OverCite: A Cooperative Digital Research Library

by

Jeremy Stribling

Submitted to the Department of Electrical Engineering
and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2005

Author ....
Department of Electrical Engineering and Computer Science
September 2, 2005

Certified by ....
M. Frans Kaashoek
Professor
Thesis Supervisor

Accepted by..
Arthur C. Smith
Chairman, Department Committee on Graduate Students

OverCite: A Cooperative Digital Research Library

by
Jeremy Stribling

Submitted to the Department of Electrical Engineering and Computer Science
on September 2, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

CiteSeer is a well-known online resource for the computer science research community, allowing users to search and browse a large archive of research papers. Unfortunately, its current centralized incarnation is costly to run. Although members of the community would presumably be willing to donate hardware and bandwidth at their own sites to assist CiteSeer, the current architecture does not facilitate such distribution of resources.

OverCite is a design for a new architecture for a distributed and cooperative research library based on a distributed hash table (DHT). The new architecture harnesses donated resources at many sites to provide document search and retrieval service to researchers worldwide. A preliminary evaluation of an initial OverCite prototype shows that it can service more queries per second than a centralized system, and that it increases total storage capacity by a factor of $n/4$ in a system of n nodes. OverCite can exploit these additional resources by supporting new features such as document alerts, and by scaling to larger data sets.

Thesis Supervisor: M. Frans Kaashoek
Title: Professor

Acknowledgments

My advisor, Frans Kaashoek, provided the encouragement and vision needed to embark on this project and write this thesis. I thank him for his time and tireless effort, and for taking a chance on a fresh-faced kid from Berkeley. In addition, the OverCite project has been joint work with Robert Morris, Jinyang Li, Isaac Councill, David Karger, and Scott Shenker, all of whom made important contributions to the success of this work.

I am eternally grateful to Frank Dabek, Jinyang Li, Max Krohn, and Emil Sit for creating and debugging the software on which OverCite runs. This work would not have been possible without their genius, creativity, and patience. Thanks also to Dave Andersen for the use of the RON network, and helping me deploy the new nodes.

The comments of Jayanthkumar Kannan, Beverly Yang, Sam Madden, Anthony Joseph, the MIT PDOS research group, and the anonymous IPTPS reviewers greatly improved previous drafts of this work. I also thank C. Lee Giles for his continued support at PSU.

I thank Dan, John, Sanjit, Frank, Nick, Thomer, Frans, Max, Jinyang, Robert, Athicha, Rodrigo, Emil, Mike, and Alex for keeping my days interesting, my mind challenged, and my desk covered in plastic yogurt lids and little round magnets.

Without the love and support of my family and friends, I never would have made it this far. Thanks to everyone.

This research was conducted as part of the IRIS project (`http://project-iris.net/`), supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660.

Portions of this thesis were previously published in

> Jeremy Stribling, Isaac G. Councill. Jinyang Li, M. Frans Kaashoek, David R. Karger, Robert Morris and Scott Shenker. OverCite: A Cooperative Digital Research Library. In *Proceedings of the 3rd IPTPS*, February 2005.

and consist of text originally written and/or edited by authors of that document.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

CiteSeer is a popular repository of scientific papers for the computer science community [23], supporting traditional keyword searches as well as navigation of the "web" of citations between papers. CiteSeer also ranks papers and authors in various ways, and can identify similarity among papers. Through these and other useful services, it has become a vital resource for the academic computer science community.

Despite its community value, the future of CiteSeer is uncertain without a sustainable model for community support. After an initial period of development and deployment at NEC, CiteSeer went mostly unmaintained until a volunteer research group at Pennsylvania State University (PSU) recently took over the considerable task of running and maintaining the system.

This thesis describes OverCite, a new system that provides the same services as CiteSeer on a distributed set of geographically-separated nodes. OverCite eliminates the need for a single institution to contribute all of the resources necessary to run CiteSeer; the resource and bandwidth requirement of each participating node is significantly reduced. The rest of this chapter describes the problems with the current CiteSeer architecture more fully, outlines OverCite's solution to these problems, and lists the contributions of the development and deployment of OverCite.

1.1 Problem Statement

If CiteSeer were required to support many more queries, implement new features, or significantly expand its document collection or its user base, the resources required would quickly outstrip what PSU, or any other single noncommercial institution, could easily provide. For example, to include all the papers currently indexed by Inspec [20] (an engineering bibliography service), CiteSeer would need to increase its storage usage by an order of magnitude. The research community needs a technical solution to overcome these resource and scalability constraints.

An ideal solution to this problem would put the research community in control of its own repository of publications. An open, self-managed, self-controlled research repository would appeal to the collaborative nature of academic researchers, and allow them to retain control over their work. Commercially-managed systems such as Google Scholar [17] or the ACM Digital Library [1], while certainly feasible solutions, place control of a valuable document collection in the hands of a single organization (and potentially allows the organization to control copyrights as well); we argue that a community-controlled solution would be more flexible and fitting to the needs of researchers.

Because of CiteSeer's value to the community, it is likely that many institutions would be willing to donate resources to CiteSeer in return for more control over its evolution. One possible solution is for PSU to accept these donations (either as grants or hardware) and use them to enhance the performance of the existing CiteSeer configuration. Unfortunately, this does not address any bandwidth or centralized management limitations currently faced by PSU. As one step toward decentralization, willing institutions could purchase and maintain machines to act as CiteSeer mirrors. However, the hardware and bandwidth requirements of this machine, as well as the complex setup and maintenance required of a CiteSeer mirror, might discourage some otherwise interested institutions from participating. If the price of adoption were not so high, a potentially large number of institutions might be willing to donate the use of machines and bandwidth at their sites.

Thus, for CiteSeer to prosper and grow as a noncommercial enterprise, it must be adapted to run on a distributed set of donated, geographically-separated nodes [21]. The challenge that this thesis addresses is how to realize, robustly and scalably, a distributed version of CiteSeer. A well-funded, professionally-maintained, centralized library service will always have more available resources (*i.e.*, storage and bandwidth), and obtain better performance, than a distributed approach constructed from volunteer resources. This thesis shows, however, that designing a cooperative library using donated hardware and network resources is feasible.

With a wide-spread and potentially diverse set of resources, such as the donated CiteSeer nodes we envision, one major difficulty is providing a unified service to the user, despite the lack of centrally-coordinated facilities. A distributed version of CiteSeer, in particular, must face the problem of maintaining and coordinating a decentralized search index and distributed storage, allowing data access from anywhere in the network without requiring any centralized elements. At the same time, adding more resources to a system introduces more potential points of failure; component and network failures must be handled gracefully. Furthermore, the system should be able to scale as more nodes are added, and handle more demanding workloads than CiteSeer experiences today (see Table 2.1). Our solution, OverCite, addresses all of these issues.

1.2 Solution Approach

OverCite is a design that allows such an aggregation of distributed resources, using a distributed hash table (DHT) infrastructure. A DHT is peer-to-peer software that enables the sharing of data in a structured overlay network, without the need for centralized components. An application can use a DHT to store data robustly on a distributed set of nodes, and efficiently locate and retrieve specific items from anywhere in the network. DHTs also mask network failures from the application, allowing the storage and retrieval of data even under significant network churn. Examples of DHTs include Chord/DHash [12, 41], Kademlia [30], Pastry [37], and Tapestry [45], among many others.

By employing a DHT as a distributed storage layer, OverCite gains several attractive properties that make it a tenable alternative to CiteSeer. First, the DHT's robust and scalable model for data management and peer communication simplifies and automates the inclusion of additional CPU and storage resources. An institution seeking to donate resources would need only download, install and configure the OverCite software on its machines, and OverCite could immediately take advantage of the new resources. Second, the DHT simplifies the coordination of distributed activities, such as Web crawling, by providing a shared medium for storage and communication. Finally, the DHT acts as a rendezvous point for producers and consumers of meta-data and documents. Once a node has crawled a new document and stored it in the DHT, the document will quickly be available to other DHT nodes, and the application itself, for retrieval.

While peer-to-peer file-sharing technologies such as BitTorrent [11] and content distribution networks such as Akamai [2] and Coral [15] provide ways to serve data in a distributed fashion, they do not solve the same problems as a DHT. In particular, they cache data, but do not store it. Instead, they rely on a Web source to be available as permanent storage. Furthermore, they do not provide a facility for storing meta-data, or the relationship between documents. For these reasons, we choose to implement OverCite using a DHT, which provides both long-term, arbitrary data storage, decentralized content distribution, and a convenient interface for application development.

