Spatial Indexing of Distributed Multidimensional Datasets [*]

Beomseok Nam and Alan Sussman
UMIACS and Dept. of Computer Science
University of Maryland
College Park, MD 20742

Abstract

While declustering methods for distributed multidimensional indexing of large datasets have been researched widely in the past, replication techniques for multidimensional indexes have not been investigated deeply. In general, a centralized index server may become the performance bottleneck in a wide area network rather than the data servers, since the index is likely to be accessed more often than any of the datasets in the servers. In this paper, we present two different multidimensional indexing algorithms for a distributed environment - a centralized global index and a two-level hierarchical index. Our experimental results show that the centralized scheme does not scale well for either insertion or searching the index. In order to improve the scalability of the index server, we have employed a replication protocol for both the centralized and two-level index schemes that allows some inconsistency between replicas without affecting correctness. Our experiments show that the two-level hierarchical index scheme shows better scalability for both building and searching the index than the non-replicated centralized index, but replication can make the centralized index faster than the two-level hierarchical index for searching in some cases.

1 Introduction

In many scientific disciplines, large amounts of sensor and simulation data are being generated continually and stored on disks and other large-scale storage systems. Because of the the demand for large storage capacity for such datasets, and because those datasets are acquired or generated at multiple sites, the datasets must be stored in multiple, distributed storage devices.

Many scientific datasets are made up of collections of multidimensional arrays. One of the most common access pattern into such scientific datasets is a spatio-temporal *range query*, which reads the subset of the arrays that overlaps the given query range. Multidimensional indexing structures, such as R-trees or its variants, allow for direct access to subsets of a dataset in order to improve data access performance [2, 12].

In this paper, we discuss the benefits that can be obtained from distributing the index itself. One way of distributing the index is to replicate the whole index file onto multiple servers, and another way is partitioning the index and storing parts of the index on multiple servers. There have been some efforts to employ the second approach, such as Master R-trees [6], but most of that work has focused on declustering methods, which determine data placement. However, for the applications we target, we cannot assume that we control where datasets are stored.

To improve the performance of range queries on scientific datasets, in previous work we have developed a generic multidimensional indexing library (GMIL) [11]. GMIL indexes datasets stored in a variety of scientific data formats, including HDF4, HDF5, and netCDF, and improves range query performance into such datasets using a technique called *data chunking*. Since scientific datasets tend to increase in size over time, and the overall size of a dataset can become very large, it is not necessarily a good idea to index individual array elements. Instead, we partition the multidimensional array into the data chunks, typically all of approximately the same size, and compute the bounding box containing all the data elements in the datasets' underlying multidimensional attribute space. The bounding boxes are then used as the keys into the index. In this way we can reduce the size of the index and accelerate range query performance, at the cost of always retrieving entire chunks of data from storage.

Geographically distributed sensor devices will store the datasets in their local machines, and large datasets generated by simulations will be stored near where they are produced. For both organizational and technical reasons, it is

[*]This research was supported by the National Science Foundation under Grants #EIA-0121161 and #ACI-9619020 (UC Subcontract #10152408), Lawrence Livermore National Laboratory under Grant #B500288, and NASA under Grant #NAG511994.

often not desirable to move or replicate huge datasets onto remote machines. However the current design and implementation of GMIL does not support indexing of distributed datasets. Hence the techniques we describe in this paper will extend GMIL to a distributed storage environment, to improve the performance of scientific applications that access large datasets in a high performance distributed (Grid) environment.

Although research on partitioned index methods has investigated index performance across a distributed set of machines, no research that we know of has measured the overhead of a more centralized index server. In a wide area network, we may expect that the master server in a centralized scheme could be a performance bottleneck, thus distributing the index into multiple servers should alleviate the overhead on an overloaded index server to make such an indexing scheme more scalable.

In this paper, we evaluate the scalability of two different distributed indexing schemes - a replicated centralized index and a two-level hierarchical index. The centralized index distributes the load of a single index server to several hosts, replicating the entire index to each host. The two-level hierarchical index has each host maintain its own local index for the data stored on that host, while the information from the root nodes of the local indexes is stored in a top level global index, that may also be replicated.

As more storage capacity is required to store large datasets, recently much research has been focused on developing more scalable distributed file systems. One of the most successful systems is the Storage Resource Broker (SRB), which is not only a distributed file system designed for a heterogeneous Grid computing environment, but also supports various data management facilities, such as data replication, third-party copies, data look-up services, etc. [1]. The SRB is the platform for our implementation and the experiments.

