Managing Metadata for Distributed Information Servers

Nigel Hinds and Chinya V. Ravishankar
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, Michigan 48109–2122
(313) 763-5243; E-mail address: nigel@eecs.umich.edu
(313) 647-1806; E-mail address: ravi@eecs.umich.edu

Abstract

In this paper we present the design and implementation of a distributed index architecture to facilitate metadata discovery on large networks. Many current information discovery systems use crawlers to search the Internet, constructing metadata from the data they find. Instead, we propose a general distributed index architecture that uses descriptive hierarchies to organize metadata and route metadata queries to sites likely to produce relevant results.

We are testing the architecture in the NASA Earth Observing System domain. We address some requirements particular to EOS, then discuss the design implications. We also present the results of preliminary performance studies on a prototype.

1 Introduction

The amount of available Web information and the access frequencies have grown rapidly, but methods to locate and access desired data remain rudimentary. Information consumers must sift inefficiently through large amounts of data. Users are increasingly interested in accessing structured data sets such as phone books which contain names, addresses and phone numbers, or product catalogs which typically contain product names, features and prices. This pattern resembles distributed database access more than traditional file or document system access. The problem of finding data of interest, from a large set of distributed sites, has been called *information discovery*.

An effective approach to facilitating information discovery is to provide and manage metadata (i.e., data that describes the original data). Recent work in naming and information systems has focused on constructing metadata to reduce the time required to find information in the Internet. Most of these approaches use centralized indices, and function by using crawlers or robots to search the Internet, constructing metadata from the data they find. Archie [1] was one of the first central indexing systems. It indexed the names of File Transfer Protocol (FTP) files located on a subset of popular Internet hosts. More recent examples are the Web search engines, which use crawlers that extract text from Web pages.

Our work is motivated by four observations. First, systems using centralized indices require considerable investment in disk space and CPU resources, without which they will not scale as the amount of metadata increases. Slow response times observed in early versions of Archie and Lycos [2] illustrate over-utilization difficulties with the centralized model. Only after additional investment in resources could reasonable performance be attained.

Second, these systems function independently, resulting in duplication of metadata and inefficient use of network resources. Because these systems are attempting to index diverse data, they use simple indexing techniques, such as text string filtering. We argue that data creators are in a better position to create metadata using domain-specific techniques that might include additional metadata not found in the original data.

Third, the widely-used metadata systems index files or Web pages, so they do not include metadata stored in databases. This class of metadata will get larger as the growth in Web-based commerce leads to more online product catalogs. Finally, many institutions have developed hierarchical structures for describing and organizing their data. We argue that these descriptive hierarchies can be used to organize distributed metadata. The Dewey Decimal System [3] and the MARC standard [4] are good examples of existing descriptive hierarchies. They are used to hierarchically organize and integrate the holdings of many diverse libraries. The Database of Occupational Titles (DOT) [5, 6] and the Earth Observing System (EOS) Global Change Master Directory parameters [7] are two other descriptive hierarchies.

Field name	Description
Granule ID	system specific granule identifier
Sensor	instrument that produced data
DAAC	name of DAAC archiving this granule
Data set	name of dataset with granule is associated
Platform	platform containing the instrument
Starting date-time	temporal cover of data
Ending date-time	temporal cover of data
Location	bounding polygon
Geophysical Parameters	one or more controlled keywords describing the data

Table 1: Subset of EOSDIS Metadata

The NASA EOS Distributed Information System (EOSDIS) is a distributed catalog of collected data, similar to a growing number of Internet catalog applications. We will use the NASA EOSDIS environment to validate our solution. In the next section we provide an overview of the EOSDIS requirements and their impact on our architecture.

1.1 Overview of the EOS Environment

The primary data managed in the EOSDIS and other catalog systems is metadata about products. In EOSDIS this is satellite or in-situ data, or in general, the results of scientific experiments. Preliminary estimates are that EOS archive centers will process millions if not billions of granules (metadata records) per day [8] when operational.

In the current version of EOSDIS, a user enters values in an X11-based forms interface that produces an SQL-like query string. The result of an EOSDIS query is a set of metadata entries (called granules) describing data held at Distributed Active Archive Centers (DAACs) throughout the world. There are presently seven DAACs, but it is expected that this number will grow to over 100 [8]. Table 1 shows a subset of EOS metadata fields returned by metadata queries. For some metadata, granule ID can be used to download the actual data from the DAACs. All data can be ordered (for postal delivery) from the DAACs using the granule ID.