In addition to the DHT, OverCite employs a Web crawler and a distributed, partitioned search index to serve queries. This combination of components allows OverCite to distribute the services of CiteSeer among many volunteer sites. Chapter 3 describes the OverCite design in more detail.

1.3 Related Work

Digital libraries have been an integral tool for academic researchers for many years. Professional societies such as ACM [1] and IEE [20] maintain online repositories of papers published at their conferences. Specific academic fields often have their own research archives, such as arXiv.org [4] and CiteSeer [23], which allow researchers to search and browse relevant work, both new and old. More recently, initiatives like DSpace [40] and the Digital Object Identifier system [13] and seek to provide long-term archival of publications. The main difference between these systems and OverCite is that OverCite is a community-based initiative that can incorporate donated resources in a peer-to-peer fashion.

Previous work on distributed library systems includes LOCKSS [36], which consists of many persistent web caches that can work together to preserve data for decades against both malicious attacks and bit rot. Furthermore, the Eternity Service [3] uses peer-to-peer technology to resist censorship of electronic documents. There have also been a number of systems for searching large data sets [5, 9, 16, 19, 29, 38, 43, 44] and crawling the Web [7, 10, 28, 39] using peer-to-peer systems. We share with these systems a desire to distribute work across many nodes to avoid centralized points of weakness and increase robustness and scalability.

As mentioned above, services like BitTorrent [11] and Coral [15] provide an alternative style of content distribution to a DHT. Like DHTs such as DHash [12], these systems can find the closest copy of a particular data item for a user, and can fetch many data items in parallel. However, the DHT model of data storage is more closely aligned to OverCite's vision of creating a full citation database and providing a complete Web service given donated resources.

1.4 Contributions

At a technical level, exploring the design of a distributed digital library raises a number of interesting research problems. For example, how does one build a distributed search engine for a domain that includes hundreds of thousands of documents, but is more specific (and smaller) than the World Wide Web? What is the best way to store terabytes of data in a DHT? How does one design a distributed, wide-area online repository to be robust against server failures? The exploration and solution of these and other problems is the main contribution of the design described in this thesis.

OverCite is also an interesting and novel application for DHTs. The scale of the data set, as well as the potential usefulness of the services to the academic community, make OverCite an intriguing and challenging use of peer-to-peer technology. The implementation and subsequent testing of OverCite highlights important issues in DHT design (and in general, distributed systems). Furthermore, OverCite serves as a case study for the use of DHTs as a substrate for complex peer-to-peer applications, and hopefully others can apply the lessons we learned to future applications.

Simple calculations (Chapter 4) show that, despite the necessary coordination between wide-area nodes, an OverCite network with n nodes will have a performance benefit of $n/3$ times the current centralized system. For example, with 100 nodes, OverCite system-wide has approximately 30 times the storage of a single CiteSeer node. In practice, we find that our initial OverCite prototype can provide approximately $n/4$ times more storage than a single node (See Section 6.7 for details).

We demonstrate the performance properties of our OverCite implementation on a small, wide-area set of nodes. In particular, our preliminary experiments show that OverCite can handle more queries per second than a centralized service, and can provide reasonable throughput while serving large files.

OverCite, by potentially aggregating many resources, could offer more documents and features to its users, enabling it to play an even more central role in the community. For example, we will make OverCite available as a service to the community, and hope it will improve and enhance the way academic researchers publish and find documents online.

1.5 Thesis Outline

This thesis describes the design, analysis, implementation and evaluation of an OverCite prototype. After a brief overview of the current CiteSeer architecture and system properties in Chapter 2, Chapter 3 details the components and data structures used in OverCite, and Chapter 4 calculates the potential performance and overhead costs incurred by OverCite.

Chapter 5 describes the implementation of our prototype, while Chapter 6 evaluates the prototype with respect to query and file throughput. Chapter 7 postulates new features that could be added to OverCite, given its wealth of resources, and Chapter 8 discusses future work and concludes.

Chapter 2

CiteSeer Background

CiteSeer's major components interact as follows. A Web crawler visits a set of Web pages that are likely to contain links to PDF and PostScript files of research papers. If it sees a paper link it hasn't already fetched, CiteSeer fetches the file, parses it to extract text and citations, and checks whether the format looks like that of an academic paper. Then it applies heuristics to check if the document duplicates an existing document; if not, it adds meta-data about the document to its tables, and adds the document's words to an inverted index. The Web user interface accepts search terms, looks them up in the inverted index, and presents data about the resulting documents.

CiteSeer assigns a document ID (DID) to each document for which it has a PDF or Postscript file, and a unique citation ID (CID) to every bibliography entry within a document. CiteSeer also knows about the titles and authors of many papers for which it has no file, but to which it has seen citations. For this reason CiteSeer also assigns a "group ID" (GID) to each title/author pair for use in contexts where a file is not required. The GID also serves to connect newly inserted documents to previously discovered citations.

CiteSeer uses the following tables:

1. The document meta-data table, indexed by DID, which records each document's authors, title, year, abstract, GID, CIDs of document's citations, number of citations to the document, etc.
2. The citation meta-data, indexed by CID, which records each citation's GID and citing document DID.
3. A table mapping each GID to the corresponding DID, if a DID exists.
4. A table mapping each GID to the list of CIDs that cite it.
5. An inverted index mapping each word to the DIDs of documents containing that word.
6. A table indexed by the checksum of each fetched document file, used to decide if a file has already been processed.
7. A table indexed by the hash of every sentence CiteSeer has seen in a document, used to gauge document similarity.
8. A URL status table to keep track of which pages need to be crawled.
9. A table mapping paper titles and authors to the corresponding GID, used to find the target of citations observed in paper bibliographies.

Property	Measurement
Number of papers (# of DIDs)	715,000
New documents per week	750
HTML pages visited	113,000
Total document storage	767 GB
Avg. document size	735 KB
Total meta-data storage	44 GB
Total inverted index size	18 GB
Hits per day	>1,000,000
Searches per day	250,000
Total traffic per day	34.4 GB
Document traffic per day	21 GB
Avg. number of active conns	68.4
Avg. load per CPU	66%

Table 2.1: Statistics of the PSU CiteSeer deployment.

Table 2.1 lists statistics for the current deployment of CiteSeer at PSU, current as of November 2004. CiteSeer uses two servers, each with two 2.8 GHz processors. Most of the CPU time is used to satisfy user searches. The main costs of searching are lookups in the inverted index, collecting and displaying meta-data about search results, and converting document files to user-requested formats. The primary costs of inserting new documents into CiteSeer are extracting words from newly found documents, and adding the words to the inverted index. It takes about ten seconds of CPU time to process each new document.

Chapter 3

OverCite Design

The primary goal of OverCite is to spread the system's load over a few hundred volunteer servers. OverCite partitions the inverted index among many participating nodes, so that each node only indexes a fraction of the documents. This parallelizes the work of creating, updating, and searching the index. OverCite makes the user interface available on many nodes, thus spreading the work of serving files and converting between file formats. OverCite stores the document files in a DHT, which spreads the burden of storing them. OverCite also stores its meta-data in the DHT for convenience, to make all data available to all nodes, and for reliability. The choice of a DHT as a shared storage medium ensures robust, scalable storage along with the efficient lookup and management of documents and meta-data. OverCite partitions its index by document, rather than keyword [25, 34, 42, 43], to avoid expensive joins on multi-keyword queries, and limit the communication necessary on document insertions.

3.1 Architecture

OverCite nodes have four active components: a DHT process, an index server, a Web crawler, and a Web server that answers queries. Isolating the components in this manner allows us to treat each independently; for example, the inverted index is not tied any particular document storage solution. We describe each component in turn.

DHT process. OverCite nodes participate in a DHT. The DHT provides robust storage for documents and meta-data, and helps coordinate distributed activities such as crawling. Since OverCite is intended to run on a few hundred stable nodes, each DHT node can keep a full routing table and thus provide one hop lookups [18, 26, 27]. Because we expect failed nodes to return to the system with disks intact in most cases, and because all the data is soft state, the DHT can be lazy about re-replicating data stored on failed nodes.

Index server. To avoid broadcasting each query to every node, OverCite partitions the inverted index by document into k index partitions. Each document is indexed in just one partition. Each node maintains a copy of one index partition; if there are n nodes, there are n/k copies of each index partition. OverCite sends a copy of each query to one server in each partition, so that only k servers are involved in each query. Each of the k servers uses about $1/k$'th of the CPU time that would be required to search a single full-size inverted index. Each server returns the DIDs, partial meta-data, scores, and matching context of the m highest-ranked documents (by some specified criterion, such as citation count) in response to a query.

