The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

COMPUTATIONAL ISSUES IN DIGITAL LIBRARY SEARCH ENGINES

A Dissertation in

Computer Science and Engineering

by

Pradeep Teregowda

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2012

The dissertation of Pradeep Teregowda was read and approved* by the following:

C. Lee Giles
Professor of Information Sciences and Technology
Dissertation Adviser
Chair of Committee

Daniel Kifer
Assistant Professor of Computer Science and Engineering

Bhuvan Urgaonkar
Associate Professor of Computer Science and Engineering

Conrad S. Tucker
Assistant Professor of Industrial and Manufacturing Engineering

Krzysztof Janowicz
Assistant Professor of Geography

Raj Acharya
Head of the Department of Computer Science and Engineering

*Signatures on file in the Graduate School.

Abstract

The open source SeerSuite digital library search engine framework, has been utilized in building several information systems such as CiteSeerX, ChemXSeer and recommender systems. The framework, its components and data collected from its instances have been utilized by researchers around the world, in topics related to information systems, machine learning, software design and systems. We describe the architecture and workflow of SeerSuite. Several features of SeerSuite instances such as dynamic workloads imposed by web traffic and document acquisition systems, maintenance requirements make cloud computing an attractive option. We discuss in detail the economics and impact of hosting instances of SeerSuite in the cloud. The framework depends on a complex metadata extraction system, capable of extracting crucial entities such as author, title, citations and their contexts necessary for building citation graphs, to link and rank documents in the collection. However, the current metadata extractor, suffers several limitations as a result of dependencies in the program, code complexity, lack of parallelization and a need for specialized infrastructure in the form of large scale shared storage. These limitations prevent the SeerSuite framework from scaling to supporting large collections. In this dissertation, we discuss the design and implementation of a metadata extraction system. Specifically, we present details of a scalable and portable system built using message oriented middleware architecture with a publish/subscribe approach and can be deployed across different physical and cloud infrastructure. Experimental results indicate the throughput of the extraction system can be increased by order of several factors. A discussion of lessons learned from our experiences in building and deploying the metadata extraction system, especially those related to scaling, reliability and cloud deployments is provided.

Table of Contents

List of Tables

List of Figures

Acknowledgments

Foremost, I would like to thank my advisor Lee Giles, for motivating me, guiding me, with his suggestions and for the many lunches he shared with us. I would like to thank my committee members for taking time to discuss and guide me in my research. I would like to thank members of our research group past and present, particularly Pucktada Treeratpituk, Sujatha Das, Jian Wu, Madian Khabsa, Neela Sawant, Isaac Councill and Yves Petinot for all the mentoring, suggestions, support and guidance they have offered me during my research. I would like to thank my parents, my wife, my son and my sisters for their invaluable patience and support. Finally I would like to thank all my teachers and staff, whose direction and instruction have enabled my work.

There's only one corner of the universe you can be certain of improving, and that's your own self. –Aldous Huxley

Chapter 1

Introduction

This dissertation we describe SeerSuite, its instance CiteSeer^x and through lessons learned from the deployment of CiteSeer^x, limitations and motivations for refactoring components of SeerSuite to improve the performance, scalability and extend its features. In this introduction, a background into digital libraries, relevant to our work is provided, followed with broad overviews of each of the topics discussed in individual chapters.

1.1 Digital Libraries

Efficient and reliable access to the vast scientific and scholarly publications on the web requires advanced citation index retrieval systems (18) such as CiteSeer (20), CiteSeer^x [1], Google Scholar [2], ACM Portal [3], etc.

Domain specific repositories and digital library systems have been very popular over the last few decades, One of the earliest was, the Greenstone digital library [4] which has been followed with arXiv (21) for physics and RePEc [5] for economics. These repository systems, unlike CiteSeer^x, depend on manual submission of document metadata, a tedious and resource expensive process. As such CiteSeer^x which crawls authors web pages for documents is in many ways a unique system, closest in design to Google Scholar.

1.1.1 SeerSuite

The SeerSuite application framework provides an unique automatic and advanced citation indexing system that is usable and comprehensive, and provides efficient access to scientific publications. Several instances of SeerSuite such as CiteSeer^x, Chem$_X$Seer and data generated by these instances are being utilized by users and researchers (35; 16; 15).

1.1.1.1 Design of SeerSuite

The popularity of services provided by the original CiteSeer and limitations in its design and architecture were the motivation behind the design of SeerSuite (9; 38). The design of SeerSuite was an incremental process involving users and researchers. User input was received in the form of feedback and feature requests. Exchange of

[1] `http://citeseerx.ist.psu.edu`
[2] `http://scholar.google.com`
[3] `http://portal.acm.org/portal.cfm`
[4] `http://www.greenstone.org`
[5] `http://repec.org`

ideas between researchers and users across the world through collaborations, reviews and discussions have made a significant contribution to the design. In addition to ensuring reliable scalable services, portability of the overall system and components was identified as an essential feature that would encourage adoption of SeerSuite elsewhere. During the process of designing the architecture of SeerSuite, other academic repository content management system (CMS) architectures such as Fedora (34) and DSpace [6] were studied. The main obstacles in adopting existing repository and CMS systems were the levels of customization and the effort required to meet SeerSuite design requirements. More specifically, implementation of the citation graph structure with a focus on automatic metadata extraction and workflow requirements for maintaining and updating citation graph structures made these approaches cumbersome to use.

A previous implementation, CiteSeer, was designed to support such services. However, CiteSeer was a research prototype and, as such, suffered serious limitations. These limitations included the ability to support large scale collections, complex code base and strong library dependencies. SeerSuite was designed to provide a framework that would replace CiteSeer, to provide most of its functionality, but designed to be extensible. SeerSuite improves on several aspects of the original CiteSeer with features such as reliability in operation and availability, robustness with its ability to support failures in individual modules and scalability. It does this by adopting a multi-tier architecture with a service orientation and a loose coupling of modules.

1.1.2 CiteSeer^x : An Instance of SeerSuite

CiteSeer^x, an instance of SeerSuite is one of the top ranked resources on the web and indexes over two million documents [7]. The collection spans computer and information science (CIS) and related areas such as mathematics, physics and statistics. CiteSeer^x acquires its documents primarily by automatically crawling authors web sites for academic and research documents related to computer science, information science and related areas using an extensive seedlist. CiteSeer^x daily receives approximately two million hits and has more than two hundred thousand documents downloaded from its cache. The personalization module of SeerSuite/CiteSeer^x, MyCiteSeer supports over ten thousand registered users.

While the SeerSuite application framework provides most of the functionality of CiteSeer, SeerSuite represents a complete redesign of CiteSeer. SeerSuite takes advantage of and includes state of the art machine learning methods such as support vector machines, conditional random fields and uses open source applications such as Solr, the Spring framework and libraries. With CiteSeer^x, SeerSuite focuses primarily on CIS. There are requests for similar focused services as in (43), components of SeerSuite have beein utilized in Chem$_X$Seer [8], a digital library search engine and collaboration service for chemistry and in ArchSeer, archaeology discussed in (50).

[6] `http://www.dspace.org`
[7] `http://citeseerx.ist.psu.edu`
[8] `http://chemxseer.ist.psu.edu`

One of the major aims of developing SeerSuite is to encourage widespread usage and application deployment. The use of customized hardware solutions and or expensive subsystems are not desirable. The design and architecture of SeerSuite is designed to enhance the ease of maintenance and to reduce the cost of operations achieved by extensive use of service oriented architecture. Thus, deploying instances of SeerSuite, presents several challenges in managing and provisioning infrastructure in order to build a reliable and scalable deployment.

1.2 SeerSuite and the Cloud

Cloud computing (2) in this context the use of infrastructure in a dynamic, pay as you go model, offers information retrieval systems, particularly digital libraries and search engines, a wide variety of options for growth and reduction of maintenance needs and encourages efficient resource use. These features are particularly attractive for digital libraries, repositories, and search engines such as CiteSeer^x. The dynamic and elastic provisioning features of cloud infrastructure allow rapid growth in collection size, acquisition workload and the need to support a larger user base, while reducing management issues. Use of Cloud computing infrastructure for information retrieval and scientific computing has focused on implementing particular features for cloud storage in (47) or studying cost benefit trade-offs in (12). The use of cloud infrastructure has also been studied in the context of private clouds for digital libraries in (30).

Cloud offerings have taken a number of forms, not limited to Infrastructure (IaaS), such as Platform (PaaS) and Software (SaaS) as a service approaches. Recent research has focused on adoption, economics and applications. In (2), Armbrust et. al., explain and quantify benefits from the elasticity of a cloud. They argue that although costs for using cloud may appear higher than buying the hardware, elasticity and the ability to transfer the risk of under/over-provisioning outweighs the calculated costs. They show that cloud cost makes sense when factors such as cooling, power, and operational costs are taken into account. (7) show using simple calculations that for OpenCirrus, the break-even point in terms of server utilization is 33%. A number of other recent papers present simple calculations showing the suitability (or lack thereof) of migrating a certain application to a cloud discussed in (57; 56; 52). Cost, ROI (Return on Investment) calculators are available from several vendors and consulting groups. A more detailed discussion of the decision making process and support for such a process is available in (36).

Other approaches to utilizing Cloud infrastructure for information retrieval systems have strong links to grid computing and distributed computing as discussed in (23). In these discussions, the focus has been either on cloud computing for data storage infrastructure or the compute infrastructure. A discussion of cloud computing for scalability in preprocessing, harvesting, transformation and storage is provided by (42) in the context of crawling the Web with Sindice.

We study improvements to SeerSuite components in improving the scalability, deployment capabilities of SeerSuite across different infrastructure such as physical and cloud/virtual instances. We propose improvements to SeerSuite along two broad areas,

the infrastructure supporting SeerSuite deployments and improvements in SeerSuite itself by means of refactoring.

1.3 Refactoring SeerSuite

Abstraction of services such as those of the repository by adopting REST (REpresentational State Transfer) based interfaces can extend the capabilities of SeerSuite and improve the portability, scalability of SeerSuite (55). We discuss the mechanics of setting up such an abstraction and evaluation of an implementation. By such an abstraction, the workflow and placement of services can be improved, we discuss the impact of such abstractions on the workflow and placement.

In addition to utilizing cloud computing infrastructure to reduce maintenance issues, adoption of automatic systems, particularly in the acquisition and ingestion of documents into the collection, can be effective. Such a system would allow several steps in the SeerSuite workflow to be automated and thus the overall efficiency and throughput of the system can be increased. This is particularly true of the metadata extraction system. A new metadata extraction system which is flexible, extensible and simplifies the code base of the extraction system is discussed along with a reference implementation and evaluation.

1.4 Scaling SeerSuite

Beyond building a flexible and extensible metadata extraction system, we explore the scalability of such a system with distributed computing and storage abstraction. Our approach to scaling the system uses message oriented middleware (3) with publish subscribe topic exchanges. We evaluate the system in different scenarios and exploit threading to further improve the throughput of the system by several orders of magnitude. We summarize the goals of this dissertation in the following section.

1.5 Goals

The overall goals of this dissertation are,

- Provide an overview of the SeerSuite framework in Chapter 2.
- Examine cloud computing as an option for hosting SeerSuite instances, in Chapter 3 .
- Discuss the design of a new metadata extraction system in Chapter 4.
- Discuss the scaling of the metadata extraction system using message oriented middleware in Chapter 5.

Chapter 2

SeerSuite

This chapter provides a description of SeerSuite, its workflow provide a background to our efforts of migrating and improving the SeerSuite framework as a whole and the repository and extraction systems in particular. This discussion flows from (53). This chapter also highlights how SeerSuite instances aim to achieve goals such as flexibility, portability, reliability and scalability. Since CiteSeer^x is the preeminent instance of SeerSuite, the this dissertation uses both of them interchangeable manner.

2.1 SeerSuite Architecture

An outline of SeerSuite architecture is shown in figure 2.1. By adopting a loosely coupled approach for modules and subsystem, SeerSuite ensures that its instances can be scaled and can provide robust service. The components of SeerSuite can be broken down according to the services they provide as web application, data storage, metadata extraction, ingestion, crawling and maintenance.

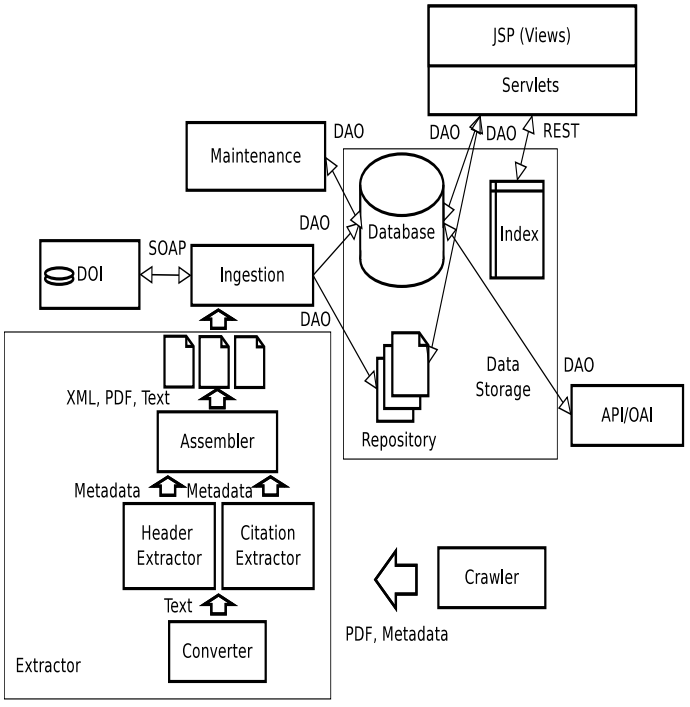

Fig. 2.1 SeerSuite Architecture

2.1.1 Web application

The web application makes use of the model view controller architecture implemented with the Spring framework. The application presentation uses a mix of java server pages and javascript to generate the user interface. Design of the user interface allows the look and feel to be modified by switching Cascading Style Sheets (CSS). The use of JavaServer Pages Standard Tag Library (JSTL) supports template based construction. The web pages allow further interaction through the use of javascript frameworks. While the development environment is mostly Linux, components are developed with portability as a focus. Adoption of the spring framework allows development to concentrate on application design, rather than designing access methods wiring of the application. The use of the spring framework supports application development by providing abstraction to the interface between the database and the application methods.