EOS metadata are currently stored in heterogeneous relational databases at the DAACs that archive the data. The distributed EOSDIS metadata is presented as a traditional distributed database global aggregate view representing the union of all DAAC metadata tables. No actual metadata is stored in the centralized EOSDIS. Instead, the central site initiates a query at each of the DAACs. The DAACs in turn translate the query into the format and semantics of the local metadatabase.

The problems with the EOSDIS are similar to those of all centralized architectures: without considerable investment in resources centralized systems are likely to suffer overloading as utilization increases, and system failure or network partitions can make the entire system unavailable. A more serious problem specific to EOSDIS is that the global view does not use indexing, so each query results in an exhaustive search of all DAACs.

One simple solution to the load problem associated with centralized systems is to replicate the index. However, maintaining consistency now requires replication/concurrency control algorithms that do not scale well. And, as the universe of metadata records and metadata sites becomes large the index will become unmanageable [9].

Below are general distributed system, as well as EOSDIS requirements:

Availability & Reliability: There should be no central point of failure. If one metadata supplier crashes the system should be able to satisfy requests for unrelated metadata.

EOSDIS need not be 100% available. However, the failure of a DAAC server should result only in the loss of metadata from that DAAC. This choice provides no redundancy, but does provide gradual degradation of service as failures occur. The system must manage the effects of component failure, including network partitions and node shutdowns at link end-points as well as at intermediate nodes. The literature offers a number of possible solutions [10, 11, 12].

Distributed Administration: The EOSDIS solution must respect the administrative boundaries that exist between DAAC organizations. In particular, each DAAC must have final control over all its resources. This means DAACs must be free to decide what metadata, if any, is made available through the system, and limit the resources used on behalf of a client.

Consistency: EOSDIS users will expect 100 percent recall, that is, expect to see all metadata in the system's universe that satisfies a query. Caching metadata results at or near the client is an obvious technique for improving response time, provided there is locality of reference. At this point it is unclear if EOS users will repeatedly search for the same data. What is more certain is that a user will want a *standing order*. A standing order is a query that remains active, so that metadata servers notify (or deliver metadata to) the user whenever new metadata becomes available.

An issue affecting caching is that EOS metadata rarely changes, but new metadata is continually created. This means accuracy of any cached metadata is almost guaranteed. However, relying solely on the cached metadata will result in progressively lower recall.

Scaling: To estimate the query load, we draw on the EOS user modeling results [13]. At 1000 queries per day, this gives us a rough query rate of 41.6 queries per hour, or slightly less than one query per minute.

1.2 Our Approach

We introduce a general distributed index architecture that uses descriptive hierarchies to organize metadata and route metadata queries to sites likely to produce relevant results. Descriptive hierarchies allow queries to be directed at any participating server, eliminating the bottleneck of directing every metadata query to a centralized site for routing. They do not require complete replication of metadata to distribute load away from a single site to mirror sites. We also use index caching to exploit user locality of reference. Similar queries benefit from cached index results. In this work we eliminate continual network crawling for data and metadata construction by expanding the notion of a referral (which describes the metadata holdings of a site). This approach permits more efficient use of network bandwidth. Finally, our system provides access to metadata contained in databases by supporting attribute-value queries and using heterogeneous database techniques to communicate with metadata servers.

Earlier work, such as Nomenclator, demonstrates the effectiveness of metadata distribution and caching in the X.500 naming service. Our work generalizes this approach to information systems. Harvest [14] (using the Inde subsystem [15]) constructs a distributed index shared between *brokers* which summarize the contents of selected metadata archives. Like Harvest, our work deals with resources and information found on the Internet. However, we have added hierarchical indices to considerably reduce the amount of metadata replicated and support the efficient search of distributed metadata.

In the remainder of this paper we present the DSMS architecture and results from experiments on a prototype. Sections 2 describes the DSMS architecture, issues and approaches. Section 3 presents results from experiments measuring distributed index search response times as well as a preliminary response time model. Finally, section 4 discusses conclusions and some future directions.

2 The Domain-Specific Metadata Service Architecture

This section describes an alternative architecture, called the Domain-Specific Metadata Service (DSMS), for providing scalable metadata service. We begin with some important assumptions about the metadata environment.

Federation of Semi-autonomous Organizations: Organizations must be willing to run a server that will accept and run pseudo-SQL queries on the local metadata. This is reasonable, since Web servers are designed to accept attribute-value queries and pass them to a database engine. At this time we assume the metadata servers return records that include unique identifiers that can be used to access the data.

Descriptive Hierarchy: The DSMS architecture assumes that there exists a hierarchy of terms. Terms in the hierarchy may have multiple parents as well as appear more than once. This is not an unreasonable assumption in many domains, particularly in EOS.

Subject field: Terms from the Subject Hierarchy appear in a metadata subject field.

