Distributed Indexing: A Scalable Mechanism for Distributed Information Retrieval

Peter B. Danzig, Jongsuk Ahn, John Noll, Katia Obraczka
Computer Science Department
University of Southern California
Los Angeles, California 90089-0782
danzig@usc.edu

Abstract

Despite blossoming computer network bandwidths and the emergence of hypertext and CD-ROM databases, little progress has been made towards uniting the world's library-style bibliographic databases. While a few advanced distributed retrieval systems can broadcast a query to hundreds of participating databases, experience shows that local users almost always clog library retrieval systems. Hence broadcast remote queries will clog nearly every system. The premise of this work is that broadcast-based systems do not scale to world-wide systems. This project describes an indexing scheme that will permit thorough yet efficient searches of millions of retrieval systems. Our architecture will work with an arbitrary number of indexing companies and information providers, and, in the market place, could provide economic incentive for cooperation between database and indexing services. We call our scheme distributed indexing, and believe it will help researchers disseminate and locate both published and prepublication material.

We are building and plan to distribute a research prototype for the Internet that demonstrates these ideas. Our prototype will index technical reports and public domain software from dozens of computer science departments around the country.

Keywords: Information retrieval, heterogeneous databases, resource location, bibliographic databases.

1 Introduction and Motivation

In several years, all books, journals, articles, technical reports, images of art work, and recordings of music will be available from thousands, perhaps millions of data retrieval companies and publishers[9]. Imagine that you, a scholar, need to find an obscure article that exists in a handful of a million databases throughout the world, but you have no idea where to start. One solution would have you submit your query to every database. This process could be automated quite simply with a computer network feature called broadcast[6]. While broadcasting your query assures that you find your article, it will not do it efficiently. The impact of millions of other people also broadcasting queries would quickly overload all the world's databases, and no one's queries would be answered. However, if the task of distributing a query to a set of appropriate databases were not automated, then the advantages of having millions of databases would diminish, because using them would be slow and cumbersome. What is needed is an automated system that only forwards a query to a set of databases that are likely to have the article that you seek. We report here an architecture which we call distributed indexing that could unite millions of autonomous, heterogeneous retrieval systems.

We say autonomous and heterogeneous because, despite cooperating together to form a single distributed retrieval system, they are neither managed nor owned by any individual agency. They cooperate to increase their individual exposure, and thereby increase the likelihood that people use (and pay for) their services. The paragraphs below motivate distributed indexing and contrast it to the current state of the art[13].

1.1 Relationship to Existing Systems

The current largest heterogeneous retrieval system connects 850 databases. This system can broadcast queries to all 850 participants, where a stub at each participant translates the query and response for local use[13]. If you search for all works whose titles contain the words "Information Retrieval", then the system sends such a query to all 850 databases, and you receive receive 850 responses. If you then ask for all works that contain title words "Database Retrieval Systems", the system broadcasts a similar query to the 850 databases, and again, you receive 850 replies. The creators of this database have organized the 850 member databases by topic, and users can reduce the number of databases contacted by limiting their queries to particular topics.

The database research community, until this year, addressed the issues of heterogeneous databases[16] by focusing on translating schemas and concurrency control schemes, rather than finding or organizing information. Several months ago, NSF identified these latter problems as important[7].

The Wide Area Information Server (WAIS), an architecture for heterogeneous retrieval systems is gaining momentum[11]. WAIS maintains a directory of databases, which is used to identify databases relevant to a user's query. Clients and servers employ the Z39-50 protocol [10]. Clients cache previous queries and their results for future reference. Cache consistency is maintained by periodically re-executing queries or by cache invalidation call-backs from the servers. We believe this approach to cache consistency will cause considerable overhead to the servers and communication network.

