Scalability Bottlenecks of the CiteSeerX Digial Library Search Engine

Jian Wu[†], Pradeep B. Teregowda[‡], Madian Khabsa[‡], Eric Treece[†], Douglas Jordan[‡], Stephen Carman[†], Prasenjit Mitra[†] and C. Lee Giles[†‡]
[†]Information Sciences and Technology
[‡]Department of Computer Science and Engineering
University Park, PA, 16802, USA
jxw394@ist.psu.edu

ABSTRACT

As the document collection and user population increase, the capability and performance of a digital library such as CiteSeerX maybe limited by some bottlenecks. This paper describes the current infrastructure of the CiteSeerX academic digital library search engine, outlines its current bottlenecks and proposes feasible solutions. These bottlenecks exist in various components of the system including hardware, web crawling, text extraction and storage. The hardware bottleneck is the increasing difficulty to maintain a cluster consisting of almost twenty physical servers. The solution is to merge some servers and implement the whole system under a virtual architecture. The web crawling bottleneck is that the seed URLs are biased on on computer science, information sciences, technology and related fields. One of the approaches to balance the domain distribution of our crawl repository, is to obtain seed URLs from generic search engines. Another bottleneck is the average time to extract text from the crawled documents. To reduce the processing time, we have proposed a new extraction model using message queues and multiple threads. Preliminary experiments indicates that the average time to extract a document can be reduced by an order of magnitude. The storage bottleneck is that as the data repository size grows, a better tool is required to manage the storage, transferring, sharing and backing up of our files. Hadoop provides a promising tool to parallelize data analysis and the Hadoop File System provides a solution for shared storage. All solutions to the current bottlenecks are either under testing or on our roadmap. Our indexing protocol does not have foreseeable bottlenecks in the near future.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models; H.3.7 [**Digital Libraries**]: System issues

1. INTRODUCTION

CiteSeerX is a digital library search engine that provides free downloads of over two million documents. There are over 32 million citations (over 9 million unique citations) and about 5 million authors (over 1.6 million unique authors). Besides paper downloads, this search engine also keeps track of citation references for most of the papers. Based on these citation references, it has built a large citation network. The search engine is visited over two million times daily and most of them are downloading activities.

The infrastructure of the CiteSeerX system is shown in Figure 1. It contains the following components: the web crawler, the ingester, the database, the repository, the indexer and the web application. The infrastructure in Figure 1 is implemented on a cluster consisting of about twenty physical servers installed with Red Hat Enterprise Linux 5 (RHEL 5). The detailed description of the CiteSeerX can be seen in [7] and [11].

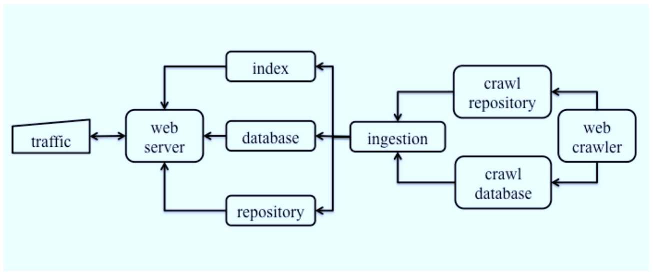

Figure 1: The infractructure of the CiteSeerX search engine.

CiteSeerX collects documents by automatically crawling the web. It has a focused web crawler which performs a scheduled batch crawl on a weekly basis. The crawler downloads between thousands and tens of thousands of documents each day. In addition, with the CDI middleware [13], we are able to crawl using a general web crawler such as Heritrix besides the original dedicated crawler written in Python (here after the SYZ crawler). The middleware also allows us to import documents downloaded by FTP in a batch mode. For example, the middleware is able to import 400,000 PDF documents in about 50 hours. Both the SYZ crawler and the CDI middleware contain four filters that only keep PDF, postscript and their zipped format documents.