We can further reduce the query load by observing that many queries over the CiteSeer data will involve only paper titles or authors. In fact, analysis of an October 2004 trace of CiteSeer queries shows that 40% of answerable queries match the title or author list of at least one document. Furthermore, a complete index of just this meta-data for all CiteSeer papers is only 50 MB. Thus, an effective optimization may be to replicate this full meta-data index on all nodes, and keep it in memory, as a way to satisfy many queries quickly and locally. Another option is to replicate an index containing common search terms on all nodes. Moreover, if we would like to replicate the full text index on all nodes for even faster queries (*i.e.*, $k = 1$), we may be able to use differential updates to keep all nodes up-to-date on a periodic basis, saving computation at each node when updating the index.

In future work we plan to explore other possible optimizations for distributed search (*e.g.*, threshold aggregation algorithms [14]). If query scalability becomes an issue, we plan to explore techniques from recent DHT search proposals [5, 16, 19, 29, 38, 43] or unstructured peer-to-peer search optimizations [9, 44].

Web crawler. The OverCite crawler design builds on several existing proposals for distributed crawling (*e.g.*, [7, 10, 28, 39]). Nodes coordinate the crawling effort via a list of to-be-crawled page URLs stored in the DHT. Each crawler process periodically chooses a random entry from the list and fetches the corresponding page. When the crawler finds a new document file, it extracts the document's text words and citations, and stores the document file, the extracted words, and the document's meta-data in the DHT. The node adds the document's words to its inverted index, and sends a message to each server in the same index partition telling it to fetch the document's words from the DHT and index them. A node keeps a cache of the meta-data for documents it has indexed, particularly the number of citations *to* the paper, in order to be able to rank search results locally. While many enhancements to this basic design (such as locality-based crawling and more intelligent URL partitioning) are both possible and desirable, we defer a more complete discussion of the OverCite crawler design to future work.

Web-based front-end. A subset of OverCite nodes run a Web user interface, using round-robin DNS to spread the client load. The front-end accepts search terms from the user, sends them to an index server in each partition, collects the results and ranks them, and displays the top b results for the user. The front-end also retrieves document files from the DHT, optionally converts them to a user-specified format, and sends them to the user.

3.2 Tables

Table 3.1 lists the data tables that OverCite stores in the DHT. The tables are not explicitly distinct entities in the DHT. Instead, OverCite uses the DHT as a single large key/value

Name	Key	Value
`Docs`	DID	FID, GID, CIDs, etc.
`Cites`	CID	DID, GID
`Groups`	GID	DID + CID list
`Files`	FID	Document file
`Shins`	hash(shingle)	list of DIDs
`Crawl`		list of page URLs
`URLs`	hash(doc URL)	date file last fetched
`Titles`	hash(Ti+Au)	GID

Table 3.1: The data structures OverCite stores in the DHT.

table; the system interprets values retrieved from the DHT based on the context in which the key was found. These tables are patterned after those of CiteSeer, but adapted to storage in the DHT. These are the main differences:

- The `Files` table holds a copy of each document PDF or PostScript file, keyed by the FID, a hash of the file contents.
- Rather than use sentence-level duplicate detection, which results in very large tables of sentences, OverCite instead uses *shingles* [6], a well-known and effective technique for duplicate detection. The `Shins` table is keyed by the hashes of shingles found in documents, and each value is a list of DIDs having that shingle.
- The `Crawl` key/value pair contains the list of URLs of pages known to contain document file URLs, in a single DHT block with a well-known key.
- The `URLs` table indicates when each document file URL was last fetched. This allows crawlers to periodically re-fetch a document file to check whether it has changed.

In addition to the tables stored in the DHT, each node stores its partition of the inverted index locally. The index is sufficiently annotated so that it can satisfy queries over both documents and citations, just as in the current CiteSeer.

Chapter 4

Calculations

OverCite requires more communication resources than CiteSeer in order to manage the distribution of work, but as a result each server has less work to do. This section calculates the resources consumed by OverCite, comparing them to the costs of CiteSeer.

4.1 Maintenance Resources

Crawling and fetching new documents will take approximately three times more bandwidth than CiteSeer uses in total, spread out among all the servers. For each link to a Postscript or PDF file a node finds, it performs a lookup in `URLs` to see whether it should download the file. After the download, the crawler process checks whether this is a duplicate document. This requires (1) looking up the FID of the file in `Files`; (2) searching for an existing document with the same title and authors using `Titles`; and (3) verifying that, at a shingle level, the document sufficiently differs from others. These lookups are constant per document and inexpensive relative to downloading the document. Steps (2) and (3) occur after the process parses the document, converts it into text, and extracts the meta-data.

If the document is not a duplicate, the crawler process inserts the document into `Files` as Postscript or PDF, which costs as much as downloading the file, times the overhead f due to storage redundancy in the DHT [12]. The node also inserts the text version of the document into `Files` and updates `Docs`, `Cites`, `Groups`, and `Titles` to reflect this document and its meta-data.

Next, the node must add this document to its local inverted index partition (which is stored at a total of n/k nodes). However, each additional node in the same index partition need only fetch the *text* version of the file from `Files`, which is on average a tenth the size of the original file. Each of these n/k nodes then indexes the document, incurring some cost in CPU time.

The additional system bandwidth required by OverCite to crawl and insert a new document is dominated by the costs of inserting the document into the DHT, and for the other nodes to retrieve the text for that document. If we assume that the average original file size is x, and the size of the text files is on average $x/10$, then the approximate bandwidth overhead per document is $fx + (n/k)(x/10)$ bytes.

We estimate the amount of storage needed by each node as follows. The DHT divides

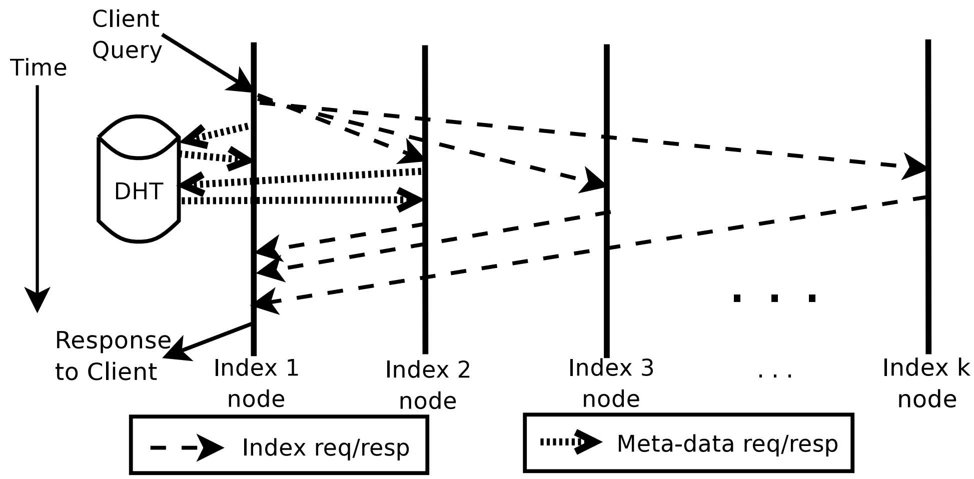

Figure 4-1: The timeline of a query in OverCite, and the steps involved. Each vertical bar represents a node with a different index partition. DHT meta-data lookups are only required at index servers without cached copies of result meta-data.

document and table storage among all n nodes in the system: this requires $(d + e)f/n$ GB, where d and e are the amount of storage used for documents and meta-data tables, respectively. Furthermore, each node stores one partition of the inverted index, or i/k GB if i is the total index size.

These bandwidth and storage requirements depend, of course, on the system parameters chosen for OverCite. Some reasonable design choices might be: $n = 100$ (roughly what PlanetLab has obtained through donations), $k = 20$ (so that only a few nodes need to index the full text of each new document), and $f = 2$ (the value DHash uses [12]). With these parameter choices, and the measurements from CiteSeer in Table 2.1, we find that the OverCite would require 1.84 MB of additional bandwidth per document (above the .735 MB CiteSeer currently uses) and 25 GB of storage per node.

These calculations ignore the cost of DHT routing table and data maintenance traffic. In practice, we expect these costs to be dwarfed by the traffic used to serve documents as we assume nodes are relatively stable.