The web application is composed of servlets responding to user queries. These servlets interact with the index and database to respond to keyword search, the database and index to generate document summaries and the repository to serve cached files. Servlets use Data Access Objects (DAO) with the support of the framework to interact with databases and the repository. The web application is deployed through a web application archive file.

2.1.2 Data storage

The databases store metadata and relationships in the form of tables providing transaction support, where transactions include adding, updating or deleting objects. The database design partitions the stored data across three main databases. This allows for growth in any one component to be handled by further partitioning the database horizontally.

The main database contains objects and is focused on transactions and version tracking of objects central to SeerSuite and digital libraries implementations. Objects stored in the main database include document metadata, author, citation, keywords, tags, and hubs (for URL). The tables in the main database are linked together by the document they appear in and are identified by the document object id.

One of the most unique aspects of SeerSuite is the database describing the citation graph, which allows the user to browse a SeerSuite collection using citations made to other documents. The nodes of this graph correspond to documents or citations and the edges to the relationships among them. This graph is stored in its own database. The citation relationships are stored in a graph table in the form of 'cited by' and 'cites' fields. The citation relationships are established by generating and matching keys for the document metadata records during ingestion or updates.

In addition the database stores a canonical representation of citation or document metadata generated using inference. The inference is obtained by grouping records across the collection by means of a cluster and allowing each member of the cluster to vote on the canonical representation of each field.

These records serve as a grouping point for metadata collected about a particular document or citation. These relationships enable the application to support autonomous citation indexing and establish links between documents. The use of triggers allows data

in the graph database to be updated when transactions such as insertions, deletions and corrections occur in the main database. In addition to triggers, application logic and maintenance functions maintain the link between the graph and the main database.

MyCiteSeer personalization portal stores user information, queries and document portfolios in a separate database. The link between the user and the main database is through references in the tags in the MyCiteSeer database and the main database version tracking tables. A conventional RDBMS with support for triggers is used to host these databases.

The repository system provides SeerSuite with the ability, to provide cached copies of the documents crawled. In addition to the cached copy, the repository stores XML files containing extracted document metadata. These files serve as backup copies of the metadata stored in the databases. This is similar to the Fedora Object XML document representation described in (34). The repository is organized into a directory tree. The top of the tree consists of a root folder containing sub directories mapped according to the segments in the document identifier. The final level contains metadata, text and cached files. The document identifier structure ensures that the there are a nominal number of sub folders under any folder in the tree structure.

An index provides a fast efficient method of storing and accessing text and metadata through the use of an inverted file. SeerSuite uses the Apache Solr an Index application [1] supported by Apache Tomcat to provide full text and metadata indexing operations. The metadata items are obtained from the database and the full text from the repository. The interaction of the application in the form of controllers is through the REST (Representational state transfer) (17) interface. This allows any indexing application supporting REST API's to be adopted by SeerSuite. In addition, this enables introduction of newer feature sets in the index or new versions of Solr without disruptions.

2.1.3 Metadata Extraction and Ingestion

Metadata extraction methods are built using Perl and C++. To enable interaction with the extraction services, a service oriented architecture is utilized using a Business Process Execution Language (BPEL) or, for convenience, scripts. Each component of the extraction system individually contributes to the final document, in this case an XML file, which is then ingested. The feedback to individual systems is manual adjusting of parameters or replacing components. The system can be rewired to include or exclude any extraction modules or applications.

The user can either batch process the incoming data or process each item individually. Addition of metadata into the system is controlled by the ingestion system, which interacts with the citeseerx main database using DAOs. The ingestion system ensures that essential metadata is correctly and uniquely labeled, with the help of an object identifier and checksum based de-duplication. In addition, the ingestion system makes use of listeners to share notification data such as alerts that inform users and programs about objects of interest.

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

The ingestion process can itself be distributed across machines, taking advantage of the Document Object Identifier (DOI), database and shared repository services. The DOI is issued by a stand alone web service supported by a database and tracks document identifiers and the time of their issue.

2.1.4 Crawler

The suite also includes a Heritrix [2] based crawler for harvesting documents from the web. The interface between the ingestion system and the crawler is based on the Java Messaging Service over ActiveMQ [3]. This system provides a buffer between the crawler and the ingestion system and allows for a more flexible scheduling of the ingestion process Due to the modularity in the design, other crawlers can be used. The ingestion system sends URL submissions messages containing a job identifier and URL through the ingestion channel, and the crawler responds with 'new content' messages pointing to the acquired resource and metadata to the ingestion system. The crawler can use other optional channels to provide status messages to listeners. The crawler uses the Heritrix job submission system which consumes messages in the submissions channel and processes these submissions.

2.1.5 Maintenance

Maintenance services support and enable associated functionality for the ingestion and presentation systems. Maintenance systems are responsible for updating the index, identifying metadata by inference, generating citation and document statistics, charts generation, and external data linking in the system. For convenience, common maintenance processes are available through a command line interface.

An evidence based inference system (10) utilizes a Bayesian inference framework to determine canonical metadata for documents in the collection. The system builds an ideal citation record or canonical representation for a document inferred from the information provided by citations and the document metadata.

Citation graphs which accompany the document view are generated from the graph database, by examining the citation relationships and the distribution across years. In addition to enabling access to extensive document metadata, SeerSuite allows documents in the collection to be linked to copies or bibliographic records in other collections. SeerSuite provides components to map and link to other services such as DBLP [4] and CiteULike [5].

The configuration of an SeerSuite instance or application is controlled through properties and context files, through which information about the file system for the repository, database, index and system parameters can be specified. The web application and the maintenance and indexing functions use similar configuration files. In addition

[2] `http://crawler.archive.org`
[3] `http://activemq.apache.org`
[4] `http://www.informatik.uni-trier.de/~ley/db`
[5] `http://www.citeulike.org`

the SeerSuite distribution provides configuration files or examples of configurations for applications used such as the Solr index and Tomcat.

2.2 Federation of Services

CiteSeer^x includes several unique services, which are not part of the SeerSuite application framework. Provisioning for these services is an unique aspect of the SeerSuite framework. Many of these services have evolved as a result of research and are still being developed. The developer builds and operates these services independently, sharing hosting infrastructure with the main application. Separate tables and databases and index operations maybe provisioned for each service. In the following sections, we briefly discuss Table search and author disambiguation search.

2.2.1 Table Search

Tables in documents often contain important data not present elsewhere. Table search services are based on TableSeer developed as part of the project and described in (40). Table search automatically extracts table metadata, and indexes ranks tables present in a SeerSuite collection. While table search shares components of the web application and shares the repository with SeerSuite, the index and extraction components are independent of SeerSuite.

Table search has served as a template for the development of similar services such as algorithm and figure search, which are in development.

2.2.2 Author Disambiguation

Author disambiguation enables users to identify whether records of publications in a SeerSuite collection refer to the same person (even those sharing the same name). The author disambiguation service provided by SeerSuite is based on an efficient integrative framework for solving the name disambiguation problem. In an earlier deployment, A blocking method retrieved candidate classes of authors with similar names and a clustering method, DBSCAN, clusters papers by author. The distance metric between papers used in DBSCAN was calculated by an online active selection Support Vector Machine algorithm(LASVM) described in (31). The current system utilizes a Random Forest based clustering (54) method to calculate the distance between papers. The disambiguation application identifies distinct authors based on header information which includes author affiliation and co-authorship.

The implementation makes use of already existing author object data in the main database, and generates cluster IDs for disambiguated authors that are stored in a separate table and index. Results for disambiguated author queries are handled by main application framework by interacting with the main database and the index. A profile page exists for each disambiguated author, with author affiliation, impact, and publications garnered from the SeerSuite instance. The profile page also provides a link, if available, to the author homepage obtained through a system HomePageSeer. An incremental algorithm replacing the offline batch algorithm currently used is in development.

2.3 SeerSuite Workflow

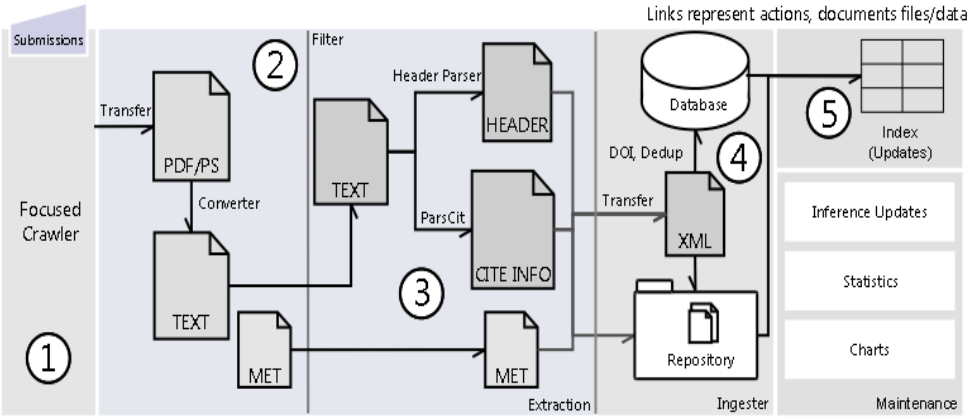

Fig. 2.2 SeerSuite Workflow: Links represent actions and documents data

The outline of the process of adding documents to SeerSuite instances is shown in Figure 2.2. Documents are harvested from the web using focused crawlers (step 1). These documents are first converted from PDF or PostScript format to text with application tools such as PDFBox and TET or GhostScript for PostScript documents in the step labeled 2.

In step 3, to prevent processing of documents such as resumes and non-academic documents part of the harvested collection, SeerSuite uses a regular expression based filter on the converted text file. The converted, filtered text is processed using state of the art automatic metadata extraction systems. These include the Support Vector Machine based header parser (27), which extracts metadata such as titles, publication information, author(s) and their affiliation, and abstract from the document. The ParsCit (32) citation extraction system extracts citation and context information from the document.

The ingestion system identifies unique documents and requests a document identifier for the document. If the document is found to be a checksum duplicate based on content, URL mappings are updated to include alternate URL(s) in step 4. In the same step, the repository is updated with a complete set of files including the document in the original format. The converted text files, citations, crawler and document metadata are placed in the relevant document directory under the repository tree. In addition to the individual metadata files, a file copy of the complete document metadata is stored in the form of an XML file. With updates and corrections, the XML files are updated and stored with a version tag. In the main database papers, author, citation and URL mapping tables are updated, triggering updates to the citation graph database. Updates to the graph database ensure that the citation relationship of the incoming metadata to the data already existing in the collection is accurately maintained. If a citation cluster exists and cluster information is available for the document, the document metadata is updated with the cluster information by the inference system.

Incremental updates are crucial in reducing the resources, time required in the upkeep of the system. Documents in the database have a time stamp indicating time of update, helping the maintenance scripts perform incremental updates of the index. With step 5, new or updated metadata are indexed. The maintenance script scans the database for updated and new documents and creates an in-memory indexable document, including the document metadata fields and citation information gathered across the main and citation graph databases. This document is then indexed by the main Solr index over the REST interface with an update command. The maintenance system optimizes the index after each update, using an built in function provided by Solr, which compacts the index and reduces the posting file footprint on disk.

Statistics provide users with a perspective of the collection from a citation, document and author ranking using aggregated citation information. Statistics are generated from the graph database, in the form of text files, which are then presented to the user through the statistics servlet interface. Maintenance scripts are manually scheduled or run by the administrator.

2.4 Deployment as CiteSeer^x

In case of federated services like tablesearch, The SeerSuite interface utilizes the main application framework for interaction with an independent table index. The query results from the index are again processed and presented by the main application framework. Independent operation of the index from the main index allows for more efficient query processing and ranking of table search results. The results utilize the SeerSuite file system infrastructure view the result of particular pages of the tables in cached documents. The ingestion, maintenance and updates services for Table search are independent of SeerSuite, allowing for flexibility in research and development. Some aspects of the table search ingestion system require access to the document metadata such as titles, authors not extracted as part of table extraction, which are acquired from the main SeerSuite instance data stores.

CiteSeer^x serves as a flagship deployment of SeerSuite. It utilizes Apache Tomcat as the supporting platform with MySQL as the RDBMS (Relational Database Management System). The deployment spans multiple machines with varied configurations. Components are distributed based on functionality on these machines. A pair of level 4 load balancing servers (label 1) direct traffic to a pair of webservers (label 2). The load balancers use a connection based metric to determine to which server a particular request will be directed. To ensure availability, the load balancers are configured as a high availability asymmetric cluster that uses open source linux high availability software.

The web servers host the application on the Apache Tomcat platform with the Tomcat instances as part of an Apache Tomcat TCP cluster with session information shared across the cluster. The application processes these requests and processes them with the help of the index (label 3), database (label 4) or the repository (label 5).