The rest of the paper is organized as follows. In Section 2 we discuss other research related to distributed indexing. In Section 3 we introduce the two distributed indexing schemes, centralized indexing and two level hierarchical indexing. We also briefly discuss replication and concurrency control mechanisms for the distributed multidimensional indexes. In Section 4 we present performance results for both centralized and two level hierarchical indexing, examining the costs of both building and searching the index. We conclude in Section 6.

2 Related Work

Liebeherr et al. [7] evaluated the performance benefits of partitioned B+-tree indexes for a single relation in distributed relational databases. The single relation is partitioned across all servers and the ranges for each fragment do not overlap each other. They introduced a partitioned global index (PGI) scheme and compared it with the classical partitioned index scheme (PI), in which each server indexes its own data but does not know about the global status, therefore a broadcast message must be sent to all the servers for each request. In PGI, each server has a master index, hence can forward range queries to the servers that have the requested data. However, since PGI has several serious problems in terms of index update, that work assumed that there will be no index update.

Since the R-tree was introduced in 1984 for indexing multidimensional objects, little work was done on parallelizing R-tree operations until Kamel and Faloutsos [4] proposed the first parallel R-trees (Multiplexed R-trees) in 1992, for a machine with a single CPU and multiple disks. They investigated various declustering methods that decide how to distribute the leaves of an R-tree across multiple disks. The limitation of that paper for the current work is that the parallel architecture has a single CPU and multiple disks. Since that paper several parallel multidimensional indexing structures have been studied to extend Multiplexed R-trees, such as Master R-trees [6] and Master Client R-trees [14].

The Master R-tree was designed for a shared nothing environment (i.e. distributed memory parallel machine) by Koudas et al. [6]. A single server maintains all the internal nodes of the R-tree except the leaf level data nodes, which are declustered across the other servers. The paper focuses on declustering methods and deriving formulas for the optimal distributed leaf node size. A Master Client R-tree, proposed by Schnitzer et al. [14], is a two-level distributed R-tree that has a single master index on a master server and local client indexes on the other servers. The Master Client R-tree is similar to the Master R-tree in the sense that it declusters leaf level nodes across data servers. However each data server creates its own local index using the leaf level nodes that are assigned to it. Therefore, the master index does not have to keep the pointers to the data objects in its master index. Instead, it contains the server address where its local index must be searched again in order to get pointers to the data objects. The authors claim that the overhead of the master server can be reduced by maintaining a smaller master index file. In our work, we make the master index even smaller by storing only the root Minimum Bounding Rectangles (MBRs) of local indexes.

Mondal et al. investigated data migration techniques that move the workload from heavily loaded servers to lightly loaded servers in shared nothing environments [9]. The master server in their scheme controls all other second level servers and maintains information about the second level index. The second level index servers periodically send messages concerning their load status to the master server so that the master server can migrate data from heavily loaded servers. In a wide area network, dynamic data migration may not be useful, especially when the size of data is very large. In the scientific applications we target, we cannot migrate the data chunks, which are typically small subarrays of large multidimensional arrays.

All the prior research discussed requires at least one dedicated server for global status information about the distributed index, which is a potential bottleneck. To avoid centralized data access, fully decentralized indexing structures have been proposed, such as Scalable Distributed Data Structures (SDDS) [8] and the recently developed P2PR-tree [10]. In both indexing structures, each participating server has partial information about global status, and servers cooperate to send and receive data to/from the right server as in peer-to-peer systems. We do not discuss fully decentralized indexing in this paper, but in the near future we will investigate its behavior as compared to the indexing schemes presented in Section 3.

3 Multidimensional Indexing in Wide Area Network

3.1 Centralized Indexing

In the centralized indexing scheme, a single index server stores all the index tree nodes. Since data items are distributed across multiple data servers, leaf level nodes in a centralized index have server names, file names, and offset information. All range queries must be forwarded to the central index server, and the central server searches its index and returns pointers to the data to the requesting client. After receiving the pointers, clients can request data objects from the specified data servers. Alternately, the central index server can multicast data read requests to the appropriate servers as in Master R-trees, which would increase the overhead on the central index server.

