OverCite: A Distributed, Cooperative CiteSeer

Jeremy Stribling, Jinyang Li[†], Isaac G. Councill[††], M. Frans Kaashoek, Robert Morris

MIT Computer Science and Artificial Intelligence Laboratory
[†]*New York University and MIT CSAIL, via UC Berkeley* [††]*Pennsylvania State University*

Abstract

CiteSeer is a popular online resource for the computer science research community, allowing users to search and browse a large archive of research papers. CiteSeer is expensive: it generates 35 GB of network traffic per day, requires nearly one terabyte of disk storage, and needs significant human maintenance.

OverCite is a new digital research library system that aggregates donated resources at multiple sites to provide CiteSeer-like document search and retrieval. OverCite enables members of the community to share the costs of running CiteSeer. The challenge facing OverCite is how to provide scalable and load-balanced storage and query processing with automatic data management. OverCite uses a three-tier design: presentation servers provide an identical user interface to CiteSeer's; application servers partition and replicate a search index to spread the work of answering each query among several nodes; and a distributed hash table stores documents and meta-data, and coordinates the activities of the servers.

Evaluation of a prototype shows that OverCite increases its query throughput by a factor of seven with a nine-fold increase in the number of servers. OverCite requires more total storage and network bandwidth than centralized CiteSeer, but spreads these costs over all the sites. OverCite can exploit the resources of these sites to support new features such as document alerts and to scale to larger data sets.

1 Introduction

Running a popular Web site is a costly endeavor, typically requiring many physical machines to store and process data and a fast Internet pipe to push data out quickly. It's common for non-commercial Web sites to be popular yet to lack the resources to support their popularity. Users of such sites are often willing to help out, particularly in the form of modest amounts of compute power and network traffic. Examples of applications that thrive on volunteer resources include SETI@home, BitTorrent, and volunteer software distribution mirror sites. Another prominent example is the PlanetLab wide-area testbed [36], which is made up of hundreds of donated machines over many different institutions. Since donated resources are distributed over the wide area and are usually abundant, they also allow the construction of a more fault tolerant system using a geographically diverse set of replicas.

In order to harness volunteer resources at many sites, a Web service needs a design that will increase its capacity as servers are added. This paper explores such a design in OverCite, a multi-site version of the CiteSeer repository of computer science research papers [30]. We choose CiteSeer for our study because of its value to the research community. However, despite its popularity, CiteSeer went mostly unmaintained after its initial development at NEC until a volunteer research group at Pennsylvania State University (PSU) took over the considerable task of running and maintaining the system.

1.1 Multi-site Web Design

Many designs and tools exist for distributing a Web service within a single-site cluster. The three-tier design is a common approach—a load-balancing front-end, application servers, and a shared back-end database—and services may also use techniques such as DDS [25], TACC [20], and MapReduce [17]. These solutions take advantage of reliable high-speed LANs to coordinate the servers in the cluster. Such solutions are less well suited to servers spread over the Internet, with relatively low speed and less reliable connectivity [3].

Existing approaches to multi-site services include mirroring, strict partitioning, caching, and more recently distributed hash tables (DHTs). Mirroring and strict parti-

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.

tioning eliminate inter-site communication during ordinary operations. Mirroring is not attractive for storage-intensive services: CiteSeer, for example, would require each mirror site to store nearly a terabyte of data. Partitioning the service among sites works only if each client request clearly belongs to a particular partition, but this is not true for keyword queries in CiteSeer; in addition, CiteSeer must coordinate its crawling activities among all sites. While content distribution networks such as Akamai [2], Coral [21], or CoDeeN [48] would help distribute static content such as the documents stored by CiteSeer, they would not help with the dynamic portion of the CiteSeer workload: keyword searches, navigation of the graph of citations between papers, ranking papers and authors in various ways, and identification of similarity among papers. Similarly, as discussed in the related work (see Section 6), no existing DHT has been used for an application of the complexity of CiteSeer.

1.2 OverCite

What is needed is a design that parallelizes CiteSeer's operations over many sites to increase performance, partitions the storage to minimize the per-site burden, allows the coordination required to perform keyword searches and crawling, and can tolerate network and site failures. OverCite's design satisfies these requirements. Like many cluster applications, it has a three-tier design: multiple Web front-ends that accept queries and display results, application servers that crawl, generate indices, perform keyword searches on the indices, and a DHT back-end that aggregates the disks of the donated machines to store documents, meta-data, and coordination state.

The DHT back-end provides several distinct benefits. First, it is self-managing, balancing storage load automatically as volunteer servers come and go. Second, it handles replication and location of data to provide high availability. Finally, the DHT provides a convenient rendezvous service for producers and consumers of meta-data that must coordinate their activities. For example, once a node at one site has crawled a new document and stored it in the DHT, the document will be available to DHT nodes at other sites.