Texts are extracted from downloaded documents to make them searchable. The extracted texts are then parsed and labeled based on a regular expression procedure. The titles, authors, text bodies and citations are extracted and saved as separate files. These files are compiled to a single XML file for indexing. We use Solr for indexing and MySQL as the database engine. The web application is deployed using Tomcat. We also have two load balancing servers to make the cluster highly available.

CiteSeerX has been growing steadily in the past years. The number of registered users grew from about 6,000 in 2008 to over 40,000 in 2012 (Figure 2). The number of crawled documents increased from less than two million to almost eight million in five years (Figure 3). The total number of documents has also increased from less than one million to over two million (Figure 4). The traffic is also growing in a modest rate. Figure 5 overlays the CiteSeerX traffic in 2011 and 2012. The figure shows that visits to our site have increased by 1.3% (6,994,697 vs. 6,904,715) while unique visitors have increased by 1.69% (4,897,548 vs. 4,816,145). With the growing number of documents, users and traffic, it is important to consider the scalability issues of our system, to ensure that it achieves comparable or better performance.

Solr is an enterprise level search and indexing platform that is built using Apache Lucene[1]. CiteSeerX uses Solr in a distributed cluster across several different servers. These servers replicate the index and balance the distribution of traffic to the index. The index is searched via RESTful requests made to the cluster, which returns JSON search results. This JSON response is then processed in Java servlets and presented to the user. Currently, the index is able to handle two million daily requests. Searching accounts for roughly 31% of overall CiteSeerX traffic. The index is required by about 620,000 per day or 7 times per second on average. This load is well below what the Solr is capable of handling.

In addition, Solr is equipped to have easily configured replication and distribution of cluster nodes. In order to scale the index further it would be logical to add more nodes to the index clusters. This scaling would be accomplished by adding a slave node into the Solr cluster, at which point the master index would replicate the index over to the slave node. However, since the index has not even began to slow down from traffic this it would be a while before this solution is necessary.

MySQL has been tested to work with large social network websites such as Facebook [4] with proper configuration. Currently, the size of the databases is about 50GB and is saved in the disk. The reading and writing can be sped up by using high speed storage devices such as solid state disk or memory disk. However, the database has not been a bottleneck for the searching performance overall performance. Tomcat is also a enterprise level software that scales well so it will not become CiteSeerX bottlenecks anytime soon. The major bottlenecks are investigated and evaluated in the sections below in terms of hardware and software.

[1]http://lucene.apache.org/

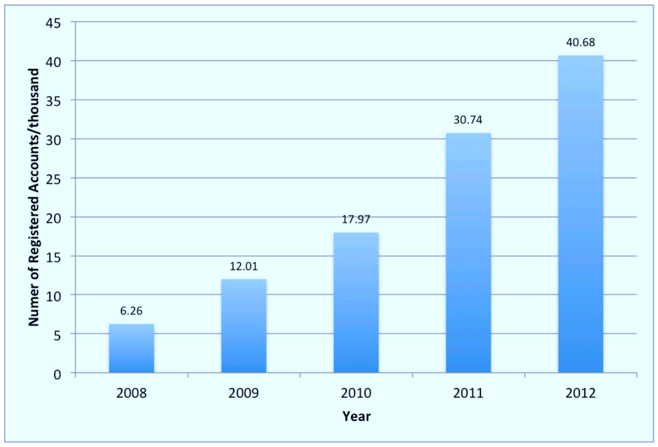

Figure 2: The number of registered accounts to CiteSeerX in the past five years. Each bar represents the number at the end of that year.