4.2 Query Resources

Because OverCite partitions the inverted index by document, each query needs to be broadcast in parallel to $k-1$ nodes, one for each of the other index partitions.[1] Each node caches the meta-data for the documents in its index partition in order to rank search results and return relevant data fields to the querying server. When all k nodes return their top m matches, along with the context of the matches, relevant meta-data, and the value of rank metric, the originating node sorts the documents and displays the top b matches. Figure 4-1 depicts this process.

The packets containing the queries will be relatively small; however, each response will

[1]We assume here that no queries match in the meta-data index; hence, these are worst-case calculations.

contain the identifiers of each matching document, the context of each match, and the value of the rank metric. If each DID is 20 bytes, and the context, meta-data, and rank metric value together are 150 bytes, each query consumes about $170mk$ bytes of traffic. Assuming 250,000 searches per day, $k = 20$, and returning $m = 10$ results per query per node, our query design adds 8.5 GB of traffic per day to the network (or 85 MB per node). This is a reasonably small fraction of the traffic currently served by CiteSeer (34.4 GB). This does not include meta-data lookup traffic for any uncached documents on the index servers, which should be an infrequent and inexpensive operation.

Serving a document contributes the most additional cost in OverCite, since the Web-based front-end must retrieve the document fragments from the DHT before returning it to the user. This will approximately double the amount of traffic from paper downloads, which is currently 21 GB (though this load is now spread among all nodes). However, one can imagine an optimization involving redirecting the user to cached pre-constructed copies of the document on specific DHT nodes, saving this addition bandwidth cost.

OverCite spreads the CPU load of performing each query across multiple nodes, because the cost of an inverted index lookup is linear in the number of documents in the index. We demonstrate this effect experimentally in Section 6.4.

4.3 User Delay

User-perceived delay could be a problem in OverCite, as constructing each Web page requires multiple DHT lookups. However, most lookups are parallelizable, and because we assume a one-hop DHT, the total latency should be low. For example, consider the page generated by a user keyword query. The node initially receiving the query forwards the query, in parallel, to $k - 1$ nodes. These nodes may need to lookup uncached meta-data blocks in parallel, potentially adding a round-trip time to the latency. Therefore, we expect that the node can generate the page in response to a search in about twice the average round trip time of the network, plus computation time.

Generating a page about a given document (which includes that document's citations and what documents cite it) will take additional delay for looking up extra meta-data; we expect each of those pages to take an average of three or four round trip times.

Chapter 5

Implementation

The OverCite implementation consists of several modules, corresponding to the architectural components listed in Section 3.1 (see Figure 5-1). The *DHTStore* module is a library that allows other modules to store and retrieve data objects from a distributed hash table. The *Index* module computes an index over the documents in the local machine's partition, and performs queries over that index. Finally, the *OCweb* module presents a Web-based interface to users, and communicates with the *DHTStore* module and both local and remote *Index* modules to perform queries and provide access to document meta-data and the documents themselves. The current implementation does not yet include a Crawler module; we have populated OverCite with existing CiteSeer data, and plan to implement a distributed crawler in the future. In this chapter, after a brief code overview, we describe each of the above components (and the interaction between components) in detail.

5.1 Code Overview

The OverCite implementation consists of nearly 10,000 lines of C++ code, and uses the SFS libasync library [31] to provide an asynchronous, single-threaded execution environment for each module. Modules on the same machine communicate locally through Unix domain sockets; modules on different machines communicate via TCP sockets. All inter-module communication occurs over the Sun RPC protocol.

5.2 The *DHTStore* Module

The *DHTStore* module acts as an interface between the rest of the OverCite system and the DHT storage layer. The module takes the form of a library that can be used by other system components to retrieve meta-data and documents from DHash [12], a distributed hash table. The library communicates with DHash over a Unix domain socket.

DHTStore can work with any DHT that provides a put/get interface for data management, and supports unauthenticated, non-content-hash blocks (*i.e.*, the application can have complete control of the key for each item inserted in the DHT). Additionally, the *OCWeb* module requires an interface that lists neighbors of the local DHT node, though we plan to remove this dependence in the future.

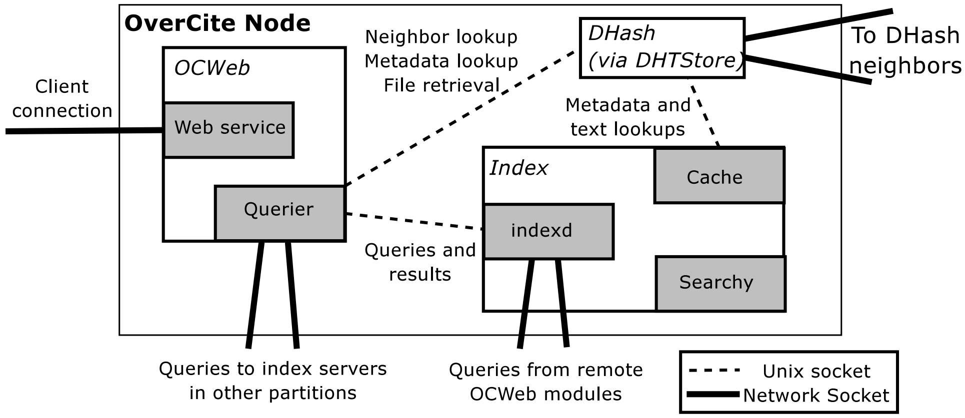

Figure 5-1: Implementation overview. This diagram shows the communication paths between OverCite components on a single node, and network connections between nodes.

5.2.1 Meta-data Storage

OverCite stores CiteSeer data in the DHT as Section 3.2 describes. Each logical table within the DHT indexes its data using a hash of its CiteSeer identifier and a well-known string; for example, the keys used to index the `Docs` table are the hash of the string "DID" concatenated with the DID of the document. This method ensures that nodes can look up information about a particular data item knowing only the appropriate CiteSeer identifier. The current implementation supports all tables listed in Table 3.1, with the exception of `Shins`, `Crawl`, and `URLs`.

While choosing the keys in this manner allows OverCite to store meta-data under a well-known identifier despite future updates, mutable blocks present a potential challenge to the DHT. When an application uses the content hash of a block as the block's key, the data can be treated as immutable, and data consistency is easily verifiable by both the DHT and the application. However, mutable data stored under arbitrary keys must be handled with care by the DHT; if multiple nodes update the data concurrently, the correct state of the block's data is not well-defined.

To overcome this problem, OverCite treats its meta-data blocks as *append-only* data structures, using new DHash data types inspired by OpenDHT [35]. When meta-data is first inserted, OverCite places the entire record as a data block under the appropriate key. For subsequent updates to the meta-data, such as a citation count increase or a title correction, OverCite appends an update field to the block. On a fetch, the *DHTStore* module receives both the original data structure and an update log of all updates; given this information the module can reconstruct the meta-data block into its correct form.

5.2.2 File Storage

OverCite stores files (both Postscript/PDF files and their text equivalents) within a file-system-like structure using immutable, content-hash blocks. An inode block contains the name and size of a file, and pointers to a number of indirect blocks, each of which point to 16KB blocks of file data. OverCite stores the content-hash key of the inode block in the document's meta-data; thus, an OverCite module can find a document file knowing only the DID of the document. The *DHTStore* module can fetch many file blocks in parallel, improving the time for file reconstruction. However, due to the absence of request congestion management in DHash, the higher-layer module fetching the blocks using *DHTStore* should limit the number of outstanding file fetches at any one time.

5.3 The *Index* Module

The *Index* module consists of a server daemon (*indexd*), a meta-data and text-file cache, and an interface to a local search engine. *indexd* listens on both TCP and Unix domain sockets for Sun RPCs, which request either local query operations or instruct the node to include new documents in its index.

As detailed in Section 3.1, each node is responsible for indexing and querying some subset (partition) of the documents. The *Index* module uses a deterministic function of the node's DHT identifier to determine responsibility for a particular partition. OverCite uses this same function to place new documents in exactly one partition. Currently the module assumes a centralized agent telling all nodes in each partition about all relevant documents; future releases of OverCite will include distributed reconciliation of the set of documents between nodes in the same partition. OverCite does not yet support citation queries or a full, in-memory meta-data index.