OverCite's primary non-storage activity is indexed keyword search, which it parallelizes with a scheme used in cluster-based search engines [7, 20]. OverCite divides the inverted index in k partitions. Each node stores a copy of one partition on its local disk; with n nodes, n/k nodes store a particular partition. OverCite sends a user query to k index servers, one for each partition, and aggregates the results from these index servers.

1.3 Contributions

The contributions of this paper are as follows[1]:

- A three-tier DHT-backed design that may be of general use to multi-site Web services.
- OverCite, a multi-site deployment of CiteSeer.
- An experimental evaluation with 27 nodes, the full CiteSeer document set, and a trace of user queries issued to CiteSeer. A single-node OverCite can serve 2.8 queries/s (CiteSeer can serve 4.8 queries/s), and OverCite scales well: with 9 nodes serving as front-end query servers, OverCite can serve 21 queries/s.
- A case study of a challenging use of a DHT. OverCite currently stores 850 GB of data (amounting to tens of millions of blocks), consuming about a factor of 4 more total storage than CiteSeer itself. Each keyword query involves tens of DHT operations and is completed with reasonable latency.

We conjecture that OverCite's three-tier design may be useful for many Web services that wish to adopt a multi-site arrangement. Services tend to consist of a mix of static content and dynamic operations. As long as the consistency requirements are not strict, a DHT can be used as a general-purpose back-end, both to hold inter-site coordination state and to spread the storage and serving load of large static documents.

1.4 Roadmap

Section 2 describes the design and operation of the current CiteSeer implementation. Section 3 gives the design of OverCite. Section 4 details the OverCite prototype implementation, and Section 5 evaluates the implementation. Section 6 discusses related work, and finally Section 7 concludes.

2 CiteSeer Background

CiteSeer [30] crawls, stores and indexes more than half a million research papers. CiteSeer's hardware consists of a pair of identical servers running the following components. 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 has not already fetched, CiteSeer fetches the file and parses it to extract text and citations. Then it applies heuristics to check if the

[1]We presented a preliminary design, without an implementation, in an earlier workshop paper [45].

Number of papers	674,720
New documents per week	1000
HTML pages visited	113,000
Total document storage	803 GB
Avg. document size (all formats)	735 KB
Total meta-data storage	45 GB
Total inverted index size	22 GB
Hits per day	800,000
Searches per day	250,000
Total traffic per day	34 GB
Document traffic per day	21 GB
Avg. number of active conns	68
Avg. load per CPU	66%

Table 1: Statistics of the PSU CiteSeer deployment.

document duplicates an existing document; if not, it adds meta-data about the document to its tables. CiteSeer periodically updates its inverted index with newly crawled documents, indexing only the first 500 words of each document. 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 found a PDF or Postscript file, and a citation ID (CID) to every bibliography entry within a document. CiteSeer knows about many papers for which it has seen citations but not found a document file. For this reason CiteSeer assigns a "group ID" (GID) to each known paper for use in contexts where a file is not required. The GID also serves to connect newly inserted documents to previously discovered citations. All the IDs are numerically increasing 14-byte numbers.

CiteSeer stores the PDF/PostScript of each research paper (as well as the ASCII text extracted from it) in a local file system. In addition, CiteSeer stores the following meta-data tables to help identify papers and filter out possible duplicates:

1. The document meta-data table (`Docs`), indexed by DID, which records each document's authors, title, year, abstract, GID, CIDs of the document's citations, number of citations to the document, etc.
2. The citation meta-data table (`Cites`), indexed by CID, which records each citation's GID and citing document DID.
3. A table (`Groups`) mapping each GID to the corresponding DID (if a DID exists) and the list of CIDs that cite it.
4. A table indexed by the checksum of each fetched document file, used to decide if a file has already been processed.

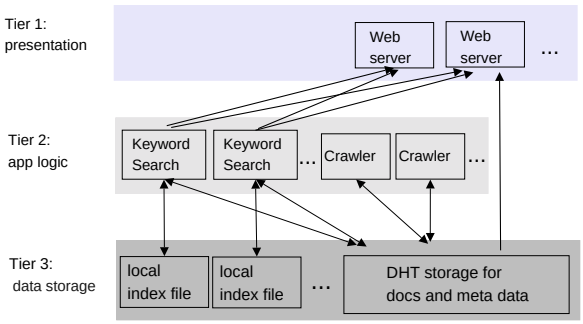

Figure 1: Overview of OverCite's three-tier architecture. The modules at all tiers are run at many (if not all) nodes. The Tier 1 Web server interacts with multiple search engines on different nodes to perform a single user keyword search. The Tier 2 modules use the underlying DHT to store documents and corresponding meta-data. The keyword search engine also stores data locally outside the DHT.