We are investigating how autonomous databases can cooperate to make an efficient, distributed retrieval system. We emphasize efficiency because the average query to existing, nondistributed retrieval systems already returns hundreds of answers on average, frequently returns thousands of answers, and occasionally tens of thousands [14]. The state of the art in distributed information retrieval systems is intelligent "front end" interfaces to non-cooperating, "back end" databases. Current research addresses this problem with user interface tools that help naive users refine their queries before unleashing them. We believe this refinement should, instead, be done by sophisticated users. For example, only a sophisticated researcher would know that physicists, biologists, and computer scientists all work on vision. We believe we can make sophisticated users out of naive users without employing artificial intelligence. We do this by making it easy to create autonomous databases that specialize in particular topics and types of queries. Companies and user communities can create databases that sift through all the world's documents, collect material relevant to a particular topic, and process queries submitted to them from around the world. We say such databases are *precomputed* because they contain the results of executing particular queries on thousands of other databases. Precomputation avoids the need for users to broadcast their queries to hundreds or thousands of databases. It eliminates work, reduces query response time, and helps locate obscure information.

Our research does not address issues of access controls, storage space, and copyright. Distributed indexing is a mechanism for efficiently *discovering* interesting objects. While it can be extended with access controls and the participating databases can be sensitive to copyright, neither this paper nor our prototype makes contributions regarding copyright enforcement, access control, or royalties. After all, these problems need only be dealt with once an object is discovered. Information suppliers that sell access to their database may still want to have their objects indexed, since the more users that discover and subsequently retrieve them, the higher their revenues.

1.2 Precomputed Queries

A simple example illustrates what we mean by precomputed indices. Suppose a company created a database of the title words of every work in each of 850 databases. Because the new database contains neither the text for the works nor even an entire citation record, the new database would require only a small fraction of the combined storage of the original 850 databases. It need only contain the title and the name of the database that has the complete record. Assuming the title word database were kept current, it would be much more efficient to forward title word queries to the title word database than to all 850 databases. The system would need only contact the subset of the 850 databases that stored appropriate articles.

The idea behind distributed indexing is to establish databases that summarize the holdings on particular topics of other databases. We call these special databases *brokers*. We describe the particular topic by a *generator query*. Generator queries are expressed in our Common Query Language, the base query language employed by users. The rest of this paper describes a system that can exploit the advantages of precomputation on a world-wide scale.

This paper is organized as follows. Section 2 describes the architecture of distributed indexing as seen by programmers and creators of indices, and Section 3 describes the architecture as viewed by users. Section 4

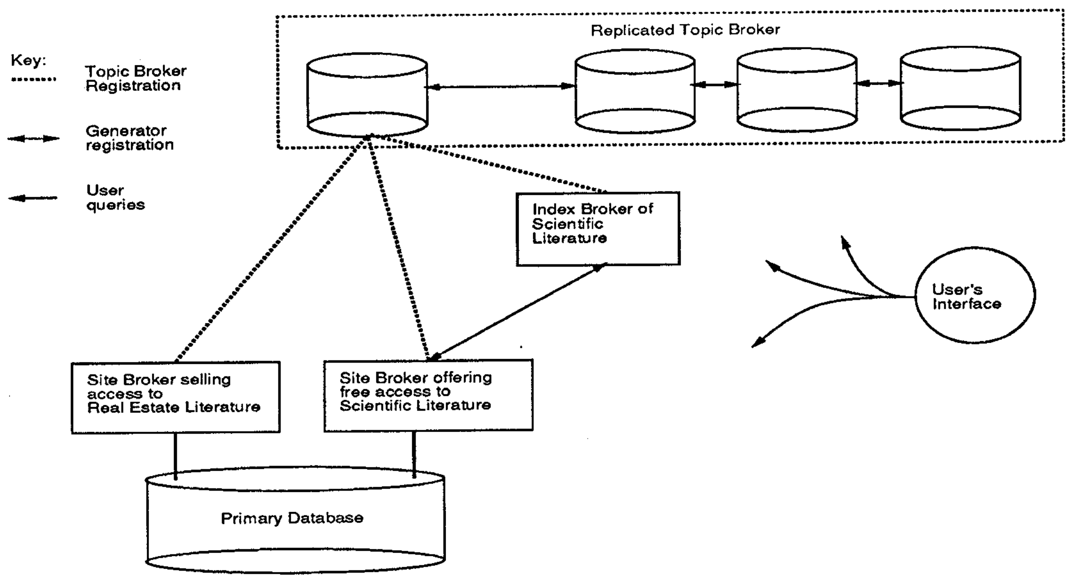
Key:
Topic Broker Registration
Generator registration
User queries
Replicated Topic Broker
Index Broker of Scientific Literature
User's Interface
Site Broker selling access to Real Estate Literature
Site Broker offering free access to Scientific Literature
Primary Database

Figure 1: Components of a distributed indexing system.

focuses attention on how to unambiguously specify the topic of an index, and describes a special indexing structure we use. Section 5 addresses how we obtain update consistency. We draw conclusions in Section 6 and describe the prototype retrieval system that our research group has designed and prototyped, and is planning to distribute.