In case of an search query, the application translates the query to a suitable format and dispatches the query to the Solr indices. The results are processed and presented to the user. CiteSeer^x uses indices for document and citation objects, table objects and disambiguated author objects. Requests for metadata are handled by the

MySQL database system. Where the document summaries are generated by the citeseerx main and citegraph databases and user interactions through the myciteseerx database. The repository is responsible for storing cached documents, the text and metadata files. Requests for cached files are handled by the application with support from the repository stored on a storage server. The repository system is shared with the ingestion system and web servers, using the Global File System within the cluster.

The processing system (label 9) is maintained separately from the web application and data storage infrastructure. The document processing systems are responsible for converting and extracting the metadata from converted text files. This operation is distributed across machines by dividing the incoming data into distinct sets, each set being processed by individual machines across the cluster. Finally these processed documents are ingested by an ingestion system (label 6). Focused crawler (label 8) and its associated storage (label 7) are also illustrated.

The deployment is supported by a staging and development system, where new features are introduced and reviewed before being introduced in the production system. Major components in the system are backed up either using component level replication services and or by file level backups. In addition to backup on site, off site backups are utilized to ensure redundancy.

CiteSeer^x depends on operating system based security and application security provided by firewalls plus intrusion detection systems and the underlying framework. Application logs and Tomcat error and access logging provide audit trails for bug fixes and troubleshooting.

Infrastructure adopted by CiteSeer^x has led to several issues. Frequent freezes occur due to deadlocks involving the shared file system. Hardware failures have led to loss of data across the repository database. In such cases back ups have helped restore data, while the locking issue has not been resolved. The ability to recover from these unfortunate losses showcase CiteSeer^x robustness.

2.5 Summary

We discussed the architecture, workflow and deployment of SeerSuite. Services not part of SeerSuite but share aspects, infrastructure of SeerSuite were discussed in Federated Services, with examples in disambiguated author search and tablesearch, in future several such services are to be introduced with the current efforts serving as templates. We discuss approaches to Cloud Computing for infrastructure services to SeerSuite and refactoring of SeerSuite in the following chapters.

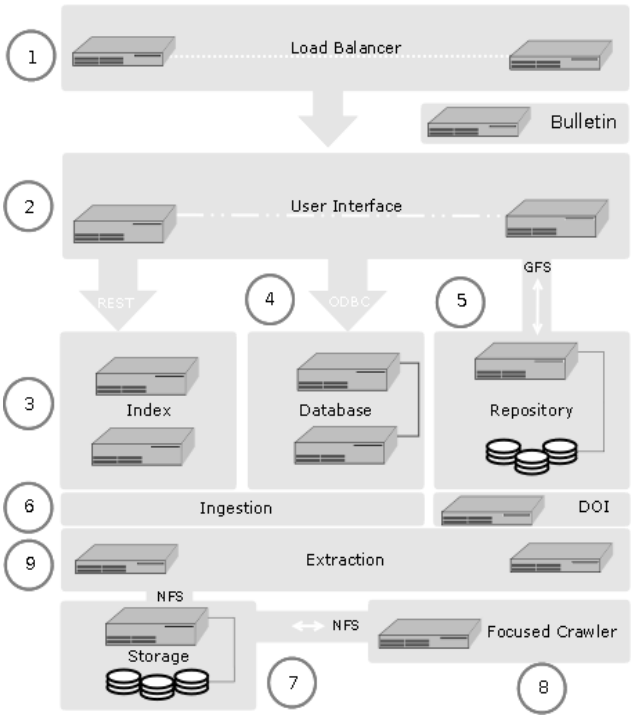

Fig. 2.3 CiteSeer^x Deployment

Chapter 3

Cloud Computing and SeerSuite

Information Retrieval systems such as SeerSuite have several characteristics which can take advantage of cloud infrastructure features. These include the ability of cloud infrastructure to support dynamic workloads both in servicing user requests and in scaling to meet document acquisition needs. The reduced maintenance possible, without the need to maintain physical systems and related infrastructure is another attractive feature.

While it is possible to deploy SeerSuite instances in the cloud, an indepth study of the economic and technical feasibility are essential ensure successful migration to the cloud. In this chapter, we study components of CiteSeer^x and their characteristics, to help understand any advantages or disadvantages obtained by placing these components on the cloud, based on (52; 51). We discuss this issue in two contexts, the effort required to migrate a component or service to cloud infrastructure and the cost of such placement. For each service we summarize the discussion by drawing a conclusion based on the economics of such a migration, and supported by the effort involved.

3.1 Services and the Cloud

SeerSuite components have been designed to allow loose coupling and modular architecture of its instances. This allows hosting of SeerSuite or its components in the cloud with minor modifications. Utilizing the grouping identified in the study of SeerSuite architecture and workflow, we examine each grouping of services; Web Application, metadata extraction and conversion, ingestion, focused crawler, maintenance in detail.

3.1.1 Web Application

Other then the user interface hosted on the Web servers, the database, the index and repository are part of the the application response to user requests. The variation in user load makes these services attractive as candidates for hosting this application on cloud infrastructure.

- Migration effort: The application would require modifications to state information stored as part of myCiteSeer.
- Costs: The Web application has a large footprint, in traffic passing through it, hence likely to be expensive to host this service on current cloud offerings.
- Other considerations: This service has significant data access requirements, hosting is linked to the location of the database, index and repository services.

The session and information stored as part of myCiteSeer are challenges to hosting the web application in the cloud, which includes managing login and the business logic for maintaining privacy and confidentiality. The web application is a good candidate if stored closer to the data storage components in terms of cost.

3.1.2 Metadata extraction and Conversion

The metadata extraction and conversion services are responsible for acquiring documents for the collection. If the documents are acquired one at a time (when users submit documents) the extraction and conversion services are not resource intensive.

Workload at the extractor is a direct result of crawling for new documents and crawls resulting from user submissions and transfers from other repositories into the repository . While the number of documents crawled by the crawler can be scheduled, the same is not possible with user submissions, This leads to varied workloads at the extractor.

The graph in Figure 3.1 shows the number of documents fetched by the CiteSeer^x crawler during a period of 31 days. The documents fetched by the crawler vary greatly each day, reaching peaks of several thousand documents per day to zero. Note that not all of the documents fetched by the crawler are actual academic documents.

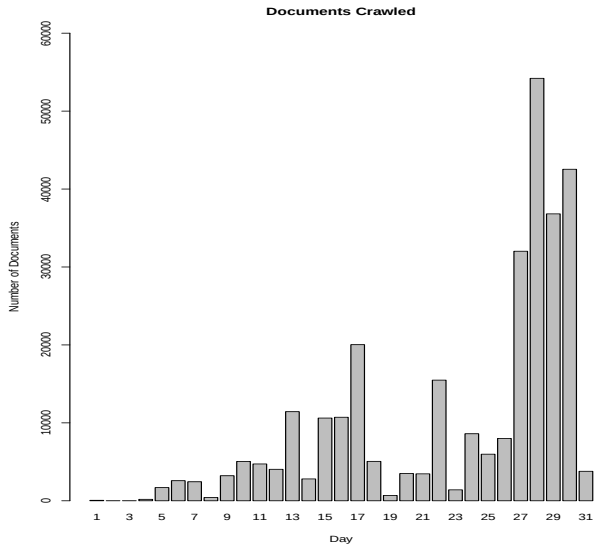

Fig. 3.1 Documents Crawled per Day

The process of identifying documents, fetching them is asynchronous and does not follow any predictable distribution. This is as a result of the crawl process and due to policies implemented at the crawler.

- Migration effort: Most extraction and conversion modules, use multiple programming languages and applications, which may not be supported by most current cloud offerings. Refactoring effort is significant.
- Costs: The costs are likely to be reasonable as the traffic through this process is limited.

Thus, metadata extraction services in the batch processing mode are strong candidates for hosting on the cloud. This fact has been borne out by earlier literature covering distributed computing (23), use of peer to peer resources (45) and cloud computing (42) efforts.

3.1.3 Ingestion

The ingestion system has one of the smallest footprints in the system, This process includes obtaining the digital object identifiers and methods for adding processed documents into the collection.

- Migration effort: Minimal, application modules in java and libraries well supported, minor configuration modifications maybe required.
- Costs: The traffic flow through this process is low. Hence costs are likely to be reasonable.
- Other considerations: If placed on the cloud by itself, modification to other services to interact with the ingestion system will be required.

Hosting of the ingestion process is strongly linked with the location of the data storage component. If placed near the component the costs are reasonable.

3.1.4 Focused Crawler

Focused crawlers are responsible for acquiring documents from the Web for addition to the collection.

- Migration effort: Minimal, the code is mostly written in java and libraries are well supported in many cloud offerings.
- Costs: The traffic flow through this service is minimal. Hence costs are likely to be reasonable.

Since the load factor of the crawler is fairly elastic, it is well suited to take advantage of the underlying infrastructure. This fact is also borne out by (42).

3.1.5 Maintenance

The maintenance functions are responsible for updating the index, generating citation graphs, statistics, and inference based corrections to document metadata. By placing these services in the cloud, we could reduce infrastructure costs.

- Migration effort: Minimal, the code is fairly homogeneous and supported in many cloud offerings.
- Costs: The maintenance system by itself consumes very little storage and traffic and does not generate significant amounts of data. Thus, we feel hosting costs are likely to be minimal.
- Other considerations: Each of the maintenance service interactions occur over large sections of data which includes those stored in the database and the repository.

Thus, the maintenance system must be based near to the database and repository. Similarly the placement of the data should be near where it is used for computation (22).

3.1.6 Federated Services

Federated services include services not part of the framework which share the main CiteSeer^x infrastructure. These services may use separate databases, index.

- Migration effort: Most services offered under federated services are still under development or in research so it will require minimal effort to adopt these services in the cloud.
- Costs: The size of the metadata processed by most of these services is much less than the size of the data processed by the production services. Hence costs are likely to be minimal.

3.2 Approaches

A varied set of approaches are considered for hosting SeerSuite considering data from the CiteSeer^x deployment. Such a discussion enables us to choose both a best fit and to identify areas which will require more thought and effort during the migration. For these estimates the following statistics from the CiteSeer^x were used, CiteSeer^x hosts more than 1.5 million documents occupying 1.5 TB of storage. The size of the database is around 120 GB, with the index occupying 40 GB of space. CiteSeer^x receives over 2 million hits consuming a bandwidth of 150 GB per day.

3.2.1 Complete Hosting: Cost Estimates

We study the cost estimate for the components based on cloud infrastructure services offered by Amazon EC2 [1] and Google App Engine [2]. Cost estimates are based on a 30 day month and provided in US dollars.

Choice of vendors is a result of support in terms of environment and libraries offered or supported by these vendors. In the case of Amazon EC2, we consider a mapping of one to one to an extra large instance for hosting the database, application, index, repository, extraction and crawler services. We assume that additional instances are provided as required in Google App Engine with no additional cost.

[1] `http://aws.amazon.com/ec2/`
[2] `https://code.google.com/appengine/`

Cost			Amazon	Google
Initial Setup	Data In	1820.4	0	182
Monthly	Stored	1820.4	182.04	273.06
	Data In	152	0	15.2
	Data Out	3072	460.8	368.64
	Trans.	368	190.77	0
	CPU	30*24	2937.6	144
Total Monthly			$3771.21	$800.9

Table 3.1 CiteSeer^x Hosting

Note that Amazon currently provides free data transfers into the cloud. If this were not the case, hosting services on Amazon would be much more expensive and also incur initial setup costs. Calculating the cost of hosting the entire application leads to a figure of $3771 for Amazon EC2 and $800 for Google App Engine. The cost of hosting components also lends support to the conclusions drawn about data and access in in (22)

The above costs indicate that moving CiteSeer^x to the cloud would be an expensive process compared to the current hosting costs and funds available for such an effort.

3.2.2 Component Hosting

From our discussion of SeerSuite, it is possible to identify specific components in SeerSuite to be hosted. Each component hosted in the cloud includes all of its associated modules and subsystems. For example, hosting of the SeerSuite index includes hosting of the storage for the index, application code, and interfaces.

We consider components of the system, choosing components based on size, data transfer, and feasibility of migration. From Table 3.1, we see the cost of hosting is dominated by the cost of computation (per instance cost). The other major cost components include the cost of data storage and data transferred in and out of the cloud. For hosting components the compute costs are a constant, since the components always operational. Therefore we now address the data storage and transfer costs.

Figure 3.2 shows the flow of data between components of CiteSeer^x. Data for creating this graph was obtained from log files and application specific monitors. In the case of crawlers, the information extraction data flow was assumed proportional to the number of documents acquired and processed. This graph is useful for determining candidates for hosting. For example, if CiteSeer^x were hosted entirely within the cloud infrastructure, the amount of data stored in the cloud would be 1.7 TB, with 3.2 TB of data transferred between the user, web and the application. We now examine the cost of hosting individual components in the cloud, with all other services hosted locally. Estimates are provided in Table 3.2 costs is in US dollars.