No Data: Actual data is stored and managed outside the proposed system, which deals with metadata only.

2.1 Architecture Overview

There are three major components of the DSMS architecture.

DSMS Server: Stores pointers to metadata and responds to queries from resolvers.

Resolver: This is the primary search engine, and contacts DSMS servers in an attempt to find metadata. The resolver typically resides on the same host as the user agent.

User Agent: A DSMS user agent accepts queries from users (e.g. via HTTP servers). The user agent repackages the query and sends it to the DSMS resolver. The agent then waits for results, which it formats for display and returns to the user.

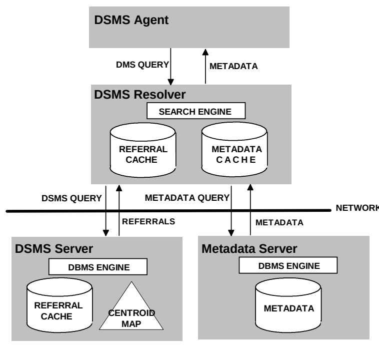

Figure 1: DSMS Architecture

Data archive sites will produce and manage the metadata, and must provide interfaces to access local data. We propose accessing metadata through Web servers (labeled Metadata Server in Figure 1). Presently, the DSMS architecture relies on external metadata production systems. We assume the existence of file-type specific metadata extractors such as the Essence indexer [16].

An additional component is the User Interface veneer. We envision the user interface will be a forms-based Web page. In this environment the agent is started by the user interface page via Common Gateway Interface (CGI). The agent in turn starts the resolver. Ideally, the agent and resolver should also run on the host running the Web browser, to distribute processing and improve performance. However, this is difficult to achieve in the current Web environment, since programs can only be run at the HTTP server site, and not from Web browsers on the user's host. There are a number of solutions to this problem. As the Web matures it is likely that the browsers will change. Alternatively, Java can be used to overcome this limitation. Another solution used by many packages is to run a separate client program that can accept a user query, then start a Web browser to display the results. For simplicity, we will adopt the separate client approach.

Figure 1 exposes the major data structures within the DSMS components. These structures are discussed in the next two sections. Sections 2.5 describes query processing.

2.2 Centroid Hierarchy

A novel feature of our architecture is the use of a descriptive hierarchy to describe entries in a distributed index. In DSMS the descriptive hierarchy is called a centroid hierarchy. Salton [17] defines a *centroid* as a synthetic average for a group of documents. In database modeling terms, the centroid hierarchy is a *generalization* structure [18], which defines an "is-a" relationship. Figure 2 depicts a partial centroid hierarchy using a portion of the EOS Global Change Master Directory (GCMD) parameter hierarchy [7]. Terms at higher levels in the structure are more general than lower level terms. In this example, *Precipitation* is the centroid for the term *Snow*, and *Atmospheric Dynamics* is the centroid for *Precipitation*. So, in our architecture the centroid hierarchy defines a mapping from lower level terms to higher level terms.

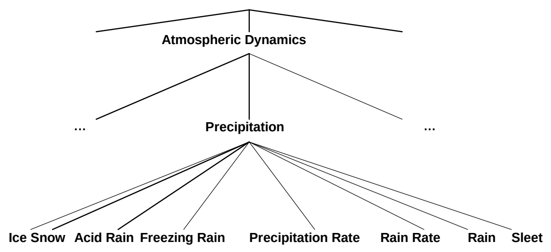

Figure 2: Partial Descriptive Hierarchy (EOS Domain)

At this time the DSMS architecture prescribes only a simple keyword-based centroid hierarchy. We understand without further guidance on centroid hierarchy construction there will be considerable variance in detail, completeness and quality of the centroid hierarchies.

Authority for a centroid hierarchy is delegated by an organization controlling a root DSMS server. A new centroid hierarchy is created by inserting a new high-level description, or *domain*, at the root DSMS. For example, our partial hierarchy is part of the EOS domain, for which an authoritative DSMS server is defined. This is the only server allowed to modify the centroid hierarchy rooted at the new domain. At the root DSMS server there is a list of all domains.

There are three types of domain updates: adding a new subdomain (centroid), changing the name of an existing subdomain, and removing an existing subdomain. In general operations on a domain can be discussed in terms of add and delete. The primary issue is to maintain consistency by ensuring updates propagate to all DSMS servers that replicate a domain.

There are two types of inconsistencies arising from updates. In the first, the domain data at a DSMS server is updated, but centroids in the referrals maintained at the resolver and/or server contain referrals that may conflict with the new domain. The second case is the opposite. New referrals have been created using the updated domain centroids, but a server has not received the domain update. Below we describe each of these anomalies in context of the updates described in the last paragraph.