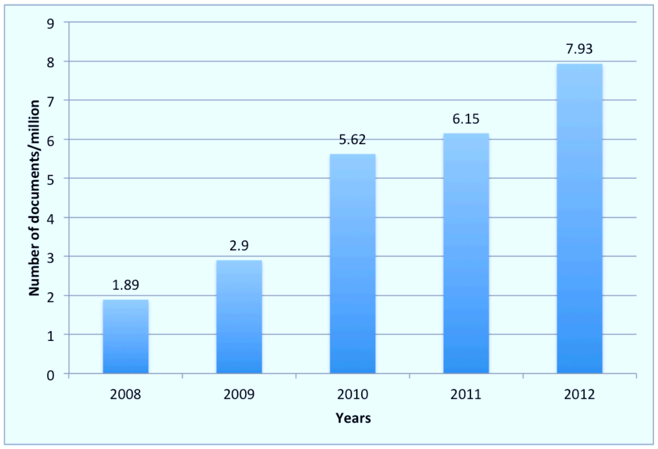

Figure 3: The number of crawled documents in CiteSeerX in the past five years. Each bar represents the number at the end of that year.

We do not see many publications talking about scalability of a digital library search engine. Some work has focused on one aspect such as the database [3] and another on search [8].

2. SCALABILITY BOTTLENECKS OF CiteSeerX

CiteSeerX has several scalability bottlenecks in both hardware and software. There are two problems in the hardware: occasional hardware failures and computational capability. The software bottlenecks lie in several aspects: web crawling, text extraction, and storage.

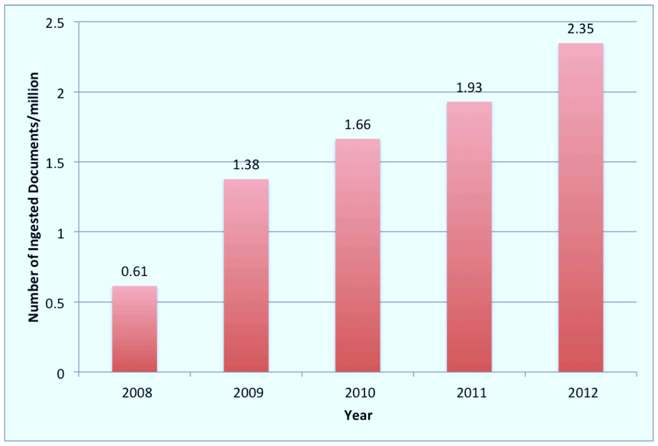

Figure 4: The number of ingested documents in CiteSeerX in the past five years. Each bar represents the number at the end of that year.

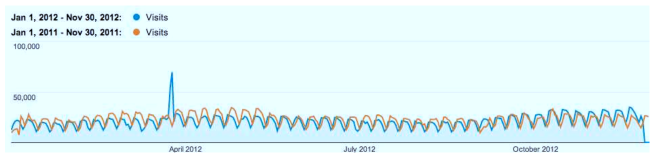

Figure 5: CiteSeerX traffic in 2012 (blue) overlaid with CiteSeerX traffic in 2011 (red).

2.1 Hardware Bottleneck

As CiteSeerX expands its service domains and enlarges its document collection, the current hardware becomes a bottleneck to limit CiteSeerX on document processing and storage. Currently, CiteSeerX applies a distributed cluster model in which different modules are located on a number of physical servers. The drawbacks of this model are two folds. On one hand, system maintenance takes a significant overhead, including replacing failed hard drives, RAID controllers, placing orders on new servers as well as re-installation at system failure. This overhead will increase as the system grows and may lead to higher cost in mechanical and software labor. With CiteSeerX, full time reliability of the system is required, while providing an architecture that can be expanded on an as needed. At times, some of the components may become unavailable, but the major parts of the system can still be accessed.

This can be achieved through the use of a virtualized architecture, which is composed of three levels (Figure 6): the storage level, followed by a processing level and finally an operating system/application level.

The storage level is composed of several servers whose sole purpose is to act as storage for the virtual machines. The processing level is composed of multiple big servers each of which contains more than ten cores of processors and up to 100GB memory. These servers are connected to the storage level. The system/application level consists of various of virtual machines running on the processing level while data and the virtual machine themselves are stored on the storage level. A hypervisor takes care of the interactions between the storage and the processing level.