Individual components hosted on the cloud have implications beyond the cost of hosting them in the cloud. Costs related to refactoring code for migration has not been accounted for in Table 3.2. In the case of Google App Engine, existing code written in languages not supported by App Engine will require significant refactoring. Along with components hosted in the cloud, components hosted locally may require refactoring. This

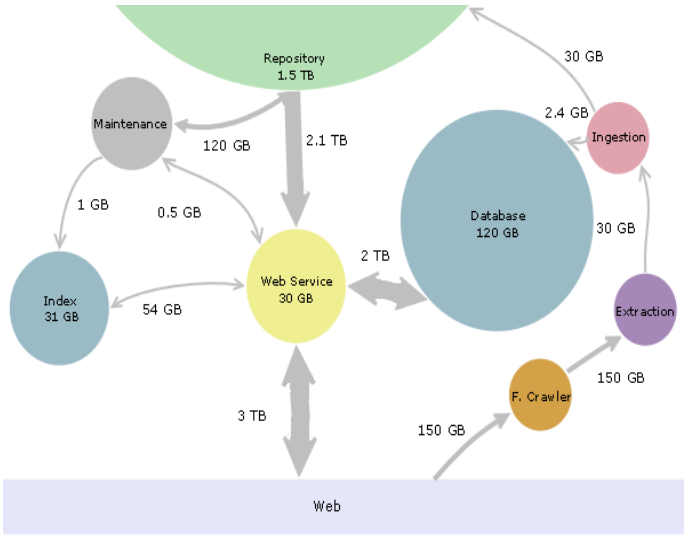

Fig. 3.2 Data Flow - CiteSeer^x Components

refactoring is minimal if the service or component utilized a service oriented interface and significant when services are closely coupled.

Lessons Learned: If the cost of hosting an entire service is prohibitive, hosting components may be a reasonable approach to taking advantage of cloud infrastructure. The cost effectiveness of such an approach depends on data transferred through the service. Loosely coupled components are easier to migrate. For existing components and code, refactoring costs will provide a closer estimates of costs. This approach is suitable, when a fixed budget constrains the placement of services or components. By identifying components, data transfer and refactoring costs a hosting solution can be identified.

3.3 Load and content based hosting

Hosting part or an entire instance of SeerSuite can be economically challenging or cumbersome due to the amount of refactoring and or data transfer between components.

We explore scenarios, through which a part of a component or the load of an instance can be hosted in the cloud. We begin by considering the web application services and discuss content based partitioning of hosting.

3.3.1 Content Hosting

Content particularly static images, stylesheets, javascript common to most web pages need not be hosted locally. An analysis of peak traffic at the web services provides an insight on how this can be achieved. From analysis of figure 3.3, we see that most requests for peak traffic are for such content. In this case the amount of computation required and data stored on the cloud is small, the cost of hosting is cost-effective. The total size of all files to be placed on the cloud is 2.24 MB. By hosting these files in the cloud, the amount of data transferred for CiteSeer^x from the cloud is 390.26 GB

Component	A. EC2		G. App Engine	
	Initial	Month	Initial	Month
Web Service	0	1448.18	0	942.53
Repository	0	1011.88	163.8	593.21
Database	0	858.89	12	348.05
Index	0	527.08	3.1	83.48
Extraction	0	499.02	0	90.6
Crawler	0	513.4	0	105

Table 3.2 Component Costs (USD) in the Cloud

costing less then $142 per month (on both Amazon EC2 and Google App Engine services including a small instance cost). While this is a small part of more than 3 TB of data volume between the application and the user, it helps the system satisfy a significant number of peak load requests.

The same approach can be used to identify elements such as a subset of the repository to be placed in the cloud. Such an approach would involve identifying the most commonly accessed documents and placing them both locally and in the cloud. During peak loads, clients can be directed to the cloud for access.

Lesson Learned: Hosting specific content relevant to peak load scenarios in the cloud can be beneficial, and the simplest approach to hosting services in the cloud.

3.3.2 Load based partitioning

This approach is particularly important for supporting the growth in traffic, flash crowds providing users access to service.

Figure 3.4 shows the requests received at the web server. From the graph we identify that the 90th percentile is represented by 60 requests per second. Most of these requests are for elements associated with presentation (javascript and stylesheets). Assuming that the traffic growth continues at the same pace and as more features (Algorithm and Figure search) are added, There is a need for provisioning more systems. Instead of procuring these systems, infrastructure at the cloud can be considered to fulfill this need.

Further examination provides evidence of self-similarity in the request arrival process, which has interesting implications for resource provisioning. The peak resource needs of several CiteSeer^x components are significantly higher than their average-case (or even a high percentile) needs.

Two strategies are possible in partitioning based on load. Of these, one strategy would be to host a copy of the entire application in the cloud, using load balancers to identify and direct traffic during peak load conditions. Table 3.3 provides the costs of such a hosting solution for CiteSeer^x in Amazon EC2 and Google App Engine. All data measurements are in GB, and transaction measurements in transactions per second obtained via iostat.

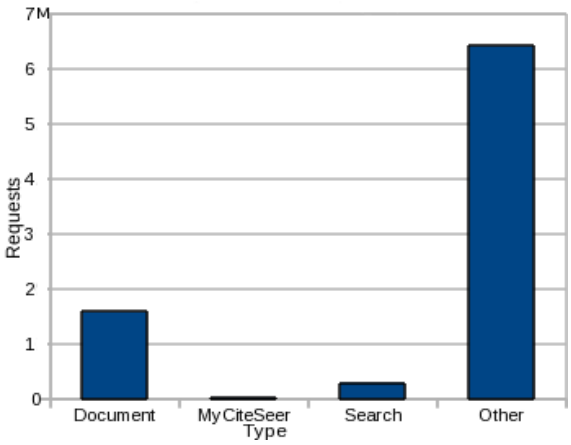

Fig. 3.3 Request Types at Peak

These costs can be considered in comparison to the cost of procuring, maintaining systems. Savings by avoiding adoption of storage systems locally add to the attractiveness of cloud infrastructure.

An alternate approach would be to host only the component under stress in the cloud, For example, a database replica to support a locally hosted database could be deployed in the cloud. If this instance were used only during peak load conditions, the costs would decrease to $385, since the instance would be in use for 70 hours.

Lessons Learned: By utilizing a replica or subset of the application for handling only peak loads, we can take advantage of cloud infrastructure in a cost-effective manner. This can resolve issues stemming from the growth of the collection and user traffic.

3.4 Summary

We explored various strategies for hosting SeerSuite in the cloud, In Section 3.3.2 we briefly mentioned the temporal nature of traffic and user behavior. By identifying user patterns (39), the hosting solutions can be optimized to take advantage these patterns. While this discussion included the Amazon EC2 and Google App Engine for cost comparison, this work needs to be extended by examining in depth options offered by other cloud offerings, private clouds and virtualization solutions.

Products such as private clouds offered Eucalyptus (44) can be utilized to take advantage of hardware already existing as part of the system. Components related to user interaction with CiteSeer^x hosting with services like Amazon Virtual Private Clouds, and local clouds can be considered for these services.

Inclusion of cloud services could require significant refactoring and changes to maintenance cycles. Issues with how issues like latency, load balancing have not been addressed. We address some issues in abstraction and refactoring in the next chapter.

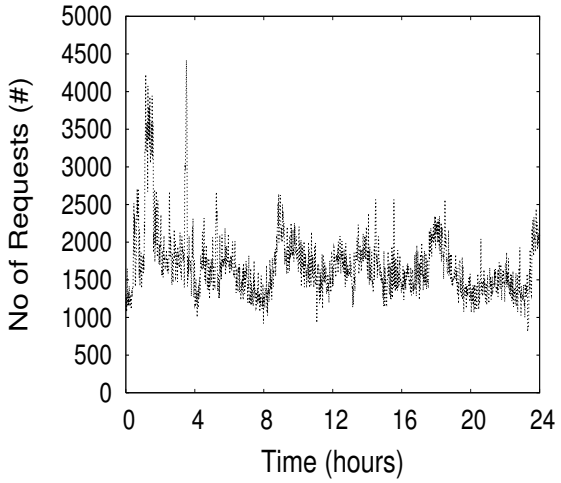

Fig. 3.4 Number of Requests per Second

Cost			Amazon	Google
Initial Setup	Data In	1820.4	0	182
Monthly	Stored	1820.4	182.04	273.06
	Data In	14.78	0	1.48
	Data Out	298.7	44.8	35.84
	Trans.	368	9.27	0
	CPU	70	285.6	7
Total Monthly			$521.71	$317.38

Table 3.3 CiteSeer^x Peak Load Hosting

Chapter 4

Refactoring SeerSuite

From our examination of the components, services and feasibility of migrating services to the cloud, we know that migrating SeerSuite to the cloud while feasible is limited by economic considerations. This examination also highlights several components which can be redesigned to better scale and aid in migration. Loose coupling of modules and services in SeerSuite provides us flexibility to modify and re-design components and services transparent to other services, we take advantage of this ability to address two main challenges to the migration of SeerSuite and scalability of SeerSuite. These challenges are centered around the size of the data that is being stored and transferred, and limitations in the architecture and functioning of the metadata extraction system.

The repository system presents the greatest challenge to migration, mainly centered around the requirement for specialized storage infrastructure such as a SAN/NAS supporting large storage and sharing of this storage with several systems concurrently. We consider the approach of abstraction of the services provided by the file system to building a new repository system in SeerSuite. The architecture and design of the metadata extraction system includes several dependencies and infrastructure requirements, which are obstacles to migration to the cloud, we propose a system which simplifies the structure and provides extensibility to the type of metadata and methods utilized for metadata extraction.

4.1 Abstraction

Abstraction of storage services can greatly benefit both SeerSuite instances hosted locally and in cloud infrastructure. Abstraction of storage removes the need for specialized hardware as the size of the collection grows. By abstracting storage services, a greater amount of flexibility can be exercised in the placement of services such as the ingestion, extraction and web applications.

4.1.1 Repository

Among the components of SeerSuite, the repository service, the database and index require significant amount of storage. The requirements of the repository process are demanding of storage services. The repository requires the repository space to be shared among multiple application components, which is particularly challenging when the system is distributed across multiple systems. The repository is responsible for storing PDF, metadata (XML), text files for use by the web applications, maintenance, backup and indexing services.

Functionally, the repository serves as a application file storage system, storing documents for the SeerSuite collection in a hierarchical structure based on the DOI (DOcument Identifier) of the document. The characteristics of these files and access patterns are as follows.

The repository contains a large number of files which are never accessed once placed in the repository. These files include the body of the document, the citation metadata and the crawler metadata. It contains files which are only read once placed, such as the XML files. An XML file contains a snapshot of the document metadata and serve as application level backup, ensuring that database and index can be created from scratch from the metadata stored in the repository. The other files stored in the repository include the the PDF/PS files and files that are accessed periodically for reading in the form of cached copy downloads by users and are never written to. Finally the text files stored in the repository are also read periodically (during index updates) and never written to.

The repository is exposed in the current setup as a shared resource between web servers in CiteSeer^x. This is due to the fact that changes to the repository which includes corrections, updates to charts, etc. have to be reflected across all services. Writes to the repository occur as a result of ingestion, corrections, or when charts are generated by the maintenance service for each document. With updates to SeerSuite, graph information has now been moved to the database and no longer requires repository storage or access.

Among the writes to the repository, corrections are initiated at the web application by the user and stored in an versioned XML file in the repository.

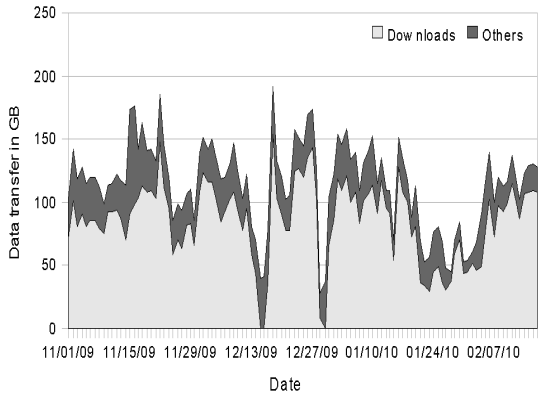

Fig. 4.1 Download Traffic

The graph in Figure 4.1 indicates that a significant portion (60-80%) of the user traffic to CiteSeer^x is a result of downloads of cached copies from CiteSeer^x. Cost efficient, scalable solutions to handle this traffic become crucial for large deployments.

Scalability of the repository presents a challenge for SeerSuite applications, especially CiteSeer^x. This is further exacerbated, by future plans to store and offer video,

images and presentations linked to the document. In order to grow beyond the present collection size, the repository services have to be hosted on cost-efficient scalable infrastructure. Solutions involving storage systems, particularly SAN (Storage Area Networks) or NAS (Network Attached Storage) can be expensive to adopt and maintain. In this regard, virtualized hardware or cloud infrastructure services offer a desirable solution.

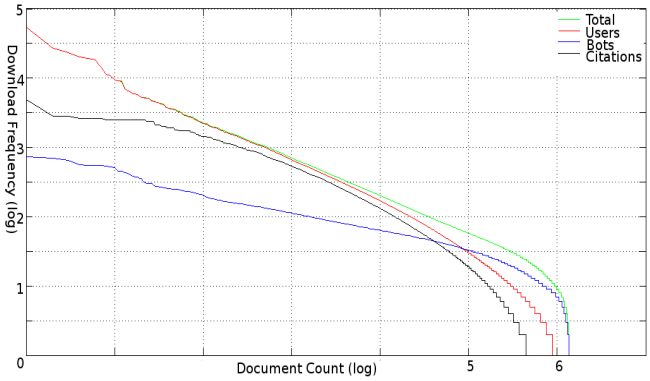

Fig. 4.2 Document Download Frequency

Partitioning and hosting of content in the cloud could be advantageous for for SeerSuite instances. For example, the repository could be partitioned and parts of the repository hosted in the cloud. In this context, an examination of web caches, placement and replacement policies could be beneficial. A partition of the repository, with documents placed either in the cloud or local storage can be translated to the problem of sizing the cache, management of the cache by understanding the placement and replacement policies. A study of the traffic pattern to the repository service provides us with some guidelines for such a study. A suitable partitioning of the repository service should take advantage of the download pattern (frequency of access), size of the media files being placed in the cloud or local repository (size of the objects) and finally the replacement policy when the placement itself is dynamic and guided by existing literature (46; 58).

