

# CiteSeer<sup>x</sup> - A Scalable Autonomous Scientific Digital Library

Huajing Li<sup>1</sup> Isaac G. Councill<sup>2</sup> Levent Bolelli<sup>1</sup> Ding Zhou<sup>1</sup> Yang Song<sup>1</sup>

Wang-Chien Lee<sup>1</sup> Anand Sivasubramaniam<sup>1</sup> C. Lee Giles<sup>1,2</sup>

<sup>1</sup>Department of Computer Science and Engineering

<sup>2</sup>The School of Information Sciences and Technology

Pennsylvania State University

State College, PA 16802, USA

{huai, bolelli, dzhou, yasong, wlee, anand}@cse.psu.edu,

{icouncil, giles}@ist.psu.edu

**Abstract**—CiteSeer is a scientific literature digital library and search engine which automatically crawls and indexes scientific documents in the fields of computer and information science. Since its inception in 1997 CiteSeer has grown to index over 730,000 documents and serves over 800,000 requests daily, pushing the limits of the current system's capabilities. In addition, CiteSeer's monolithic architecture inconveniences system maintenance and reduces the flexibility of the system in terms of new feature development, algorithm updates, and system interoperability. In this paper, we discuss the problems of the current CiteSeer architecture and propose a new architecture for a next generation CiteSeer application. The new architecture is based on modular web services and pluggable service components. Preliminary results based on a prototype system show the new architecture enhances flexibility, scalability, and performance for CiteSeer. In addition, new services in development for the next generation CiteSeer system are discussed.

## I. INTRODUCTION

CiteSeer has arisen as a web-based scientific literature digital library and search engine that focuses primarily on the literature in the fields of computer and information science. The hallmark feature of CiteSeer is Autonomous Citation Indexing (ACI) [1], which focuses on extracting citation information from scholarly publications in electronic format. CiteSeer automatically discovers and retrieves online scientific documents. Upon the acquisition of new contents, the new documents are parsed in order to extract citations information and other document metadata (such as titles, authors, abstracts, etc.). Documents and citations are indexed, and a standard search engine query interface is provided to receive incoming requests for system content.

From its birth in 1997 until now, CiteSeer has grown into a collection of over 730,000 documents with over 8 million citations. CiteSeer receives over 800,000 hits daily, is accessed by over 100,000 unique users monthly, and serves approximately 30 gigabytes of data daily. However, rising demands from system use and the increasing size of CiteSeer's archive are causing query latencies to rise as well as significant degradation of system stability. CiteSeer suffers from design deficiencies. The most obvious problems are its lack of scalable storage and transaction-safe updates, which

bring down the system's performance as well as its stability significantly. These problems motivate our research interest in pursuing a new architecture which naturally provides scalable storage and transaction-safe processes.

Current internal design problems also increase the system's maintenance cost. CiteSeer is monolithic, making the system hard to administer, configure, and modify. Not only does CiteSeer serve as a large digital library application, but as an *automatic* library it also provides an excellent platform for relevant research (such as data mining, information retrieval, etc). With the support of CiteSeer's architecture as well as its resources, algorithms and techniques can be developed and tested in a real-world digital library application. However, the current system architecture discourages such use since incorporation of new modules and algorithms requires significant labor.

The introduction of a modular CiteSeer architecture offers a great opportunity to exploit newly emerged web technologies to improve the system's performance and configurability. Accordingly, the system will become more powerful and robust: more simultaneous transactions can be supported; tasks like system monitoring and logging will be more convenient. The next generation CiteSeer is far more than a debugged version of CiteSeer. It is a system with new elements: new services, new resources, and new data models, all working in a new framework.

Generally speaking, our goal is the development of a new CiteSeer architecture, in which current services and modules will be selectively integrated. In summary, we design the system to be:

**Scalable**, in terms of the amount of processed content as well as the number of simultaneous transactions the system can handle.

**Flexible**, or easily reconfigurable, making it a good platform for conducting research and maintaining the system.

**Self-adaptive**, which can proactively detect and fix system errors, automatically balance server loads and autonomously improve the quality of services.

**Service-based**, which provides application-level service

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

INFOSCALE '06. Proceedings of the First International Conference on Scalable Information Systems, May 29-June 1 2006, Hong Kong

© 2006 ACM 1-59593-428-6/06/05...\$5.00

APIs as well as web interfaces.

**User-oriented**, in order to better serve the research community and offer a more open environment for academic use.

In this paper, we present our efforts to construct the next generation CiteSeer - *CiteSeer<sup>x</sup>*. Our contribution can be summarized as four-fold:

- 1) We propose a new data model, which is fundamentally different from that of the old CiteSeer. The new model meets our requirement of data flexibility. We also provide solutions to make our storage strategy both scalable and efficient.
- 2) Based on the new data model, we present a new CiteSeer architecture, which aims to overcome current system design drawbacks.
- 3) Some features and services of the new system are introduced, which gives an overview of the next generation academic digital library.
- 4) A prototype system is setup and related experiments are performed. Evaluation results suggest the new architecture works well under various settings, providing both good scalability and query efficiency.

The rest of the paper is organized as follows. In Section II, we briefly review some related research in ACI systems and digital libraries. An analysis of the old CiteSeer system, including the implemented architecture and problems, is presented in Section III. We give the description of the new data model in Section IV. After that, system architecture of *CiteSeer<sup>x</sup>* is presented in Section V. Some implementation details are given in Section VI. We present a prototype system and analyze our experimental results in Section VII. Section VIII shows features and services of the new system. Finally, we present our concluding remarks and future plans in Section IX.

## II. RELATED WORK

Since the introduction of CiteSeer [2] as the pioneer in the field of Autonomous Citation Indexing and one of the most popular digital libraries in computer science, the amount of scientific papers on the web has increased significantly, creating the need for an efficient and scalable system architecture. Namely, it is highly desired to construct large-scale digital library systems, which can provide interoperability, extensibility, high performance and security under a distributed and heterogeneous environment. A number of specialized [3], [4], [5], [6] and generic [7], [8], [9] public search engines and digital libraries has emerged to facilitate the access to scientific publications, including Arxiv, ScienceDirect and Corr. Whether centralized [3], [7], [10] or de-centralized [11], [12], every digital library architecture employs modularization of their services to achieve scalability.