Figure 1 shows the sequence of operations for a request to the centralized index. In step 1 a client (A) connects to one of the servers (server 2, in this example). After the connection is established, in step 2 that server accesses the metadata service server to get the necessary system metadata needed for processing client requests (i.e. access permissions, index server location, etc). After server 2 gets the centralized index server location from the metadata server, it forwards the range query to the centralized index server (server 1) in step 3. Index server 1 then opens the index file and performs the search or insert operation. If the request from the client is a search operation, server 1 returns the set of pointers to the requested data to server 2 in step 4. After server 2 parses the set of pointers, it sends read requests to other servers (server 3 and N in Figure 1) in step 5, merges the data returned from the servers, and finally returns the

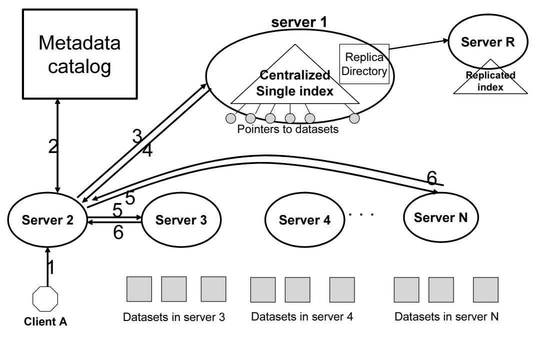

Figure 1. *Searching with Centralized Indexing*

data to the client. For an insert operation, centralized index server 1 updates the index to complete the operation.

3.2 Two Level Hierarchical Indexing

In the centralized indexing scheme, a dedicated index server may become a performance bottleneck, especially when a large number of clients update the index concurrently since an insert/update operation needs exclusive access to the index file. We can implement the index update operation as an atomic blocking operation using a semaphore. If possible, the number of update operations should be minimized for performance reasons, since concurrent blocking operations are expensive. Although searching the index does not need to be a mutually exclusive operation, we expect that the centralized index server will become a performance bottleneck when a large number of clients search the index simultaneously.

For these reasons, we have designed and implemented the two level hierarchical distributed indexing scheme shown in Figure 2. Because of the behavior of multidimensional indexing structures, updating or searching an index file in parallel is a good way to distribute the load on a centralized server. There are two ways to parallelize index operations. One method is to simply replicate the index, while the other is to partition the index and distribute the parts to multiple servers. Partitioning not only spreads clients' requests across multiple index servers, but also decreases the amount of the work to be done by each server for an index request (search or insert) because each server has a smaller index to operate on.

In two level hierarchical indexing, each data server has its own index for local data (a *local index*). For searching the index, a *global index* is used to determine which local index(es) must be accessed. The global index stores the Minimum Bounding Rectangles (MBRs) of the root nodes of the local indexes. When a range query is submitted to the server owning the global index, the server compares the

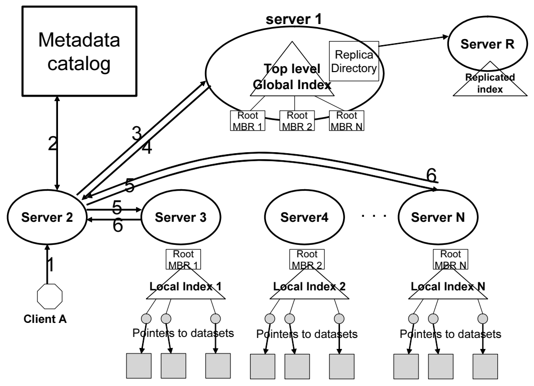

Figure 2. *Searching with Two Level Hierarchical Indexing*

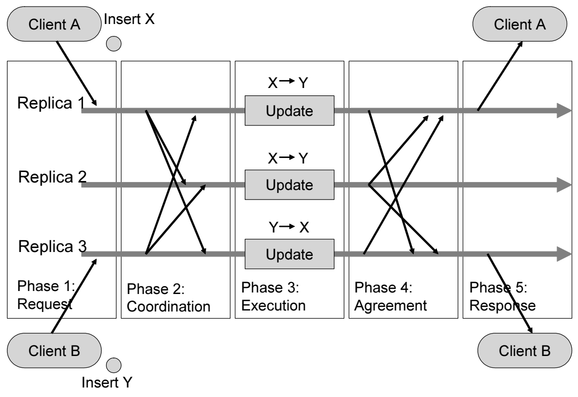

Figure 3. *Functional model of the replication strategy that allows inconsistency without affecting correctness*

range with those MBRs and returns the list of servers that have overlapping MBRs with the given range. Since the global index does not contain any information about the actual data stored in the servers, it is possible for the global index server to return local servers for a query when, in fact, those local servers do not have any data that overlaps the query range.