The advantages of this architecture in regards to CiteSeerX are three folds. First, this increases the server reliability. If one processing server fails, the hypervisor can respond to this and move the virtual machine to another processing server. We have noticed minimal to no downtime in testing environments when this occurs. The second advantage is a smaller footprint in the datacenter which equates to less physical space used in racks as well as a lower operating temperature and thus more efficient use of electricity, such as reported in [6] and [5]. This smaller footprint allows us to add physical servers to our cluster should we need more processing power or storage. We then can move more mission critical VMs to the newly added physical servers while keeping the old servers for less critical works such as research or experiments. The third advantage is a reduction in time to setup a new server. CiteSeerX has been running on twenty or more physical servers in current and past incarnations. While this has been scalable, the scalability has been greatly limited by time as well as cost constraints. By using a template-based workflow in a virtualized architecture, setup time has been reduced from a day, not including the time for a vendor to deliver a system, to a matter of minutes. This allows CiteSeerX features to be added in a rapid fashion. As the limit of the chosen hypervisor allows a maximum of 2,000 physical hosts [12], the virtual machines at the processing level, a limit on scalability in the foreseeable future is not expected. Should CiteSeerX require a more processing or storage space in the future, physical servers can be added to the cluster with no downtime to the system.

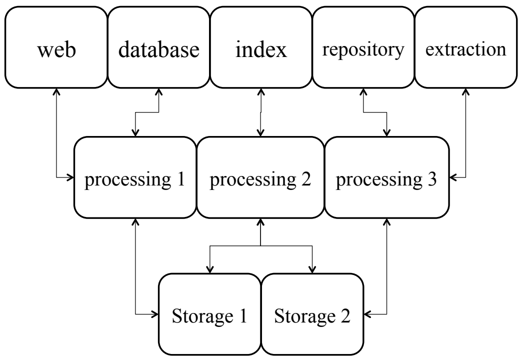

Figure 6: The structure of the three-level virtualized structure as a solution to the hardware bottleneck.

2.2 Web Crawling Bottleneck

CiteSeerX uses a dedicated multi-thread focused crawler, *citeseerxbot*, written in Python (the SYZ crawler). Besides components of a general web crawler, the SYZ crawler applies a series of filters, which are specially customized for the purpose of obtaining academic papers in PDF and postscript files. We apply a breadth-first crawling policy and crawl up to a depth of two (not including the seed level). The size of each downloaded document is limited to 100-megabytes, which covers almost all academic papers and the majority of books. If a document resource passes all filters, its URL information is saved to the document table in the crawl database. The document itself and the associated metadata files in XML format are saved to the crawl repository.

As argued by [14], providing high quality seeds is essential to focused crawlers, so we carefully select and rank seed URLs from our crawling history based on the document numbers and citation rates. Following the work in [14], we have constructed a revised whitelist which contains more than 120,000 seed URLs, selected from more than 700,000 candidate seed URLs. All URLs in the revised whitelist are verified to be alive as of April, 2012. The domains are not in our blacklist and are ranked by the number of "useful" documents linking to them. We classify a document to be useful if it is identified to be an academic document by the ingestion system.

However, due to the initial seed selection of the CiteSeerX crawler, most of our seed URLs are in computer science related domains. For example, homepages of faculties and researchers in computer or information science departments. We have a large document collection in chemistry, mathematical and physical subjects, but Medical sciences, economics and finance only take up a small portion. The domain bias of the seed URLs is a bottleneck of the CiteSeerX, which significantly limits the maximum number of documents we can crawl each day. The user population is also limited to computer and information related fields.