The attempts to organize heterogeneous systems or distributed resources using a specific protocol can be taken as first efforts in designing an open architecture. The Simple Digital Library Interoperability Protocol (SDLIP) [13] supports search queries to be performed over multiple information sources. The results can be returned synchronously or streamed when

they are available. In traditional library literature, Z39.50<sup>1</sup> is a widely-used protocol which supports queries on distributed metadata repositories. Recently, the emerging OAI-PMH protocol<sup>2</sup> is adopted in many digital libraries as well as some search engines to improve the system-level interoperability. Remote metadata can be harvested and queried as if they were locally stored. Additionally, in [11], [14], [15], solutions to distributed digital collection integration are proposed.

Meanwhile, a lot of efforts have been performed to find a systematic solution for an open digital library architecture. UC Berkeley Library's GenDL (Generic Digital Library) [16] provides a modular object management environment, which includes a web-based content management system, a preservation repository and an access system. Greenstone [17] and DSpace [18] are outstanding digital library systems which offers users much freedom in customizing digital resources. Greenstone provides users a quick process to distribute digital documents online. The document formats can be extended using plug-ins. Also, classifiers can be added to build indices for customized file formats. DSpace can meet a variety of digital archiving needs. It supports all forms of digital materials and manages related metadata.

Fedora [19], [20] is another open-source digital repository management system, which demonstrates how distributed digital library architecture can be deployed using web-based technologies, including XML and web services. Fedora is more like a reusable middleware than a complete digital library system because the services are published in the format of APIs, which make it easier to be integrated into a digital library application.

In the literature of ACI study, Google recently introduced The Google Scholar<sup>3</sup> that incorporates ACI to index over 560 million documents and citation records<sup>4</sup>. Microsoft's Libra system [21] contains one million papers, 650,000 authors and over 2,000 publication venues. Collectively, their dataset contains 7 million links representing cited-by, authored-by and published-by relationships. SMEALSearch – a niche search engine based on the CiteSeer technology – utilizes ACI for the documents primarily in the fields of business, e-business and related areas. Rosetta [10] is a digital library that indexes publications in the field of computer science. Instead of using the content of documents for indexing, it uses the anchor text from the citing documents to represent each document.

## III. CURRENT CITESEER SYSTEM

CiteSeer has proven its usefulness to the computer science community as a digital archive for research publications and through its autonomous citation indexing feature. However, CiteSeer currently faces significant challenges of interoperability and scalability that must be overcome in order to improve the quality of the services provided and to guarantee that CiteSeer will continue to be a valuable, up-to-date resource

<sup>1</sup><http://www.niso.org/z39.50/z3950.html>

<sup>2</sup><http://www.openarchives.org/OAI/openarchivesprotocol.htm>

<sup>3</sup><http://scholar.google.com>

<sup>4</sup>This estimate was obtained by using "the" as the search term

well into the foreseeable future. The CiteSeer service is currently being made more available to the world community through the advent of several mirrors. At the time of this writing there are CiteSeer mirrors hosted in MIT, Switzerland, Canada, England, Italy, and Singapore in various stages of completion.

CiteSeer consists of three basic components: a focused crawler or harvester, the document archive and related indices, and the query interface [22]. The focused spider or harvester crawls the web for relevant documents in PDF and Postscript formats. After filtering crawled documents for academic documents, these are then indexed using autonomous citation indexing [2], which automatically links references in research articles to facilitate navigation and evaluation. Automatic extraction of the context of citations allows researchers to determine the contributions of a given research article quickly and easily [23]; and several advanced methods are employed to locate related research based on citations, text, and usage information. CiteSeer is a full text search engine with an interface that permits search by document or by numbers of citations or fielded searching, not currently possible on general-purpose web search engines.

The architecture of CiteSeer is depicted in Figure 1. Documents are ingested into the system directly into a file system repository. The raw files are processed and associated data is stored in relevant databases and indices. These storage and index containers are queried through a web application that is integrated with each system component, and obtains document files directly from the local file system. Database and indices can be locked at the table or index level through a centralized lock server during updates. Reads into locked resources are failed.

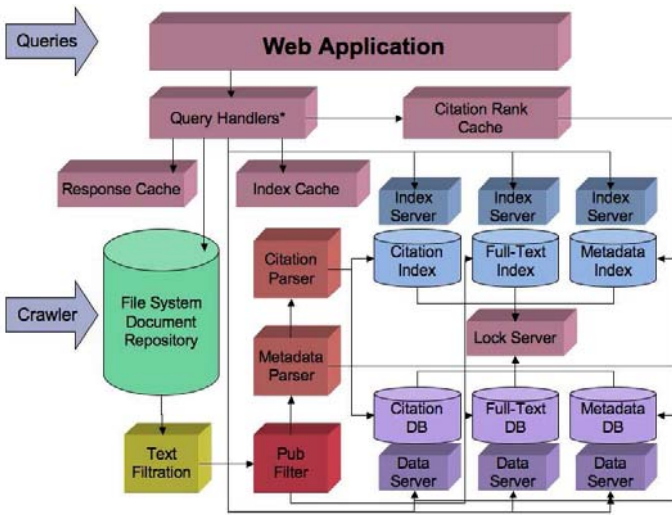


Fig. 1. Current CiteSeer architecture.

#### A. Deployment

The main installation of CiteSeer at Pennsylvania State University is currently deployed using three Dell PowerEdge

servers, each with two processors and 4-6 GB of RAM. Two of the machines serve public CiteSeer traffic and one machine is used for development and data acquisition. Additionally, two low-end single processor machines are used to provide fault-tolerant load balancing between the two public servers. Storage is handled via three Dell PowerVault SCSI disk arrays, each with 2 TB of available storage. Each disk array is connected to a single PowerEdge server and each carries a fully redundant copy of the CiteSeer data.

