the high availability load balancing cluster. The two Linux directors have two NICs. NIC1 is used for processing incoming packets from internet and outgoing packets to real servers. NIC2 is used for multi-cast between two directors. Although csxlb01 is the default master director, csxlb02 can take the VIP and become the master when csxlb01 is down.

Use the following command to enable the change to the `sysctl.conf` file:

```
$ sysctl -p /etc/sysctl.conf
```

LVS Installation

Follow the procedures below to install the software packages.

csxlb01 and csxlb02 The following software packages need to be installed:

- heartbeat.x86_64
- heartbeat-devel.x86_64

- heartbeat-gui.x86_64
- heartbeat-lldirectord.x86_64

The other packages should be installed as dependences automatically.

1. To install the packages above, first install the EPEL repository


```
$ wget http://ftp.osuosl.org/pub/fedora-epel/5/x86_64/epel-release-5-4.noarch.rpm
$ sudo rpm -ivh epel-release-5-4.noarch.rpm
$ sudo yum update
```
2. You must install `ipvsadm` before installing `heartbeat-lldirectord`.


```
$ sudo rpm -ivh ipvsadm-1.24-13.el5.x86_64.rpm
```
3. Install `heartbeat` and `heartbeat-lldirectord`

```
$ sudo yum install heartbeat.x86_64 heartbeat-devel.x86_64
$ sudo yum install heartbeat-gui.x86_64 heartbeat-lldirectord.x86_64
```

After installation, the `/etc/ha.d/` should be automatically created. There are four configuration files, including `lldirectord.cf`, `ha.cf`, `/etc/ha.d/haresources` and `authkeys`.

`iptables` should be installed. The content of `/etc/sysconfig/iptables` can be copied directly from [proc0](#) with some modifications. The version on [csxlb01](#) is attached with this document. The `iptables` must allow all ssh, and TCP accesses (open port 22, 80, 8080, 443 and 8443).

[csxweb01](#) and [csxweb02](#) The software installations and configurations must be exactly the same on both of web servers. The following packages must be installed

- JDK1.5+. The `JAVA_HOME` environment variable must be set in the `.bashrc` file (source this file after update).


```
export JAVA_HOME='/usr/local/sbin/jdk1.7.0_10'
```
- Tomcat. The `CATALINA_HOME` variable should be set in the `.bashrc` file:


```
export CATALINA_HOME='/usr/local/tomcat-csx'
```

 For testing purposes, a default page can be created under `/usr/local/tomcat-csx/webapps/ROOT/healthcheck` with string `HEALTH CHECK PASSED`.

- *iptables*. The content of the *iptables* can be copied directly from serv1. A version that is currently in use on csxweb01 is attached with this document. The *iptables* must accept traffic to 80 (default HTTP port), 8080 (default Tomcat port), 443 (default HTTPS port), 8443 (default Tomcat HTTPS port). *iptables* should provide the port forwarding service in the “nat” chain using the following statement:

```
-A PREROUTING -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 8443
-A PREROUTING -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 8080
```

The first statement redirects incoming traffic from 443 (default HTTPS port) to 8443 (default HTTPS port for Tomcat) and from 80 (default TCP port) to 8080 (default Tomcat port).

```
$ service iptables restart
```

LVS Configuration

csxlb01 and csxlb02 The contents of all configuration files are shown below.

authkeys The *authkeys* is for two (or more) Linux directors to communicate and verify identities of each other. The first line provides the method-id. The second line list the method and key that go with the method-id. You normally only have one authentication method-id listed in this file. “sha1” is believed to be the “best”. The mod of this file must be set to 600.

```
auth 2
2 sha1 ultramonkey
```

ha.cf The *ha.cf* tells *heartbeat* which servers are within the Linux director clusters and how to communicate between them.

```
logfacility local0
mcast eth0 255.0.0.1 694 1 0
mcast eth1 255.0.0.1 694 1 0
auto_failback off
node csxlb01.ist.psu.edu
node csxlb02.ist.psu.edu
ping 130.203.136.254
respawn hacluster /usr/lib64/heartbeat/ipfail
apiauth ipfail gid=haclient uid=hacluster
```

- *mcast*. This is used to keep synchronization signal between the active and standby Linux directors. Refer to the original file comments for details of parameters.