We examine the download pattern of documents, to identify any patterns which we can take advantage in our approach. The graph in Figure 4.2 identifies document download frequency on a log/log scale (for 6 months). Download frequency for user initiated downloads (red) and crawler initiated downloads (blue) are provided along with the citation ranking for documents in the collection. The frequency of documents downloaded in the graph follows a power law distribution. The behavior of crawlers and users can be clearly differentiated, the bots download documents less frequently but download more documents overall. From the graph we can infer which subset of the actual documents (1.5 Million) are actually downloaded either by users or bots. Another approach would be to pick only those documents cited in the collection (564,818), which is a conservative estimate. On the other end is the set of documents downloaded by crawlers - 1,394,669 documents. The above figures can be misleading if overlap between individual sets is not considered. The overlap between the set of documents cited in the

collection with the ones downloaded by users is 334,264. But documents downloaded by both bots and users number 874,100.

From the above observations it is clear that identifying the most frequently accessed documents, can aid in identifying a subset of the collection which can be migrated to the cloud based on economics. If the download costs of the cloud are not particularly prohibitive, we can host these documents in the cloud. A more important factor is that, identifying these documents and utilizing this information can lead to a more efficient design of the abstraction. By storing redundant copies of documents most frequently accessed, we can ease the load across the storage infrastructure. The patterns also mean that placement of documents is a complex operation, unlike the placement of static content identified earlier.

4.1.2 An ad-hoc solution

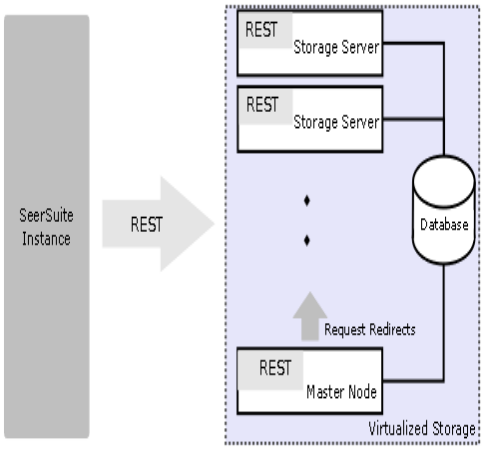

Fig. 4.3 Virtualized Storage

The SeerSuite architecture provides flexibility in adding new features. Replacing components of the system with other components is possible as long as the two interfaces are compliant. We take advantage of this fact, and experiment with an abstraction in the form of a virtualized storage. This solution allows us to identify and estimate with greater accuracy parameters and bottlenecks for adopting a cloud infrastructure in SeerSuite.

The virtualized storage solution uses a REST based interface for the entire system. The storage allows GET, POST, and DELETE commands for downloading, adding, and deleting documents. Aspects of this system were inspired by distributed file systems discussed in (19; 6). The architecture of the system is shown in Figure 4.3. The master node serves as an index for finding documents in the storage system. Requests for documents are made to the master node, which redirects the client to the storage server containing the particular file. Writes or adds bypass the master. In case there are multiple copies of the document with the same version number available, the master redirects the user to the latest copy as per the database.

The storage node is the basic unit of the system, storing files on the underlying file system. Storage nodes allow users access to cached documents through GET requests. Storage nodes accept additions, which are logged into a central database. Uploads or PUT log the location/storage server to where the upload was made. The files are stored in the same hierarchical format as in the SeerSuite repository with the top directory and leaves labeled with the repository id and with the complete DOI, respectively.

The system is rudimentary in that it provides no assurance on consistency or locking by itself and is dependent on the operating system underlying the service. However the use of checksums and versioning can address some of the issues with this approach. Features such as version information, document checksums, and sizes are currently stored in the database. An important feature of this approach is that it can be deployed over already existing collections (with collaborators) without incurring significant costs by simply updating the database with document locations at the collaboration source.

4.1.3 Impact on other services

The introduction of virtualized storage has a major impact on the ingestion, maintenance, and web application processes and services which we further elaborate further. It has relatively little or no impact on the crawler and metadata extraction services. As such the use of a virtual storage service is invisible to the user, except for access to the cached copies now provided through this service.

The ingestion system now uses POST calls to add documents to the repository. This allows the ingestion process to run on different systems other than those hosting the repository. The impact on ingestion performance is minimal. The web application now points to the director for cached document downloads. For corrections applied to documents, a POST request is made to the repository containing the document to update metadata. Significant changes occur to the document index update process and to the citation chart generation. Indexing maintenance scripts now obtain body and full text of documents with GET requests to the storage servers. Citation charts are generated by javascript libraries, removing the need for generation of static image files and storage in the repository. This approach adds costs to the application and clients. The Web application must obtain data for these charts from the database.

In summary, the system is more scalable as dependence on shared storage systems is reduced. Another positive feature is metadata about files in the repository can easily be exposed to users and researchers. Queries on dimensions such as size and checksum and can be made to the storage master. This approach allows CiteSeer^x to distribute load across collaborative engines. By identifying the closest repository containing the document across collaborating engines, faster downloads can be achieved.

4.1.4 Performance

A deployment of the ad-hoc system discussed in the earlier sections, containing documents (1 TB) available in the CiteSeer^x collection was built. The deployment employs 3 systems, each with dual core processors, 3 GB of RAM, and 1.5 TB of storage, as two storage nodes and one master node (the master node also hosts the database).

To identify bottlenecks, we subject this setup to a JMeter (25) stress test. Test case data was obtained from log analysis presented earlier. Requests are replayed to emulate scenarios. Tests with different number of users (virtual users) and user introduction rate (ramp up time) with five repeats in each test were performed. Results for one set of ramp up times are shown in figure 4.4. The number of virtual users was varied,

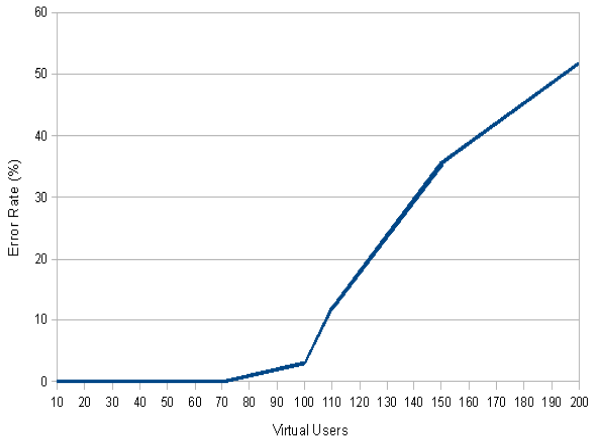

Fig. 4.4 Ad-Hoc Cloud Performance

with arrival times for all users set at 10 seconds. Each request was for documents chosen at random. From the graph as the number of virtual users increase, the error rate rises in an significant manner with 70 virtual users introduced in 10 s. Test results indicate errors occur when the number of active users is > 97. An examination of the master error logs indicates that database connection errors were the cause for server errors.

When compared to the CiteSeer^x system which was capable of handling only a limited number of active users (¡ 20), the ad-hoc system has a significantly higher throughput. These results indicate that particular care must be taken in scaling the database, access available when building or deploying CiteSeer^x on virtualized infrastructure. Both CiteSeer^x and the underlying system must support the expected load.

4.2 Metadata Extraction

We began with a review of the metadata extraction system and its limitations. The metadata extraction system is responsible for processing documents fetched by the crawler and extracting from the documents, metadata crucial in building the database and citation graphs. We examine the existing metadata extraction in detail to understand the process of extracting metadata and limitations in the design and architecture.

The process of metadata extraction begins by converting PDF documents into text. This text document is examined by a regular expression based filter to identify scholarly documents. These documents which have passed the filter process are then processed by multiple extraction methods to extract metadata. The extractors include

a header parser (27), responsible for extracting the title, authors and citations and a citation parser (32), which extracts the citations, individual fields from the citations and the context of individual citations from the document text. These parsers utilize methods such as Support Vector Machines (33), advanced text segmentation methods (28) for parsing the document to extract the header, and Conditional Random Fields (37) to extraction citation metadata. The modules of the metadata extraction can be operated either in a batch mode or as individual web services coordinated through a Business Process Execution Language (BPEL).

The metadata extraction system has several limitations. The processing of documents is a strict pipeline operation, where each process operates in serial and the output of one stage forms the input for the next process in the pipeline. This leads to limited throughput. Parallelizing these processes is not straight forward due to code complexity and limited threading support. Documents to be processed by the extractor have to be located local to the modules. These limitations imply that increasing the resources such as available computation power, memory or speed of components does not aid in improving the throughput of the process.

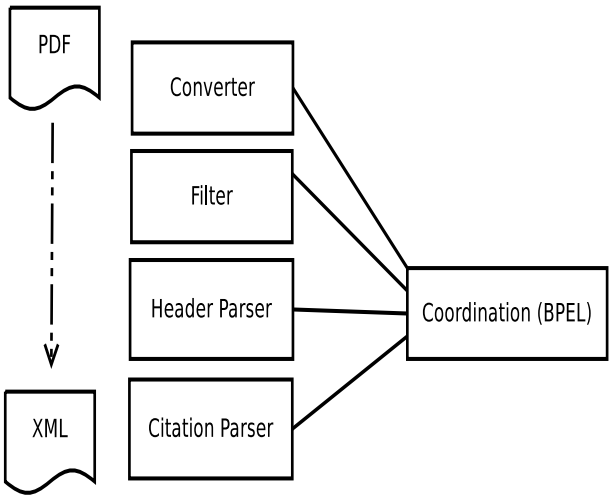

Fig. 4.5 Current Document Conversion and Extraction

Figure 4.6 provides a breakdown for processing 2000 documents (variance: 0.0002, average time for document conversion 0.81, average time for filtering 2.10, average time for extracting headers 8.37, and for extracting citations is 8.88). We notice that the header extraction is the most expensive step, followed by citation extraction, filtering and finally conversion. While some of the documents will convert (14%), not all succeed. This means that the relative throughput of the process is low as a result of time consumed in processing documents which fail the process. The throughput of the process does not increase with increase in resources provided. In an effort to improve the metadata extraction system, we built a metadata extraction system. We begin the process of

building a new metadata extraction system by simplifying the process and reducing the complexity of the metadata extraction system.

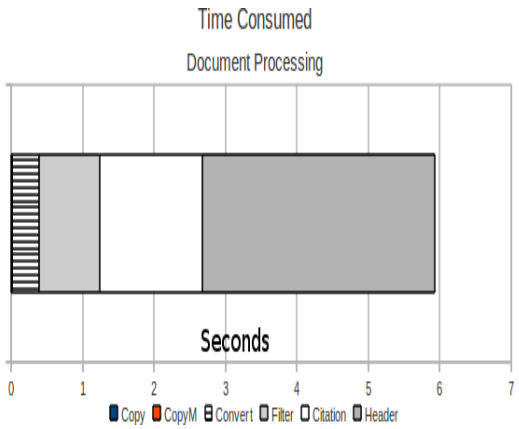

Fig. 4.6 Current Document Conversion and Extraction

4.3 Extractor Architecture

We propose a new extractor design, which overcomes several of the limitations of the present extractor identified earlier. These include the portability, extensibility of the extractor. Specifically we develop a framework for building SeerSuite extractors. The framework allows users to develop and design their own extractors to recover entities of their interest.

The extractor consists of three distinct modules, the document conversion software such as a PDF to text converter, a classification system or an ensemble of classifiers for extracting document metadata and an assembler to aggregate and assemble extracted metadata. The framework makes use of java interfaces as the base for modules allowing the use of dependency injection.

4.4 Reference Implementation

The proposed metadata extraction framework is shown in Figure 4.7, it includes the document conversion subsystem, the header parser, the metadata assembly system and supplements the citation parser in the document acquisition and processing pipeline. The proposed extractor focuses on a select set of metadata; title, author names, section headers, citation contexts and the citations themselves.

A crucial improvement introduced in the extractor, is the ability to make use of font and layout information embedded in PDF documents. These features are made available either through interfaces provided by the document conversion application itself (PDFlib TET through tetml) or by modifying the document conversion system as in open source software like PDFBox.

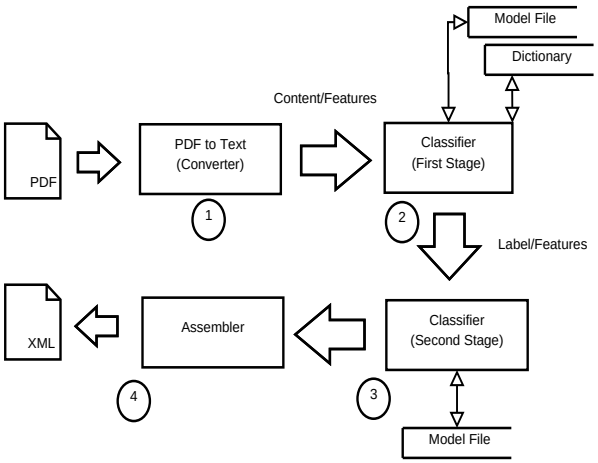

Fig. 4.7 Reference Extractor

The supervised machine learning classifier, which processes the output of the document conversion service, by labeling lines is built using an open source library Weka (26). This allows us to adopt different methods, combined with the dependency injection approach followed by the system, to enable different features, entities to be captured to be included with only minor refactoring.

The reference system utilizes a random forest classifier, trained on 207 labeled titles, 326 labeled author name(s), 960 labeled section headers, 2397 labeled citations and 1010 labeled contexts for a total of 4090 labeled lines. It utilizes features generated from the content, layout, font information and domain information from dictionaries such as author names (dblp, census), title keywords (dblp), geographical entities, academic entities and stopwords. We were encouraged to use these features such as the font information by results presented in (4) and the use of domain knowledge represented by the dictionaries are discussed in (27).