Figure 2 illustrates the current deployment of CiteSeer. Linux Virtual Server [24] is used to provide fault tolerance and load balancing between the two public servers. All incoming traffic is routed through a single active director node, which hands off the connection to the least loaded real server. The real server responds directly back to the client, avoiding a bottleneck that could occur if both incoming and outgoing traffic were routed through the director. The public machine group is transparent to users as users must only know a single IP address to gain access to the cluster, as is typical in large-scale search engines. Real servers are monitored by the director nodes and are removed from the routing tables if a failure occurs. If the active director fails, the backup director is prepared to immediately take over routing.

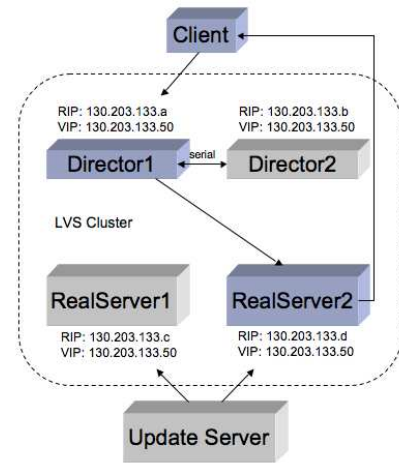


Fig. 2. Deployment of CiteSeer using Linux Virtual Server.

#### IV. DATA MODEL

As discussed in Section III, the core component of CiteSeer's storage is the document repository, in which the primary digital objects are documents, citations, and citation groups. Each document is assigned a document ID, each citation a citation ID and each citation group is assigned a group ID. These objects are stored in separate index spaces. This ID system brings unnecessary overhead in terms of storage, system efficiency, and data integration. CiteSeer<sup>x</sup>, in accordance with our design goals and emerging requirements, employs a new data model.

### A. Extended Storage Scope

The current CiteSeer system can be viewed as a document-centric digital library, which crawls, stores, and indexes text documents. Processing and data modeling flows from these documents as the core system objects. In the new system, the focus is on digital object records rather than text documents. Each harvested document is associated with a document record, and each citation is associated with an individual citation record as well as a related document record, even if the real document referenced by a citation is not present in the repository. Additionally, authors and publication venues are no longer considered as metadata "belonging to" a document, but as peer digital objects that are linked to documents as well as to each other. Figure 3 illustrates the new storage structure in simplified form. Other digital objects that cross the boundaries of individual documents, such as institutional affiliations or acknowledged entities, are stored with the same model of functional separation. Only data that is specific to a single document, such as titles, abstracts, and keywords, are stored within document records.

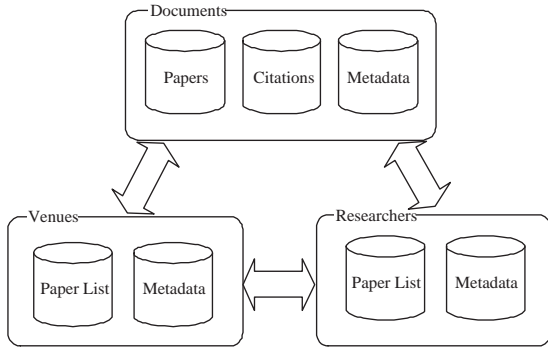


Fig. 3. Community-Centric Storage.

### B. New ID System and Virtual Documents

In CiteSeer<sup>x</sup>, the ID system is simplified for documents. Previously, we maintain citation IDs and citation group IDs for offline citation clustering and matching algorithms. Citations and documents are treated differently in the old CiteSeer system, ignoring the fact that the cited objects are themselves documents. Hence, we only keep document IDs in the new document repository. The citation relationship is kept using links between document records, which we call the *citation graph*. For author and venue repositories, we also employ the one ID per record principle, respectively. This approach has the benefit of separating record matching and linking routines from the storage model. Name and document disambiguation as well as record linkage services can be plugged into the system as fully separate modules.

Two problems arise with the new ID system for the document repository:

- 1) It is possible that a matching document record for a new citation cannot be found in CiteSeer's document repository. It is thus necessary to create a temporary

document record as a placeholder for a "real" document record, that is, a record for a document that is known within the system.

- 2) The existing offline algorithm for matching citations to document records depends on the prior clustering of citations into groups that reference a single document. A new solution is needed to build and maintain the citation graph.

To solve the first problem, we introduce the notion of *virtual document records* into our system. Once a new citation is ingested into the repository, the system queries the existing repository in order to identify a document record that matches the citation. If a match is found, an edge is added to the citation graph. Otherwise, the system creates a virtual document record based on the citation metadata. The record is called *virtual* because the cited document has not been located and stored by CiteSeer. Once the document is crawled and downloaded, another query into the system is performed in order to match the document with any existing virtual document records. If a match is found, the corresponding virtual record is updated with a pointer to the document file, making it a *real* document record. This new organization makes the CiteSeer repository more unified and complete. There are no citation edges pointing to an external unknown resources. All edges are internal in the document database. Real and virtual documents can be searched in the same index space. Virtual document records need not be created by citations alone. There are some other sources for virtual document information, such as the proceeding list of a conference and metadata obtained from other digital libraries. As long as a document record contains only metadata without the corresponding document file, the record is marked as virtual.

An online process has been developed to match citations with documents [25]. In realistic cases, there are often abbreviations and typos in the observed citation metadata. Our algorithm is capable of handling some data inaccuracy. The Lucene indexer<sup>5</sup> is used to match records based on fuzzy queries over the metadata fields in records. If a result is returned whose similarity (based on Levenshtein Distance [26]) reaches a predefined threshold, our algorithm indicates a match between a document and a citation record and creates a new entry in the citation graph. Otherwise, the algorithm determines that no record in the repository matches the new citation. A virtual document record is then created.

### C. Canonical Metadata

In CiteSeer, the identity of a document, author, or venue can be defined as its perfect metadata record. Even in situations where human-generated metadata is available, the data can be incomplete or incorrect. This problem is exacerbated by digital libraries such as CiteSeer, in which the system is responsible for building its own collection without the aid of human-produced metadata. Records in CiteSeer enter the system with information identified by automatic parsing algorithms. For

<sup>5</sup><http://jakarta.apache.org/lucene/>

documents, the information is usually restricted to the title, author names, and any other metadata typically available in document headers. The parsed information is often noisy or incorrect, and almost always incomplete. Citation information can be used to fill in missing information of document records and to correct erroneous data. When citations to a document record are found, the document’s metadata can be updated with information from citations. Information from multiple citations can be fused to form a best guess as to the correct metadata values for the document. This process will be referred to subsequently as the process of building the *canonical* metadata for a document. Similarly, document records can be gradually refined from an online proceeding list of a venue or a publication list of an author. In return, canonical metadata for venue or author records can be inferred from a document’s canonical metadata or a citation. The process of refining document canonical metadata with citations is illustrated in Figure 4.