- *auto_failback*. This parameter determines if a resource will automatically fail back to its “primary” node, or remain on whatever node is serving it until that node fails, or an administrator intervenes.
- *node*. Node names must be the same as the output of `uname -n`.
- *respawn*. We must make sure that the `/usr/lib64/heartbeat/ipfail` exists. In a 32-bit system, it may be located under `/usr/lib/heartbeat/`.

ldirectord.cf This file tells *ldirectord* the IPs of real servers and allowable connections for healthy check and TCP requests. Indents from the “fallback” line are necessary.

```
checktimeout=10
checkinterval=2
autoreload=no
logfile='local0'
quiescent=yes
virtual=1
    fallback=127.0.0.1:80
    real=130.203.136.60:0 gate
    real=130.203.136.69:0 gate
    service=http
    request='healthcheck'
    receive='HEALTH CHECK PASSED'
    scheduler=lc
    persistent=60
    checktype=negotiate
```

- *autoreload*. Automatically load this configuration file to make any changes effective.
- *quiescent*. When a real server is to be removed, its weight is set to zero but it remains part of the virtual service. Existing connections to the real server may continue, but no new connections will be allocated. This is particularly useful for gracefully taking real servers offline. Setting **quiescent=no** changes this behavior by removing the real server from the virtual service.
- *virtual*. Must be the firewall-mark value (in our case, it is an integer “1”), consistent with *iptables*. All incoming traffic are marked with this integer regardless of which port is used. Alternatively, the value can be `130.203.136.95:80` if FWM is not used. In this case, it only distributes traffic to the VIP and port 80.
- *fallback*. Local HTTP service is provided if none of web servers are available.

- *real*. Real server IPs and ports. “0” means all ports. “gate” means through the default gateway.
- *scheduler*. “lc” means “least connection”. Other algorithms are available, such as “wlc” and “rr”.
- *persistent*. Persistent to a single web server in a single session.

haresources This file tells *heartbeat* the service resources to start and the virtual IP (VIP) to bring up. The indents are necessary.

```
csxlb01.ist.psu.edu \
    LVSSyncDaemonSwap::master \
    ldirectord::ldirectord.cf \
    IPAddr2::130.203.136.95/24/eth0
```

- *csxlb01.ist.psu.edu*. Change node name to *csxlb02.ist.psu.edu* on csxlb02.
- *IPAddr2*. *IPAddr* uses the `ifconfig` command to create an interface, and *IPAddr2* uses the `ip` command.

Make sure that *ldirectord* does not startup on reboot and *heartbeat* starts up because *heartbeat* will startup *ldirectord*.

```
$ chkconfig --level 2345 heartbeat on
$ chkconfig --del ldirectord
```

To setup the fallback page on csxlb01 and csxlb02, we create an `index.html` under the HTTPD default web folder. In our case, it is under `/data/websites/`. The page should tell users that the system is not available now. In our case, we just prints a string like

```
THIS IS THE FALLBACK PAGE ON csxlb01
```

The *iptables* should be updated to set forward-mark to 1 under the “mangle” chain:

```
-A PREROUTING -d 130.203.136.95 -p tcp -j MARK --set-mark 0x1
```

Note that because we are not masquerading, do not put anything under the “nat” chain. As long as packets belong to a connection are marked with the FWM (in this case “1”, as per `--set-mark 1`), that corresponds to an LVS virtual service, the connection will be load balanced.

csxweb01 and csxweb02 The real-servers needs to be configured to see traffic for the VIP address as local. This can be done by making use of an IP alias on the

loopback device. But before doing this, ARP advertisements need to be restricted so that the real servers do not advertise the VIP, which should only be advertised by the active Linux director.

```
# Enable configuration of arp_ignore option
$ echo 1 > /proc/sys/net/ipv4/conf/all/arp_ignore

# When an arp request is received on eth0, only respond if that address is configured
# on eth0. In particular, do not respond if the address is configured on lo
$ echo 1 > /proc/sys/net/ipv4/conf/eth0/arp_ignore

# Enable configuration of arp_announce option
$ echo 2 > /proc/sys/net/ipv4/conf/all/arp_announce

# Be announced only by the Linux director
$ echo 2 > /proc/sys/net/ipv4/conf/eth0/arp_announce
```