Adding a new subdomain (centroid)

The authority issues an add operation to the DSMS servers that subscribe to the domain. Each of those servers, in turn, notifies their peers (other DSMS servers).

old domain – new referral: a resolver may retrieve a referral that uses the new centroid and query a DSMS server that has not received the domain update. In this case the centroid search would return an error to the resolver, which would then disregard the referral for the duration of the metadata search. The effect is a possible reduction in recall, as a potential metadata source is skipped.

new domain – old referral: Add is non-destructive, so old referrals are not affected.

Moving an existing subdomain

This operation re-links all subdomains (centroids) from one domain to another domain.

old domain – new referral: a resolver may retrieve a referral that uses a centroid in the new domain and a query that uses the old centroid. In this case the server will not find the new referral. The effect is a possible reduction in recall, as a potential metadata source is skipped.

new domain – old referral: If a DSMS server has received the domain update, then queries using old referrals will fail unless the query uses centroid paths (see centroid paths at the end of this section).

Removing an existing subdomain

This operation removes all subdomains (centroids) from a target subdomain.

old domain – new referral: There will be no new referrals for a deleted subdomain. A server may contain and return old referrals that have not been expunged.

new domain – old referral: If a DSMS server has received the domain update, then it will not use the old referrals unless the query uses centroids paths (see centroid paths at the end of this section).

These consistency problems are a combination of file system consistency and search scope-performance trade-offs. As a result, our basic design for domain consistency draws heavily from solutions in the file systems literature [12, 11]. The choice of consistency model depends on the environment. Three major factors influencing that choice are; how crucial is data accuracy, write propagation costs and frequency. If accuracy is important, locking is one popular method used to implement strong consistency. Locking protocols, however, typically require several rounds of communication with all replicas to accomplish their task [19, 20]. If updates are few and the cost of inaccurate data high, then the cost of locking may be tolerable.

Standards bodies typically work in cycles in which they accumulate and review a number of proposed changes. Hence, we do not expect updates to the centroid hierarchy to be continuous. We expect domains to change more often. If a referral contains a new centroid that has not been inserted into a local DSMS server being queried, the lookup on the centroid will return a "centroid not found" error, causing the referral lookup process to discard that referral. The metadata query will continue, but recall may be reduced as a result of the discarded referral. We feel the reduced recall does not justify the cost of strong consistency, so a weaker model is used to disseminate domain updates.

Multiple inheritance within a domain is also an issue. In our sample domain, in addition to being a type of *Atmospheric Dynamics*, *Precipitation* is also a type of *Hydrologic Property*. This ambiguity is resolved by modifying the query to accommodate a *centroid path*. Since the centroid hierarchy is a tree each node can be uniquely identified by a path composed of nodes/centroids starting at the root. The DSMS server requires an unambiguous centroid path. This can be supplied by the user or by an aliasing mechanism.

2.3 Referral Index/Cache

Many systems use ideas similar to *referrals*. In such systems, the metadata refers to or gives the location of data. Our referrals are like meta-metadata in that they give the location of metadata or other referrals. A DSMS referral contains a centroid and the name of a metadata site that has metadata described by the centroid. Referrals are used to restrict the number of sites searched during a metadata query. Centroids are used in referrals to describe the contents of a metadata archive.

No server is expected to store all the referrals exported on the network. Instead, servers join replication groups that maintain consistent subsets of all referrals. The subset maintained by any replication group is defined by a centroid.

Using referrals has a number of advantages. First, metadata can now be created by organizations more familiar with the data. Also, there is no need for robots to crawl around the network looking for data. Finally, referrals change less frequently than metadata holdings, because they describe the contents of a metadata site in general terms. Referrals stored at the various DSMS servers make-up a distributed index.

The DSMS approach does require that either a metadata site or broker create a referral describing metadata to be exported. While this does impose more work on the data creator we feel the resulting data accessibility out-weighs the additional effort. Furthermore, we are now seeing that Web page creators are willing to use the HTML metadata tag to include descriptive keywords.

The consistency issues for the distributed index are similar to those of the centroid hierarchy. As with the centroid hierarchy, DSMS servers form groups to maintain consistency of the distributed index. However, referrals may also be cached by resolvers not participating in the replication group. Finally, any server can add a referral to the index. Consistency within the replication group is based on a weak model to avoid the cost of locks. Fault recovery for the distributed index will use the same technique as the centroid hierarchy.