2 The Architecture

Our system consists of several components, illustrated in Figure 1. An *index broker* is a database that builds indices. The title word database from the example above is an index broker. Index brokers can index the contents of *primary databases* and, in fact, other index brokers. Primary databases are today's single-site retrieval systems such as a library catalogues, indexing services and CD/ROM databases. Each primary database and index broker operates in concert with one or more *site brokers* that perform several functions. Site brokers store the *generator queries* of all index brokers that index their associated database, and are responsible for keeping these index brokers' indices current. Site brokers also translate the system's query language for execution on the primary database, translate results from the primary database to the system's query language[21], and implement access controls.

When a site broker accepts a new generator, it translates and executes it on its associated database, and reliably sends the results to the index broker that registered the generator. A database sends all new and deleted records to its associated site brokers. Site brokers apply their generator queries against these updates and reliably forward appropriate changes to the index brokers that originally registered the generators. Because we anticipate that popular site brokers may register millions of generators, efficient storage and index structures for evaluating them are essential. We discuss these structures in Section 4.

Notice that index brokers are not caches. Rather they are a consistent view of the databases that they index. The architecture's complexity is due to this consistency. This contrasts with Kahn's Knowbots or WAIS's caches which must be re-executed or invalidated to remain consistent [15].

Index brokers index other databases, performing a function similar to telephone yellow pages. An index broker registers generator queries with site brokers of primary databases and other index brokers. These site brokers forward updates to the index broker as their associated databases change. The index broker stores these updates and passes them to the index broker's associated site brokers.

Index brokers, like telephone yellow pages, do not compete with primary databases; they simply make the primary databases more visible (and profitable). The primary database is not obligated to give an index broker a copy of the object, nor is it prevented from doing so. Users of the index broker may have to contact the primary databases to retrieve the object itself. Previous work on yellow page services, known as attribute-based naming, has been restricted to network name servers

[17, 4, 19, 20]. Our architecture extends attribute-based naming to bibliographic databases.

The system's final component, the replicated Topic Broker, describes every site and index broker [12]. Users employ an instance of the Topic Broker to identify brokers relevant to their queries. Creators of brokers use it to find relevant brokers and primary databases to index. Topic broker replication is implemented with a flooding algorithm.

2.1 Creating Index Brokers

How is a new index broker created? Creating a new index broker entails describing its contents and selecting a set of primary databases and index brokers to index. The broker's generator, the query that it registers at the site brokers of the databases that it indexes, defines an index broker. A new index broker announces its presence by recording its generator and an abstract of its contents with the replicated *Topic Broker*, the one logically centralized component of the system (site brokers of primary databases also register a generator and an abstract with the topic broker).

The creator of a new index broker queries the Topic Broker for a list of site brokers whose descriptions pertain to the new index broker's generator and abstract. The creator then attempts to register its generator at each of the appropriate site brokers, and with the Topic Broker. Henceforth it collects and stores indexing information from these site brokers. The instance of the Topic Broker that first registered an index broker informs the index broker when a pertinent new database or index broker is created, or when changes in the description of an existing database make it gain or lose pertinence. This permits index brokers to choose whether or not to register their generator at the new site broker. We are developing interactive tools to automate broker creation, especially the process of writing generator queries.

3 User-System Interaction

So far, we have examined how information is organized in the distributed indexing architecture. In this section, we examine how users can retrieve this information.

3.1 Specification of Queries

We use a common query language to express generators and user queries (note that user interface tools, or *user agents*, can hide the common query language from users). The common query language permits Boolean expressions over traditional bibliographic attributes such as author name, title, publication date, journal name, and publication type; and non-traditional attributes such as geographic location and search cost. It also permits Boolean expressions over ranges of Library of Congress numbers. Range queries efficiently restrict a query's domain of interest, and restrict it less ambiguously than traditional subject keywords. We discuss how ranges are stored and employed in Section 4.