5. A table indexed by the hash of every sentence CiteSeer has seen in a document, used to gauge document similarity.
6. A table (`URLs`) to keep track of which Web pages need to be crawled, indexed by URL.
7. A table (`Titles`) 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 deployment of CiteSeer at PSU as of September 2005. The CiteSeer Web site uses two servers each with two 2.8 GHz processors. The two servers run independently of each other and each has a complete collection of PDF/Postscript documents, inverted indices, and meta-data. Most of the CPU time is used to satisfy keyword searches and to convert document files to user-requested formats.. The main costs of searching are lookups in the inverted index, and collecting and displaying meta-data about search results.

3 Design

A primary goal of OverCite's design is to ensure that its performance increases as volunteers contribute nodes. OverCite addresses this challenge with a three-tier design (see Figure 1), similar to cluster-based Web sites, but with the database replaced with a DHT. The modules in Tier 1 accept keyword queries and aggregate results from Tier 2 modules on other nodes to present the traditional CiteSeer interface to users. The Tier 2 modules perform keyword searches and crawling for new documents. The Tier 1 and 2 modules use the DHT servers in Tier 3 to store and fetch

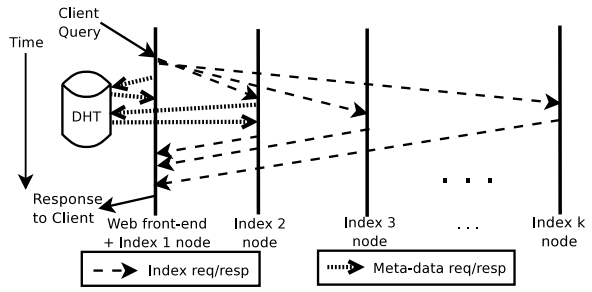

Figure 2: The timeline of a query in OverCite, and the steps involved. Each vertical bar represents a node with a different index partition. DHT meta-data lookups are only required at index servers without cached copies of result meta-data.

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

Table 2: The data structures OverCite stores in the DHT. The `Files` table stores immutable content hash blocks for PDF/Postscript documents, indexed by a root content hash key FID. All the rest of the tables are stored as append-only blocks.

the documents, meta-data, and shared state for coordination. When a site contributes one or more nodes to the system, each node starts a DHT server to put the donated disk space and network bandwidth to use.

3.1 Overview: The Life of a Query

Figure 2 depicts the timeline of a query. A subset of OverCite nodes run a Web user interface, using round-robin DNS or OASIS [22] to spread the client load. The front-end accepts a query with search terms from the user and uses a scheme similar to the ones used by cluster-based search engines to parallelize the search: the front-end sends the query to k index servers, each responsible for $1/k$th of the index.

Given a keyword query, each index server searches for the m most highly-ranked documents in its local index; the search engine ranks documents by the relative frequency and position of the query keywords within the text as well as by citation count. The index server looks up the meta-data for these m documents in the DHT to supply additional information like citation counts back to the front-end. Servers cache this meta-data locally to speed up future searches. The front-end is responsible for merging the results from the k index servers and displaying the top m to the user.

3.2 Global Data Structures

OverCite stores document files and meta-data in a DHT shared among all the nodes. Table 2 lists the data tables that OverCite stores in the DHT. In addition to existing CiteSeer data structures, `Crawl` and `URLs` tables are added to coordinate distributed crawling activity as explained in Section 3.4.

OverCite needs the following four properties from the underlying DHT. First, the DHT should be able to store a large amount of data with a put/get interface, where a block is named by the hash of the block's content. To balance the storage of data well across the nodes, OverCite's blocks are at most 16 KB in size. OverCite stores large files, such as PDF and PostScript files, as a Merkle tree of content-hash blocks [35]; the file's identifier, or FID, is the DHT key of the root block of the Merkle tree.

Second, the DHT must support append-only blocks [38]. OverCite stores each entry in the meta-data tables (`Docs`, `Cites`, and `Groups`) listed in Table 2 as an append-only block, indexed using a randomly-generated 20-byte DID, CID or GID. OverCite treats the append-only blocks as an update log, and reconstructs the current version of the data by applying each block of appended data as a separate update. Append-only logs can grow arbitrarily large as a document's meta-data is updated, but in practice is usually small.

OverCite uses append-only blocks, rather than using fully-mutable blocks, because append-only blocks simplify keeping data consistent. Any OverCite node can append to an append-only block at any time, and the DHT ensures that eventually all replicas of the block will see all appended data (though strict order is not necessarily enforced by the DHT).

Third, the DHT should try to keep the data available (perhaps by replicating it), although ultimately OverCite can regenerate it by re-crawling. Furthermore, the DHT should be resilient in the face of dynamic membership changes (churn), though we do no expect this to be a major issue in a managed, cooperative system like OverCite. Finally, the DHT should support quick (preferably one-hop) lookups in systems with a few hundred nodes.