Metadata queries are processed in two phases. During the first phase the distributed index is used to collect referrals from DSMS servers. The implicit assumption to this point has been that all referrals are of equal quality. So, during the second phase referrals can be processed in any order. In reality, referrals are not typically equal. Referral quality can be quantified by the number of metadata entries returned or metadata server access speed. For now we choose return size estimate not only because it is the simplest measure, but because it has a direct impact on the query recall. The more results the higher the recall. We would like the system to search the sites with the highest results size estimates first. We intend to use the *GlOSS* technique for ranking metadata sites [21]. *GlOSS* uses keyword *frequency* and *weight* to rank a set of candidate data sites in an order that produces the highest recall from a minimal number of sites.

2.4 Metadata Cache

The metadata is the highest volume data managed by DSMS. Although exact metadata results size depends on the query, EOS metadata results can easily include 1000 records. Under the current assumptions the results will be HTML pages. When the resolver receives new metadata results from metadata servers, it places the pages at the end of a circular buffer. This allows the resolver to work in parallel, querying metadata sites while the agent reads results from the head of the buffer. Subsequent searches on the same query will access the metadata buffer before contacting any DSMS servers.

There is considerable heterogeneity among metadata archives sites. Sites will archive different metadata and as a result have different schemas. Sites are also likely to store the metadata in different database systems. Space does not permit a discussion on heterogeneity in this paper. We refer the reader to the extensive literature on the issue [22, 23, 20, 24, 25]. Since

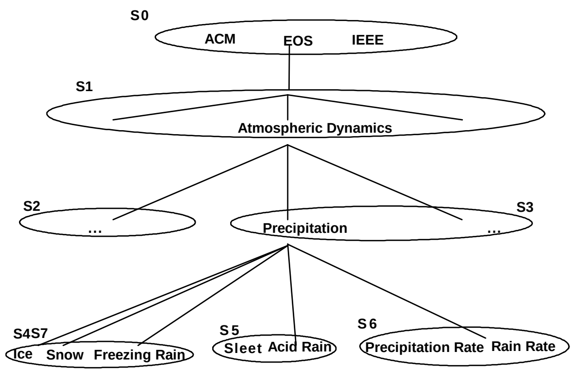

Figure 3: Sample Server Configuration

we are primarily concerned with product metadata that can be stored in tables, this discussion assumes the use of relational databases which can be used to manage tabular data.

2.5 DSMS Operation

This section describes how DSMS servers are connected and how metadata queries are processed.

When a new metadata site comes on-line, the administrators must locate an existing DSMS server that will accept referrals from them. At least one of these bilateral agreements is required for a new site to become part of the DSMS system. It would be most efficient to connect DSMS servers in a hierarchy. For example, the new DSMS server in the Space Sciences department at a university might connect to the DSMS server in the Engineering College of that university. In this scenario, the two servers have become peers. Upon connecting to its peer, the new server chooses a domain to replicate and joins a replication group for a centroid under that domain. This is effectively accomplished by performing a metadata query and registering as a replica with one of the servers that responds with referrals. The new server will also export its referrals to the replication group. Figure 3 illustrates how servers can be used to partition a domain.

S0 is the DSMS root server. It contains referrals for all the domains in the system. S3 contains referrals for precipitation; while S4, S5, S6 and S7 contain metadata for various referrals under precipitation. Given the partial hierarchy in Figure 3, the query

```
SELECT granules
        WHERE centroid = "Snow"
        AND location={lat. 82.50,
                    lon. 90; ...}
```

has the centroid *Snow*, and *Precipitation* is the centroid for *Snow*. For simplicity this version of the architecture requires that the user supply one and only one centroid per query. Future additions to the architecture could resolve multiple centroids by choosing the closest common parent centroid, or combine the results from multiple searches.

Like other recent discovery systems, we divide the data search process into several stages. During the query construction stage, the user specifies the data of interest by entering a list of attribute-value pairs. A resolver using the query centroids searches for referrals to metadata holdings with relevant metadata. Metadata servers referenced in the referrals are queried and the results returned to the user. The user accesses the data via a server specific protocol. We now demonstrate how the centroid hierarchy is used to constrain a metadata query. With the partial hierarchy in Figure 3 and the query given above processing proceeds as follows:

1. User input on the HTML search page is sent to the user agent as an attribute-value string. The agent packages the query string, sends it to its pre-assigned local DSMS resolver and waits for a response.
2. The resolver receives the query and first searches its local metadata cache. If metadata exists, it is returned to the user agent.
3. If there is no cached metadata, the resolver searches the referral cache at its pre-assigned DSMS server. If referrals are found the resolver will send the original query to each of the referral sites. The metadata results of the query are cached and returned to the user agent.
4. If no referrals are found, the resolver determines the parent centroid (C_{i+1}) of the current centroid (C_i) from the centroid map. The resolver searches its centroid cache for referrals for the new centroid. If referrals are found then those sites are sent a query for referrals on centroids C_i; or the original metadata query if C_i is the centroid of the original query.
5. If no referrals are found for centroid C_{i+1} the resolver checks the cache for the next higher centroid. This continues until the resolver reaches the root of the centroid map. At that point the query can fail or access a universal referral to a master broker.