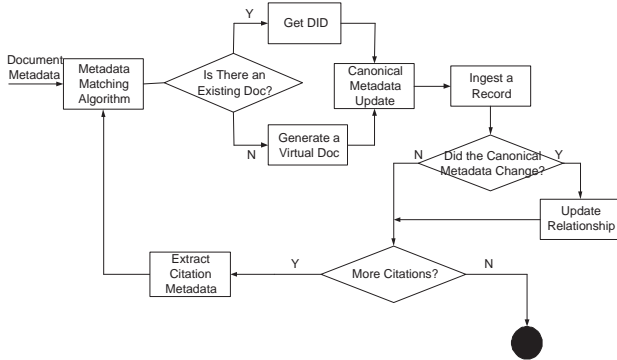


Fig. 4. Online Citation Matching Workflow.

Due to the page limitation, the detailed discussion of our canonical record construction will not be presented in this paper.

#### D. Digital Objects

In the current CiteSeer system, data storage is explicitly exposed to services and algorithms. This approach is problematic in that there is no uniform interface to the underlying objects. The relationship between functional modules and data repositories is tight-coupled. Therefore, if the data definition is changed or extended, for example, a new media type should be managed by the system, the services cannot work properly without significant revision. Accordingly, if it is desirable to install new services into the system, the new modules must be interfaced through custom routines into the system data model.

To make CiteSeer<sup>x</sup> more flexible and extensible in terms of storage types and service types, a new virtual storage layer is introduced into the system, separating physical storage from service access. With the assistance of this layer, a level of abstraction can be defined on top of physical storage. Any modification from the applications will not affect the underlying physical storage, as long as the uniform access interface is followed. Also, actual physical storage can be distributed across multiple machines and sites. Tasks such as

query routing and load balancing can be performed in this layer.

The idea of *digital objects* is proposed in [27], in which a universal container for various resource types and metadata formats is presented. Typically, digital resources are represented according to their media type in most resource repositories. A movie is no more than a movie and a PDF file is simply a PDF file. When a digital resource is obtained, its identity and format can be recognized instantaneously. This approach is convenient in terms of data retrieval and representation. However, it may be desirable to have more flexibility in terms of unified object retrieval or the construction of complex data structures. A direct resource mapping can make such processes and representations impractical due to the necessity of codifying appropriate resource handlers according to media types. The idea of digital object is somewhat similar with the object-oriented idea in programming languages. A digital object is a group of tightly-related digital resources wrapped within a common container, along with the associated distribution methods[19], [20], [27]. Commonly, a description file (written in XML) is used to explain what is included in the object. Conceptually, a digital object is comprised of three parts:

- 1) Data: Multiple streams of data can be stored and inter-related. This feature is powerful for expressing complex data types such as books and papers, which may contain information in various media formats (e.g. text versus image data).
- 2) Methods: Methods define a set of behaviors that can be used for external systems to access or manage the data objects.
- 3) Metadata: Metadata can be viewed as another portion of data. It is special because often in digital libraries users may query metadata attributes stored in separate index slices.

The heterogeneity between different resource types are hidden behind the wall of digital objects. From external systems, all digital objects are identical in character. In summary, the digital object model offers the strengths and advantages of:

**Abstraction:** The object model is identical from outside the system, and various data objects such as papers, authors, or venues can be retrieved via the same access methods.

**Flexibility:** Data access methods and data composition can be designed to best represent the system requirements and modified without greatly affecting the related applications when requirements change.

**Scalability:** Digital Objects do not manage physical storage, which can be located locally, in distributed repositories, or remotely on the web.

#### V. ARCHITECTURE OF CITESEER<sup>x</sup>

Based on the new data model, CiteSeer<sup>x</sup> adopts a new system architecture to overcome the limitations outlined in the previous sections. This section provides details on the components of the new architecture and our solution towards achieving flexibility and scalability within the new system.

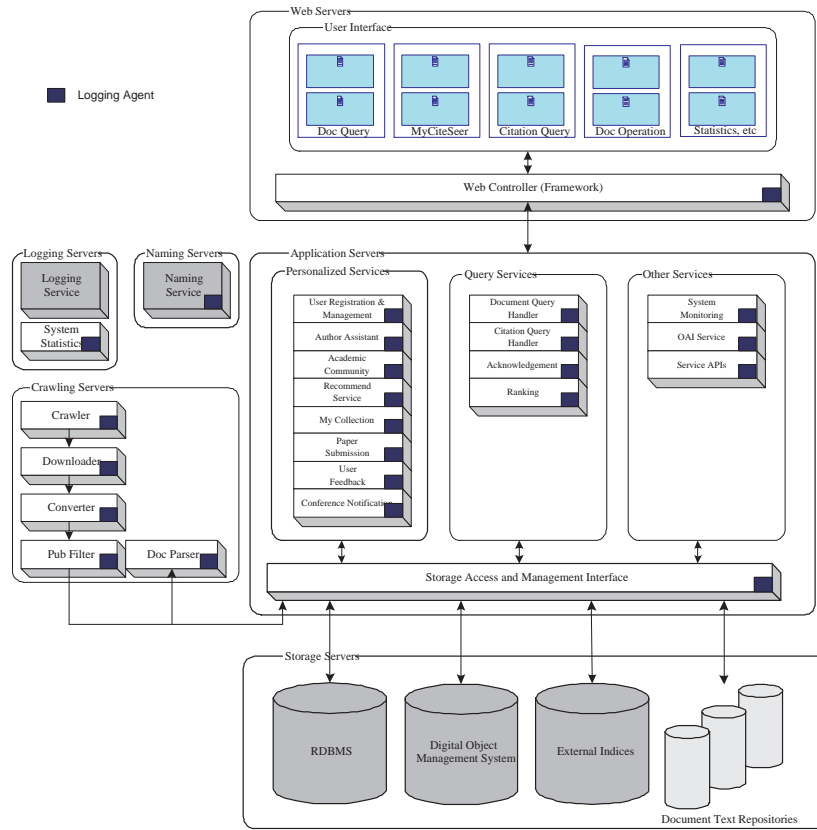


Fig. 5. CiteSeer<sup>X</sup> Architecture.