When a sensor device or a simulation stores data in a local server, the data will first be inserted into the local index for that server in the two-level hierarchical indexing scheme. When a data object is inserted that is outside the current MBR of the root node of the local index, that MBR must be extended to include the new data object. When the root MBR changes, a root node update notification is forwarded to the top level global index server. When the global index server receives the update notification, it searches its index, deletes the old MBR and inserts the new one. Many insertion operations are done only in local servers, which can greatly improve index insertion performance, as will be shown in Section 4.

The size of the top level global index depends on the number of local indexes, not on the total number of data objects being indexed. Therefore, the size of top level global index is much smaller than for the centralized index, so searching the global index is faster than searching the centralized index. If there are N local indexes and the capacity of a leaf node in the global index is greater than N, searching the global index using a spatial indexing tree structure would be the same as for a sequential scan through the leaves of the tree. However, with a 4KB page size and 3 dimensional bounding boxes, a leaf node can hold information for approximately 13 servers in our current implementation. Hence, by maintaining the global index as a spatial tree structure, we can make searching the global index server faster than a simple linear scan.

Figure 2 shows how to search the two level hierarchical index. In general it is similar to searching the centralized index, except that the top level global index server (server 1) returns local data server information (which data servers to do local searches on), rather than pointers into the datasets on each local server, back to server 2. After server 2 receives the list with local server information, it forwards the range query to those servers, in this case servers 3 and N. Note that in the centralized indexing scheme, the client sends pointers to the datasets for the data to be read, not the range query, to the data servers. In two-level hierarchical indexing, the data servers are responsible for searching their own local indexes, reading the parts of the datasets pointed to by the index, and returning the data to the requesting server (server 2 in this case), which then forwards it to the client.

3.3 Replication and Concurrency Control

The main challenges in distributed indexing services are dealing with a wide area network and scalability issues when a large number of of client applications connect to the index server. When datasets are stored in multiple sites in a wide area network, replication of index files can decrease the network costs for index searches.

Replication has been studied in many areas including distributed systems and in databases. While replication in distributed systems is done mainly for fault tolerance, database research focuses on its performance implications [16]. In this paper, we focus on the performance of a replicated index and leave fault tolerance issues for future work.

Figure 3 depicts the replication protocol for the multi-dimensional index. Wiesmann et al. [16] classified various replication protocols using five generic distinct phases as in the figure. In the first phase, clients send requests to one (or more) replica. In the coordination phase, the replica servers coordinate an execution order so that all the replicas have the same data dependencies. In the execution phase, the requests are executed in the order agreed upon. In the agreement phase, the replicas should agree that they updated data in the same order as did the other replicas. The client receives a response in the response phase.

For the centralized index the whole index is replicated, but for the two-level scheme only the global index is replicated. In our replication protocol for a multidimensional index, a client submits a request to one known replicated index server. Clients do not need to know where all the replicas are located, but they must know the location of at least one of them. After receiving a request from a client, if that request is an insertion operation the index server multicasts the request to the other replicated index servers using its directory, which contains the location of all other replicas. If the request is an index search, the search is executed on the local index replica.

In order to ensure consistency across replicas, distributed locking protocols or atomic broadcasts have been extensively researched. However, due to the properties of the index, we can obtain correct search results without strong consistency mechanisms, as we now describe. Multidimensional spatial indexing trees have nondeterministic internal structure. If we insert the same data objects in different orders, the resulting tree structures can be different. Nonetheless, any of those indexes will return the same result for any given query, as long as all the structures contain the same data objects in their leaf nodes (i.e., the same data objects have been inserted). We allow some inconsistency between replicas of an index in order to improve the performance of insertion operations, so that clients can insert data concurrently into different replicas, as if there are no *write-after-write* data dependences across insertions. In Figure 3, clients A and B simultaneously insert data into different replicas. In replicas 1 and 2, X is inserted first and Y next, but in replica 3, Y is inserted first and X next, thus the index tree structure of replica 3 may be different from that of replicas 1 and 2. However, that will have no effect on the results of a search into any index replica.