3 Experimental Results

The success of the DSMS architecture depends on the following issues. Within the context of a real application we must, first, demonstrate the advantages of the distributed index. Here, we do not explicitly show the centroid hierarchy is able to constrain metadata queries. The performance advantages of indexing has been well documented in the literature. Instead we want to show the potential improvement of distributed indices over their centralized counterparts. Second, we must measure the effects of referral & metadata caching. Finally, we must demonstrate that the architecture will function on the scale of tens of thousands of servers.

In this section we describe the DSMS implementation and present the results of a study comparing the DSMS distributed index performance with that of a centralized system. The results show the DSMS model offers superior performance as work load increases.

3.1 Environment

A DSMS agent, resolver and server have been implemented for Sun Solaris. Unless otherwise stated the experiments were performed on Sun4 CPUs running Solaris 5.1 with 64MB of memory. Experiments were run during off hours on lightly loaded machines to reduce the affects of competition for processor cycles. The centroid hierarchy domain used is the Global Change Master Directory parameters version 23 [7]. Referral data was synthesized.

A 500 byte referral record was used for all tests. The mSQL Beta 2.0 relational database server has been used to implement the referral cache and centroid hierarchy. The measurements are for referral lookup only and do not include subsequent search times at the metadata sites. Only the query centroid is used in these tests. Queries contain a single centroid, such as clouds, or ozone.

As expected, preliminary results show that up to about one million records indexing will make search response independent of the total database size. We use this result to simplify the experiments by using a fixed referral database record count of 100K.

3.2 DSMS Response time

This study is intended to demonstrate that DSMS provides acceptable response in the EOS operating environment. Results of this study will also be used as input parameters to the modeling studies described later in this section. Since there are no widely accepted benchmarks for comparing discovery system response times, our goal was to provide response times that approach the maximum I/O capabilities of the tested machines.

First, we gathered measurements of raw disk I/O bandwidth. Our test machines were using a SCSI-2

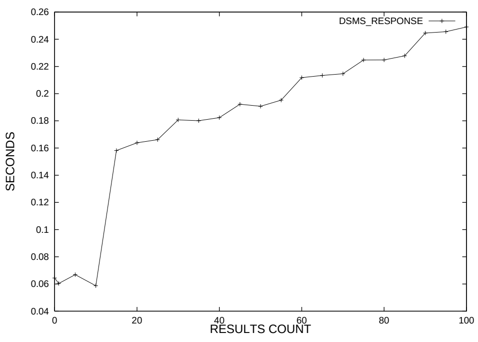

Figure 4: DSMS Server response time as number of results returned increases. 100K referral database.

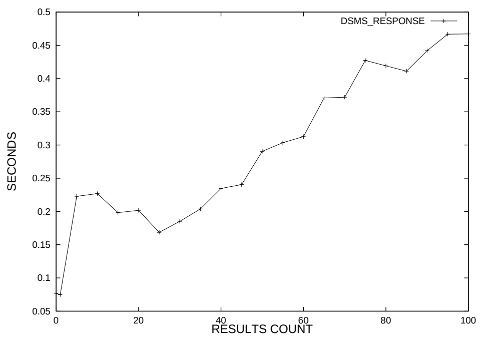

Figure 6: No-load DSMS Response Times: 1000-byte referral records.

channel so maximum I/O is 20 Mbytes per second. Our measurements of raw disk read times are within an order of magnitude of the maximum 20 Mbytes per second. mSQL overhead reduces I/O to approximately 200 Kbytes per second.

These DSMS lookup experiments were designed to measure referral lookup performance at a single DSMS server with no load, as a function of the results size. The main results are shown in Figure 4. Response time in the figure is measured between the DSMS server receiving a referral query and the time that all results are sent. These figures do not include run times for the subsequent metadata server search phase. The jump after *results_count* 10 may be explained by increased context switching due to transmission buffer overflow. This explanation is supported by the fact that actual CPU time used during the search does not display this behavior. DSMS introduces an approximately 150 percent increase in response This overhead primarily represents the time consuming task of moving results data to I/O buffers. We believe this effect can be greatly reduced by optimizing the internals of the engine.

Some of the overhead arises from our use of the mSQL engine, which we have chosen because it is freely available with source code. Better search engines, such as those used by the commercial search systems, would yield better performance. Our goal here is not to pursue higher query engine performance, but rather to show that distributed indices work well.