People in specific disciplines want to specify searches in that discipline's preferred classification system[7]. Distributed indexing does not prevent this, and permits constructing domain specific user interfaces and broker creation tools. These tools can map the domain's preferred classification system to the Library of Congress standard. Objects described in a discipline's terminology are easily found by keyword searches on brokers that index the appropriate range of Library of Congress numbers.

3.2 Processing Queries

Executing a query invokes a sequence of steps that eventually identifies or retrieves a set of relevant objects. These steps are query specification, query translation, broker location, broker querying, primary database location, primary database querying, and object retrieval.

The process begins when a user issues a query. While we are not seriously addressing how user's specify the query, possibilities include hand-typed common query language expressions, form or template queries, and even natural language queries.

Once specified, the user agent translates the query into the common query language. For hand-typed common query language expressions, no actual translation is required. The more sophisticated the user agent, the more elaborate the transformation.

Take, for example, the following translated query:

```
Topic:     QA76-QA77 and TK5105-TK5106
Attributes:
Date >= 1989;
Type =  technical report;
Keywords: {computer networks, queueing theory,
not(neural networks, artificial intelligence),
giga-bit, flow control}
```

This query is sent to an instance of the Topic Broker,

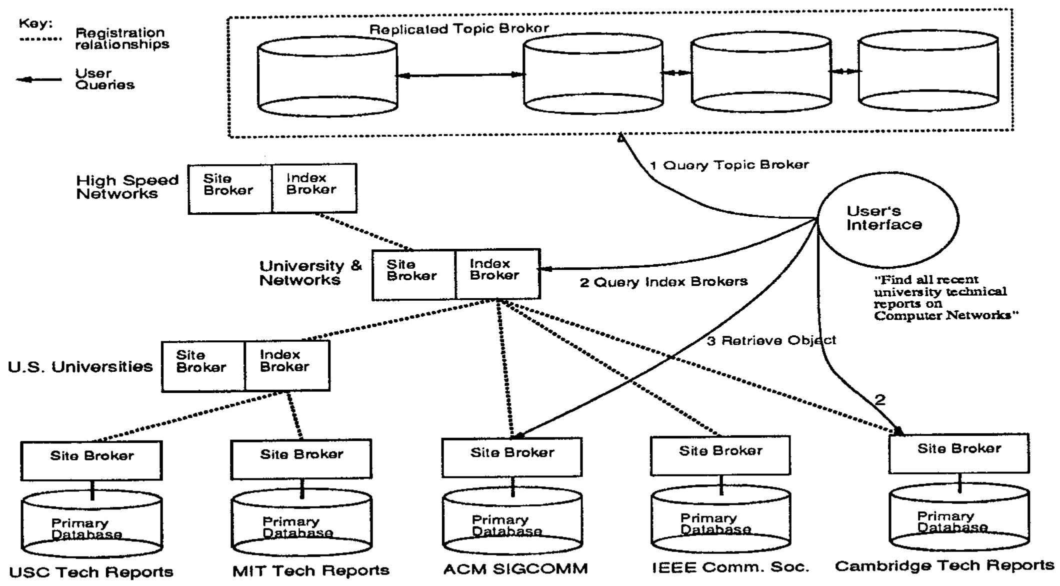

Figure 2: Interaction between system components.

which computes and returns to the user agent a list of possible target index brokers that may be able to satisfy the query. For our example query, this list could contain an entry for the ACM SIGCOMM broker, the IEEE Communication Society Broker, a university technical report index broker on Computer Networks, and several foreign university site brokers (See Figure 2).

The user agent ranks the list of target brokers according to promise, and sends the query to one or more of them. These execute the query and return a list of object identifiers and attributes.

In our example, the user agent might send the query to the *University & Networks* index broker and the site broker at Cambridge University, although it turns out that the first broker happens to already index the Cambridge broker. Both brokers return a list of object identifiers and attributes. The user examines these, and may chose to retrieve a copy of an interesting report from an MIT full text retrieval system. His user agent contacts the appropriate site broker to retrieve the object for browsing.