In the execution phase, each update transaction must be done atomically. When several client requests arrive at the same server, the server creates a thread for each request. Hence the local index file replica needs to have a lock to handle those concurrent write requests. Note that this lock is only for the local index replica, and we do not need to obtain a lock that spans all replicas. A server thread must obtain the lock to open the local index file, and releases it when the thread closes the file. Instead of this coarse grained locking strategy, we could also use a fine grained locking algorithm for the multidimensional index, as in [15, 5], in order to increase concurrency. We leave this optimization for future work.

In the agreement phase, each replica sends back either confirmation messages or requested data to the replica the client initially contacted. Since we allow different execution orders across replicas, we do not require a distributed commit protocol in this phase. In the final response phase, the replica the client contacted simply returns the result data to the requesting client.

4 Experiments

4.1 Storage Resource Broker

We have employed the Storage Resource Broker (SRB) in the implementation of the distributed indexing structures. The SRB is a client-server system developed at the San Diego Supercomputer Center (SDSC) that provides a uniform interface for connecting to heterogeneous data resources, such as storage area networks (SANs), high performance multi-level storage systems such as HPSS [3], Unix file systems, Oracle databases, etc., over a wide area network [1, 13]. The SRB provides a well-defined storage interface to heterogeneous storage resources by mapping from those interfaces to the underlying storage resource interfaces. Datasets managed by SRB can be accessed through the MCAT (MetaData Catalog) service, which is a relational database designed to enable attribute-based querying and identification of data, via metadata attached to the datasets. However MCAT does not support multidimensional indexing operations.

We have implemented multidimensional spatio-temporal indexing modules on top of the basic SRB infrastructure, to support multidimensional range queries into datasets accessible by the SRB. We have chosen Spatial Hybrid trees (SH-trees) [12] for the indexing data structure, which provides the same basic functionality as R-trees [2]. SH-trees have been shown to provide better performance, both in terms of the building and searching, than R-trees [12]. Functions for building and searching SH-trees are implemented as SRB proxy functions. The proxy functions enable an SRB server to forward client requests to other SRB servers without any interaction with the clients [1]. Thus, clients do not need to know where the local indexes and datasets are located.

For performance reasons, we decided not to register index files in the MCAT, because the MCAT can be a serious performance bottleneck. If we register an index file in MCAT to make it easy to find, then whenever we open or close an index file the SRB server contacts the MCAT server to update the metadata for the file.

4.2 Experimental Environment

We measured the performance of the centralized and two-level indexing schemes on two clusters geographically distributed over a wide area network. The first is a Linux cluster at U. Maryland, where each of 40 nodes has a Pentium III 650 MHz processor, and the nodes are connected by a 100Mb/sec switched Ethernet network. The second is a Linux cluster at Ohio State University, where each of 20 nodes has a Pentium III 933 MHz processor, also connected by switched 100 Mb/sec Ethernet. These two clusters are connected by the Internet2 wide area network.

Three dimensional satellite image datasets were used to evaluate the two indexing schemes. The satellites gather remotely sensed AVHRR (Advanced Very High Resolution Radiometer) GAC (Global Area Coverage) level 1B datasets, stored as a set of arrays. Satellite datasets includes geo-location fields, time fields, and some additional metadata. As the satellite moves along a ground track over the earth, it records longitude, latitude, and time values, as well as sensor values. Because the sensor swings across the ground track, the sensor values and meta values are stored as two dimensional arrays. We partitioned those arrays into equal sized rectangular chunks, built three dimensional bounding boxes (latitude, longitude, and time) for each of them, and stored the data server address, file name, and the array offset as a pointer to the chunk.

The dataset used was collected over one month (January 1992), and has a total size of more than 30GB, with the volume of data for a single day about 1GB. The dataset was partitioned into 400,000 chunks. We assigned 10,000 chunks to each of 40 servers, 20 at Maryland and 20 at Ohio State. In order to create range queries, we modeled common query behaviors, including hot spots in the data corresponding to areas of high interest, which matches realistic workloads for this type of data..

4.3 Experimental Results

We evaluated the index creation time and search time for both centralized indexing and two-level hierarchical indexing. Once the scientific datasets are stored on disks, they are not likely to change. Thus we do not measure index deletion performance, but it should be similar to insertion performance. Figure 4 shows the total elapsed wall clock time to insert 10,000 objects per client into the index. A client waits for an insertion to complete before performing the next insertion. We ran a single client on each server and increased the number of servers from 4 to 40. For the centralized scheme the entire index and for the two-level scheme the global index were located on a Maryland server. However, the average insertion time for the clients in Ohio is only 2% slower than for Maryland, and the search times also are about the same. Whether the index is on a local or a remote cluster does not affect the overall performance of index accesses greatly. We think that this is partly because the clusters are connected by Internet2, which has much greater bandwidth than the local area network (but much higher latencies), and also most of time for index operations is spent on disk I/O rather than in network delay even for remote index accesses. The time to insert data into the centralized index increases rapidly as the number of clients increases. Meanwhile the time to insert data into the two-level hierarchical index is almost independent of the number of concurrent clients. In the two-level hierarchical indexing scheme, most of the insertion operations are done completely locally, and connecting to the server containing the global index only when the root MBR on that server changes.