The key to remove this bias is to import more seed URLs in other domains. These seed URLs can be obtained by taking advantage of general search engine APIs, e.g., the Bing API. Academic documents crawled by these general search engines are not focused on particular fields, so their document collection provides is more complete. These document URLs can be obtained by searching a set of keywords in the fields that CiteSeerX is short of. Alternatively, we can just search stop words and the content type to obtain document URLs in random research subjects. Another way is to directly import documents to the crawler database by downloading FTP websites. For example, we have successfully imported over 400,000 PDF papers from PubMed using the CDI middleware. Removing the domain bias and setting up a complete document collection is essential to attract more users and increase the cite visibility of the site.

2.3 Extraction Bottleneck

A necessary step to make documents searchable is to extract text from the PDF/postscript files. This step is before the ingestion so a slow extraction can significantly lag the ingestion behind the crawling and becomes a bottleneck. The current extraction applies a batch processing model, in which the program first retrieves a list of un-ingested documents from the crawler database using an API. The actual crawled documents are then retrieved from the crawler repository in a shared storage. A text extractor is then used, e.g., PDFlib TET PDF IFilter, 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* tool. 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 academic related. Documents that pass this filter are then processed by multiple parsers to extract metadata, including titles, authors, citations and the text bodies.

The current metadata extraction system has several limitations. First, the document processing procedure is a pipeline operation, i.e., each sub-procedure operates in serial and the output of one stage becomes the input for the next sub-procedure, which leads to limited throughput. In Figure 7, we present a diagram which shows the processing time for a typical document. The total amount of time to extract text

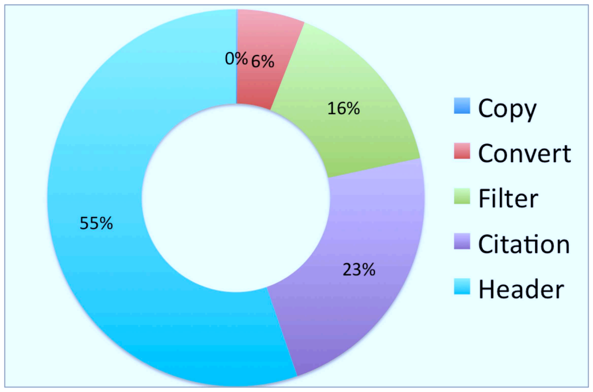

Figure 7: The time distribution of extracting a typical document.

from one PDF file is about six seconds, which means that it can extract about 14,400 documents per day. While this is comparable to the number of documents we crawl daily at present, it will lag behind our future crawling. In addition, the text extraction is a semi-automated batch process, so it requires a lot of manual operations to run commands. Due to the code complexity and limited threading support, it is not straightforward to parallelize these processes. Second, documents to be processed have to be local to the modules. These limitations imply that merely improving the computational resources such as the memory and processor speed does not significantly improve the throughput of the process.

To solve this issue, we designed a new document extractor by implementing two major improvements. First, the new extractor makes use of font and layout information embedded in PDF documents. The new extractor also utilizes a random forest classifier, which uses features generated from content, layout, font and domain information from dictionaries such as author names, title keywords, geographical entities, academic entities and stopwords. The new code achieves a comparison quality to the current code. Second, we develop a message oriented middleware for the extraction. Message oriented middleware offers several advantages in building distributed system involving heterogeneous components, networks and infrastructure. With the middleware, once the crawler fetches a document from the web, it posts a message to the extractor queue. This message points to the URL of the document. The extractor then consumes this message from the queue and process the document, generating an ingestable document on success. A message of completion is posted onto the ingestion queue, which is handled by the ingestion system. Many of the performance issues are related to I/O operations, shared objects and network issues, which are independent processes. This allows us to take advantage of parallelizing the application to boost performance. Figure 8 presents the average processing time for each document as the number of processing instance increases from 2 to 16, which indicates that the mean processing time is significantly reduced by applying multiple threads [10].