3.3 Local Data Structures

Each OverCite node stores data required for it to participate in keyword searches on its local disk. This local data includes an inverted index yielding the list of documents containing each word and extracted ASCII text for each document in the inverted index that is used to present the context around each search result.

OverCite partitions the keyword index by document, so that each node's inverted index includes documents from only one partition. Partitioning reduces the index storage requirement at each node, and reduces latency when load is low by performing each query in parallel on multiple nodes. OverCite typically has fewer index partitions than nodes, and replicates each partition on multiple nodes. Replication increases fault-tolerance and increases throughput when the system is busy since it allows different queries to be processed in parallel. This index partitioning and replication strategy is used by cluster-based search engines such as Google [7] and HotBot [20]. Compared with several other peer-to-peer search proposals [31, 37, 46, 47], partitioning by document is bandwidth efficient, load-balanced, and provides the same quality of results as a centralized search engine.

OverCite uses k index partitions, where k is less than the number of nodes (n). Each node stores and searches one copy of one index partition; if there are n nodes, there are n/k copies of each index partition. The front-end sends a copy of each query to one server in each partition. Each of the k servers processes the query using $1/k$'th of the full index, which requires about $1/k$'th the time needed to search a complete index.

A large k decreases query latency at low load due to increased parallelism, and may also increase throughput since a smaller inverted index is more likely to fit in each node's disk cache. However, a large k also increases network traffic, since each node involved in a query returns about 170 bytes of information about up to m of its best matches. Another reason to restrict k is that the overall keyword search latency may be largely determined by the response time of the slowest among the $k-1$ remote index servers. This effect is somewhat mitigated because the front-end sends each query to the lowest-delay replica of each index partition. We also plan to explore forwarding queries to the least-loaded index partition replicas among nearby servers.

A node's index partition number is its DHT identifier mod k. A document's index partition number is its DID mod k. When the crawler inserts a new document into the system, it notifies at least one node in the document's partition; the nodes in a partition periodically exchange notes about new documents.

3.4 Web Crawler

The OverCite crawler design builds on several existing proposals for distributed crawling (*e.g.*, [9, 12, 32, 41]). Nodes coordinate the crawling effort via a list of to-be-crawled Web page URLs stored in the DHT. Each crawler process periodically chooses a random entry from the list and fetches the corresponding page.

For each link to a Postscript or PDF file a node finds on a Web page, the crawler performs a lookup in the `URLs` table to see whether the document has already been downloaded. After the download, the crawler parses the file – extracting meta-data (*e.g.*, title, authors, citations, etc.) as well as the bare ASCII text of the document – and 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. OverCite uses shingles [8] instead of individual sentences as in CiteSeer for duplicate detection. Checking for duplicates using shingles is effective and efficient, resulting in a small `Shins` table. If the document is not a duplicate, the crawler inserts the document into `Files` as Postscript or PDF. The node also updates `Docs`, `Cites`, `Groups`, and `Titles` to reflect this document and its meta-data. The crawler puts the extracted ASCII text in the DHT and informs one index server in the document's partition of newly inserted document's DID.

While many enhancements to this basic design (such as locality-based crawling and more intelligent URL partitioning) are both possible and desirable, we defer optimizations of the basic crawler design to future work. Crawling and fetching new documents will take approximately three times more bandwidth than CiteSeer uses in total, spread out among all the servers. We have shown these calculations in previous work [45].

4 Implementation

The OverCite implementation consists of several software modules, corresponding to the components described in Section 3. The current implementation does not yet include a *Crawler* module; we have populated OverCite with existing CiteSeer documents. The OverCite implementation consists of over 11,000 lines of C++ code, and uses the SFS libasync library [34] to provide an event-driven, single-threaded execution environment for each module. Figure 3 shows the overall interactions of different OverCite modules with each other and the DHT. Modules on the same machine communicate locally through

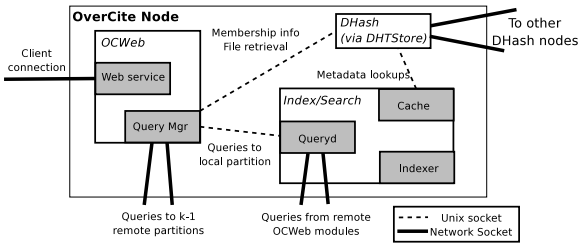

Figure 3: Implementation overview. This diagram shows the communication paths between OverCite components on a single node, and network connections between nodes.

Unix domain sockets; modules on different machines communicate via TCP sockets. All inter-module communication occurs over the Sun RPC protocol.

The *OCWeb* Module. The *OCWeb* module provides a Web interface to accept keyword queries and display lists of matching documents. *OCWeb* is implemented as a module for the OK Web Server (OKWS) [29], 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.