The meta-data and text-file cache manages the information needed by the *Index* module to index documents and return query results. The cache coordinates the retrieval of all DHT information via the *DHTStore* module, ensuring that only one fetch for a particular data item is happening at any one time. The cache can also throttle the rate of its requests to ensure it doesn't overwhelm the DHT. It keeps all fetched document meta-data in an in-memory cache, and keeps the fetched text files in both an in-memory cache of limited size and an on-disk cache.

OverCite uses Searchy [24] as a local search engine. Searchy is a fast, flexible, updatable search engine developed at MIT, and indexes each document using its DID as an identifier. The *Index* module interfaces with Searchy through command line calls, and parses Searchy's output to retrieve the DIDs, search term positions, and scores of the results. The module then uses this information to retrieve the meta-data and text file from the DHT (or, more likely, the cache) for each of the top m results, extracts the context information, and returns all the relevant information in an RPC response via *indexd*. OverCite can harness any local search engine, provided a class exists for the engine that implements OverCite's expected search engine API.

We chose to deploy OverCite using Searchy, rather than reusing the current homegrown CiteSeer search engine solution, for several reasons. The current CiteSeer implementation does not perform well at high loads, as shown in Section 6.2. Furthermore, Searchy proved to be more extensible, flexible, and easier to debug than the CiteSeer engine, due to its modular design and active development status.

5.4 The *OCWeb* Module

OverCite uses the *OCWeb* module to provide an interface between a user's Web browser and the storage and index layers of the system. This component is implemented as a module for the OK Web Server (OKWS) [22], a secure, high-performance web server. Because OKWS also uses libasync, this choice of Web server allows *OCWeb* to access the *DHTStore* library and interact with DHash directly.

To display document and citation meta-data to users, the *OCWeb* module looks up the information using the *DHTStore* module and formats it as HTML. To perform queries on behalf of a user, the module contacts the DHT to learn about the local node's neighbors. In our DHash-based implementation, this operation returns all of the nodes in the DHash location table. Based on the DHT identifiers of these neighbors, the module chooses one from each index partition (excluding the partition of the local index server). For simplicity, our current implementation assumes that at least one neighbor in each index partition is running an *Index* module; however, to decouple the index from the DHT a future version may use a source other than the DHT to learn about index servers. Currently, the module picks the live node in each partition with the lowest ping time, and opens a TCP connection to the *indexd* running on the node. If the node is not running *indexd*, the module switches to the next closest node. When a user submits a query, the *OCWeb* module sends the full query to each of the chosen nodes (including the local node), aggregates and sorts the results, and returns the top results (including title and context) to the user.

Chapter 6

Evaluation

OverCite harnesses distributed resources to provide a scalable, community-controlled digital research library. However, the extra computation and storage resources afforded by this design benefit the user only if the system can perform queries and retrieve requested data with reasonable performance, even under heavy load. This chapter explores the performance of our OverCite implementation with respect to query rate, throughput, and storage.

Although the scale of the following preliminary experiments is smaller than the expected size of an OverCite system, and the code is currently an early prototype, this evaluation demonstrates basic performance properties of the OverCite design. We plan to perform a more in-depth analysis of the system once the code base matures and more nodes become available for testing.

6.1 Evaluation Details

For this evaluation, we deployed OverCite on five geographically-spread nodes in North America, and three local MIT nodes. These eight nodes ran all components of the system; an additional three nodes at MIT participated only in the DHT. The DHash instance on each node spawned three virtual nodes for load-balancing purposes. All results represent the average of five experiments (except Figures 6-1 and 6-2, due to time constraints).

We inserted 14,755 documents from the CiteSeer repository into our OverCite deployment, including the meta-data for the documents, their text and Postscript/PDF files, and the full citation graph between all documents. Unless otherwise stated, each experiment involves two index partitions ($k = 2$). All *Index* modules return up to 30 results per query ($m = b = 30$), and the context for each query contains one highlighted search term. Furthermore, each node has a complete on-disk cache of the text files for all documents in its index partition (but not the document meta-data or Postscript/PDF files). *Index* modules do not throttle their DHT requests for this evaluation.

To evaluate the query performance of OverCite, we used a trace of actual CiteSeer queries, collected in October 2004. Because we are using a subset of the full CiteSeer document repository, queries returned fewer results than in CiteSeer. The client machine issuing queries to OverCite nodes is a local MIT node that is not participating in OverCite, and that can generate requests concurrently to emulate many simultaneous clients.

6.2 CiteSeer Performance

First, we tested the performance of the current CiteSeer implementation on an unloaded CiteSeer mirror at PSU. In this experiment, the client machine issues queries at varying levels of concurrency to emulate many simultaneous clients. For each query, the client opens a new TCP connection to the server, issues the query, and waits to receive the full page of results from the server. At each concurrency level, the client issues 1000 total queries.

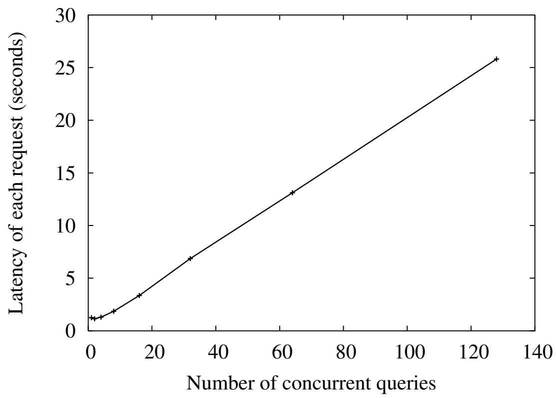

Figure 6-1: Average latency of all queries on CiteSeer, as a function of the number of concurrent clients.

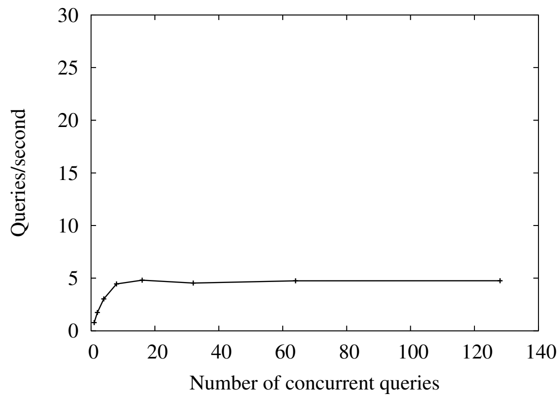

Figure 6-2: Average query throughput on CiteSeer, as a function of the number of concurrent clients.

Figures 6-1 and 6-2 show CiteSeer's performance, as a function of the number of concurrent queries. Query latency increases linearly as concurrency increases, while query throughput holds roughly constant at 5 queries per second. These results are not directly comparable to the following OverCite results, since the CiteSeer mirror uses the full repository (more than 700,000 documents) and runs on different hardware. A more direct comparison is an objective for future work.

6.3 Centralized Baseline

As a point of comparison for the fully-distributed OverCite system, we first evaluated a single-node version of OverCite. The single node indexes the full document set using one index partition, and has a complete in-memory cache of all document meta-data; thus, the node never communicates with the DHT or *Index* modules on remote nodes. The queries in these experiments involve no network hops other than the one between the client and the server.

Figures 6-3 and 6-4 show the performance (in terms of latency and throughput) of OverCite on a single node, as the number of concurrent queries increases. We see that latency increases linearly with the number of concurrent queries, while the number of queries that can be served per second roughly levels off around 14 queries/sec. Figure 6-3 shows only the latency of queries that returned more than 5 results.

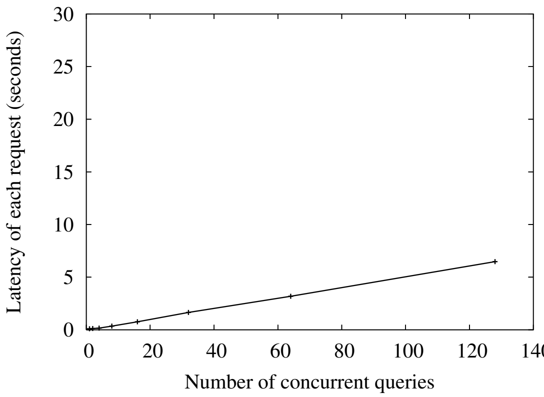

Figure 6-3: Average latency of queries with more than 5 results on a centralized server, as a function of the number of concurrent clients.