Since that ARP advertisement of the VIP have been controlled, the VIP can be added as an alias to the loopback interface to allow the real server to accept packets for the virtual service. Create the file `/etc/sysconfig/network-script/ifcfg-lo:0` and add the following lines

```
DEVICE=lo:0
IPADDR=130.203.136.95
NETMASK=255.255.255.255
NETWORK=130.203.0.0
BROADCAST=130.203.0.255
ONBOOT=yes
NAME=loopback
```

To bring up the IP alias, the `ifup` command is used:

```
$ ifup lo
```

To verify this, use the following command

```
$ ip addr sh lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
inet 130.203.136.95/32 brd 130.203.0.255 scope global lo:0
```

Note that a netmask of 255.255.255.255 on the `lo:0` interface indicates that this interface only accepts traffic from 130.203.136.95.

The real servers need to be set up so that their default route is set to the gateway router's address on the server network and not an address on one of the

Linux directors. To do this, change file
`/etc/sysconfig/network-scripts/ifcfg-eth0`

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
HWADDR=00:50:56:ac:35:6c
IPADDR=130.203.136.60
NETMASK=255.255.255.0
NETWORK=130.203.136.0
```

The `/etc/sysconfig/network` file may not be changed, but for reference, we print its content below

```
NETWORKING_IPV6=no
HOSTNAME=csxweb01.ist.psu.edu
NETWORKING=yes
GATEWAY=130.203.136.1
```

Restart the networking

```
$ /etc/init.d/network restart
```

To verify the gateway, use the following command

```
$ ip route show 0/0
```

```
default via 130.203.136.1 dev eth0
```

Or just the `route` command to display the routing table.

If you just need to use an HTTP service, the settings above should be sufficient, but if you need to use HTTPS connection (for CiteSeerX), you need an HTTPS certificate on web servers. This has to be done after CiteSeerX is installed (see Section ??). A piece of code need to be copied to `/usr/local/tomcat-csx/conf/server.xml`.

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443
This connector uses the JSSE configuration, when using APR, the
connector should be using the OpenSSL style configuration
described in the APR documentation -->
<Connector port="8443" minSpareThreads="5" maxSpareThreads="75"
enableLookups="true" disableUploadTimeout="true"
acceptCount="100" maxThreads="200"
scheme="https" secure="true" SSLEnabled="true"
keyStoreType="pkcs12"
keyStoreFile="/key/store/file" keyStorePass="passcode"
clientAuth="false" sslProtocol="TLS"/>
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

The blue text block should be copied and pasted into `server.xml` above the red text block. For security reasons, we do not public the values of `keystoreFile` and `keystorePass`. Do this for both of the web servers.

Make sure that *openssl* is installed and then create the folder `/etc/ssl/`. Copy the file `csxcert.p12` under that folder. To make it effective, restart Tomcat. To test it, after starting CiteSeerX, click “Login” on the CiteSeerX front page. If you do not have the certificate, you will get an error message. After copying the certificate, you should be able to see the login page. If the database is configured, you should be able to login using your login credential.

Start Load Balancers

First, start up Tomcat on `csxweb01` and `csxweb02`. This can be tested locally by running

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

or from client:

```
$ elinks http://csxweb01.ist.psu.edu
```

Note that *iptables* will do port forwarding in the *PREROUTING* chain. Localhost requests are after *PREROUTING* so you must access through port 8080.

Startup the *heartbeat* on the master load balancer (`csxlb01` in our case).

```
$ service heartbeat start
```

```
$ service heartbeat status
```

```
heartbeat OK [pid 29479 et al] is running on csxlb01.ist.psu.edu [csxlb01.ist.psu.edu]...
```

This should bring up *ldirectord*. To check it, use

```
$ /etc/ha.d/resource.d/ldirectord /etc/ha.d/ldirectord.cf status
```

```
ldirectord for /etc/ha.d/ldirectord.cf is running with pid: 29515
```

Note that `/etc/init.d/ldirectord` should be stopped:

```
$ service ldirectord status
```

```
ldirectord is stopped for /etc/ha.d/ldirectord.cf
```

You should be able to see the output of “*ipvsadm*”

```
$ ipvsadm -L -n
```

```
->RemoteAddress:Port Forward Weight ActiveConn InActConn
FWM 1 lc persistent 60
->130.203.136.60:0 Route 1 0 0
->130.203.136.69:0 Route 1 0 0
```

If both of web servers are down, the local loopback is displayed in the output.