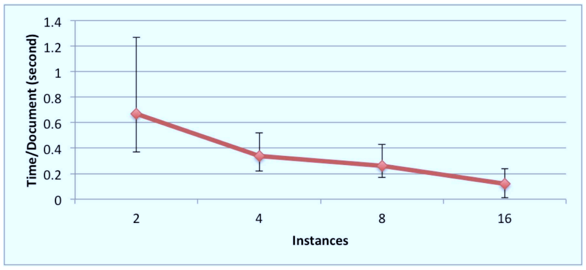

Figure 8: The average time used to process each document as a function of number of thread instances.

2.4 Storage Bottleneck

The current storage architecture of the system is based on having two repository servers, with one as a backup of another. The repositories are running on the Global File System (GFS) on top of the RHEL and are accessed for reads and writes. The read access happens when a document is requested for download where the web server accesses the files on GFS and streams it through the web. The writes happen when new documents are ingested or when users enter manual corrections to correct document metadata. When that happens, the updates are inserted into the database, and the system creates an XML file describing a new version and write it to the repository. While this file system works for the current production, we are expecting some potential issues that may affect the data service as the CiteSeerX document collection grows and/or the platform is upgraded.

One of the problems related to the file system in the current deployment is that the storage module occasionally causes the entire application to lock, slowing down CiteSeerX significantly. These locks happen when the system is trying to write an XML file triggered by a manual correction while an ingestion process is adding new documents. We have observed that traffic spikes tend to aggravate the locking problem when combined with ingestion and manual corrections. Our analysis and debugging traced the problem back to the file system. While we are coping with these problems for now as our repository size has just surpassed 2 million documents, this can not be a tolerable issue as the repository continues to grow.

There are several approaches to be pursued for handling the aforementioned scalability problems in the current repository architecture. A simple fix is to use a single large repository, deployed using RAID 5, which can scale to support all the concurrent reads and writes. The direct benefit is that it provides the sufficient bandwidth required by the digital library. Striping the data across many disks would also enhance the bandwidth. While this approach is the simplest to install and maintain, the cost does not scale linearly with the storage. In addition, it would be a single point of failure in case the RAID controller malfunctions, which we experienced on a couple of occasions.

Teregowda [10] proposed a web API to access the file system where read and write requests are handled by a web server, therefore shifting the synchronization responsibility to the API itself rather than the file system. The API would provide write and read operations to the caller, while it handles any possible synchronization issues. This approach would provide a layer of abstraction for the storage system, and allow using other file systems underneath without modifications to the code base. However, it requires a careful design and implementation at the API level to ensure that the synchronization problems do not merely travel from the file system level to the API level without solving the problem in hand.

Another promising approach is to use the Hadoop Distributed Filesystem (HDFS)[2] which uses commodity hardware to create multiple replicas of the same document on different machines [9]. The HDFS has been under active development in the open source community, as well as by many consulting companies that provide enterprise level support, i.e. *Cloudera*[3]. Under this approach, the application would deal with a single repository, which is on top of HDFS, and the reads and writes are handled by the filesystem itself. To the best of our knowledge, HDFS does not exhibit the locking behavior of GFS as the number of transactions increase, though we have not benchmarked it on our repository yet. The downside of using HDFS is the loss of physical storage to provide redundancy. Effectively, when HDFS uses 3 nodes replica, only one third of the storage is available to the application, and the other two thirds are used for replication.

Currently, we are implementing the API approach to provide scalable storage, while HDFS is our next step in the direction of future work.

3. DISCUSSION AND SUMMARY

In this paper, we reviewed the infrastructure of the CiteSeerX digital library search engine and addressed several bottlenecks which can potentially affect CiteSeerX performance as the numbers of documents and users grow. We propose solutions to overcome the bottlenecks. Some of the

[2]http://hadoop.apache.org/
[3]http://www.cloudera.com/

solutions are tested and advantages over the current model are reported. For example, to break the extraction bottleneck, we have proposed a new extraction system using more text features and a message queue. The text extraction is also parallelized to speed up the processing. The new extraction proposed out-performs the current one in terms of both quality and speed.