### A. Architecture Overview

Figure 5 gives an overview of the new architecture. Basically, the system comprises of the following three layers:

1) *Storage Layer:* Storage layer handles the management of and access to locally stored data objects of CiteSeer<sup>X</sup>. These objects are maintained by a digital object management system. Each digital object is accompanied by a description file that contains the metadata of the object. For efficiency purposes, some of the data that are generated from the objects (i.e. citation graph) are stored in databases. The parsed documents are stored in local disks as raw text files, which are linked by digital object records.

2) *Application Layer:* Application layer is the collection of the following function modules and servers in CiteSeer<sup>X</sup>:

- **Naming Server:** The highly distributed architecture of CiteSeer<sup>X</sup> enables us to efficiently utilize multiple servers across the network. A naming server lookup forwards incoming requests to one of CiteSeer<sup>X</sup> servers, which provides to be effective for load balancing among our servers as well as directing requests to the closest server node.
- **Logging Server:** We utilize separate logging services to aggregate and manage system logs. Logging servers are where the logs are created and stored. Each working module of the system has a logging agent which sends logging event to the logging server.

- **Crawling Server:** Crawling processes populate the document repository. Each document that is identified as a publication in the field of computer science is forwarded to the Storage Access and Management Interface for permanent storage.
- **Storage Access and Management Interface:** This layer provides an interface to the storage servers that serves multiple purposes. First, by intermediating the accesses to the storage servers, it provides a data access standard for the methods that directly accesses the repository/index. Second, an access control mechanism regulates the access to the stored information. Third, it acts as a resource location resolver for mapping internal resource identifications to actual storage addresses. This gives us the flexibility to implement distributed storage, which has further benefits of data migration and load balancing across multiple servers.
- **Application Servers:** Application server provides service access entries for web user interfaces as well as external applications. Application servers host 3rd-party software modules and native modules.

3) *User Interface Layer:* This layer provides an abstraction for the web interface of CiteSeer<sup>X</sup> by acting as a gateway between the user interface and application modules. This gives us the flexibility to update application logic without worrying about the user interface as well as providing personalized

services to users.

## B. Collection and Scalability Solution

As described in Section IV, CiteSeer<sup>X</sup> contains three inter-related repositories: documents, authors and venues. Among them, documents are considerably much larger than the other two. Hence, we primarily emphasize on addressing the scalability issues in our document repository in the new architecture.

The documents are distributed across the storage servers to avoid the bottleneck of directing requests to a single storage server. A new concept is introduced to improve the scalability of CiteSeer<sup>X</sup>. Document records are separated into *collections*. Distinct from repositories that represent physical storage, collections are virtual concepts. In CiteSeer<sup>X</sup>, one storage server is specified as the *active* server, which receives crawled documents. In this server, a temporary collection (not serving public traffic) ingests new documents until either a predefined volume threshold is reached or the system requests the collection to be *finalized*. New documents that enter the collection are assigned a Document ID (DID) by the system, which includes the collection identification number. Hence, performing a document identifier lookup returns the DID along with the identifier of the collection that contains the document. The new finalized collection, in turn, is transferred to the migration service in the application layer, which keeps a configuration of all available storage servers. An entry of the configuration looks like this:

*Server 1: Total Storage: 75G; Used Storage: 40G; Access Frequency in Last Month: 200,000*

The logging service periodically updates access frequencies in the configuration. Our policy is to ask the migration service to store the new collection in the least-loaded server which has enough storage space to accommodate it. Following the migration of this new collection, the naming server updates the collection-server mapping registry. The active storage server creates a temporary collection for subsequently crawled documents. This process can be illustrated in Figure 6.

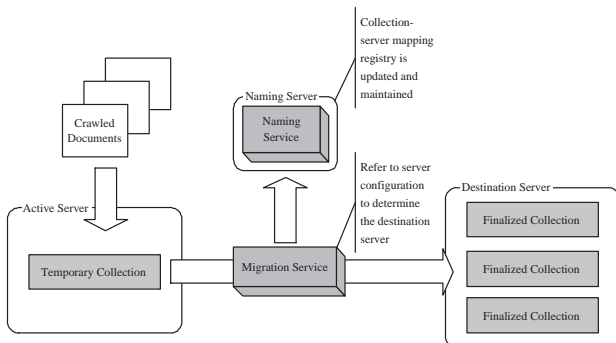


Fig. 6. Temporary Collection Migration Process.

However, document popularity distributions vary significantly over time, presenting a dynamic behavior in user access patterns. Therefore, despite the fact that each collection is deployed optimally in its creation time, the access pattern for

the servers can eventually become unbalanced. To overcome this problem, the migration service keeps observing the access status of each storage server and each collection. The load balancing algorithm and migration policy is presented in Algorithm 1.

A further level of data distribution is achieved by the fact that the digital objects are *not* the documents, but rather, descriptors of documents. Hence, we can separate the storage of digital objects and the corresponding documents as shown in Figure 5.

---

### Algorithm 1 Load Balancing Algorithm

---

**Input:**

$f_i$  ( $i = 0, 1, \dots, n$ ): Access count for each deployed storage server  $S_i$  in a given time period.

$f_{ij}$  ( $i = 0, 1, \dots, n; j = 0, 1, \dots, m$ ): Access count for each collection  $c_{ij}$  in a given time period.  $c_{ij}$  represents the  $j$ -th collection in Server  $S_i$ .

$\alpha$ : Overload factor.