3.3 Hypertext and Brokers

Hypertext systems provide an interface to information that emphasizes browsing through linked chunks of data. Hypertext can benefit from distributed indexing in three ways: as a method to identify starting points for browsing, as a way to locate interesting information to place in a hypertext, and as a mechanism to provide "dynamic links".

Consider an existing large scale hypertext containing hundreds of millions of nodes stored in the primary databases of our architecture. How can a user begin to browse through such an enormous information space? Clearly, it is impractical to visit every object in the hope of finding something interesting. However, a query to index brokers might identify a reasonable subset of objects for browsing. By specifying the general topic of interest in the form of a query, the user can ask the brokers for nodes that are worth examining[8].

Brokers can also aid in constructing a hypertext by finding objects to serve as nodes. A query yields a set of interesting items; these may be merged with other sets to form a flat space of objects. Then, links are added between objects to compose the hypertext structure.

Finally, a query to the distributed indexing system can be used to implement dynamic links: links that are constructed on demand. For example, a user viewing a node might click on a link anchor. Rather than following an existing link, the system sends a query to the distributed index. The result is displayed as the endpoint of the link, either taking the first object returned, or presenting a linked list of objects. In this way, a link endpoint can change over time as new objects are added to the index space.

Thus, there is a complementary relationship between hypertext and brokers: brokers add associative access to the browsing mechanism of hypertext; hypertext adds organization to objects so they can be easily retrieved in the future.

3.4 Implementation Issues

We hope that, ultimately, a multitude of user agents, brokers, and primary databases will cooperate to form global distributed indexing systems. These brokers and databases will be heterogeneous, and, by necessity, will incorporate existing databases. Network protocol standards will avoid heterogeneity problems between broker components and user agents. However, indexing an existing primary database requires building a new site broker that knows how to communicate with it. Besides this, site brokers have another primary database related task. Suppose a user indentifies an interesting object and wants to retrieve it from its primary database. The site broker must retrieve the object from the database and transfer it to the user. We do this to hide the primary database's access protocol from the user. Brokers and site brokers will employ a common communication protocol[1].

The preceding discussion presents a rather involved series of interactions with the system in order to process a query. We should emphasize that these need not be explicitly managed by the human searcher; rather, we imagine that user agents will do most of the work, asking for user input at important decision points.

For example, a user agent might take the initial query specification, translate it into the common query language, then perform the necessary communications with brokers and primary databases to obtain a set of relevant objects for presentation to the user. In the process, it may do weighting and ordering of brokers and objects according to rules specified in some user or system specific configuration file.

After a set of objects is found, the user agent may evaluate their relevance to the search at hand by processing user feedback while browsing. This information could be used to modify the weighting criteria used to order lists of brokers and objects.

This finishes our discussion on the system's architecture. The remainder of this paper discusses implementation issues. The next two sections discuss indexing data structures and issues in consistency, concurrency control, and recovery.

4 Indexing Structures

Site and topic brokers store generator queries, while index brokers store object descriptors (and possibly may cache the objects themselves). This section describes the indexing structures that brokers use. Because topic and site brokers may store millions of generator queries, generator queries must be indexed to optimize system response time, and the index must efficiently use disk space. Evaluating the query language's keyword Boolean expressions requires an attributed, inverted index, a structure we use but do not describe here. However, inverted indices do not support range queries over Library of Congress numbers. We support Boolean expressions over Library of Congress ranges (e.g. QA76-QA77 & QA244-QA248) as an attempt to confront the query vocabulary problem. This section describes an indexing data structure we use to support such range queries.

When a primary database or index broker passes a new object to its site brokers, the site brokers must determine which generators need to be evaluated. Likewise, when a user agent queries the Topic Broker for a list of target brokers, the Topic Broker must determine which of the abstracts and generators to evaluate. These operations require collecting pointers to generators from the keyword inverted index and from the range query index, and efficiently evaluating the Boolean operations on the indices.

We must identify the list of generators whose Library of Congress ranges intersect with the Library of Congress numbers of the new items. In essence, this is a set membership problem. Similar range problems exist in VLSI design and computational geometry[1]; we have chosen an index structure called a *segment tree* from the latter.

4.1 Segment Trees

In its purest form, a segment tree is a balanced binary tree that represents a set of overlapping line segments on some interval, say zero to one. The root node covers the entire interval and the leaf nodes cover fixed sized segments of the interval. A node's left and right children cover the first and second half of the parent's interval. The tree's height is determined by the segment size of the leaf nodes, which is set at design time.

