OverCite: A Cooperative Digital Research Library

Jeremy Stribling*, Isaac G. Councill†, Jinyang Li*,
M. Frans Kaashoek*, David R. Karger*, Robert Morris*, Scott Shenker‡

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 proposal for a new architecture for a distributed and cooperative research library based on a distributed hash table (DHT). The new architecture will harness resources at many sites, and thereby be able to support new features such as document alerts and scale to larger data sets.

1 Introduction

CiteSeer is a popular repository of scientific papers for the computer science community [12], 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 recently took over the considerable task of running and maintaining the system (see Table 1).

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. A commercially-managed system, such as Google Scholar, is one feasible solution; however, because of CiteSeer's value to the community, it is likely that many institutions would be willing to donate the use of machines and bandwidth at their sites in return for more control over its evolution. Thus, for CiteSeer to prosper and grow as a noncommercial enterprise, it must be adapted to run on a distributed set of donated nodes [11].

OverCite is a design that allows such an aggregation of distributed resources, using a DHT infrastructure. Our emphasis is not on the *novelty* of the design, but on its *benefits*. The DHT's role as a distributed storage layer, coupled with its robust and scalable models for data management and peer communication, allows the decentralization of the CiteSeer infrastructure and the inclusion of additional CPU and storage resources. Besides serving as a distributed, robust archive of data, 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.

By potentially aggregating many resources in this manner, CiteSeer could offer many more documents and features, enabling it to play an even more central role in the community. We are currently developing an OverCite prototype, and hope to make it available as a service to the community in the future.

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 displays 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 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

*MIT Computer Science and Artificial Intelligence Laboratory: {strib, jinyang, kaashoek, karger, rtm}@csail.mit.edu
†PSU School of Information Sciences and Technology: igc2@psu.edu
‡UC Berkeley and ICSI: shenker@icsi.berkeley.edu

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. Isaac G. Councill receives support from NSF SGER Grant IIS-0330783 and Microsoft Research.

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 1: Statistics for the PSU CiteSeer deployment.

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.

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 that contain 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.

Table 1 lists statistics for the current deployment of CiteSeer at PSU. 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.

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 executes the user interface 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 [13, 18, 21, 22], 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 [9, 14, 15]. 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, so that 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 only the DIDs 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 [7]). If query scalability becomes an issue, we plan to explore techniques from recent DHT search proposals [1, 8, 10, 17, 19, 22] or unstructured peer-to-peer search optimizations [4, 23].

Web crawler. The OverCite crawler design builds on several existing proposals for distributed crawling (*e.g.*, [3, 5, 16, 20]). 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 query words from the user, sends them to inverted index servers, collects the results and ranks them, fetches meta-data from the DHT for the top-ranked results, and displays them to 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.

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 2: The data structures OverCite stores in the DHT.

3.2 Tables

Table 2 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 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* [2], 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.

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 over 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 [6]. 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 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 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 [6]). With these parameter choices, and the measurements from CiteSeer in Table 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.

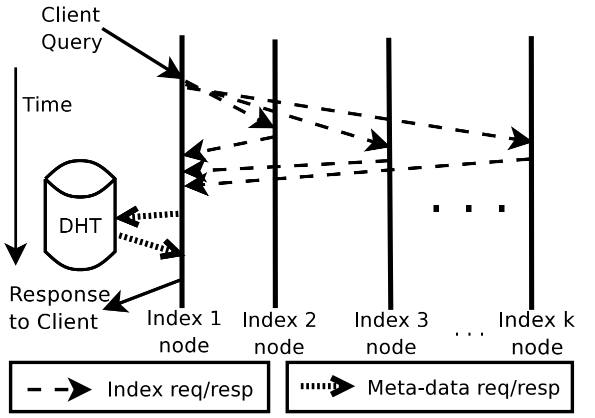

Figure 1: The timeline of a query in OverCite, and the steps involved. Each vertical bar represents a node with a different index partition.

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; this cache need not be up to date. When all k nodes return their top m matches, along with the context of the matches and the value of rank metric, the originating node looks up the meta-data for the top b matches. Figure 1 depicts this process.

The packets containing the queries will be relatively small; however, each response will contain the identifiers of each matching document, the context of each match, and the value of the rank metric. If there are n participating nodes, each DID is 20 bytes, and the context and rank metric value together are 50 bytes, each query consumes about $70mk$ bytes of traffic. Assuming 250,000 searches per day, $k = 20$, and returning $m = 10$ results per query per node, our query design adds 3.5 GB of traffic per day to the network (or 35 MB per node). This is a reasonably small fraction of the traffic currently served by CiteSeer (34.4 GB). This does not include the meta-data lookup traffic for the top b matches, which is much smaller (a reasonable value for b is 10 or 20).

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 cur-

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

rently 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.

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. After receiving responses from all nodes, the node looks up the meta-data for the top matches in parallel. 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.