1: calculate the mean access count for all storage servers:

$$f = \frac{1}{n} \sum_{i=1}^n f_i$$

2: **for** each server  $S_k$  whose  $f_k$  is more than  $(1 + \alpha)f$  **do**

3:   find collection  $c_{kj}$  whose  $f_{kj}$  is closest to  $\alpha f$  in  $S_k$

4:   find server  $S_l$ , whose  $f_l$  is least among all servers that have enough space to store  $c_{kj}$

5:   migrate  $c_{kj}$  to  $S_l$

6: **end for**

---

## VI. SYSTEM IMPLEMENTATION

CiteSeer<sup>X</sup> is supported by J2EE framework<sup>6</sup>, utilizing a number of open-source modules and middleware.

This section presents the open-source technologies that have been successfully integrated into CiteSeer<sup>X</sup> in the order from bottom to top, as shown in Figure 5.

Fedora [19], [20] serves as the digital object management system in the storage layer. It provides a level of abstraction through well defined access mechanism based on web services. This feature makes it well suited to CiteSeer<sup>X</sup> as a storage middleware, connecting to both the front-end (application servers) and the back-end (physical storage).

Full-text search is handled by querying Lucene index of the whole document collection. Relational tuples are stored in a PostgreSQL<sup>7</sup> database.

Application layer contains a number system applications, exposed as web services that are accessible by the user interface layer and external applications.

Apache's Struts<sup>8</sup> bridges web requests to appropriate application level logics in the user interface layer. Struts takes the *Model 2* approach, which is a variation of the classic Model-View-Controller (MVC) paradigm. It plays a *controller* role with other technologies providing the model and the view, providing a clear separation of the web pages and the applications.

<sup>6</sup><http://java.sun.com/j2ee/>

<sup>7</sup><http://www.postgresql.org/>

<sup>8</sup><http://struts.apache.org/>

## VII. PERFORMANCE EVALUATION

### A. Experimental Setup

Experiments were conducted in order to evaluate CiteSeer<sup>x</sup>'s scalability in terms of: (1) simultaneous online requests and (2) the number of stored documents. Also, various distributed storage deployments were tested in order to study their influence on system performance.

The experiments were implemented using Java on multiple workstations (CPU: Pentium 4 2.40GHz, Memory: 1GBytes, OS: Solaris 10). The raw text and metadata of documents within the CiteSeer repository were extracted and encoded for ingestion into CiteSeer<sup>x</sup>. The records were ingested into Fedora repositories specified during the migration process. Meanwhile, Lucene was used to build indices on the metadata and full texts respectively. Inside Fedora, the metadata was also indexed for the purpose of comparing Fedora's native search capability with Lucene.

Concurrent user transactions were simulated through mock clients. For the current CiteSeer system, the most common operations performed by users are document search and document retrieval. To better simulate the behavior patterns of real users, CiteSeer's system logs were analyzed to find the top 10,000 most popular query terms in a given month. Each time a client thread was initialized, it was set to randomly choose a term from the popular term vocabulary. Each thread issued its query to the system independently of other threads. The mock clients randomly selected  $n$  ( $0 \leq n \leq 10$ ) records from the search result list and issued new requests to the storage server to fetch the selected records.

The query handler process was set to capture incoming queries and forward them to appropriate storage servers. Inside the prototype, there are three ways to handle a query, for the purpose of comparison: (1) metadata search with Lucene index, (2) metadata search with Fedora's in-built index, and (3) full text search with Lucene index. System performance is measured according to query latencies. To minimize the possible errors, each experiment was performed multiple times and the mean value of the observations is taken as the final result. The experimental settings are listed in Table I. In the following subsections, if not explicitly specified, default values are chosen in the settings.

System Parameter	Range	Default
Number of documents in a server	12,500 – 50,000	12,500
Number of distributed servers	1 – 4	4
Number of simultaneous user threads	1,000 – 5,000	1,000

TABLE I  
EXPERIMENT SETTINGS

### B. Experimental Results

1) *Effect of the Size of a Repository*: For this experiment, document content is not distributed among multiple servers. Within a single server, the number of documents is controlled and the system's performance is observed, shown in Figure 7(a).

As expected, query latencies rise according to the number of documents within the repository. Generally, as the document repository grows, more hits are returned by the index and more documents need to be retrieved, requiring more communication with the index and storage services. Also, it is observed that querying Lucene's metadata index is more efficient than querying Fedora's in-built metadata index, indicating that a specialized index service is required.

2) *Effect of Simultaneous Requests*: Next, the system was configured with 4 repository servers and each was loaded with 12,500 documents. The number of simultaneous requests was varied from 1,000 to 5,000 to study how the system behaves under different workloads. The results are shown in Figure 7(b).

Generally, the overall performance of the system decreases with the increase of simultaneous queries, as expected. It should be noted that as the number of users reaches 5,000, there is a large increase in the system's response time. However, the public CiteSeer service currently does not typically receive more than 50 simultaneous queries at any time. Tracing the execution of the system revealed that the performance bottleneck occurred in the query handler, which ran out of memory at approximately 5000 queries and began queuing queries. If necessary, this problem could be overcome by increasing the RAM available to the query handler or balancing load across multiple redundant index servers.

3) *Effect of Data Distribution*: The effect distributing storage across multiple Fedora repositories on dedicated machines was investigated. The number of documents in the storage repository was fixed at 50,000 and the number of repository services was adjusted from 1 to 4.

Figure 7(c) shows that data distribution can improve overall system performance. As the Lucene index is not distributed across servers, the improvement brought by data distribution is caused by the document retrieval efficiency from the Fedora repository services. As more storage servers are deployed, more server resources (CPU, memory) are allocated to data retrieval tasks. The performance improvements are most noticeable under heavy workloads, where distributed servers can support more concurrent data retrieval operations.