```

->RemoteAddress:Port Forward Weight ActiveConn InActConn
FWM 1 lc persistent 60
->130.203.136.60:0 Route 0 0 0
->130.203.136.69:0 Route 0 0 0
->127.0.0.1:80 Local 1 0 0

```

In addition, the current Linux director should be the master load balancer. This can be checked by

```

$ /etc/ha.d/resource.d/LVSSyncDaemonSwap master status
master running.

```

The IP configuration can be checked using

```

$ ip addr sh eth0

```

An example output is

```

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen
1000
link/ether 00:50:56:ac:65:7a brd ff:ff:ff:ff:ff:ff
inet 130.203.136.70/24 brd 130.203.136.255 scope global eth0
inet 130.203.136.95/24 brd 130.203.136.255 scope global secondary eth0

```

The red line indicates that the VIP is attached to eth0.

After csxlb01 is started, you can start csxlb02 in the same manner.

```

$ service heartbeat start

```

If *ldirectord* is not started, it can be started using

```

$ /etc/ha.d/resource.d/ldirectord /etc/ha.d/ldirectord.cf start

```

You can verify that csxlb02 is not the master.

```

$ /etc/ha.d/resource.d/LVSSyncDaemonSwap master status
master stopped.

```

At this point, the load balancer and the high availability cluster has been setup. To check the high availability really works, stop *ldirectord* on csxlb01

```

$ service heartbeat stop

```

After a while, csxlb02 should take the VIP and become the master Linux director. However, after starting *heartbeat* on csxlb01, csxlb02 is still the master (because *auto_failback* is set to off in *ha.cf*).

3.2.2 Repository Servers

The repository servers include *csxrepository01* and *csxrepository02*. For a testing purpose, we decide to use different configurations to these two servers. For *csxrepos-*

itory01, we use Pradeep's API and for *csxrepository02*, we use GFS2 to mount the server to other servers to implement the document access.

Configure *csxrepository01*

This VM is allocated 10TB of space. Before formatting the disk, we cannot see anything under this directory (using `$ df -h` command). First, we need to make a file system in a *ext4* format. The command is `$ sudo mke2fs -t ext4 /dev/mapper/vg_data-lv_data01`

The `-t` specifies the file system type. The device part of this command can be obtained by checking the result of

```
$ sudo fdisk -l
```

You should find the one with about 10TB space, like the one below:

```
$ Disk /dev/mapper/vg_data-lv_data01: 10995.1 GB, 10995095306240 bytes
```

Then, we create a `/data` directory and mount this device to it:

```
$ sudo mkdir /data
```

```
$ sudo mount -t ext4 /dev/mapper/vg_data-lv_data01 /data/
```

Configure *csxrepository02*

This VM is allocated 10TB of space. Before formatting the disk, we cannot see anything under this directory (using `$ df -h` command). In order to mount this storage to other servers using GNBD, we must format it to GFS2, install GNBD and then mount the GNBD device to the designated directory. It should be noted that the GNBD model is tested and implemented to only RHEL 5.8 (or CentOS 5.8), it should work in RHEL 5.x but it *does not* work for RHEL 6.x. The steps for our server configurations are described below. For a different CPU architecture or Linux release, the specific packages may vary but it should be easy to find the right ones from the same or similar places. For convenience, we save the downloaded packages under the `supplement/` folder associated with this manual.

1. Install the *rpmforge* packages to get the necessary

2. Get the Linux *rpmforge* packages:

```
$ wget http://packages.sw.be/rpmforge-release/rpmforge-release-0.5.2-2.el5.rf.x86
```

3. Install the DAG' GPG Key:

```
$ rpm --import http://apt.sw.be/RPM-GPG-KEY.dag.txt
```

4. Verify the package you have downloaded:

```
$ rpm -K rpmforge-release-0.5.2-2.el5.rf.*.rpm
```

5. Install the package

3.2.3 Database Servers

The database servers are `csxdb01` and `csxdb02`. `csxdb01` is the *master* server and `csxdb02` is the *slave* server. After setting up `csxdb01` using the instructions in § 3.1.3, the next step is to setup `csxdb02` as a replication by following the procedures below.

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
```

A complete example of `my.cnf`, which is used by `taos` can be found in Appendix ??.

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.