How do we use this structure to record line segments? A line segment is inserted at the highest level node or nodes in the tree such that the segment subsumes the subinterval covered by the node, but not that of the node's parent. This means that a line segment may appear in many nodes. Take, for example, the segment tree in Figure 3. The tree's root covers the interval $0-1$, and its leaves break this interval into 0.125 length segments. We would insert the interval $s_1 = 0.125 - 1.0$ into nodes i, e, and c because i covers $0.125 - 0.25$, e

[1]It is possible that this protocol will be the Z39.50 Information Retrieval Protocol, but this decision has yet to be made.

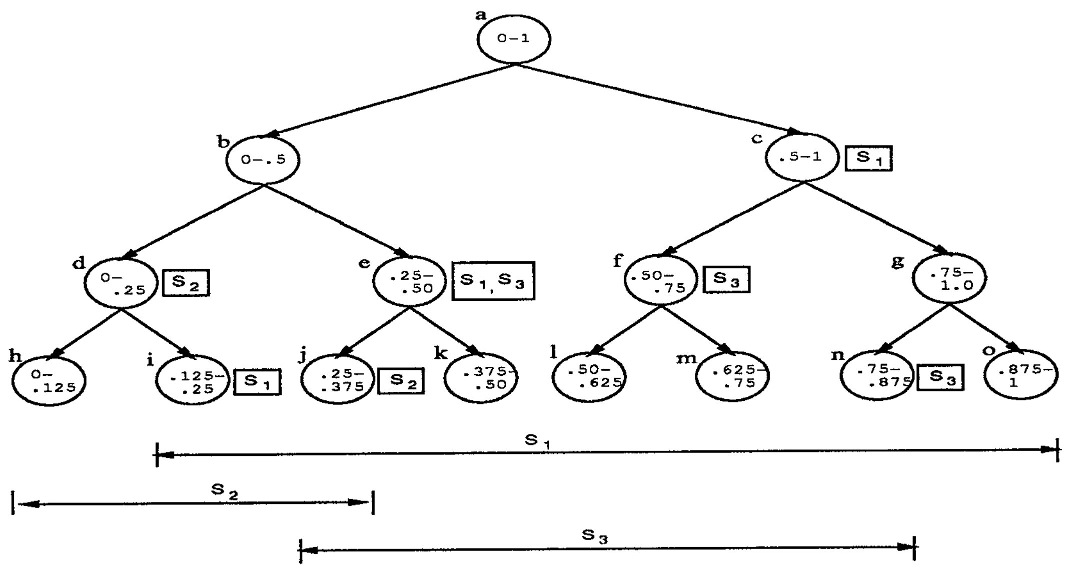

Figure 3: An example segment tree

covers 0.25 – 0.5, and c covers 0.5 – 1.0. Likewise, we would insert interval $s_2 = 0.00 - .30$ into nodes d and j, and interval $s_3 = 0.25 - 0.80$ in nodes e, f, and n.

How do we use this structure to find all segments that contain a given point? One simply collects references to intervals while traversing the tree from the root to the leaf into which the point falls. Take, for example, the point 0.20. Neither the root nor its left child index any segments, however node d refers to segment s_2, and the leaf i refers to segment s_1. Hence, s_2 and s_1 both contain the point 0.20.

Site brokers use our segment tree to identify the set of index brokers whose generator queries should be evaluated when they learn of new items. How do we represent Library of Congress ranges using a segment tree, and how does one find all ranges that contain a given Library of Congress number? We tailor the segment tree structure to fit the relatively static Library of Congress schedules by selecting the subintervals that nodes represent so that they fall on Library of Congress classification boundaries. Another way to look at it is that we're mapping the Library of Congress numbers onto the interval 0 – 1. Based on the current size of the Library of Congress schedules (about 10,000 printed pages), we expect the actual size of the index to be about 10 megabytes. Hence, a substantial portion of the index can reside in main memory. Additionally, we would expect blocks containing segments attached to the upper nodes in the tree to be cached in main memory, since they are shared by many paths.