Figure 5 shows the index creation time when the index is replicated. We fixed the number of clients at 20, which run at Maryland, and each client inserts 10,000 objects. As the number of replicas increases from 1 to 20, the insertion time for the centralized index increases by 32%, and the insertion time for two-level hierarchical index increases by 58%, but from a much lower starting point. When there are

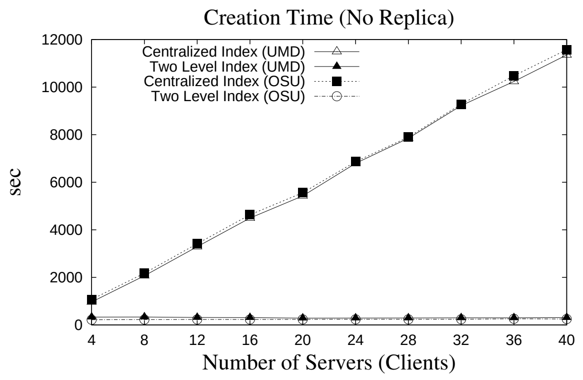

Figure 4. *Creation Time without Replication*

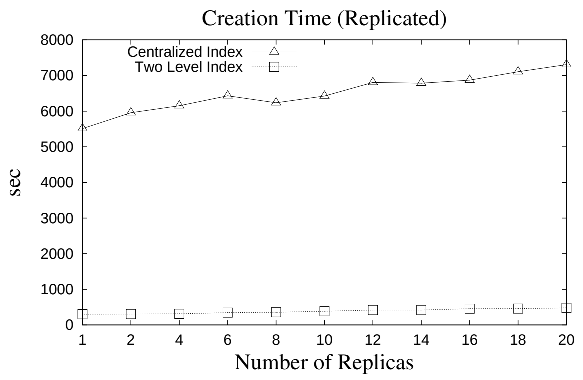

Figure 5. *Creation Time with Replication*

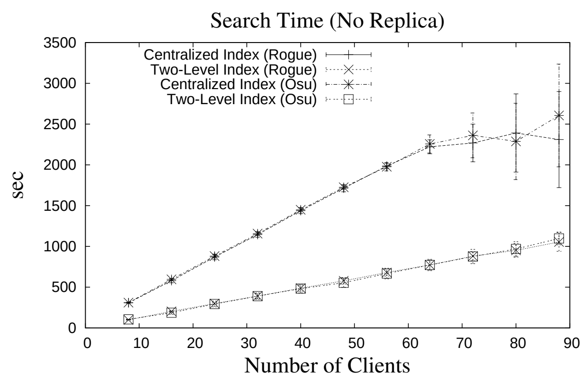

Figure 6. *Search Time without Replication*

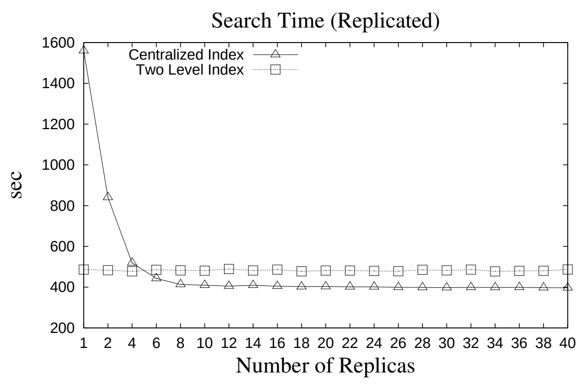

Figure 7. *Search Time with Replication*

20 replicated indexes, each client inserts data into its own local index and the local SRB server forwards the update to the other replicas. Inserting data into multiple replicated indexes is a non-blocking operation and is performed in parallel, therefore the cost for additional replicated indexes is not very high. If we had employed a blocking insertion operation for strict consistency among replicas, the cost of having more replicas would have been quite substantial.