To illustrate the effectiveness of features obtained from the font and layout, we present the results of a ranked features obtained using information gain (1) in 4.1.

Relative Position	Absolute Position
Relative Font size	X margin
Count of number tokens	Count of ”.”
Count of alpa tokens	Geometric length
Count of decimal tokens	Count of Stopwords

Table 4.1 Top features

Finally an assembler, which aggregates the labeled entities in the document and links entities such as the citations to the context.

4.4.1 Evaluating Quality of Extraction

To illustrate the effectiveness of features obtained from the font and layout, we present the results of ranked features obtained using information gain (1).

Since the reference extractor would replace most of the existing extractor we examine the data quality of the extraction. Performance is measured across two criteria, the ability to extract the complete entity and the percentage recovered in case there are several entities of the same time. A less conservative measure could be to allow partial matches, since the inference system in SeerSuite would recover the complete entity from these partially extracted entities. Fields such as title of the document, author names and citations made by the document to other documents are crucial, We manually examined 50 documents for titles, authors and citations recovered. The accuracy of the extraction is compared to that of the existing extraction system. Figure 4.8 shows the relative performance for extracting titles. While the reference implementation has better performance, the overlap in the confidence intervals indicate that there is no significant difference. In case of author extraction shown in Figure 4.9, the performance is once again comparable and there was no significant difference, with the present extractor having a better average performance. In case of the number of authors extracted shown in Figure 4.10, the number of authors recovered was much more than the reference implementation. For extracting citations show in Figure 4.11, the average performance is the same, while the the number of citations recovered is much higher with the reference implementation.

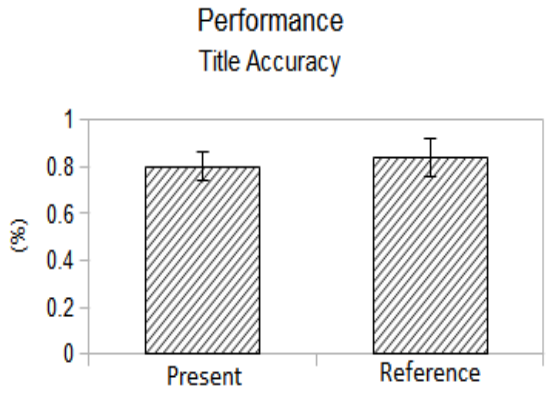

Fig. 4.8 Accuracy in extracting titles

Average results for each entity is shown in Table 4.2.

From the results, it is clear that the reference metadata extraction improves on the extracting titles, citations, while its performance is marginally worse than the existing extractor for authors. An examination of the errors for the of the reference implementation shows that there were issues with identifying author names containing hypens and

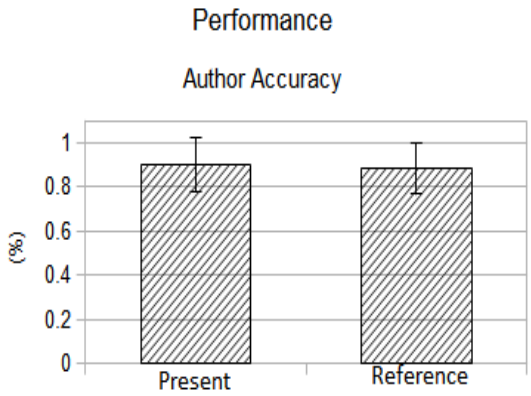

Fig. 4.9 Accuracy in extracting author names

Extractor	Titles	Authors (Recovered)	Citations (Recovered)
Reference	84%	88% (86.05%)	98% (87.90%)
Present	80%	90% (93.70%)	98% (79.99%)

Table 4.2 Quality of extraction

unicode characters, and certain entities such as the university name, department were labeled as author names.

4.5 Summary

We discussed two major challenges in migrating SeerSuite to the cloud and our approach to overcoming these obstacles. We presented an abstraction for the repository services and an implementation of a metadata extractor. The results from our evaluation of the repository and the metadata extractor show that the repository can be abstracted successfully and extends both the performance and features of the existing system. The metadata extractor improves on the complexity of the system and quality of extraction.

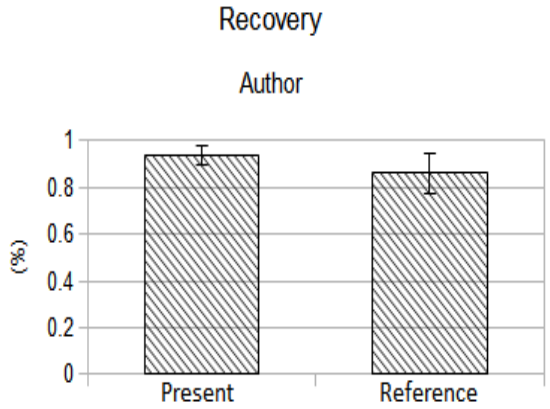

Fig. 4.10 Recovery of author names

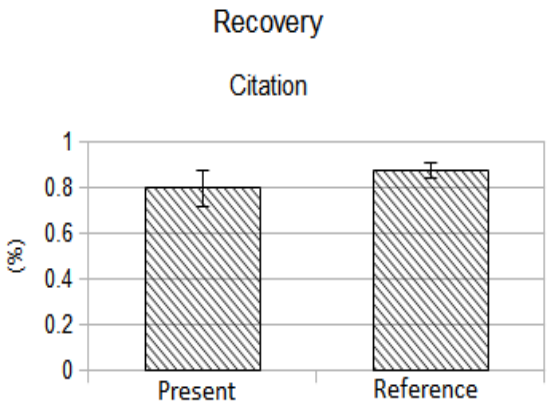

Fig. 4.11 Recovery of citations

Chapter 5

Scaling SeerSuite

We addressed the improvements to the metadata extractor code base to simplify and reduce the code complexity of the metadata extractor, while these changes are significant, issues with low throughput of the extractor are still a significant concern. At the time of writing this document, several hundred thousand documents had yet to be processed by the extractor at CiteSeer^x. Increase in the resources allocated to the existing extractor do not lead to an increase in the throughput, we take advantage of the simplified architecture, and code base to take advantage of distributed computing approaches to scale the metadata extractor.

5.1 Stand alone performance

We present the evaluation of the existing and proposed extractor to illustrate the need for scaling the metadata extractor. To evaluate the performance of the extractor in terms of throughput, we make use of a collection of documents obtained from the CiteSeer^x collection and the web. Both extractors are run in batch mode on the existing processing infrastructure [1].

We examine the average time taken for processing a document on the existing infrastructure, through the average number of wall clock time taken for converting a set of 2,498 documents. The documents had an average size of 700 Kilobytes and 97% of the documents were smaller than 2 Megabytes in size. In this case the existing extractor consumed 3.32 seconds on average to process a single document, compared to the 3.20 seconds consumed by the proposed extractor. Thus there has been a marginal improvement (4%) in the time taken on average.

5.1.1 Stand alone performance in the cloud

To evaluate the operation of the new metadata extractor in a cloud environment, we hosted the extractor along with the necessary libraries, dictionaries and files to process documents in different cloud computing environments such as the Amazon EC2 and Windows Azure. In this case the performance of the system in the time taken to process a document was examined.

The extractor processed 120 documents (97% less than 2 Megabytes) all located in the local storage/filesystem for this evaluation. The output of the extractor (XML) was also stored on the local file system. There were no changes made to the application

[1] Dual core processor, 3 GB of RAM, TB of disk space

code, except for changes to the packaging according to the cloud infrastructure to which the extractor would be deployed.

In case of Amazon EC2 [2], the application was shipped into the instance as a single zip file along with a copy of java, pdfs and models, while in the case of Windows Azure [3], we made use of available plugins to package java, pdfs and models. A local system similar to the cloud instances [4] was utilized for this evaluation. Results are shown in table 5.1

Infrastructure	Average Time (sec/doc)
Local	1.55
Amazon EC2	2.25
Azure	3.6

Table 5.1 Performance in the Cloud

Amazon EC2 instance provides the closest performance to the local deployment, while the performance on Azure indicates that there maybe optimizations to be made to improve performance.

5.2 Goals

There are several approaches to scaling systems and in particular information systems in the cloud. To better identify the most appropriate choice, we study constraints placed on SeerSuite components due to the overall design goals of SeerSuite. These are closely identified with the requirements for SeerSuite itself.

5.2.1 Management

The most effective approach to management of a component or process would be to allow the system to function in an automatic manner, with manual intervention required only in anomalous situations.

5.2.2 Flexibility

There is a constant need for adding new features and improving the existing features. In the present scenario, components such as the metadata extractor would need to undergo large scale modifications to support new features.

5.2.3 Heterogeneous Environment and Portability

One of the main requirements for SeerSuite is the hosting of its components in environments with various operating systems, storage and access.

[2] Medium Extra Large: 4 CPU (8 ECU), 15 GB RAM, 8 GB Storage
[3] Extra Large: 8 Cores, 14 GB, 2 GB Storage
[4] 8 Cores, 16 GB RAM, 1 TB Storage

5.2.4 Reliability

Reliability of the component and its ability to function in case of failures within the component or in the supporting components is crucial, to maintain uninterrupted service to users.

5.2.5 Discussion

With these goals in context, we also identify constraints placed on the metadata extraction system due to its design. Among these is the number of model files and dictionaries being utilized which would need to be distributed along with the extractor. The input to the extractor is a large number of small files, in the PDF/PS format.

These requirements and constraints mean that MapReduce implementations such as the Hadoop framework cannot be utilized for metadata processing. The requirement for reliability and portability mean that the system cannot be based on scale up, but may also need to take advantage of scaling out. This discussion thus guides us to the use of message oriented middleware.

5.3 Message Oriented Middleware and Extraction

While the issues related to flexibility and portability are addressed with the use of the new extractor, the issue of scalability still needs to be addressed. The performance of the system has been marginally improved, however the throughput of the system remains constrained with the same limitations of the existing extractor. To address these issues, we make use of a message oriented middleware system, utilizing a publish subscribe pattern.

Message oriented middleware (3; 11) offers several advantages in building distributed systems involving heterogeneous components, networks and infrastructure. We were influenced by the use of grid based service oriented middleware in WSPeer (29) for message exchanges, Meghdoot (24) for content based publish/subscribe in Peer to Peer environments, of particular interest is the discussion on software engineering and middleware presented in (13). Issues related to scalability and heterogeneity discussed in the paper have been addressed in our work by utilizing queue servers with the properties of replication, persistence and availability in AMQP based queue servers and using popular formats such as JSON to marshal messages. Message oriented middleware is already part of SeerSuite and CiteSeer^x deployment,a pre-existing messaging framework for interaction between components of the crawler and a legacy messaging system for interacting with the crawler through JMS already exists as part of SeerSuite.

Message oriented middleware has been utilized for applications as diverse as cyberinfrastructure in TeraGrid (48), sensor networks (49) and supply chain management (8). In addition the publish/subscribe (14) model allows for loose coupling and scalability which are suited to the goals identified for SeerSuite. Addition of new metadata extractors (41; 5) as subscribers, grouping extractors is possible without modification to the existing pipeline.

5.3.1 Wrapper

The proposed metadata extraction service by itself offers limited ability to interact with a queue or other interfaces. We design a wrapper which provides the metadata extractor with methods to interact with interfaces. The wrapper design does not limit the operations of the extractor in any manner, the extractor could be utilized in the same fashion as the existing extractor.

5.3.1.1 REST Module

The wrapper provides common REST API functionality to the extractor. While the GET method in many cases could already be part of the pdf converter, a POST method for uploading processed documents is included in the wrapper.

5.3.1.2 Subscriber and Queue Client

To interact with queue's the wrapper includes client functionality for AMQP and Amazon SQS. These interfaces allow the extractor to be a consumer for queues.

5.4 Workflow and Deployment with Wrapper

The use of the wrapper enables the metadata extraction system to interact with message oriented middleware A typical deployment is shown in figure 5.1.

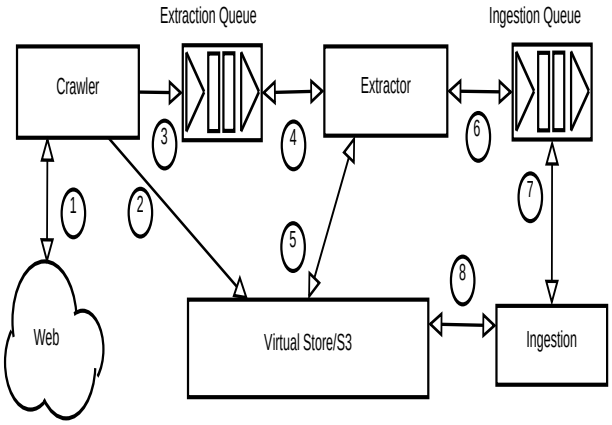

Fig. 5.1 Deployment Architecture

The workflow of the process proceeds as follows. The crawler on a successful identification and fetching of a document from the web (steps 1 and 2), posts a message to the extractor queue (step 3). This message either points to the URL of the document or the URL at the virtual storage. The extractor then consumes messages (step 4) from the queue and processes the document (step 5), generating an ingestable document on success. A message of completion (step 6) is posted onto the ingestion queue, which then handled by the ingestion system (steps 7 and 8).