The *DHTStore* Module. The *DHTStore* module acts as an interface between the rest of the OverCite system and the DHT. The module takes the form of a library that can be used by other system components to retrieve meta-data and documents. The implementation uses the DHash [16] DHT. Each OverCite node runs three DHash virtual nodes per physical disk to balance the storage load evenly.

Meta-data Storage. OverCite stores its tables (see Table 2) in the the DHT's single 20-byte key space, with each row of each table represented as a DHT data block. A table row's DHT key is the hash of the corresponding OverCite identifier (e.g. DID, CID, etc.) concatenated with the table's name; for example, the keys used to index the `Docs` table are the hash of the string "Docs" concatenated with the DID of the document. The hashing spreads the meta-data evenly over the DHT nodes. OverCite appends data to some kinds of table rows; for example, OverCite appends to a `Groups` block when it finds a new citation to a document. The current implementation supports the tables listed in Table 2 except `Shins`, `Crawl`, and `URLs`.

The *Index/Search* Module. The *Index/Search* module consists of a query server daemon (*Queryd*), a meta-data and text-file cache, and an index generator (*Indexer*). We chose to implement our own search engine instead of reusing the current built-in CiteSeer search engine, because we have found it difficult to extend CiteSeer's search engine to include new functions (e.g., to experiment with different ranking functions).

The *Indexer* periodically retrieves the ASCII text and meta-data for new documents to be included in the node's index partition from the DHT and caches them on the local disk. It updates the local inverted index file based on the local disk cache. The inverted index consists of posting lists of document numbers and the ASCII file offset pairs for each word. Compared to CiteSeer's inverted index structure, *Indexer*'s inverted index optimizes query speed at the cost of slower incremental index updates. The *Indexer* indexes the first 5000 words of each document (CiteSeer indexes the first 500). The document and offset pairs in a posting list are ranked based on the corresponding citation counts.

Upon receiving a query, *Queryd* obtains the list of matching documents by intersecting the posting lists for different keywords. On-disk posting lists are *mmap*-ed into the memory, causing them to be paged in the buffer cache. *Queryd* scores each result based on the document ranks, the file offsets where a keyword occurs, and the proximity of keywords in the matching document. *Queryd* returns the top m scored documents as soon as it judges that no further lower ranked documents can generate higher scores than the existing top m matches.

For each of the top m matches, *Queryd* obtains the context (the words surrounding the matched keywords) from the on-disk cache of ASCII files. In parallel, it also retrieves the corresponding meta-data from the DHT. Upon completion of both context and meta-data fetches, *Queryd* returns the results to either the local Query manager or the remote *Queryd*.

5 Evaluation

This section explores how OverCite scales with the total number of nodes. 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 the basic scaling 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.

5.1 Evaluation Methods

We deployed OverCite on 27 nodes: sixteen at MIT and eleven spread over North America, most of which are part of the RON test-bed [3]. The DHash instance on each node spawned three virtual nodes for each physical disk to balance load; in total, the system uses 47 physical disks. These disks range in capacity from 35 GB to 400 GB.

We inserted the 674,720 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. CiteSeer uses several heuristics to determine whether a document in its repository is a duplicate of another, and indexes only non-duplicates; OverCite indexes the same set of non-duplicate documents (522,726 documents in total). OverCite currently stores only the *original* copy of each document (*i.e.*, the document originally discovered and downloaded by the crawler), while CiteSeer stores Postscript, compressed Postscript, and PDF versions for every document. We plan to store these versions as well in the near future.

Unless otherwise stated, each document is indexed by its first 5000 words, and each experiment involves two index partitions ($k = 2$). All *Queryd* modules return up to 20 results per query ($m = 20$), 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). The results represent the average of five trials for each experiment.

To evaluate the query performance of OverCite, we used a trace of actual CiteSeer queries, collected in October 2004. 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.

5.2 Query Throughput

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 Tier 1 and 2 servers is non-obvious.

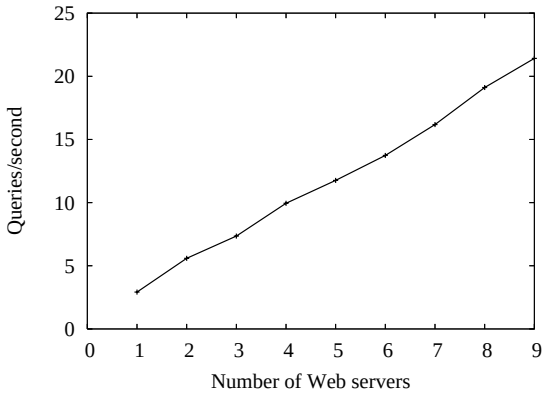

Figure 4: Average query throughput on a distributed OverCite system, as a function of the number of Web (Tier 1) servers. The client issues 128 concurrent queries at a time.