5 Consistency

Adding an item to a primary database or index broker may require updating thousands of brokers, which in turn can cascade additional updates to thousands more. Updates introduce two sources of problems. First, unless we restrict the system's topology, we must take measures to prevent an endless cycle of updates. This is because an index broker may register with other index brokers as well as primary databases. We could solve this problem by requiring that site brokers reject registration attempts when they would cause cycles. However, this requires that each site broker have knowledge of the complete topology of the system at all times, an expensive assumption. Furthermore, restricting the topology to acyclic graphs reduces the probability of indexing relevant objects, and may expose proprietary information.

We take an alternate approach based on flooding. We require that each update message include the object identifier of the object that caused the update. Upon receiving an update, an index broker examines its database for the presence of the object identifier.

In the case of an addition, if the identifier is not present in the broker, then the broker has not yet seen the update. It processes the update and forwards it to other registered brokers as appropriate. Conversely, if the object identifier is already present, then the broker has already received the update via another path. It therefore ignores the update, thus breaking the cycle.

Deletion is quite similar. If the object identifier is present in the database, then the update has not been seen. The broker processes and forwards the deletion. If, however, the object identifier is not present, then the update has already been processed, and is thus ignored.

We have to maintain consistency if index brokers or primary databases disappear or generators are unregistered. When the topology is acyclic, it is easy to maintain consistency. Permitting cycles, however, requires that certain path information be exchanged in addition to object identifiers. The deletion algorithm may need further work.

Also, because it is unlikely that several thousand databases are simultaneously functional, we cannot update several index brokers simultaneously in a single atomic action. We investigate these problems below.

5.1 Atomic Actions

When a database adds a new object, it is the responsibility of the database's site brokers to identify index brokers that may be interested in the new object. This is done by comparing the new object against the list of generator queries of all index brokers who have registered at the site. It is entirely possible that the result may include thousands of index brokers; it is also unlikely that all of these brokers will be functional at any given instant of time. Thus, it is unreasonable to assume that notification of the new object can take place as an atomic transaction.

To solve this problem, we require that new objects be timestamped with their creation time, and add a timestamp to each index broker's registered generator query. The latter timestamp indicates when the last update was successfully reported to the index broker. Site brokers repeatedly attempt to contact each index broker and report the new object. When contact is established, the generator timestamp is set to the timestamp of the new object. The process stops when all generator timestamps are equal to the timestamp of the last object created.

Deletions are handled in the same way. Each broker that had expressed interest in the deleted object is notified in turn of the object's deletion. All brokers have been notified when the timestamp of every generator query is the same as the deletion timestamp of the deleted object.

5.2 Media Recovery

We assume that primary databases perform their own media recovery and that index brokers can recover their list of generators. The index broker's and site broker's timestamps can be used to recover either the index or site broker after a disk crash[2]. For this reason, index brokers need not recover the indices they store. These are automatically recovered by resetting the generator's timestamp at the site broker. To ensure that site brokers can recover an index broker's generator query, each index broker regularly polls all site brokers where it has registered a generator. If a site broker suffers a media failure, its generators are recovered as each index broker polls it and reregisters its generator and generator timestamp.

6 Conclusions

Building a retrieval system to incorporate ten million databases has never been attempted. The largest existing distributed database is the DNS name server which has very limited functionality and only encompasses less than twenty thousand sites [18, 3]. We believe that distributed indexing presents a new perspective on uniting autonomous retrieval systems.

6.1 Prototype

We are implementing the system sketched above to distribute to the Internet community. The prototype will run on Sun UNIX, which should make it easily ported to other UNIX systems. At this time, we have a prototype running in our lab. We expect to complete the prototype by the time this paper is presented.

With the cooperation of ten to thirty other computer science and electrical engineering departments, we will create a national collection of computer science technical report index brokers covering the various subdisciplines of our field. Each department will create primary databases of their technical reports, and make a site broker available over the Internet. As it is currently difficult to obtain technical report lists, we believe our distributed indexing project could easily become a permanent part of the research Internet community. This prototype, albeit small, will demonstrate the system's functionality. We also plan to interface with DNS databases.