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

³If `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`;

3. Connect to the master server and determine its current replication coordinate by executing the `SHOW MASTER STATUS`:

```
FLUSH TABLES WITH READ LOCK;  
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. The command we used to backup all the databases on `taos` on October, 13, 2012 can be found in `supplement/csxdatabasebackup.sh`. See § 4.4.1 for a details explanation of the command options.
5. Connect to the slave and use `CHANGE MASTER TO` to configure it with the parameters for connecting to the master server and the initial replication coordinates:

```
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:

```
START SLAVE;
```

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

```
SHOW SLAVE STATUS;
```

statement on this 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 and memory. This section talks about the procedures to install the text extraction codes on the paper extraction server installed with RHEL6.4. We used to install it on RHEL5.8.

1. Make sure that the web server is up on the crawler machine and the SYZ crawler is installed, even if you are not running any crawling jobs. This is because an API is required to transfer documents from the crawler repository to the text extraction machine. In the following context, we assume the repository is located at `/data/crawldata` on the crawl server.
2. On the text extraction server, create a user `citeseerx` and a group `citeseerx` but do not issue any password to this user.

```
$ sudo groupadd citeseerx
$ sudo useradd -G citeseerx citeseerx
```

3. Copy the folder `importer/` to the home directory of `citeseerx`. The `importer` contains the following sub-directories: `bin/`, `converters/`, `crfpp/`, `lib/`, `resources/`, `svm-light/`, and `tmp/` (create it if it does not exists). The `bin/` contains the perl scripts used to run the extraction; the `converters/` contains the actual software tool that does the initial text extraction job. For us, we use `TET-3.0-Linux`⁴; the `crfpp/` contains the SVM training data and binary code; the `lib/` folder contains the head parsers, document filters and `ParsCit` tools⁵; the `resources/` contains databases and models. This folder should be copied under the home directory of `citeseerx`
4. 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.

⁴The `licensekeys.txt` file is empty. We are not allowed to distribute the licence we bought from PDF Lib. Users should make their own purchasers or use PDFBox, which is an open source text extractor.

⁵The version of `ParsCit` may not be update to date but it is tested to work with our current codes. Users can try recent versions of `ParsCit`.

5. Install perl packages to ActivePerl library.

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

6. 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 be 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 permission⁶. 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.

7. 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
```

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
```

8. Now, you should be able to run `processRemotes.pl` (See Section 4.2 for instructions on configuring this perl script before running it.). This package should print the total number of documents retrieved from the crawler web server, and the following messages:

⁶If `/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) 130.203.136.0/24(rw,sync,no_root_squash,insecure)`

```

COPY [OK] COPY MET [OK] PDFlib TET: PDFlib Text Extraction Toolkit,
3.0p1 (c) 2002-2009 PDFlib GmbH www.pdflib.com sales@pdflib.com CONVERT
[OK] FILTER [OK]

```

All .pdf and .met files should be copied without any problem. If text cannot be extracted, such as a scanned pdf file, you will not see CONVERT [OK]. If documents are not identified as an academic document, you will not see the FILTER [OK] message.

You must install TET and purchase its license to make it work like above.