3.3 Distributed Referral Index Configuration and Model

We have claimed that the distributed referral index will scale as the query load increases. This study provides support for that claim and demonstrates the performance advantage of the distributed referral indices over centralized systems. We proceed by constructing prototypes of the two alternative architectures. The first is a centralized referral database using a single host running a DSMS server and multiple clients running a DSMS resolver and agent. This is similar to the setup used to measure DSMS server responses in the last section. The second setup is a distributed referral index using 10 DSMS servers shown in Figure 5. OH is the EOS domain root and contains, among 100K total, referrals to brokers with referrals for *Atmospheric Composition* (CA), *Atmospheric Dynamics* (AZ) and *Ocean Dynamics* (VA). CA, AZ & VA contain referrals to brokers with *Atmospheric Composition/Greenhouse Gases*, *Atmospheric Dynamics/Winds* and *Ocean Dynamics/Pressure*, respectively. NY, TX, NV, FL, RI & UT have referrals to a subtopic of their respective parent nodes in the figure. WI represents one of a number of test clients containing a single referral to the domain root. Each server has a total of 100K referrals. The referral record size was increased to 1000 bytes to accommodate centroid paths. The new single server no-load response times are shown in Figure 6.

The two experiments consist of configuring a set of resolvers to generate a random referral query stream with Poisson arrival rates. The stream of queries observed at the server is the sum of the individual Poisson streams generated at each resolver. Multiple resolvers were required because a single resolver would be overloaded attempting to handle the data resulting from the queries generated. Seven arrival rates were applied to both architectures and the server response times measured.

In the distributed index test the query load is distributed evenly among the DSMS servers. Resolvers randomly select one of three possible queries. The pairs of servers under CA, AZ and VA contain referrals that satisfy one of the three queries. This effectively partitions the referral search space allowing better parallelism. Users searching for *Ocean Dynam-*

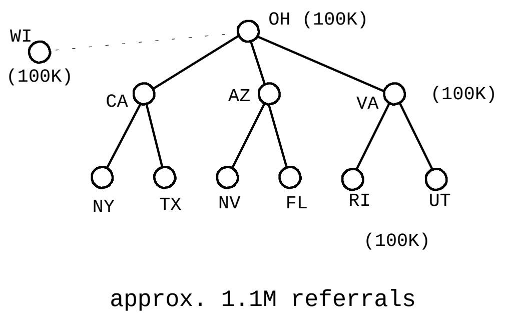

Figure 5: Distributed referral index configuration. Links represent referrals.

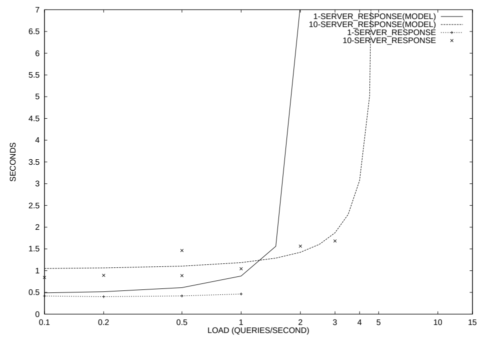

Figure 7: Measured and Modeled Performance, 1- and 10-server configurations.

ics referrals do not compete with users searching for *AtmosphericComposition* referrals. We have used this configuration for our first set of tests, but we are exploring the impact of other configurations as well.

3.4 Measured and Modeled Performance Results

We evaluated the performance of our scheme using both experimental measurements of response times as well as through analytical modeling to supplement the experiments we have been conducting.

Figure 7 shows the average agent response times for a referral query returning 100 results for the cases of a single server, and multi-site servers, respectively. The measurements were made at each agent as load increased. Measurements included the time between the resolver receiving a query from the agent and the time all referrals were received. 1-SERVER_RESPONSE is the average response time measured at a client, using a centralized index architecture. The measured values are consistent with response times observed in the no-load experiment from the earlier study (Figure 6). 10-SERVER_RESPONSE shows the average response time measured, using a distributed index configuration for a referral query returning 100 results. The lines 1-SERVER_RESPONSE(MODEL) and 10-SERVER_RESPONSE(MODEL) show the results from our analytical model for the 1-server and 10-server cases, respectively. As the figure shows, the model is close enough to the measured values to be useful.

Not surprisingly, the 1-server case breaks down badly past a query load of 1 query/second. The CPU was incapable of handling higher loads. The distributed index begins to outperform the centralized architecture by a large margin as the load increases past this point. For the configuration chosen, the cross-over point is at a query load of just over 1 query/second. For this same configuration, the distributed architecture begins to show the effects of saturation beyond a query load of 5 queries/second. However, it is easy to accommodate heavier query loads by increasing the number of servers used.