5 Features and potential impact

Given the additional resources available with OverCite's design, 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 [11] 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.[2] 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 subdisciplines 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.

Acknowledgments

The comments of Jayanthkumar Kannan, Beverly Yang, Sam Madden, Anthony Joseph, the MIT PDOS research group, and the anonymous reviewers greatly improved

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

this work. We also thank C. Lee Giles for his continued support at PSU.

References

[1] BAWA, M., MANKU, G. S., AND RAGHAVAN, P. SETS: Search enhanced by topic segmentation. In *Proceedings of the 2003 SIGIR* (July 2003).

[2] BRODER, A. Z. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences* (June 1997).

[3] BURKARD, T. Herodotus: A peer-to-peer web archival system. Master's thesis, Massachusetts Institute of Technology, May 2002.

[4] CHAWATHE, Y., RATNASAMY, S., BRESLAU, L., LANHAM, N., AND SHENKER, S. Making Gnutella-like P2P systems scalable. In *Proc. of SIGCOMM* (August 2003).

[5] CHO, J., AND GARCIA-MOLINA, H. Parallel crawlers. In *Proceedings of the 2002 WWW Conference* (May 2002).

[6] DABEK, F., KAASHOEK, M. F., LI, J., MORRIS, R., ROBERTSON, J., AND SIT, E. Designing a DHT for low latency and high throughput. In *Proceedings of the 1st NSDI* (March 2004).

[7] FAGIN, R., LOTEM, A., AND NAOR, M. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences 66* (2003), 614–656.

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

[9] GUPTA, A., LISKOV, B., AND RODRIGUES, R. Efficient routing for peer-to-peer overlays. In *Proceedings of the 1st NSDI* (Mar. 2004).

[10] HUEBSCH, R., HELLERSTEIN, J. M., LANHAM, N., LOO, B. T., SHENKER, S., AND STOICA, I. Querying the Internet with PIER. In *Proceedings of the 19th VLDB* (Sept. 2003).

[11] KANNAN, J., YANG, B., SHENKER, S., SHARMA, P., BANERJEE, S., BASU, S., AND LEE, S. J. SmartSeer: Continuous queries over CiteSeer. Tech. Rep. UCB//CSD-05-1371, UC Berkeley, Computer Science Division, Jan. 2005.

[12] LAWRENCE, S., GILES, C. L., AND BOLLACKER, K. Digital libraries and autonomous citation indexing. *IEEE Computer 32*, 6 (1999), 67–71. `http://www.citeseer.org`.

[13] LI, J., LOO, B. T., HELLERSTEIN, J. M., KAASHOEK, M. F., KARGER, D., AND MORRIS, R. On the feasibility of peer-to-peer web indexing and search. In *Proceedings of the 2nd IPTPS* (Feb. 2003).

[14] LI, J., STRIBLING, J., KAASHOEK, M. F., AND MORRIS, R. Bandwidth-efficient management of DHT routing tables. In *Proceedings of the 2nd NSDI* (May 2005).

[15] LITWIN, W., NEIMAT, M.-A., AND SCHNEIDER, D. A. LH* — a scalable, distributed data structure. *ACM Transactions on Database Systems 21*, 4 (1996), 480–525.

[16] LOO, B. T., COOPER, O., AND KRISHNAMURTHY, S. Distributed web crawling over DHTs. Tech. Rep. UCB//CSD-04-1332, UC Berkeley, Computer Science Division, Feb. 2004.

[17] LOO, B. T., HUEBSCH, R., STOICA, I., AND HELLERSTEIN, J. M. The case for a hybrid P2P search infrastructure. In *Proceedings of the 3rd IPTPS* (Feb. 2004).

[18] REYNOLDS, P., AND VAHDAT, A. Efficient peer-to-peer keyword searching. In *Proceedings of the 4th International Middleware Conference* (June 2003).

[19] SHI, S., YANG, G., WANG, D., YU, J., QU, S., AND CHEN, M. Making peer-to-peer keyword searching feasible using multi-level partitioning. In *Proceedings of the 3rd IPTPS* (Feb. 2004).

[20] SINGH, A., SRIVATSA, M., LIU, L., AND MILLER, T. Apoidea: A decentralized peer-to-peer architecture for crawling the world wide web. In *Proceedings of the SIGIR 2003 Workshop on Distributed Information Retrieval* (Aug. 2003).

[21] SUEL, T., MATHUR, C., WU, J.-W., ZHANG, J., DELIS, A., KHARRAZI, M., LONG, X., AND SHANMUGASUNDARAM, K. 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).

[22] TANG, C., AND DWARKADAS, S. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of the 1st NSDI* (Mar. 2004).

[23] YANG, B., AND GARCIA-MOLINA, H. Improving search in peer-to-peer networks. In *Proceedings of the 22nd ICDCS* (July 2002).