3.2.5 Ingestion Machine

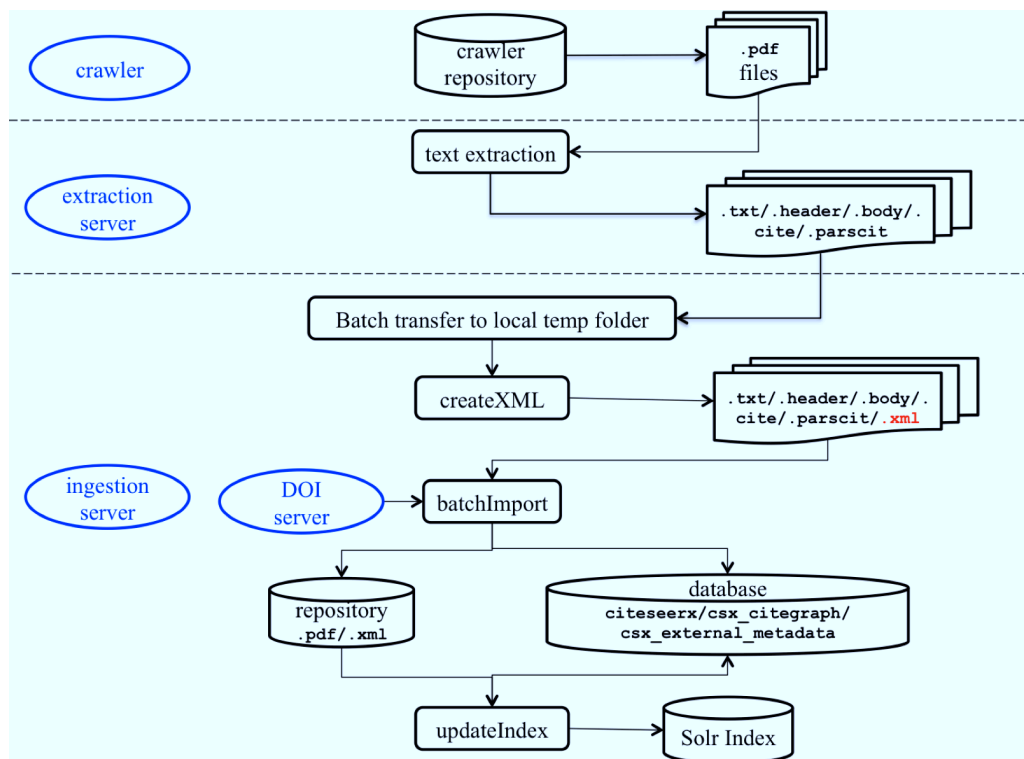


Figure 3.3 The text extraction and ingestion processes.

The ingestion system is currently installed on the repository server⁷, so the ingestion server is essentially the same as the repository server. As illustrated in

⁷This is the production repository server, which is different from the crawl repository server.

Figure 3.3, 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.
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 ??).
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 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
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.csxmetadata.driverClassName=com.mysql.jdbc.Driver
jdbc.csxmetadata.url=jdbc:mysql://dbserver/csx_external_metadata?useUnicode=true&characterEncoding=UTF
jdbc.csxmetadata.username=dbuser
jdbc.csxmetadata.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 ?? 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 ??), 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`. See Section ?? for instructions on configuring this file before actually running it.

3.2.6 Web Crawler Machine

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. In this section, we focus on the crawler designed by a formal graduate student, Shuyi Zheng. This crawler (hereafter SYZ) is written in Python. We also designed a CDI middleware for importing documents to the crawl database and repository from multiple contents sources (Heritrix, FTP, direct downloads etc).

The SYZ crawler has two major units: the *crawler*, which executes the downloading jobs and *csxbot*, which pipes the crawled content to CiteSeerX and does some work for the crawler web, such as handling URL submission and providing documents to the ingestion process. For CiteSeerX purposes, these two units must be both installed. But if you just want to download contents from a number of URLs as mirrors, you do not need the *csxbot* module.

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 resources and affect the performance of the web crawler machine, e.g., when updating the crawler history. Because the crawler web server relies on the *csxbot* and *crawler* units, both of them must be installed, but it is not necessary to run the SYZ crawler. The crawler web server was deployed on a server with RHEL5.9 installed. We have not perform a test installation on a higher version of RHEL. In the following contest, *csxbot_code_base* is set to be the top level folder after decompressing the crawler tarball.

The following packages are prerequisites.

1. **apache**. The HTTP server, both `httpd` and `httpd-devel` should be installed.

2. **mod_wsgi 2.6.**
3. **Python2.4+.** Python2.4 is the default Python with RHEL5. Installing multiple Python versions may result in system instability.
4. **django v1.1.1+.** We have tested that it works for Django 1.3.

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 SYZ crawler.** See Section ??.
3. **Install the csxbot.** Before installing it, make sure that the database connection for django project *citeseerx.crawl* is configured in *csxbot-0.3/citeseerx.crawl*. See Section ?? for details. You should also make sure that you can connect to the your (local or remote) crawl database.
4. **Deploy web files.** Just copy the web contents to the Apache home directory.

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

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
<code>.pdf</code>	Original PDF file copied from crawler repository
<code>.txt</code>	Text extracted from the PDF file
<code>.body</code>	Text body only (no bibliography)
<code>.cite</code>	Citation (bibliography) text
<code>.file</code>	XML file containing text converter (e.g., PDF Lib, TET), file type (e.g., PDF) and SHA1
<code>.header</code>	XML file containing header parser, title, author names, addresses and the abstract
<code>.met</code>	XML file containing crawled date, original URL, parent URL and SHA1 (copied from crawler repository)
<code>.parscit</code>	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 and Restoration

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 losing 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/csxdbackup.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.