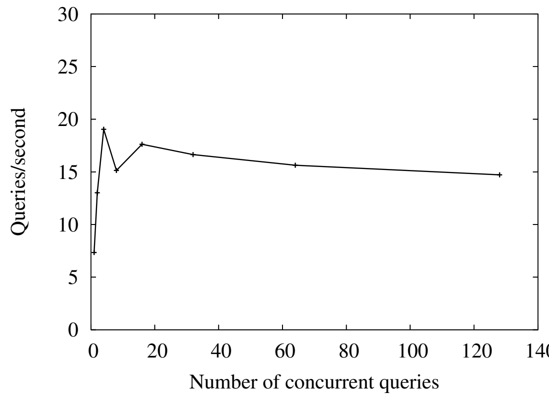

Figure 6-4: Average query throughput on a centralized server, as a function of the number of concurrent clients.

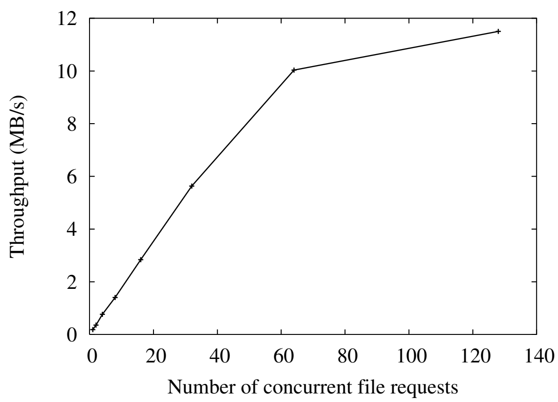

Figure 6-5: Average throughput of a single node serving text files from an on-disk cache, as the number of concurrent file requests varies.

Our client then requested text files from the single node server, rather than query results, and measured the resulting throughput. Figure 6-5 shows the throughput as the number of concurrent client requests varies. As serving these files requires no DHT or remote index communication for the server, this performance represents the maximum throughput OverCite can achieve while serving files.

6.4 Distributed OverCite

Next, we evaluated the full OverCite deployment. In these experiments, the client continues to communicate with a single OverCite server, but now the server must forward the query to a node indexing documents in the second partition as well. Furthermore, each *Index* module must contact the DHT to look up meta-data about the top results for a query. Nodes clear their in-memory meta-data cache between each experiment.

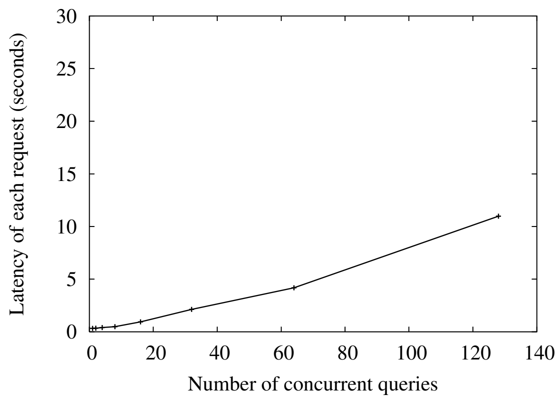

Figure 6-6: Average latency of queries with more than 5 results on a distributed OverCite system, as a function of the number of concurrent clients.

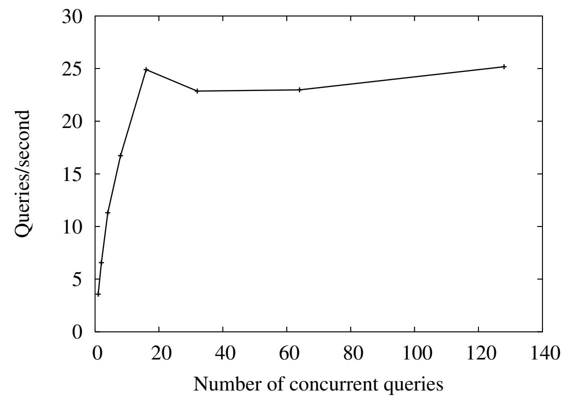

Figure 6-7: Average query throughput on a distributed OverCite system, as a function of the number of concurrent clients.

Figures 6-6 and 6-7 plot latency and throughput for a multiple-partition, DHT-enabled system as a function of concurrent queries. Again we see that latency rises linearly as concurrency increases; latencies of queries with more than 5 results are higher than those for the single partition case, due mostly to DHT lookups.

Figure 6-7 shows that using multiple nodes to process queries achieves greater throughput than in the single partition case (Figure 6-4), even though the nodes access the DHT to download document meta-data resulting in longer queries. However, it is important to notice that Figures 6-3 and 6-6 show latencies of queries with more than 5 results; for queries with no results (which account for more than 50% of the queries), the multiple-partition case is actually faster, despite the additional round trip time to transfer results between OverCite nodes.

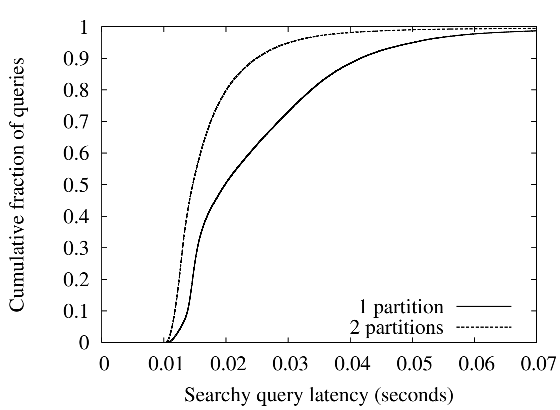

Figure 6-8: The distribution of Searchy query latencies. The tail of both lines reaches 1.3 seconds.

We infer then that the network round-trip times involved in this scenario do not bottleneck the throughput. Instead, we can explain the throughput disparity by examining the difference in Searchy query times. Figure 6-8 shows the distribution of Searchy queries during the above experiments. We see that the search times for multiple partitions (especially those above the median) are faster than those in the single partition case, accounting for the increased throughput. This is due to the difference in index sizes: in the single partition case, one server performs a query over the full index, while in the multiple partition case two serves each perform parallel queries over an index of half the size. We verified that this scaling trend holds true for larger document sets, as well.

6.5 Multiple OverCite Servers

One of the chief advantages of a distributed system such as OverCite over its centralized counterparts is the degree to which OverCite uses resources in parallel. In the case of OverCite, clients can choose from many different web servers (either manually or through DNS redirection), all of which have the ability to answer any query using different sets of nodes. Because each index partition is replicated on multiple nodes, OverCite nodes have many forwarding choices for each query. We expect that if clients issue queries concurrently to multiple servers, each of which is using different nodes as index neighbors, we will achieve a corresponding increase in system-wide throughput. However, because the nodes are sharing (and participating in) the same DHT, their resources are not entirely independent, and so the effect on throughput of adding servers is non-obvious.

Figure 6-9 shows the number of queries per second achieved by OverCite, as a function of the number of servers used as Web hosts. Each host uses a different neighbor for its remote index partitions, and the client issues a total of 128 queries concurrently to all of the available servers.

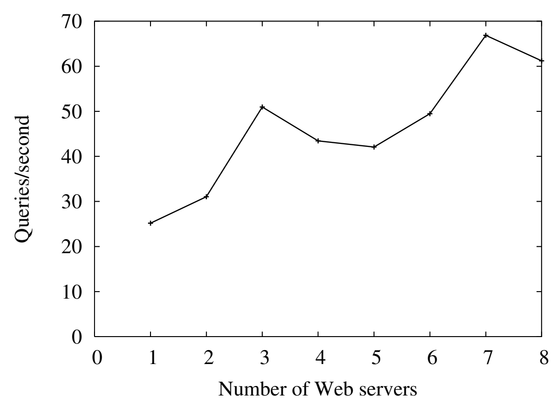

Figure 6-9: Average query throughput on a distributed OverCite system, as a function of the number of Web servers increases. The client issues 128 concurrent queries at a time.

Though the trend in the graph fluctuates somewhat across the number of servers, due to wide variation in DHT lookup times between experiments and nodes, in general adding more Web servers has the effect of increasing query throughput. Despite the fact that the servers share a common DHT backbone (used when looking up document meta-data), the resources of the different machines can be used by OverCite to satisfy more queries in parallel.

6.6 File Downloads

A potential bottleneck in OverCite is its ability to fetch, reconstruct, and return Postscript and PDF documents to a user. Such a request involves fetching potentially many blocks in parallel from the DHT. As the archiving and dispensing of these documents is one of main features of CiteSeer, it is essential that OverCite performs this task with reasonable performance.