To evaluate the performance of index searches, we ran 2 clients per node and each client submitted 100 queries. Figure 6 shows search performance for both indexing schemes without replication. Error bars are shown in this graph to emphasize the standard deviation in execution time across clients, because the variance is quite large in contrast to the other experiments. Each query accessed data on 4 to 5 servers on average. The search time for the centralized indexing scheme in Figure 6 includes the time for the centralized index server to connect to the local servers that have the desired data for all 100 queries, even though the servers do not read the actual dataset to be returned (since the same data will end up being read for both indexing schemes). We also measured the time for the centralized index server to do its index lookup without connecting to the local servers, which was only 2-4% faster. That means up to 98% of the search time is spent on searching the index for the centralized server. The time to search the two-level hierarchical index includes the time to connect to the local servers, since the local index must be searched to obtain the query result. The actual dataset in the local server can then be read.

The two-level hierarchical indexing scheme is up to 3 times faster than the centralized index without replication. As the number of clients that submits queries increases, the performance gap between the two schemes increases. When the number of clients is over 60, significant resource contention begins to occur in the centralized index server, as shown by the large standard deviation across queries for large numbers of clients.

Figure 7 shows search performance when the indexes are

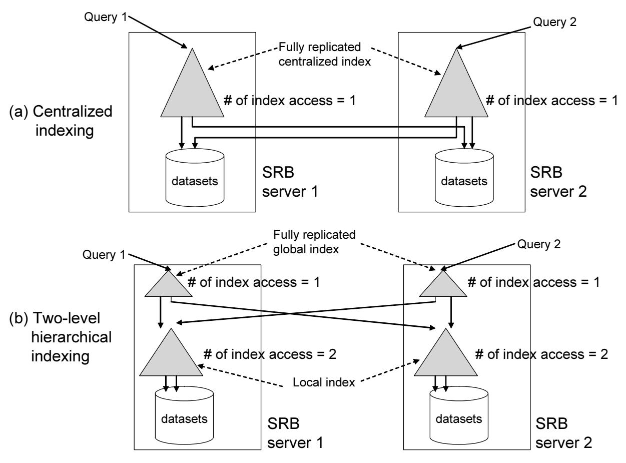

Figure 8. *Two-level hierarchical index is not always better than replicated centralized index.*

replicated onto multiple servers. We ran 40 clients on 40 hosts, and varied the number of replicated indexes from 1 to 40. In this experiment, the performance of the two-level hierarchical indexing scheme does not depend on the number of replicated indexes, because the server overhead in the global index server is very low even when there is only a single global index and 40 clients submit range queries to the same server. The file size of the global index is only 167KB for 40 data (and local index) servers, so searching such a small index does not cause much overhead. On the other hand, the performance of the centralized index improves significantly with more index replicas. When there are more than 4 replicas, searching the centralized index became faster than for the two-level hierarchical index.

When the centralized index is fully replicated, any query will access its global index in its local server. However some queries will be forwarded from other servers to the local server to do searches into the local index for the two-

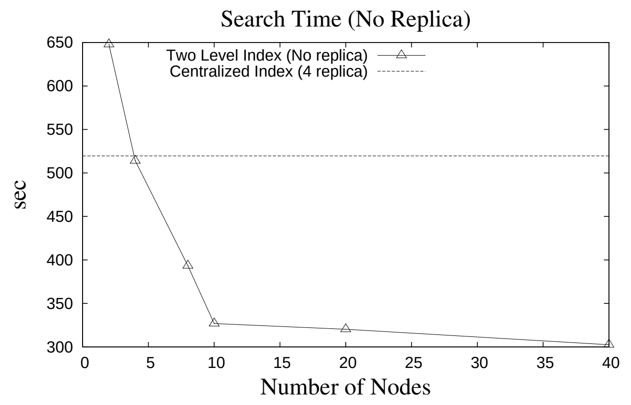

Figure 9. *The Effect of Number of Replicas*

level scheme. Thus the number of accesses to the local indexes, which is a dominant factor in the response time for the two-level scheme, does not decrease when we replicate the global index. In Figure 8, the total number of accesses to the index files in a single server is 3 for the two-level hierarchical indexing scheme, while the centralized index is accessed only once. The index file sizes and the expected number of local indexes involved for a submitted query, which is determined by the query window size, are important factors that decide whether the fully replicated centralized indexing or the two-level hierarchical indexing performs better.