6.2 Future Work

We plan to evaluate how these ideas scale to retrieval systems beyond this prototype. Scalability depends on statistical properties of the data, queries, and broker topology. We intend to load CD/ROM databases onto the disks of various workstations, and build experimental index brokers. We will investigate the index broker size as the number of indexed databases grows, and will develop better tools for writing generator queries so that they comprehensively yet selectively retrieve what we expect. We are also working on extensions of our ideas to object oriented databases[5]. Below we consider questions of scalability and expressiveness.

Generator queries must describe subjects precisely if brokers are to precompute useful views of millions of retrieval systems. However, topics ordinarily span many ranges. Someone searching "Computer Networking" expects to see articles on Queueing Theory and Network Technology as well. For this reason, we have included ranges of Library of Congress numbers in our queries. We do not expect users themselves will specify ranges, but, as mentioned earlier, domain-specific user interfaces can append a Library of Congress range to a user's keyword query. Broker creation tools help broker creators specify Library of Congress ranges that appropriately cover the creator's intended subject matter.

Acknowledgements

We would like to thank Carl Braganza for several useful discussions and ShihHao Li for his work on the prototype. We would also like to thank LBT for reading earlier drafts of this paper.

References

[1] Jon Louis Bentley and Derick Wood. An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers*, 29(7):571–577, July, 1980.

[2] Philip Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[3] Andrew Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[4] Mic Bowman, Larry Peterson, and Andrey Yeatts. Univers: An attribute-based name server. *Software Practice and Experience*, 1989.

[5] Peter Danzig, Michael Arbib, and Shahram Ghandeharizadeh. Brane: Brain analysis environment. Technical Report 91-14, University of Southern California, May 15, 1991.

[6] Steven Deering. Multicast routing in internetworks and extended LANs. *1988 ACM SIGCOMM Symposium*, pages 55–64, August 16-19, 1988.

[7] James C French, Anita K Jones, and John L Pfaltz. Summary of the final report of the NSF workshop on scientific database management. *SIGMOD Record*, 19(4):32–40, Dec, 1990.

[8] Mark E. Frisse. Searching for information in a hypertext medical handbook. In *Hypertext '87 Proceedings, Chapel Hill, North Carolina, November, 1987*, pages 57–66, New York, 1987. The Association for Computing Machinery.

[9] William Gardner. The electronic archive: Scientific publishing for the 1990s. *Psychological Science*, To Appear.

[10] ISO. Information retrieval service definition and protocol specifications for library applications. Technical report, National Information Standards Organization, 1989.

[11] Brewster Kahle. Wide area information server concepts. Alpha Release Documentation, anonymous FTP from think.com:/public/wais/wais-8-aXX.tar.Z.

[12] Butler Lampson. Designing a global name service. *ACM Principles of Distributed Computing*, August 1986.

[13] Witold Litwin, Leo Mark, and Nick Roussopoulos. Interoperability of multiple autonomous databases. *1990 ACM Computing Surveys*, 22(3):267–293, September, 1990.

[14] Clifford Lynch. Melvyl show statistics command. Personal Communication, February 21, 1991.

[15] John Markoff and Robert Kahn. Creating a giant computer highway. *New York Times*, Sept 2, 1990.

[16] Dennis McLeod. An approach to controlled sharing among autonomous, heterogeneous database systems. *IEEE Database Engineering*, pages 17–41, 1987.

[17] B. Clifford Neuman. The virtual system model: A scalable approach to organizing large systems: A thesis proposal. Technical Report TR-90-05-01, University of Washington, Seattle, May 1990.

[18] Marshall T. Rose. *NYSERNet White Pages Pilot Project: Administrator's Guide.* NYSERNet, March 26, 1990.

[19] M. F. Schwartz. A scalable, non-hierarchical resource discovery mechanism based on probabilistic protocols. Technical Report Technical Report CU-CS-474-90, Department of Computer Science, University of Colorado, Boulder, Colorado, June 1990.

[20] M. F. Schwartz, D. R. Hardy, W. K. Heinzman, and G. Hirschowitz. Supporting resource discovery among public internet archives using a spectrum of information quality. Technical Report Technical Report CU-CS-487-90, Department of Computer Science, University of Colorado, Boulder, Colorado, September 1990.

[21] Michael F. Schwartz, John Zahorjan, and David Notkin. A name service for evolving heterogeneous systems. *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, 21(5):52–62, November 1987.