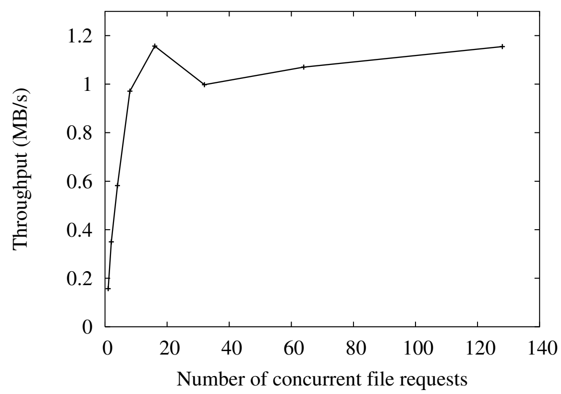

Figure 6-10: Average throughput of OverCite serving Postscript/PDF files from the DHT, as the number of concurrent file requests varies.

Figure 6-10 shows the throughput of the OverCite file-serving system, for a single Web server at various levels of request concurrency. Because OverCite throttles the number of outstanding DHash fetch operations, throughput quickly reaches a maximum throughput of 1.2 MB/s and plateaus there, despite the number of concurrent clients.

This limitation appears to be an artifact of having a small DHT: DHash encodes data in such a way that it needs seven block fragments to reconstruct each block, hence in a small network every node is involved in nearly every block fetch. As a simple exercise, assume each DHash database lookup involves two disk seeks, and each seek takes 10 ms. Thus each node can satisfy at most 50 fetches per second. Because each block fetch involves retrieving fragments from seven of the eleven DHT nodes, each node participates in $7/11$ of all fetches. Therefore, system-wide, we would expect only $50/(7/11) = 78$ block fetches to be possible per second. OverCite uses 16 KB blocks for document data, and at 78 blocks per second, could theoretically serve only 1.22 MB/s.

Without throttling requests, throughput drops to nearly zero after reaching the 1.2 MB/s level. During this period of low throughput, DHash (specifically its lookup layer, Chord) reports many RPC timeouts, even though all nodes are still alive and connected. However, given the above scenario, this behavior is not surprising: if each node is serving data from its disk as fast as possible, it would spend nearly all of its time blocking on disk operations, and almost no time servicing RPC messages. This creates livelock conditions [33]; without timely RPC responses, DHash nodes will give up on a fetch request, assuming the other node has failed. A situation emerges in which requests queue up while blocking on the disk, but once completed the results are no longer desired by the requester. Thus, the DHT does very little useful work, and throughput drops significantly.

We do not expect to see this behavior once we deploy OverCite on a larger DHT. With more nodes, each fetch will involve a much smaller fraction of the nodes, and OverCite will be able to request more blocks per second without overwhelming the disks of all DHT nodes.

6.7 Storage

Property	Measurement
Document storage	4.5 GB
Meta-data storage	.15 GB
Index size	.40 GB
Total storage	5.05 GB

Table 6.1: Storage statistics for a centralized server.

Property	Measurement
Document storage per node	1.15 GB
Meta-data storage per node	.35 GB
Index size per node	.21 GB
Total storage per node	1.71 GB

Table 6.2: Average per-node storage statistics for the OverCite deployment.

Finally, we compare the storage costs of OverCite to those of a centralized solution. Given that the current OverCite deployment stores 14,755 documents, Table 6.1 shows the storage costs faced by a single node running all components of the system. This node would incur a total storage cost of 5.05 GB.

Table 6.2 shows the average per-node storage costs measured on our 11-node, 2-partition OverCite deployment. Each node stores a total of 1.71 GB. This result is higher than expected given the storage formula from Section 4.1, which predicts 1.09 GB (using $f = 2$ for content hash data and $f = 5$ for other data, the default DHash parameters). The additional storage is due to several different factors: (a) storage overhead incurred in OverCite's file data format (*i.e.*, inode and indirect blocks); (b) extra replicas of non-content hash blocks stored on many nodes by an imperfect DHash replication implementation; and (c) a small amount of additional data stored in the DHT for testing purposes.

Despite this overhead, OverCite uses less than a factor of four more storage than CiteSeer in total. Adding 11 nodes to the system decreased the *per-node* storage costs by nearly a factor of three; assuming this scaling factor holds indefinitely, adding n nodes to the system would decrease per-node storage costs by a factor of roughly $n/4$. Therefore, we expect that an OverCite network of n nodes can store $n/4$ times as much as a single CiteSeer node.

Chapter 7

Features and Potential Impact

Given the additional resources available with OverCite's design (as demonstrated in Chapter 6), a wider range of features will be possible; in the long run the impact of new capabilities on the way researchers communicate may be the main benefit of a more scalable CiteSeer. This section sketches out a few potential features.

Document Alerts: As the field of computer science grows, it is becoming harder for researchers to keep track of new work relevant to their interests. OverCite could help by providing an *alert* service to e-mail a researcher whenever a paper entered the database that might be of interest. Users could register queries that OverCite would run daily (e.g., alert me for new papers on "distributed hash table" authored by "Druschel"). This service clearly benefits from the OverCite DHT infrastructure as the additional query load due to alerts becomes distributed over many nodes. A recent proposal [21] describes a DHT-based alert system for CiteSeer.

Document Recommendations: OverCite could provide a *recommendation* feature similar to those found in popular Web sites like Amazon. This would require OverCite to track individual users' activities. OverCite could then recommend documents based on either previous downloads, previous queries, or downloads by others with similar interests.

Plagiarism Checking: Plagiarism has only been an occasional problem in major conferences, but with increasing volumes of papers and pressure to publish, this problem will likely become more serious. OverCite could make its database of shingles available to those who wish to check whether one paper's text significantly overlaps any other papers'.

More documents: Most authors do not explicitly submit their newly written papers to CiteSeer. Instead, they rely on CiteSeer to crawl conference Web pages to find new content. CiteSeer could be far more valuable to the community if it could support a larger corpus and, in particular, if it included more preprints and other recently written material. While faster and more frequent crawling might help in this regard, the situation could only be substantially improved if authors took a more active role in adding their material.

As an extreme case, one could imagine that funding agencies and conferences require all publications under a grant and submissions to a conference be entered into OverCite, making them immediately available to the community.[1] Going one step further, one could imagine that program committees annotate submissions in OverCite with comments about the contributions of the paper. Users could then decide based on the comments of the PC which papers to read (using the document-alert feature). This approach would have the additional benefit that users have access to papers that today are rejected from a conference due to limited program time slots.

Potential impact: Radical changes, such as the one above, to the process of dissemination of scientific results are likely to happen only in incremental steps, but are not out of the question. Theoretical physics, for example, uses a preprint collection as its main document repository; insertion into the repository counts as the "publication date" for resolving credit disputes and, more importantly, researchers routinely scan the list of new submissions to find relevant papers. This manual mode works less well for computer science, due in part to the diverse set of sub-disciplines and large number of papers. OverCite, however, could be the enabler of such changes for computer science, because of its scalable capacity and ability to serve many queries.

[1] This would require rethinking anonymous submissions or providing support for anonymous submissions in OverCite.

Chapter 8

Discussion and Conclusion

8.1 DHT Application Design

OverCite is among the first generation of complex DHT applications, and benefits from using a DHT in three major ways. First, using a DHT greatly simplifies a complex, distributed system like OverCite by serving as a distributed, robust archive of data while automatically handling failures. By leveraging existing DHash functionality, the OverCite implementation is relatively small at only 10,000 lines of code (DHash and Chord are roughly 40,000 lines of code). Second, the DHT simplifies the coordination of distributed activities, such as crawling. Finally, the DHT acts as a rendezvous point for producers and consumers of meta-data and documents. OverCite, by employing a DHT in this manner, can harness and coordinate the resources of many cooperating sites, and will be able to provide a scalable, feature-rich service to the community.

While the DHT has proven to be a useful tool for OverCite, a number of design challenges arose during the course of building our prototype. For example, putting values into the DHT with an unrestricted choice of key is a recognized need of DHT applications [35]. However, not all DHTs (DHash, in particular) support this feature by default, partly due to the complexity of maintaining the integrity of such data. We found that the ability to store mutable data under an arbitrary key was essential to OverCite, and recommend that future DHT designs include this feature. Block update logs are a useful and elegant way to implement non-content-hash blocks, and provide viable, flexible consistency semantics to applications.