## VIII. NEW FEATURES AND SERVICES

### A. CiteSeer as Semantic-Enabled Services

Program interfaces for the current CiteSeer are limited by the lack of clean interfaces into the component system features. CiteSeer-API [28] allows users to programmatically access the CiteSeer service similarly to how a human user would through CiteSeer's regular web interface. CiteSeer-API merely presents the CiteSeer service as a search engine service coupled with a bibliographical database. However, the service fails to expose the basic functions of the system that might be of interest to researchers and third-party application developers. For instance a semantic web agent could simply be interested in downloading a cached document. In that situation it is common to locate a document hosted by CiteSeer through a generic search engine such as Google. Another example is

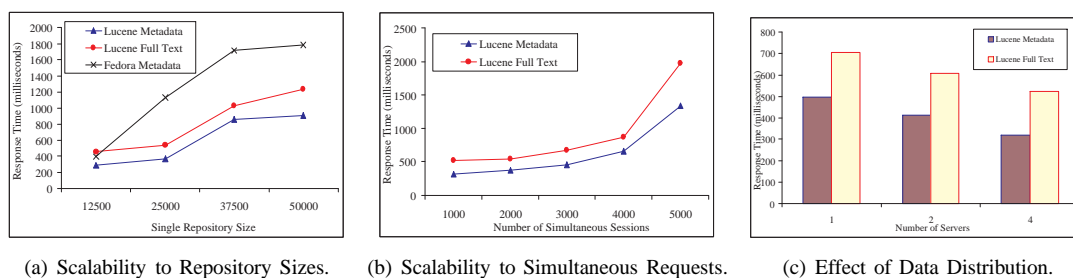


Fig. 7. Comparison Based on Different Cluster Sizes.

that of a semantic web agent that needs to extract citations from a piece of raw text: here, most of the CiteSeer service is irrelevant to the task, and unless this agent is willing to submit the full document to CiteSeer for processing, the task cannot be performed.

The elementary functionalities of CiteSeer can be divided into two categories. The first category is that of the functionalities which are specific to the CiteSeer service and which therefore are the most valuable regarding their integration on semantic web. The second category is that of the functionalities currently integrated in the CiteSeer but that are not specific to the application as they are recurrent to many information retrieval systems. The most recent service descriptions (WSDL and OWLS) for each service are available on the web <sup>9</sup>.

The CiteSeer<sup>x</sup> architecture is based upon discrete modules for each fundamental service component, facilitating the development of web service interfaces into each module. Mapping each elementary functionality of CiteSeer to independent web services, agents on the semantic web will gain access to the most unique functionalities of CiteSeer service, which are automated citation analysis and document interlinking based on citations.

Beyond the benefits of increased system flexibility and accessibility, modularizing core system components increases the configurability of the CiteSeer<sup>x</sup> deployment environment. If any single service is found to be a bottleneck in a pattern of system use, the service can be mirrored on an arbitrary amount of other machines, in a single site or across sites. Load can then be balanced across the service mirrors as necessary. This is particularly convenient for stateless services such as information extraction or document conversion modules, as no data migration is necessary.

### B. OAI Compliance

OAI-PMH enables heterogeneous digital library systems to cooperate via higher-level aggregations that abstract metadata records away from the fundamental incompatibilities between systems. Using APIs into CiteSeer<sup>x</sup>, arbitrary clients can directly determine a link to a specific resource hosted by a given CiteSeer<sup>x</sup> service. A direct application of such functionalities is the interlinking of digital library systems. Not only with this interoperability allow external digital libraries to access CiteSeer<sup>x</sup> repository content, but it will also allow the direct

importation of virtual document records (discussed previously) into CiteSeer<sup>x</sup> from trusted external sources supporting OAI-PMH. This will serve to increase confidence in the verity of virtual document records as well as to provide metadata information that can be used to locate missing document content. For instance, title and author metadata can be employed to query the web in order to discover textual documents based upon their metadata.

This concept has been used in the current CiteSeer system in order to interoperate with sister search engines employing the same code base. For example, OAI-PMH has been used to share content among CiteSeer, eBizSearch [29], [30], and SMEALSearch<sup>10</sup>. eBizSearch and SMEALSearch are niche search engines for e-Business and general academic business publications, respectively. Using OAI-PMH, the three search engines are able to request overlapping content from each other. However, due to the inability of the current CiteSeer architecture to handle metadata records for unavailable documents, special metadata elements were required within the OAI framework in order to complete the interoperability. The virtual record structure of CiteSeer<sup>x</sup> will enable the system to ingest OAI records in any common format.

### C. Logging Service

As a hybrid search engine and digital library, the logging service of CiteSeer<sup>x</sup> must consider logging from multiple perspectives in order to provide a unified view of system operations. This is particularly the case given the modular nature of CiteSeer<sup>x</sup> service deployment. The perspectives are as follows:

- 1) **System perspective:** Each service within CiteSeer<sup>x</sup> must keep usage logs; however, aggregation of those logs can require significant resources. In addition, most system use cases will involve multiple working modules, creating a problem of session and transaction tracking.
- 2) **Data perspective:** Heterogeneity of log formats for individual services can significantly increase overhead in log aggregation. Due to the lack of a well accepted self-descriptive log standard, the sharing of usage logs of web information retrieval systems and has historically been restricted.
- 3) **Use perspective:** Usage mining in search engines is usually restricted to the query logs. An important problem

<sup>9</sup><http://citeseer.ist.psu.edu/api>

<sup>10</sup><http://smealsearch.psu.edu>

with existing logging methods is the lack of logging for system responses. The lack of response logs has become an obstacle for applications of data mining to many web information retrieval systems. In addition, session and transaction information is lacking in most web log formats, which is critical in inferring user access patterns.

An XML based description language for information retrieval system usage logs is introduced in CiteSeer<sup>x</sup> by modeling a user-system interaction ontology. The language encompasses rich semantic descriptions of the events being logged such as dependency between successive actions. In addition, logs are maintained for both requests and system responses for the purposes of data mining and system monitoring.

The logging service architecture CiteSeer<sup>x</sup> reflects the concept of detaching the logging service from the target system [31]. It is no longer the duty of each module in the system to write document usages. Instead, an independently running logging service collects and manages logs from every module. In this way, the standard logging ontology can be applied without burdening module developers with strict log format criteria. We follow previously proposed guidelines [31] and build our logging service based on web services in order to facilitate logging in a heterogeneous system environment.

## IX. CONCLUSION

This paper presents CiteSeer<sup>x</sup> – the next generation CiteSeer architecture that is designed to overcome the challenges of interoperability, extensibility and scalability of the current CiteSeer system. Rather than being a debugged version of CiteSeer, CiteSeer<sup>x</sup> is a completely new architecture that is based on modular web services, pluggable service components, distributed object repositories and transaction-safe processes, all utilizing an enriched data model.