Appendix A

Configuration Examples

A.1 Databases

A.1.1 my.cnf for MySQL Database Server

config/database/master/my.cnf

```
1 [mysqld]
2 datadir      = /data/mysql
3 socket       = /var/lib/mysql/mysql.sock
4 # Default to using old password format for compatibility with mysql 3.x
5 # clients (those using the mysqlclient10 compatibility package).
6 old_passwords = 1
7
8 log-bin      = /data/mysql/mysql-bin
9 server-id    = 5
10 # The statement below enables multiple tablespaces.
11 # With multiple tablespaces enabled, InnoDB stores each newly created table
12 # into its own tbl_name.ibd file in the database directory where the table
13 # belongs. This is similar to what the MyISAM storage engine does, but MyISAM
14 # divides the table into a tbl_name.MYD data file and an tbl_name.MYI index
15 # file. For InnoDB, the data and the indexes are stored together in the .ibd
16 # file. The tbl_name.frm file is still created as usual.
17 innodb_file_per_table
18 # The next statement let the log buffer be written out to the log file at each
19 # transaction commit and the flush to disk operation is performed on the log
20 # file. innodb_flush_log_at_trx_commit has three possible values.
21 innodb_flush_log_at_trx_commit = 1
22 # The paths to individual data files and their sizes. The full directory path
23 # to each data file is formed by concatenating innodb_data_home_dir to each
24 # path specified here.
25 innodb_data_home_dir    = /data/mysql/
26 innodb_data_file_path    = ibdata1:50M:autoextend
27 # The directory path to the InnoDB log files. If you do not specify any InnoDB
28 # log variables, the default is to create two 5MB files names ib_logfile0 and
29 # ib_logfile1 in the MySQL data directory.
30 innodb_log_group_home_dir = /data/mysql/
31 # The next statement makes the MySQL server synchronizes its binary log to disk
```

```

32 # after every "1" writes to the binary log. There is one write to the binary log
33 # per statement if autocommit is enabled, and one write per transaction
34 # otherwise. The default value of sync_binlog is 0, which does no synchronizing
35 # to disk. A value of 1 is the safest choice because in the event of a crash you
36 # lose at most one statement or transaction from the binary log. It is also the
37 # slowest.
38 sync_binlog      = 1
39 # You can set ..buffer_pool_size up to 50 - 80 %
40 # of RAM but beware of setting memory usage too high
41 # Set ..log_file_size to 25 % of buffer pool size
42 # "innodb_buffer_pool_size" - The size in bytes of the memory buffer InnoDB
43 # uses to cache data and indexes of its tables. The default value is 8MB. The
44 # larger you set this value, the less disk I/O is needed to access data in
45 # tables. On a dedicated database server, you may set this to up to 80% of the
46 # machine physical memory size.
47 innodb_buffer_pool_size = 8G
48 # "innodb_additional_mem_pool_size" - The size in bytes of a memory pool InnoDB
49 # uses to store data dictionary information and other internal data structures.
50 # The more tables you have in your application, the more memory you need to
51 # allocate here. Default is 1MB.
52 innodb_additional_mem_pool_size = 20M
53 # "innodb_log_file_size" - The size in bytes of each log file in a log group.
54 # The combined size of log files must be less than 4GB. The default value is
55 # 5MB. Sensible values range from 1MB to 1/N-th of the size of the buffer pool,
56 # where N is the number of log files in the group. The larger the value, the
57 # less checkpoint flush # activity is needed in the buffer pool, saving
58 # disk I/O.
59 innodb_log_file_size = 256M
60 # The size in bytes of the buffer that InnoDB uses to write to the log files on
61 # disk. The default value is 1MB. Sensible values range from 1MB to 8MB
62 innodb_log_buffer_size = 4M
63
64 #transaction-isolation=READ-COMMITTED
65
66
67 [mysql.server]
68 user      = mysql
69 basedir   = /var/lib
70
71 [mysqld_safe]
72 log-error = /var/log/mysqld.log
73 pid-file  = /var/run/mysqld/mysqld.pid

```

Bibliography