To evaluate scalability, we measured throughput with varying numbers of front-end nodes. The value of k was 2, and the total number of nodes in each configuration is twice the number of front ends. Each front-end is paired with a non-front-end holding the other partition. A client at MIT keeps 128 queries active, spread among all of the available Web servers (randomly choosing which Web server gets which query). The test uses servers chosen at random for each experiment trial.

Figure 4 shows the number of queries per second processed by OverCite, as a function of the number of front-end servers. Adding additional front-end servers linearly increases the query throughput. With a single front-end server OverCite serves about 3 queries per second, while 9 front-end servers satisfy 21 queries per second. Despite the fact that the servers share a common DHT (used when looking up document meta-data), the resources of the different machines can be used by OverCite to satisfy more queries in parallel.

For comparison, a single-server CiteSeer can process 4.8 queries per second with the same workload but slightly different hardware. CiteSeer indexes only the first 500 words per document. The corresponding single-node throughput for OverCite is 2.8 queries per second.

5.3 Performance Breakdown

This subsection evaluates the latency of individual queries. The *Queryd* daemon on a node performs three

Context

k	Search	Context	DHT wait	Total
2	118.14	425.11 (37.29 each)	21.27	564.52
4	78.68	329.04 (32.97 each)	23.97	431.69

No context

k	Search	DHT wait	Total
2	53.31	170.03	223.34
4	28.07	208.30	236.37

Table 3: Average latencies (in ms) of OverCite operations for an experiment that generates context results, and one that does not give context results. **Search** shows the latency of finding results in the inverted index; **Context** shows the latency of generating the context for each result (the per-result latency is shown in parentheses); **DHT wait** shows how long OverCite waits for the last DHT lookup to complete, *after* the search and context operations are completed. Only one query is outstanding in the system.

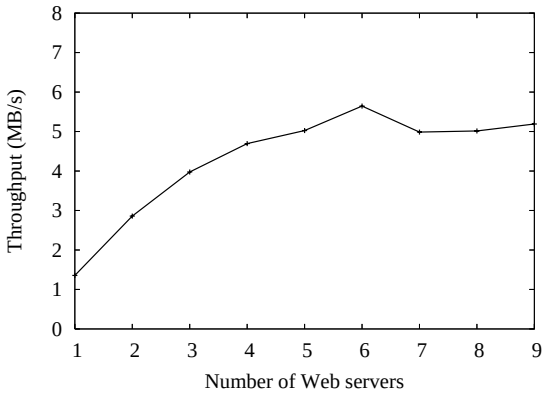

Figure 5: Average throughput of OverCite serving Postscript/PDF files from the DHT, as a function of the number of front-end Web servers.

main operations for each query it receives: a search over its inverted index, the generation of context information for each result, and a DHT meta-data lookup for each result. All DHT lookups happen in parallel, and context generation happens in parallel with the DHT lookups.

Table 3 summarizes the latencies of each individual operation, for experiments with and without context generation. **Search** shows the latency of finding results in the inverted index; **Context** shows the latency of generating the context for each result; **DHT wait** shows how long OverCite waits for the last DHT lookup to complete, *after* the search and context operations are completed. The table gives results for two different values of k, 2 and 4.

The majority of the latency for a single query comes from context generation. For each result the server must read the ASCII text file from disk (if it is not cached in memory), which potentially involves several disk seeks: OverCite currently organizes the text files in a two-level directory structure with 256 directories per level. In the future, we plan to explore solutions that store all the ASCII text for all documents in a single file, to avoid the overhead of reading the directory structures. Because context generation dominates the search time, and DHT lookups occur in parallel with the context generation, OverCite spends little time waiting for the DHT lookups to return; less than 4% of the total latency is spent waiting for and processing the meta-data lookups.

Increasing k decreases the latency of the search over the inverted index, because each partition indexes fewer documents and the size of the inverted index largely determines search speed. Interestingly, search latency is significantly lower when OverCite does not generate context, presumably due to more efficient use of the buffer cache for the inverted index when the ASCII text files are not read.

Without context generation, the meta-data lookups become a greater bottleneck to the latency, and in fact increasing k causes DHT lookups to slow down. With a larger k value, there are more total lookups happening in the system, since each node retrieves its top m results.

5.4 File Downloads

This section evaluates how well OverCite serves PDF and Postscript documents. We measured the rate at which a single front-end at CMU could serve documents to the client at MIT. The client kept 128 concurrent requests active. The network path from CMU to MIT is capable of carrying 11.4 megabytes/second, as measured with `ttcp` using UDP.

OverCite's document throughput averaged 1.3 megabytes/second. The download rate from the single front-end is limited by the rate that the server can download blocks from the DHT, which is determined by the bottleneck bandwidth between the front-end and the slowest node in the system [16]. The measured UDP network capacity over the slowest access link was 1.14 MB/s.