5.4.1 Deployment with Wrapper

For the purpose of evaluation, we deployed the extractor with the wrapper across physical systems, with a single AMQP [5] queue server hosting a topic exchange. The message template for the deployment is described in 5.2. The message format provides flexibility and allows the message to identify the output location for a document. This is particularly relevant as the extractor can be deployed in different environments, which include deployments on physical servers, computations on physical servers with virtual/cloud storage, cloud deployment with physical storage, cloud deployment with cloud storage and combinations of physical and cloud based deployment with local and cloud based storage.

The message includes, a description of the resource to be processed in the form of an URL or location in the physical storage. The destination in a similar fashion could represent either a local file location or a virtual/cloud storage location. The destination and destination type allow us to use virtual/cloud storage and different service oriented approaches (WSDL,REST) to interacting with the storage. In our deployment, we serialized the message submitted to the queue as a JSON string.

ID
URL (input)
Destination
Destination End Point (output)
Destination Type

Table 5.2 Message Template at the Extractor Queue

5.4.2 Shared Storage

The motivation for operating the metadata extractor in this mode is the presence of a shared storage system such as cluster storage or network storage device. The messaging system in this case is used to coordinate various extractors and serves as a buffer between the crawler, metadata extraction system and other components in the pipeline (Figure 5.2). By utilizing the message template discussed earlier, we can select the extractor output location at the crawler.

This mode does not take advantage of most features of message oriented middleware other than the coordination between independent extractors, which could be accomplished using other mechanisms. The performance in this case is also marginally worse than that of the standalone extractor without the wrapper system in place. Operational limitations restrict the extractor to processing sets which can be comfortably hosted locally or on the shared storage.

The table 5.3 shows the performance of the metadata extractor in this mode of operation, both when compute and storage are in the cloud host (extra large instance)

[5] `http://www.amqp.org/`,`http://www.rabbitmq.com/`

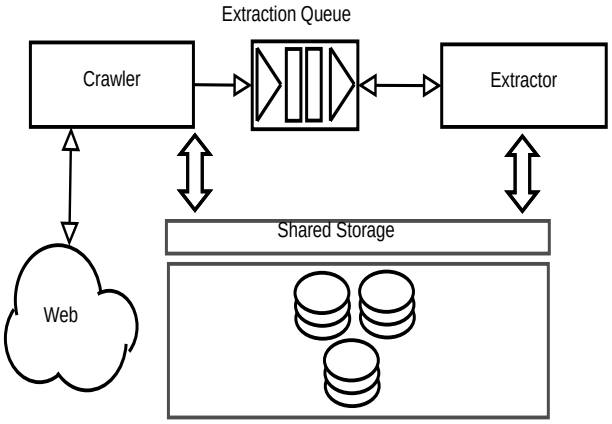

Fig. 5.2 Local or Shared Storage

or hosted on physical hardware/operating system with other SeerSuite components such as the crawler. The AMQP queue server was hosted on physical systems. During the evaluation, we emulated a crawler by a simple client, generating crawl messages of with different input, output locations on demand.

Scenario	Input Location	Computation	Output Storage	Avg. Time (s)
Local	Same Host	Physical	Same Host	1.57
Cloud	Same Host	Cloud	Same Host	2.28

Table 5.3 Local Storage

5.5 Processing with Message Oriented Middleware

We consider the case of deploying the extractor in an distributed environment, where extractors make use of a virtual storage device for processing incoming documents. The virtual storage server (discussed in the previous chapter) in this context provides methods such as GET, PUT, DELETE for storing documents using a REST interface. A more detailed discussion of the virtual storage system has been presented in (52).

The use of a virtual storage system is enabled by features available as part of the wrapper. The documents to be processed are fetched from the virtual storage, once the document is processed it is again stored with the output in the virtual storage. This enables many extractors to function at the same time without the need for a shared storage system. This also frees the extractor from being needs to be modified.

With these we can examine the various scenarios for operating the metadata extractor.

5.6 Scenarios of Operation

To take advantage of the features introduced with the virtual storage, middleware and multiple extractors, We consider deployment scenarios with multiple extractors working in parallel. This is done by distributing the extractors across several systems both physical servers and in the cloud as shown in figure 5.3 and 5.4. The systems can also be distributed by locating one instance on a physical server and another in the cloud, resulting in a hybrid deployment.

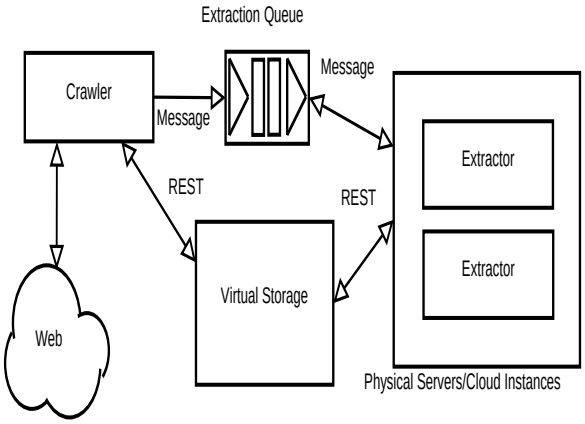

Fig. 5.3 Operating Modes (a) Cloud or Physical

5.6.1 Existing physical infrastructure

In this scenario, the extractor instances are located on physical servers. An AMQP server and virtual storage are utilized. This mode allows us to take advantage of existing infrastructure, without recourse to deploying instances in the cloud.

5.6.2 Cloud infrastructure

As the needs of extraction grows beyond the capacity of our present infrastructure, cloud based instances can be utilized. In this case, we host the virtual storage and queue locally and the extractor instances are hosted in the cloud.

5.6.3 Hybrid infrastructure

A combination of instances hosted in the cloud and locally on physical servers, serves the need of dynamic workloads. In this scenario, while a instance can be maintained in physical servers, instances can be initialized in the cloud on demand. This allows the extraction system to meet dynamic workloads as a result of crawl or submission, more efficiently. This mode of operation is enabled by using a local queue server, virtual storage.

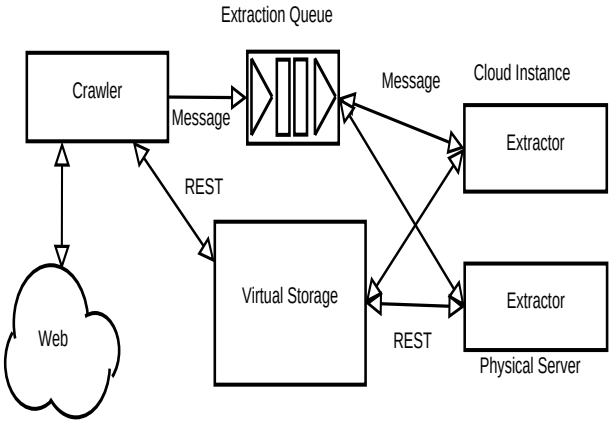

Fig. 5.4 Operating Modes (b) Hybrid

5.6.4 Evaluation

With the infrastructure scenarios for physical, cloud and hybrid infrastructure in mind, we evaluate the performance of the extractor system illustrated in Table 5.4 (T/Doc represents time consumed per document). In case SQS were used, we made use the message visibility feature to avoid processing of the same message by multiple extractors (by marking a message being consumed by an extractor as hidden).

Scenario	Hosts	Input Host	Compute	Storage	T/Doc (s)
Local	2	Web Host	Local	Virtual	10.53
Cloud	2	Web Host	Cloud	Virtual	14.58
Hybrid	2	Web Host	Local, Cloud	Virtual	9.96

Table 5.4 Average time consumption in various modes of operation

From the evaluation the operation of the hybrid service has the best performance of the combinations. This is marginally (5%) better than the distribution over physical servers. The cloud based instances have the worst performance which can be explained by network latencies. It is also clear from the evaluation that the overall performance of the system is much worse than the performance of the standalone mode or existing extractors. This leads us to examine aspects of the extractor which we can take advantage of for further improvements in performance.

5.7 Threading

From the evaluation of the scenarios of operation, we realize that the many of the issues related to performance are linked to I/O operations, shared objects, the network and related issues. In the scenarios we have examined, the individual extractors operated

as independent processes. We can take advantage of threading in the application, possible with the new extractor to boost performance.

With this in focus, we examined the utilizing a thread pool pattern for the extractor. The use of thread pools is guided by the fact that while the extractor has been simplified, it still has a large memory footprint (due to classifiers and model files). We limit the number of active threads by the use of a thread pool.

We evaluate the performance of the system with threading in the same scenarios. Results are showing in Table 5.5 (T/Doc represents time consumed per document) and Figure 5.5.

Scenario	Hosts	Input Host	Compute	Storage	T/Doc (s)
Local	2	Web Host	Local	Virtual	0.67
Cloud	2	Web Host	Cloud	Virtual	0.59
Hybrid	2	Web Host	Local, Cloud	Virtual	0.30

Table 5.5 Average time consumption in various scenarios with threading

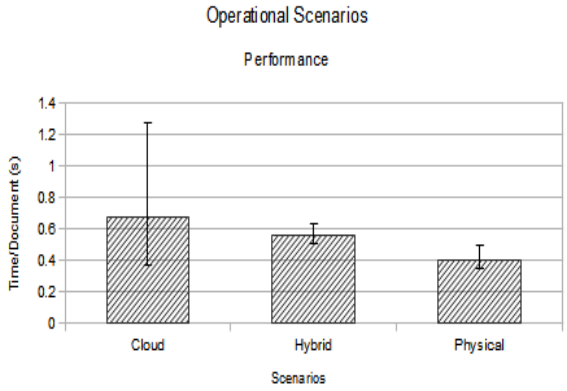

Fig. 5.5 Performance in different scenarios

It is apparent that there has been tremendous improvement in the operation of the extraction performance when threading is used along with scale out. The wide distribution of the throughput also points to issues with network issues.

We examine the scaling capabilities of the metadata extractor with different number of instances to identify improvements in performance.

We note that the performance of the distributed system for extraction in Table 5.6 and Figure 5.6 scales well with increase in the number of instances.

Instances/Systems	Time/Document (s)
2	0.67
4	0.34
8	0.26
16	0.12

Table 5.6 Average time consumed utilizing multiple instances

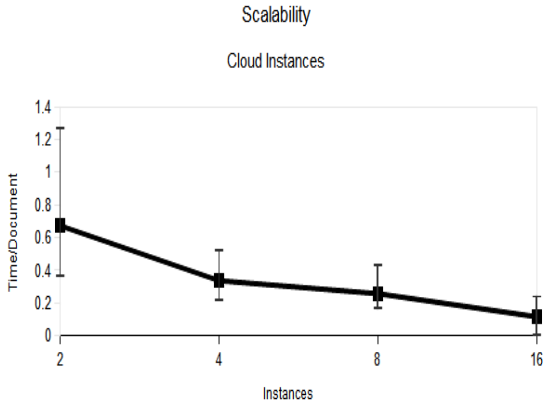

Fig. 5.6 Scalability across instances

5.8 Lessons Learned

We outline some of the several lessons learned in our experience in scaling the metadata extraction system. These focus on the designing a scalable system, reliability of the system.

5.8.1 Scaling

While the choice of threading seems obvious, the approach to threading in the case of metadata extraction needs to be considered carefully. In our implementation of the reference metadata extraction system we utilize model, dictionary files. The threading process has to pre load the model files into shared variables to avoid the penalty of loading these files in each thread. A naive implementation of threading will not result in any improvement in throughput. An approach to threading the extractor has to consider the individual modules and their dependencies (model used) within the application and on external entities (models) to be successful.

5.8.2 Reliability

The use of third party libraries, the processing of documents obtained from the web can result in the application becoming unstable due to failures in the libraries or while processing malformed or corrupt documents, suitable exception handling methods

are essential. In addition by distributing the workload across several systems can allow us to process documents, even when one of the application instances is blocked or failed. This supports the use of scale out along with scaling up for improving throughput, while maintaining reliable and robust operations.

5.9 Summary

We identified the need for scaling the metadata extractor by evaluating its performance in a stand alone mode. We examined the constraints and goals placed on the design of the distributed extraction system. With these goals and constraints in context, we proposed a message oriented middleware system for distributing extraction. We evaluated the performance of the system across local and cloud infrastructure. By utilizing threading, we were able to improve the performance of the system significantly. We also discuss some of the most important lessons learned during the design and implementation of the extraction system.

Chapter 6

Conclusions

We presented the architecture and design of SeerSuite, an open source digital library framework. SeerSuite, its components and its workflow were discussed in detail. SeerSuite by itself is an improvement on CiteSeer and progress towards building a reliable, robust information retrieval system. Its attributes such as loose coupling of components, service oriented architecture, state of art machine learning methods enable researchers and users in building information retrieval systems including the whole or components of SeerSuite. As such, the lessons learned in building SeerSuite are valuable far beyond its instances. Our experiences with SeerSuite and an understanding of the limitations of SeerSuite motivated us to examine infrastructure and components whose design would extend scalability of SeerSuite instances.

Features of such as reduced maintenance requirements while providing the ability to handle dynamic workloads, the pay as you go model make cloud infrastructure attractive to hosting instances of SeerSuite. We studied the feasibility of hosting SeerSuite instances in the cloud by profiling the components and identifying the costs, effort required for such a migration. While the cost of hosting an instance in the cloud was prohibitive, we show that hosting instances of SeerSuite during peak load conditions, partitioning content across application and data components can be useful for taking advantage of cloud infrastructure.

Instances of SeerSuite are limited by the existing repository architecture. Building large scale collections requires expensive storage subsystems. To overcome this challenge, we virtualized the repository storage system, distributing the storage across systems and utilizing a REST interface to interact with other components of SeerSuite. We evaluated this storage system and found that it improves on the services provided by the present system.