The preliminary evaluation shows flexibility, scalability and performance improvements for CiteSeer<sup>x</sup> compared to its predecessor design. The new architecture along with new features such as personalized services and acknowledgement search will enable CiteSeer<sup>x</sup> to continue to be a valuable tool for the computer science research community.

## X. ACKNOWLEDGEMENTS

We gratefully acknowledge partial support from Microsoft Research, NSF and NASA.

## REFERENCES

- [1] S. Lawrence, K. D. Bollacker, and C. L. Giles, "Indexing and retrieval of scientific literature," in *CIKM*. ACM, 1999, pp. 139–146.
- [2] C. L. Giles, K. Bollacker, and S. Lawrence, "CiteSeer: An automatic citation indexing system," in *Digital Libraries 98 - The Third ACM Conference on Digital Libraries*, I. Witten, R. Akscyn, and F. M. Shipman III, Eds. Pittsburgh, PA: ACM Press, June 23–26 1998, pp. 89–98.
- [3] "Smealsearch," <http://smealsearch.psu.edu>.
- [4] H. Anan, X. Liu, K. Maly, M. Nelson, M. Zubair, J. C. French, E. Fox, and P. Shivakumar, "Preservation and transition of ncsrl using an oai-based architecture," in *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*. New York, NY, USA: ACM Press, 2002, pp. 181–182.

- [5] "Computing research repository. <http://arxiv.org/corr/home>."
- [6] G. Crane, "Building a digital library: the perseus project as a case study in the humanities," in *DL '96: Proceedings of the first ACM international conference on Digital libraries*. New York, NY, USA: ACM Press, 1996, pp. 3–10.
- [7] C. Lagoze, W. Arms, S. Gan, D. Hillmann, C. Ingram, D. Krafft, R. Marisa, J. Phipps, J. Saylor, C. Terrizzi, W. Hoehn, D. Millman, J. Allan, S. Guzman-Lara, and T. Kalt, "Core services in the architecture of the national science digital library (nsdl)," in *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*. New York, NY, USA: ACM Press, 2002, pp. 201–209.
- [8] "Science direct. <http://www.sciencedirect.com>."
- [9] "Google scholar. <http://scholar.google.com>."
- [10] S. Bradshaw, A. Scheinkman, and K. Hammond, "Guiding people to information: providing an interface to a digital library using reference as a basis for indexing," in *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*. New York, NY, USA: ACM Press, 2000, pp. 37–43.
- [11] J. Stribling, I. G. Councill, J. Li, M. F. Kaashoek, D. R. Karger, R. Morris, and S. Shenker, "Overcite: A cooperative digital research library," in *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS05)*, Ithaca, NY, February 2005.
- [12] R. Kahn and R. Wilensky, "A framework for distributed digital services," <http://www.cnri.reston.va.us/home/cstr/arch/k-w.html>, 1995.
- [13] "The simple digital library interoperability protocol (sdliip-core)," <http://dbpubs.stanford.edu:8091/testbed/doc2/SDLIP/>.
- [14] M. D. Giacomo, M. Martinez, and J. Scott, "A large-scale digital library system to integrate heterogeneous data of distributed databases," in *Euro-Par*, 2004, pp. 391–397.
- [15] A. Kumar, R. Saigal, R. Chavez, and N. Schwertner, "Architecting an extensible digital repository," in *JCDL '04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*. New York, NY, USA: ACM Press, 2004, pp. 2–10.
- [16] "Gendl – generic digital library," <http://elib.cs.berkeley.edu>.
- [17] "Greenstone digital library software," <http://www.greenstone.org/cgi-bin/library>.
- [18] "Dspace digital repository system," <http://www.dspace.org/>.
- [19] T. Staples, R. Wayland, and S. Payette, "The fedora project: An open-source digital object repository system," *D-Lib Magazine*, vol. 9, April 2003.
- [20] C. Lagoze, S. Payette, E. Shin, and C. Wilper, "Fedora: An architecture for complex objects and their relationships," *Journal of Digital Libraries, Special Issue on Complex Objects*, 2005.
- [21] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma, "Object-level ranking: bringing order to web objects," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*. New York, NY, USA: ACM Press, 2005, pp. 567–574.
- [22] S. Lawrence, C. L. Giles, and K. Bollacker, "Digital libraries and Autonomous Citation Indexing," *IEEE Computer*, vol. 32, no. 6, pp. 67–71, 1999.
- [23] E. Garfield, "Science citation index – a new dimension in indexing," *Science*, vol. 144, pp. 649–654, 1964.
- [24] *Linux Virtual Servers for Scalable Network Services*, 2000.
- [25] I. Councill, H. Li, Z. Zhuang, S. Debnath, L. Bolelli, W. Lee, A. Sivasubramanian, and C. Giles, "Learning metadata from the evidence in an on-line citation matching scheme," submitted.
- [26] V. I. Levenshtein, "Binary codes capable of correcting spurious insertions and deletions of ones," *Problems of Information Transmission*, vol. 1, pp. 8–17, 1965.
- [27] R. Kahn and R. Wilensky, "A framework for distributed digital object services," *Working Paper, cnri.dlib/m95-01*, 1995.
- [28] Y. Petinot, C. L. Giles, V. Bhatnagar, P. B. Teregowda, H. Han, and I. G. Councill, "Citeseer-api: towards seamless resource location and interlinking for digital libraries," in *CIKM*, D. Grossman, L. Gravano, C. Zhai, O. Herzog, and D. A. Evans, Eds. ACM, 2004, pp. 553–561.
- [29] Y. Petinot, P. B. Teregowda, H. Han, C. L. Giles, S. Lawrence, A. Rangaswamy, and N. Pal, "ebizsearch: An oai-compliant digital library for ebusiness," in *JCDL*. IEEE Computer Society, 2003, pp. 199–209.
- [30] "ebizsearch," <http://www.ebizsearch.org>.
- [31] D. Horn, H. Balakrishnan, B. T. Maniampadavathu, J. Warnes, and D. A. Elko, "A logger system based on web services," *IBM Systems Journal*, vol. 43(4), pp. 723–733, 2004.