If the client uses multiple front-ends to download files, it can achieve a higher throughput. Figure 5 shows the file-serving throughput of OverCite as a function of the number of front-end Web servers used by the client, when the client requests a total of 128 files concurrently from the system. The throughput plateaus at nearly five times the throughput from a single server, and is similar to the throughput for the same number of servers measured in

Property	Cost
Document/meta-data storage	270 GB
Index size	14 GB
Total storage	284 GB

Table 4: Storage statistics for a centralized server.

a previous evaluation of DHash [16]. Table 1 shows that currently CiteSeer serves only 35 GB/day (or 425 KB/s), a load that our OverCite implementation can easily handle.

5.5 Storage

Finally, we compare the storage costs of OverCite to those of a centralized solution. The centralized solution keeps one copy of the original crawled file (as noted in Section 5.1), the extracted ASCII text, and the full set of CiteSeer meta-data for each document, in addition to a full inverted index built using OverCite's *Indexer* module. We compare this to the storage costs measured on our 27-node, 47-disk, 2-partition OverCite deployment. OverCite storage costs do not include the on-disk cache of ASCII text files used to generate context information; these files are included in the DHT storage costs, and the cache can always be created by downloading the files from the DHT.

Table 4 shows the storage costs of the centralized solution. The total space used is 284 GB, the majority of which is documents and meta-data. Table 5 shows the average per-node storage costs measured on our OverCite deployment. The system-wide storage cost is 1034.3 GB. An individual node in the system with one physical disk and one *Index/Search* module has a cost of 24.9 GB.

If we assume that each individual disk is its own node and has its own copy of the index, the full OverCite system would use 4.1 times as much space as the centralized solution. Given that OverCite's DHash configuration uses a replication factor of 2, this overhead is higher than expected. Some blocks are actually replicated more than twice, because DHash does not delete old copies of blocks when copying data to newly-joined nodes. Storing many small blocks in the database used by DHash also incurs overhead, as does OverCite's Merkle tree format for storing files.

For our current implementation, adding 47 nodes to the system decreased the *per-node* storage costs by about a factor of 11.4; 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 handle $n/4$ times as many documents as a single CiteSeer node.

Property	Individual cost	System cost
Document/meta-data storage	18.1 GB	18.1 GB $\times$ 47 = 850.7 GB
Index size	6.8 GB	6.8 GB $\times$ 27 = 183.6 GB
Total storage	24.9 GB	1034.3 GB

Table 5: Average per-node storage statistics for the OverCite deployment. There are 27 nodes (and 47 disks) in the system.

6 Related Work

Many digital libraries exist. Professional societies such as ACM [1] and IEE [27] maintain online repositories of papers published at their conferences. Specific academic fields often have their own research archives, such as arXiv.org [5], Google Scholar [24], and CiteSeer [30], which allow researchers to search and browse relevant work, both new and old. More recently, initiatives like DSpace [43] and the Digital Object Identifier system [18] 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 at multiple sites across the Internet.

Previous work on distributed library systems includes LOCKSS [39], 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 [4] uses peer-to-peer technology to resist censorship of electronic documents. There have also been a number of systems for searching large data sets [6, 11,23,26,33,40,47,49] and crawling the Web [9,12,32,41] using peer-to-peer systems. We share with these systems a desire to distribute work across many nodes to avoid centralized points of failure and performance bottlenecks.

Services like BitTorrent [14] and Coral [21] provide an alternative style of content distribution to a DHT. Like DHTs such as DHash [16], 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-based three tier design is more closely aligned with Web services that have both dynamic and static content.

DHTs have been used in many applications (e.g., [15, 19, 44]), but few have required substantial storage, or perform intensive calculations with the data stored. PIER [26] is a distributed query engine for wide-area distributed systems, and extends the DHT with new operators. The Place Lab Web service [10] is an investigation into how an unmodified DHT can be used to implement complex functions such as range-queries, but computes with little data (1.4 million small records). Usenet-

DHT [42] stores a substantial amount of data but doesn't require computation on the data. Because these applications are simpler than CiteSeer, they do not require the three-tier design used in this paper.

7 Conclusion and Future Work

Using a three-tier design, OverCite serves more queries per second than a centralized server, despite the addition of DHT operations and remote index communication. 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.

For example, 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 [28] describes a DHT-based alert system for CiteSeer. Other possible features include Amazon-like document recommendations, plagiarism detection, or including a more diverse range of documents, such as preprints or research from other fields.

Since OverCite's architecture allows it to include new resources as they become available, it can scale its capacity to meet the demands of imaginative programmers. We plan to create an open programming interface to the OverCite data (similar to CiteSeer's OAI interface [13]), allowing the community to implement new features and services on OverCite such as those listed above. We plan to launch OverCite as a service for the academic community in the near future to encourage these possibilities.