The proposed solutions to other bottlenecks are either under testing or on the roadmap and will be fully tested before putting it into production. For example, the hardware bottleneck can be solved by implementing the entire system to a virtual infrastructure instead of a physical cluster. Part of the CiteSeerX components have been successfully migrated to the VM infrastructure such as the web crawler. This solution can significantly reduces the server maintenance overhead, saves costs and make it easy to expand by adding new storage and processing servers. The crawler bottleneck is the seed URL domains. We need to obtain seed URLs from generic search engines to obtain a complete document set. The file system of the current storage system need to be upgraded and Hadoop file system is a promising solution.

As a traditional relational database management system, MySQL is still sufficient for our current scale of data. Although it has not become a major bottleneck for our search engine, it is shown to be a bottleneck for our crawl history analysis. A non-relational database solution, such as HBase, is considered to be a replacement candidate when the number of searchable papers grows by a factor of ten. HBase is part of the Apache Hadoop project, which mimics Google's Bigtable [2]. Once integrated with Hadoop, HBase has shown to a boost to large scale data analysis, which is already seen in Facebook's real-time messaging system [1]. We are investigating the feasibility to implement this database with other components of CiteSeerX.

4. ACKNOWLEDGMENTS

This work is financially supported by NSF grant 0000-0000.

5. REFERENCES

[1] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 1071–1080, New York, NY, USA, 2011. ACM.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[3] J. Chmura, N. Ratprasartporn, and G. Ozsoyoglu. Scalability of databases for digital libraries. In *Proceedings of the 8th international conference on Asian Digital Libraries: implementing strategies and sharing experiences*, ICADL'05, pages 435–445, Berlin, Heidelberg, 2005. Springer-Verlag.

[4] Facebook. Facebook at 13 million queries per second recommends: Minimize request variance. MySQL Tech Talk, November 2010.

[5] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.

[6] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 22:1–22:12, New York, NY, USA, 2011. ACM.

[7] H. Li, I. G. Councill, L. Bolelli, D. Zhou, Y. Song, W.-C. Lee, A. Sivasubramaniam, and C. L. Giles. Citeseerx: a scalable autonomous scientific digital library. In *Proceedings of the 1st international conference on Scalable information systems*, InfoScale '06, New York, NY, USA, 2006. ACM.

[8] W. Meng, Z. Wu, C. Yu, and Z. Li. A highly scalable and effective method for metasearch. *ACM Trans. Inf. Syst.*, 19(3):310–335, July 2001.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[10] P. B. Teregowda. *COMPUTATIONAL ISSUES IN DIGITAL LIBRARY SEARCH ENGINES*. PhD thesis, Pennsylvania State University, 2012.

[11] P. B. Teregowda, I. G. Councill, R. J. P. Fernández, M. Khabsa, S. Zheng, and C. L. Giles. Seersuite: developing a scalable and reliable application framework for building digital libraries by crawling the web. In *Proceedings of the 2010 USENIX conference on Web application development*, WebApps'10, pages 14–14, Berkeley, CA, USA, 2010. USENIX Association.

[12] VMware. *VMware vSphere 5.0 Configuration Maximums*. VMware, Inc., 3401 Hillview Ave., Palo Alto, CA 94304, 2011.

[13] J. Wu, P. Teregowda, M. Khabsa, S. Carman, D. Jordan, J. San Pedro Wandelmer, X. Lu, P. Mitra, and C. L. Giles. Web crawler middleware for search engine digital libraries: a case study for citeseerx. In *Proceedings of the twelfth international workshop on Web information and data management*, WIDM '12, pages 57–64, New York, NY, USA, 2012. ACM.

[14] J. Wu, P. Teregowda, J. P. F. Ramírez, P. Mitra, S. Zheng, and C. L. Giles. The evolution of a crawling strategy for an academic document search engine: whitelists and blacklists. In *Proceedings of the 3rd Annual ACM Web Science Conference*, WebSci '12, pages 340–343, New York, NY, USA, 2012. ACM.