One potential restriction facing a DHT-based application is the rate at which the DHT can satisfy puts and gets. The OverCite implementation throttles DHT requests at many places in the code; otherwise, high client load could easily overwhelm the DHT. Similarly, the DHash data maintenance algorithm for mutable blocks did not originally throttle its own bandwidth usage. Thus, when a new node joined the system, it would immediately become flooded with traffic creating replicas of many meta-data blocks simultaneously, causing congestion. The OverCite deployment uses an altered version of DHash that throttles this data maintenance traffic.

8.2 Future Work

Though the basic OverCite implementation is complete, there remain many ways to improve the system in the future. We plan to enhance and extend OverCite in the following ways:

- One barrier to large-scale OverCite deployment, in addition to obtaining and deploying more resources in the wide area, is the lack of an automatic document set reconciliation algorithm between nodes in the same index partition. We plan to explore Merkle trees [8, 32] as an efficient means to gossip the identifiers of indexed documents between nodes.

- Before OverCite can be fully operational, it should include a crawler module that discovers, inserts, and indexes new research documents appearing on the Web. The OverCite design in Section 3 includes a preliminary description of a distributed crawler, but is not yet implemented.

- We plan to deploy OverCite on many more nodes over the course of the next several months. As we scale up the network, we hope to gain a better understanding of OverCite's performance on a larger set of nodes.

- We plan to launch OverCite as a service for the academic community in the near future. Along with the above items, such a service would require inserting the full CiteSeer repository into the DHT, creating an attractive HTML appearance for Web users, and thoroughly testing and debugging the robustness of all aspects of the code.

- The features listed in Section 7 would provide valuable services for the research community, and would help draw users to OverCite. We will focus on adding new features to OverCite once a large-scale deployment is live.

- One goal for OverCite is to provide an open programming interface to its meta-data and documents. Such an API would allow the community to implement new features on OverCite (such as those listed in Section 7) and analyze the data in arbitrary ways.

- We expect OverCite to run on a relatively small number of nodes, and the Chord lookup service used by DHash is optimized for much larger deployments. A new lookup protocol under development at MIT, Accordion [26], performs well at both small and large scales. Accordion will ensure that OverCite achieves good performance for low overhead while the size of the network is small, and will adapt to fit the needs of an expanding network as well. We plan to run OverCite on Accordion as soon as a stable implementation is available.

8.3 Conclusion

OverCite is a distributed digital research library solution. As a community-controlled alternative to CiteSeer and Google Scholar, OverCite can aggregate donated resources at many institutions to provide document search and retrieval service to researchers worldwide. OverCite uses a DHT to store data and coordinate services, and partitions its index across nodes for performance and scalability.

The OverCite implementation presented in this thesis can service more queries per second than a centralized server, despite the addition of DHT operations and remote index communication. These extra resources can be used to sustain interesting new digital library features, such as plagiarism detection and document alerts, that may be too resource-intensive to deploy in a centralized environment. Future deployments of OverCite, including many of the features discussed in this thesis, will be made available to the research community in the near future.

Bibliography

[1] The ACM Digital Library. `http://portal.acm.org/dl.cfm`.

[2] Akamai technologies, inc. `http://www.akamai.com`.

[3] Ross J. Anderson. The Eternity Service. In *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology*, 1996.

[4] arXiv.org e-Print archive. `http://www.arxiv.org`.

[5] Mayank Bawa, Gurmet Singh Manku, and Prabhakar Raghavan. SETS: Search enhanced by topic segmentation. In *Proceedings of the 2003 SIGIR*, July 2003.

[6] Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, June 1997.

[7] Timo Burkard. Herodotus: A peer-to-peer web archival system. Master's thesis, Massachusetts Institute of Technology, May 2002.

[8] Josh Cates. Robust and efficient data management for a distributed hash table. Master's thesis, Massachusetts Institute of Technology, May 2003.

[9] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of the 2003 SIGCOMM*, August 2003.

[10] Jungchoo Cho and Hector Garcia-Molina. Parallel crawlers. In *Proceedings of the 2002 WWW Conference*, May 2002.

[11] Bram Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003.

[12] Frank Dabek, M. Frans Kaashoek, Jinyang Li, Robert Morris, James Robertson, and Emil Sit. Designing a DHT for low latency and high throughput. In *Proceedings of the 1st NSDI*, March 2004.

[13] The Digital Object Identifier system. `http://www.doi.org`.

[14] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66:614–656, 2003.

[15] Michael J. Freedman, Eric Freudenthal, and David Maziéres. Democratizing content publication with Coral. In *Proceedings of the 1st NSDI*, March 2004.

[16] Omprakash D Gnawali. A keyword set search system for peer-to-peer networks. Master's thesis, Massachusetts Institute of Technology, June 2002.

[17] Google Scholar. `http://scholar.google.com`.

[18] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. In *Proceedings of the 1st NSDI*, March 2004.

[19] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the Internet with PIER. In *Proceedings of the 19th VLDB*, September 2003.

[20] Inspec. `http://www.iee.org/Publish/INSPEC/`.

[21] J. Kannan, B. Yang, S. Shenker, P. Sharma, S. Banerjee, S. Basu, and S. J. Lee. SmartSeer: Continuous queries over CiteSeer. Technical Report UCB//CSD-05-1371, UC Berkeley, Computer Science Division, January 2005.

[22] Maxwell Krohn. Building secure high-performance web services with OKWS. In *Proceedings of the 2004 Usenix Technical Conference*, June 2004.

[23] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999. `http://www.citeseer.org`.

[24] Jinyang Li. Searchy. `http://pdos.csail.mit.edu/~jinyang/searchy`.

[25] Jinyang Li, Boon Thau Loo, Joseph M. Hellerstein, M. Frans Kaashoek, David Karger, and Robert Morris. On the feasibility of peer-to-peer web indexing and search. In *Proceedings of the 2nd IPTPS*, February 2003.

[26] Jinyang Li, Jeremy Stribling, M. Frans Kaashoek, and Robert Morris. Bandwidth-efficient management of DHT routing tables. In *Proceedings of the 2nd NSDI*, May 2005.

[27] Witold Litwin, Marie-Anna Neimat, and Donovan A. Schneider. LH* — a scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.

[28] Boon Thau Loo, Owen Cooper, and Sailesh Krishnamurthy. Distributed web crawling over DHTs. Technical Report UCB//CSD-04-1332, UC Berkeley, Computer Science Division, February 2004.

[29] Boon Thau Loo, Ryan Huebsch, Ion Stoica, and Joseph M. Hellerstein. The case for a hybrid P2P search infrastructure. In *Proceedings of the 3rd IPTPS*, February 2004.

[30] Petar Maymounkov and David Maziéres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st IPTPS*, March 2002.

[31] David Mazières. A toolkit for user-level file systems. In *Proceedings of the 2001 Usenix Technical Conference*, June 2001.

[32] Ralph C. Merkle. A digital signature based on a conventional encryption function. pages 369–378, 1988.

[33] Jeffrey Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. In *Proceedings of the 1996 Usenix Technical Conference*, January 1996.

[34] Patrick Reynolds and Amin Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of the 4th International Middleware Conference*, June 2003.

[35] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: A public DHT service and its uses. In *Proceedings of the 2005 SIGCOMM*, August 2005.

[36] David S. H. Rosenthal and Vicky Reich. Permanent web publishing. In *Proceedings of the 2000 USENIX Technical Conference, Freenix Track*, June 2000.

[37] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, November 2001.

[38] Shuming Shi, Guangwen Yang, Dingxing Wang, Jin Yu, Shaogang Qu, and Ming Chen. Making peer-to-peer keyword searching feasible using multi-level partitioning. In *Proceedings of the 3rd IPTPS*, February 2004.

[39] Aameek Singh, Mudhakar Srivatsa, Ling Liu, and Todd Miller. Apoidea: A decentralized peer-to-peer architecture for crawling the world wide web. In *Proceedings of the SIGIR 2003 Workshop on Distributed Information Retrieval*, August 2003.

[40] MacKenzie Smith. Dspace for e-print archives. *High Energy Physics Libraries Webzine*, (9), March 2004. `http://dspace.org`.

[41] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, pages 149–160, 2002.

[42] Torsten Suel, Chandan Mathur, Jo-Wen Wu, Jiangong Zhang, Alex Delis, Mehdi Kharrazi, Xiaohui Long, and Kulesh Shanmugasundaram. ODISSEA: A peer-to-peer architecture for scalable web search and information retrieval. In *Proceedings of the International Workshop on the Web and Databases*, June 2003.

[43] Chunqiang Tang and Sandhya Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of the 1st NSDI*, March 2004.

[44] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22nd ICDCS*, July 2002.

[45] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.