Acknowledgments

We thank Frank Dabek, Max Krohn, and Emil Sit for their tireless help debugging and discussing; David Karger and Scott Shenker for formative design conversations; Dave Andersen, Nick Feamster, Mike Howard, Eddie Kohler, Nikitas Liogkas, Ion Stoica, and all the RON host sites for volunteering machines, bandwidth, and time; the reviewers for their insightful comments; and C. Lee Giles for his continued support at PSU.

References

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

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

[3] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, M. F., AND MORRIS, R. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)* (2001).

[4] ANDERSON, R. J. The Eternity Service. In *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology* (1996).

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

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

[7] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems 30* (1998).

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

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

[10] CHAWATHE, Y., RAMABHADRAN, S., RATNASAMY, S., LAMARCA, A., SHENKER, S., AND HELLERSTEIN, J. A case study in building layered DHT applications. In *Proceedings of the 2005 SIGCOMM* (Aug. 2005).

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

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

[13] CiteSeer.PSU Open Archive Initiative Protocol. `http://citeseer.ist.psu.edu/oai.html`.

[14] COHEN, B. Incentives build robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems* (June 2003).

[15] DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)* (Oct. 2001).

[16] 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* (Mar. 2004).

[17] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Dec. 2004).

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

[19] DRUSCHEL, P., AND ROWSTRON, A. PAST: Persistent and anonymous storage in a peer-to-peer networking environment. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (May 2001), pp. 65–70.

[20] FOX, A., GRIBBLE, S. D., CHAWATHE, Y., BREWER, E. A., AND GAUTHIER, P. Cluster-based scalable network services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (1997).

[21] FREEDMAN, M. J., FREUDENTHAL, E., AND MAZIÈRES, D. Democratizing content publication with Coral. In *Proceedings of the 1st NSDI* (Mar. 2004).

[22] FREEDMAN, M. J., LAKSHMINARAYANAN, K., AND MAZIÈRES, D. OASIS: Anycast for any service. In *Proceedings of the 3rd NSDI* (May 2006).

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

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

[25] GRIBBLE, S. D., BREWER, E. A., HELLERSTEIN, J. M., AND CULLER, D. Scalable, distributed data structures for Internet service construction. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000)* (Oct. 2000).

[26] 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).

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

[28] KANNAN, J., YANG, B., SHENKER, S., SHARMA, P., BANERJEE, S., BASU, S., AND LEE, S. J. SmartSeer: Using a DHT to process continuous queries over peer-to-peer networks. In *Proceedings of the 2006 IEEE INFOCOM* (Apr. 2006).

[29] KROHN, M. Building secure high-performance web services with OKWS. In *Proceedings of the 2004 Usenix Technical Conference* (June 2004).

[30] 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`.

[31] 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).

[32] 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.

[33] 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).

[34] MAZIÈRES, D. A toolkit for user-level file systems. In *Proceedings of the 2001 Usenix Technical Conference* (June 2001).

[35] MERKLE, R. C. A digital signature based on a conventional encryption function. In *CRYPTO '87: Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology* (1988), pp. 369–378.

[36] PlanetLab: An open platform for developing, deploying and accessing planetary-scale services. `http://www.planet-lab.org`.

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

[38] RHEA, S., GODFREY, B., KARP, B., KUBIATOWICZ, J., RATNASAMY, S., SHENKER, S., STOICA, I., AND YU, H. OpenDHT: A public DHT service and its uses. In *Proceedings of the 2005 SIGCOMM* (Aug. 2005).

[39] ROSENTHAL, D. S. H., AND REICH, V. Permanent web publishing. In *Proceedings of the 2000 USENIX Technical Conference, Freenix Track* (June 2000).

[40] 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).

[41] 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).

[42] SIT, E., DABEK, F., AND ROBERTSON, J. UsenetDHT: A low overhead Usenet server. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems* (Feb. 2004).

[43] SMITH, M. Dspace for e-print archives. *High Energy Physics Libraries Webzine*, 9 (Mar. 2004). `http://dspace.org`.

[44] STOICA, I., ADKINS, D., ZHUANG, S., SHENKER, S., AND SURANA, S. Internet indirection infrastructure. In *ACM SIGCOMM* (Aug. 2002).

[45] STRIBLING, J., COUNCILL, I. G., LI, J., KAASHOEK, M. F., KARGER, D. R., MORRIS, R., AND SHENKER, S. OverCite: A cooperative digital research library. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems* (Feb. 2005).

[46] 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).

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

[48] WANG, L., PARK, K., PANG, R., PAI, V. S., AND PETERSON, L. Reliability and security in the codeen content distribution network. In *Proceedings of the USENIX 2004 Annual Technical Conference* (June 2004).

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