The metadata extraction system suffers due to its design and code complexity limiting the extensibility, portability and scalability of the entire instance. We designed and developed a metadata extraction framework, and a reference implementation to address these issues. The quality of the metadata extracted is comparable to the existing system, while the complexity of the code, dependencies have been greatly reduced.

While we addressed the extensibility and portability of the metadata extraction system, the question of scaling the metadata extraction system to meet user needs to be addressed. We discuss the constraints and proposed a message oriented middleware, publish subscribe topic exchange approach to scaling out the metadata extraction system. We significantly improve the throughput of the system by using threading.

Our contributions to understanding SeerSuite, its workflow. The study of migrating SeerSuite instances to the cloud, addressing limitations in the repository and

metadata extraction system should greatly enhance the scalability, extensibility of SeerSuite instances.

Bibliography

[1] Abramson, N. *Information theory and coding*, vol. 61. McGraw-Hill New York, 1963.

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28* (2009).

[3] Banavar, G., Chandra, T., Strom, R., and Sturman, D. A case for message oriented middleware. *Distributed Computing* (1999), 846–846.

[4] Beel, J., Gipp, B., Shaker, A., and Friedrich, N. Sciplore xtract: Extracting titles from scientific pdf documents by analyzing style information (font size). *Research and Advanced Technology for Digital Libraries* (2010), 413–416.

[5] Bhatia, S., Tuarob, S., Mitra, P., and Giles, C. An algorithm search engine for software developers. In *Proceeding of the 3rd international workshop on Search-driven development: users, infrastructure, tools, and evaluation* (2011), ACM, pp. 13–16.

[6] Borthakur, D. The hadoop distributed file system: Architecture and design. *Hadoop Project Website* (2007).

[7] Campbell, R., Gupta, I., Heath, M., Ko, S., Kozuch, M., Kunze, M., Kwan, T., Lai, K., Lee, H., Lyons, M., et al. Open CirrusTM Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)* (2009).

[8] Chen, T., Yu, X., Zhang, Q., and Wang, J. Scm-oriented dynamic service architecture and collaborative application for internet of things. In *Advances in Electronic Engineering, Communication and Management Vol.1*, D. Jin and S. Lin, Eds., vol. 139 of *Lecture Notes in Electrical Engineering*. 2012, pp. 455–462.

[9] Councill, I. G., Giles, C. L., Iorio, E. D., Gori, M., Maggini, M., and Pucci, A. Towards next generation citeseer: A flexible architecture for digital library deployment. In *Research and Advanced Technology for Digital Libraries, ECDL 2006* (2006), pp. 111–122.

[10] Councill, I. G., Li, H., Zhuang, Z., Debnath, S., Bolelli, L., Lee, W. C., Sivasubramaniam, A., and Giles, C. L. Learning metadata from the evidence in an on-line citation matching scheme. In *JCDL* (2006), pp. 276–285.

[11] Curry, E. Message-oriented middleware. *Middleware for communications* (2004), 1–28.

[12] Deelman, E., Singh, G., Livny, M., Berriman, B., and Good, J. The cost of doing science on the cloud: the montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), pp. 1–12.

[13] Emmerich, W. Software engineering and middleware: a roadmap. In *Proceedings of the Conference on The future of Software engineering* (2000), ACM, pp. 117–129.

[14] Eugster, P., Felber, P., Guerraoui, R., and Kermarrec, A. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR) 35*, 2 (2003), 114–131.

[15] Feitelson, D., and Yovel, U. Predictive ranking of computer scientists using citeseer data. *Journal of Documentation 60*, 1 (2004), 44–61.

[16] Fiala, D. Mining citation information from citeseer data. *Scientometrics 86*, 3 (2011), 553–562.

[17] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[18] Garfield, E. "Science Citation Index"a new dimension in indexing. *Science 144*, 3619 (1984), 649 – 654.

[19] Ghemawat, S., Gobioff, H., and Leung, S. The Google file system. *ACM SIGOPS Operating Systems Review 37*, 5 (2003), 43.

[20] Giles, C. L., Bollacker, K. D., and Lawrence, S. Citeseer: An automatic citation indexing system. In *Digital Libraries* (1998), pp. 89–98.

[21] Ginsparg, P. Can Peer Review be better Focussed. `http://people.ccmr.cornell.edu/~ginsparg/blurb/pg02pr.html`, 2010.

[22] Gray, J. Distributed computing economics. Tech. rep., Microsoft Research, 2003.

[23] Grossman, R., and Gu, Y. Data mining using high performance data clouds: experimental studies using sector and sphere. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), ACM, pp. 920–927.

[24] Gupta, A., Sahin, O., Agrawal, D., and Abbadi, A. Meghdoot: content-based publish/subscribe over p2p networks. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware* (2004), Springer-Verlag New York, Inc., pp. 254–273.

[25] Halili, E. Apache JMeter.

[26] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter 11*, 1 (2009), 10–18.

[27] Han, H., Giles, C. L., Manavoglu, E., Zha, H., Zhang, Z., and Fox, E. A. Automatic document metadata extraction using support vector machines. In *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries* (2003), pp. 37–48.

[28] Han, H., Manavoglu, E., Zha, H., Tsioutsiouliklis, K., Giles, C. L., and Zhang, X. Rule-based word clustering for document metadata extraction. In *SAC* (2005), pp. 1049–1053.

[29] Harrison, A., and Taylor, I. Service-oriented middleware for hybrid environments. In *Proceedings of the 1st international workshop on Advanced data processing in ubiquitous computing (ADPUC 2006)* (New York, NY, USA, 2006), ADPUC '06, ACM, pp. 2–.

[30] Hongjun, Y. Innovation of digital reference service model in the cloud computing environment [j]. *Information Studies: Theory & Application 1* (2010).

[31] Huang, J., Ertekin, S., and Giles, C. L. Efficient name disambiguation for large scale databases. In *The 10th European Conference on Principles and Practice of Knowledge Discovery in Databases* (2006), pp. 536–544.

[32] Isaac Councill, C. L. G., and Kan, M.-Y. Parscit: an open-source crf reference string parsing package. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)* (Marrakech, 2008), European Language Resources Association.

[33] Joachims, T. Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98* (1998), 137–142.

[34] Kahn, R., and Wilensky, R. A framework for distributed digital object services. *International Journal on Digital Libraries 6*, 2 (2006), 115–123.

[35] Kataria, S., Mitra, P., and Bhatia, S. Utilizing context in generative bayesian models for linked corpus. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (2010).

[36] Khajeh-Hosseini, A., Sommerville, I., Bogaerts, J., and Teregowda, P. Decision support tools for cloud migration in the enterprise. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on* (2011), IEEE, pp. 541–548.

[37] Lafferty, J., McCallum, A., and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[38] Li, H., Councill, I., Lee, W.-C., and Giles, C. L. Citeseerx: an architecture and web service design for an academic document search engine. *Poster Session 15th International World Wide Web Conference* (2006).

[39] Li, H., Lee, W., Sivasubramaniam, A., and Giles, C. Workload analysis for scientific literature digital libraries. *International Journal on Digital Libraries 9*, 2 (2008), 139–149.

[40] Liu, Y., Bai, K., Mitra, P., and Giles, C. L. Tableseer: automatic table metadata extraction and searching in digital libraries. In *JCDL* (2007), pp. 91–100.

[41] Liu, Y., Mitra, P., Giles, C., and Bai, K. Automatic extraction of table metadata from digital documents. In *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries* (2006), ACM, pp. 339–340.

[42] Mika, P., and Tummarello, G. Web semantics in the clouds. *IEEE Intelligent Systems* (2008), 82–87.

[43] Mitra, P., Giles, C. L., Sun, B., and Liu, Y. Chemxseer: a digital library and data repository for chemical kinetics. In *CIMS '07: Proceedings of the ACM first workshop on CyberInfrastructure: Information Management in eScience* (2007), pp. 7–10.

[44] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid-Volume 00* (2009), IEEE Computer Society, pp. 124–131.

[45] Pinto, H., Staab, S., Tempich, C., and Sure, C. Y. Semantic web and peer-to-peer.

[46] Podlipnig, S., and Böszörmenyi, L. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR) 35*, 4 (2003), 374–398.

[47] Singh, A., Srivatsa, M., and Liu, L. Search-as-a-service: Outsourced search over outsourced storage. *ACM Trans. Web 3*, 4 (2009), 1–33.

[48] Smith, W. An information architecture based on publish/subscribe messaging. In *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery* (2011), ACM, p. 27.

[49] Souto, E., Guimarães, G., Vasconcelos, G., Vieira, M., Rosa, N. S., and Ferraz, C. A. G. A message-oriented middleware for sensor networks. In *Middleware for Pervasive and Ad-hoc Computing* (2004), pp. 127–134.

[50] Tan, Q., Mitra, P., and Giles, C. Metadata extraction and indexing for map search in web documents. In *Proceeding of the 17th ACM CIKM* (2008), pp. 1367–1368.

[51] Teregowda, P., Urgaonkar, B., and Giles, C. L. Citeseer^x: A cloud perspective. *HotCloud 2010, 2nd USENIX Workshop on Hot Topics in Cloud Computing* (2010).

[52] Teregowda, P., Urgaonkar, B., and Giles, C. L. Cloud computing: A digital libraries perspective. *Cloud Computing, IEEE International Conference on 0* (2010), 115–122.

[53] Teregowda, P. B., Councill, I. G., Fernandez, J. P. R., Kasbha, M., Zheng, S., and Giles, L. C. Seersuite: Developing a scalable and reliable application framework for building digital libraries by crawling the web. In *USENIX Conference on Web Application Development* (2010).

[54] Treeratpituk, P., and Giles, C. Disambiguating authors in academic publications using random forests. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries* (2009), ACM, pp. 39–48.

[55] Vinoski, S. Rest eye for the soa guy. *IEEE Internet Computing* (2007), 82–84.

[56] Walker, E., Brisken, W., and Romney, J. To lease or not to lease from storage clouds. *Computer 43* (2010), 44–50.

[57] Wood, T., Cecchet, E., Ramakrishnany, K., Shenoy, P., van der Merwey, J., and Venkataramani, A. Disaster recovery as a cloud service: Economic benefits & deployment challenges. In *2nd USENIX Workshop on Hot Topics in Cloud Computing* (2010).

[58] Zhu, T., Gandhi, A., Harchol-Balter, M., and Kozuch, M. Saving cash by using less cache. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)* (2012).

Vita
Pradeep Teregowda

Education

The Pennsylvania State University University Park, Pennsylvania 2008–Present
Ph.D. in Computer Science and Engineering, expected in December 2012

The Pennsylvania State University University Park, PA 2001–2004
Masters in Electrical Engineering and High Performance Computing

University of Mysore Mysore, India 1996–2000
Bachelors in Electrical and Electronics Engineering

Research Experience

Doctoral Research The Pennsylvania State University 2008–Present
Dissertation Advisor: Prof. C. Lee Giles

Graduate Research The Pennsylvania State University 2001–2004
Research Advisor(s): Prof. Constantino Lagoa, Prof. C. Lee Giles, Nirmal Pal, Prof. Arvind Rangaswamy

Undergraduate Research Mysore University 1999–2000
Research Advisor: A. S. Arvindamurthy

Publications

Pradeep Teregowda, Madian Khabsa, Clyde Giles, A System for Indexing Tables, Algorithms and Figures. JCDL 2012.

Jian Wu, C. Lee Giles, Pradeep Teregowda, Juan Pablo Fernndez Ramrez and Prasenjit Mitra. A Study of the Crawling Strategy Evolution for Academic Document Search Engines. WebSci 2012.

Ali Khajeh-Hosseini, Ian Sommerville, Jurgen Bogaerts, Pradeep B. Teregowda, Decision Support Tools for Cloud Migration in the Enterprise. IEEE CLOUD 2011.

Pradeep B. Teregowda, Bhuvan Urgaonkar, C. Lee Giles: Cloud Computing: A Digital Libraries Perspective. IEEE CLOUD 2010.

Pradeep B. Teregowda, Isaac G. Councill, Juan Pablo Fernndez Ramrez, Madian Khabsa, Shuyi Zheng, C. Lee Giles. USENIX WebApps 2012.

Pradeep Teregowda, Bhuvan Urgaonkar, C. Lee Giles, CiteSeerx: A Cloud Perspective, USENIX HotCloud 2010.

Pucktada Treeratpituk, Pradeep Teregowda, Jian Huang, and C. Lee Giles, SEERLAB: A System for Extracting Keyphrases from Scholarly Documents, SemEval 2010.

Yves Petinot, C. Lee Giles, Vivek Bhatnagar, Pradeep B. Teregowda, Hui Han, Isaac G. Councill, CiteSeer-API: towards seamless resource location and interlinking for digital libraries, CIKM 2004.

Yves Petinot, C. Lee Giles, Vivek Bhatnagar, Pradeep B. Teregowda, Hui Han, Isaac G. Councill, A service-oriented architecture for digital libraries, ICSOC 2004.

Yves Petinot, C. Lee Giles, Vivek Bhatnagar, Pradeep B. Teregowda, Hui Han, Enabling interoperability for autonomous digital libraries: an API to citeseer services, JCDL 2004.

Yves Petinot, Pradeep B. Teregowda, Hui Han, C. Lee Giles, Steve Lawrence, Arvind Rangaswamy, Nirmal Pal, eBizSearch: An OAI-Compliant Digital Library for eBusiness, JCDL 2003.

C. Lee Giles, Yves Petinot, Pradeep B. Teregowda, Hui Han, Steve Lawrence, Arvind Rangaswamy, Nirmal Pal, eBizSearch: a niche search engine for e-business, SIGIR 2003.