4 Conclusions

We have introduced DSMS, a novel prototype distributed index system. We have tested the architecture using the NASA Earth Observing System Global Change Master Directory parameters to construct a centroid hierarchy. DSMS has demonstrated the feasibility of using distributed indices to manage network information discovery. The preliminary results support our claim that a distributed index can reduce metadata query response time when compared to centralized systems. THE DSMS approach is applicable even when a centralized service is appropriate and feasible. In this case, the servers could all be located at the same site, typically on the same local network, and the referrals divided among the servers.

We intend to continue developing DSMS by adding caching and refining the analytical model. The referral cache is likely to have a high hit rate because it is reasonable to expect EOS user interests to remain constant with respect to a small number of topics; or change slowly over time. So, the queries will show high locality of reference. Modeling will allow us to investigate the effects of scaling on the number of metadata and DSMS server sites as well as the query load. Currently the scale of the prototype is limited by available resources.

Finally, work is need to define the characteristics of centroid hierarchies. This would help define guidelines, perhaps a standard, for constructing domains.

References

[1] A. Emtage and P. Deutsch. Archie - an electronic directory service for the internet. In *Proceedings of USENIX Winter Conference*, pages 93–110, January 1992.

[2] Michael L. Mauldin and John R. R. Leavitt. Web-agent related research at the CMT. In *Proceedings of the ACM Special Interest Group on Networked Information Discovery and Retrieval (SIGNIDR-94)*, August 1994.

[3] Lois Mai Chan. *Cataloging and classification: an introduction.* McGraw-Hill, 1994.

[4] Marion E. Matters. *Introduction to the USMARC format for archival and manuscripts control.* Society of American Archivists, Chicago, IL, 1990.

[5] United States/Employment and Training Administration. *Dictionary of occupational titles.* Career Press, 4th edition, 1992.

[6] Bureau of the Census U.S. Dept. of Commerce. Dictionary of occupational titles. http://immigration-usa.com/dot_index.html, Information Technology Associates, 1997.

[7] NASA. Global change master directory proposed keyword list v23. http://gcmd.gsfc.nasa.gov/keywords/homepage_keywords.html, 1995.

[8] H. Ramapriyan. The EOS data and information system. Technical report, American Institute of Aeronautics and Astronautics Inc., 1990.

[9] Joann Ordille and Barton P. Miller. Database challenges in global information systems. In *Proceedings of the 1993 ACM SIGMOD International Conference on the Management of Data*, pages 403–407, May 1993.

[10] B. Kantor and P. Lapsley. Network news transfer protocol. RFC-977, Internet Request for Comments, February 1986.

[11] Sape Mullender, editor. *Distributed Systems.* Addison-Wesley, Reading, Mass., 2nd edition, 1993.

[12] George Coulouris, Jean Dollimore, and Tim Kindberg, editors. *Distributed Systems Concepts and Design.* Addison-Wesley, Reading, Mass., 2nd edition, 1994.

[13] Lori J. Tyahla. ECS user characterization methodology. Report 194-00313TPW, Hughes Applied Information Systems, September 1994.

[14] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, August 1994.

[15] P. B. Danzig, S. Li, and K. Obraczka. Distributed indexing of Autonomous Internet Services. *Computing Systems*, 5(4):433–459, 1992.

[16] D. Hardy and M. Schwartz. Essence: A resource discovery system based on semantic file indexing. In *Proceedings of USENIX Winter Conference*, pages 361–373, January 1993.

[17] G. Salton and M. J. McGill. *Introduction to modern information retrieval.* McGraw-Hill, New York, NY, 1983.

[18] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings Publishing, Redwood City, CA, 1989.

[19] Stefano Ceri and Giuseppe Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill, New York, NY, 1984.

[20] M. Tamer Ozsu and Patrick Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Englewood Cliffs, N.J., 1991.

[21] Luis Gravano and Hector Garcia-Molina. Generalizing GlOSS to vector-space databases and broker hierachies. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 78–89, 1995.

[22] D. Notkin, N. Hutchinson, J. Sanislo, and M. Scwhartz. Heterogeneous computing environments: report on the ACM sigops workshop on accommodating heterogeneity. *Communications of the ACM*, 30(2), February 1987.

[23] A. Sheth and J. Larson. Federated database systems for managing distributed heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[24] Larry Smarr and Charles E. Catlett. Metacomputing. *Communications of the ACM*, 35(6), June 1992.

[25] Won Kim, Injun Choi, Sunit Gala, and Mark Scheevel. On resolving schematic heterogeneity in multidatabase systems. *Distributed and Parallel Databases*, 1(3):73–95, 1993.