Although our experiments showed that increasing the number of replicas of the global index for the two-level hierarchical indexing scheme did not increase performance, if the number of data servers is much larger than the 40 used in the experiments shown, and the number of clients that submit queries is also large, the global index may become a performance bottleneck. We have not yet tested this hypothesis, but will do so in the future.

For the experiments shown in Firgure 9, we declustered 400,000 data objects across from 2 to 40 data servers (200,000 local data when there are 2 data servers, and 10,000 local data when there are 40 data servers.) When the number of data server is 1, it would work like centralized index. The centralized index is independent on the number of data servers but the number of data objects. In the Figure 9, the performance of centralized index is from when there are 4 replicated centralized index. As the number of data servers N increases, the query response time for two-level index decreases significantly.

5 Performance Model

We modeled both indexing schemes using the parameters in Table 1. The number of network messages in both indexing schemes are proportional to $2 \cdot \sum_{i=1}^{Q} W_i$. Note

N	# of nodes
Q	# of queries
R	# of replicas
D_C	average # of disk access to centralized index
D_G	average # of disk access to global index
D_L	average # of disk access to local index
W_i	expected # of data servers involved for a query i ($0 \leq W_i \; legN$)

Table 1. Variables

that W_i in the two-level indexing will be a little higher than that of centralized indexing, because two-level indexing may search some local indexes which do not have the data that overlap the given range query due to dead space. The average number of disk access to one of the replicated index for a single server is $\lceil Q/R \rceil \cdot D_C$ in the centralized indexing scheme, when the workload is balanced. Also, the average number of disk access to the top level global index for single server in two-level hierarchical indexing scheme is $\lceil Q/R \rceil \cdot D_G$ The average number of disk access to local index for a single server in two-level hierarchical indexing scheme is $\sum_{i=1}^{Q} W_i/N \cdot D_L$. Therefore the average number of disk access in two-level hierarchical indexing scheme is

$$\lceil Q/R \rceil \cdot D_G + \sum_{i=1}^{Q} W_i/N \cdot D_L$$

Note that we have shown that D_G is ignorable since D_C is much higher than D_G due to the index file size.

The size of index file depends on the number of data objects, the number of fan outs, and node utilization. If we assume that the node utilization is 100%, the number of leaf nodes (m) in the index will be $(number\ of\ data\ objects)/k$, where k is the number of fan-outs. Then, the size of the index (S_C) will be

$$\begin{aligned} S_C &\approx m + \frac{m}{k} + \frac{m}{k^2} + ... + \frac{m}{k^{\log_k^m}} \\ &= m + (\frac{m}{k} \cdot (1 - (\frac{1}{k})^{\log_k^m}))/(1 - \frac{1}{k}) \\ &= m + \frac{m-1}{k-1} \end{aligned}$$

When we decluster the data objects across N data servers, the index file size S_L will be

$$S_L \approx \frac{m}{N} + \frac{\frac{m}{N} - 1}{k-1} = \frac{1}{N} \cdot (m + \frac{m-N}{k-1})$$

And the size of global index in two-level hierarchical scheme will be

$$S_G \approx \frac{N}{k} + \frac{\frac{N}{k} - 1}{k-1} = \frac{N-1}{k-1}$$

Since the number of data objects is usually much larger than the number of data servers, we may think of $S_L \approx S_C/N$. The average number of disk access D_G, D_L, and D_C depends on the size of index files and the window size of the queries. Thus we may assume D_C is in proportion to S_C, D_L in proportion to $S_L = S_C/N$ and D_G in proportion to $S_G = \frac{N-1}{k-1}$. (W is 1 when a query is unbounded.)

Now we can rewrite the average number of disk access to the centralized index as

$$\lceil Q/R \rceil \cdot D_C \approx \lceil Q/R \rceil \cdot S_C$$

The average number of disk access to the global and local index in two-level hierarchical indexing is

$$\lceil Q/R \rceil \cdot D_G + \sum_{i=1}^{Q} W_i/N \cdot D_L$$

$$\approx \lceil Q/R \rceil \cdot \frac{N-1}{k-1} + \sum_{i=1}^{Q} W_i \cdot S_C/N^2$$

From these formula, we claim the followings.

(1) As the number of replicas increases, centralized index becomes better than two-level index.

(2) As the number of nodes increases, two-level indexing becomes better than centralized index.

From the experiments shown in Figure 7 and Figure 9, it is evident that the two claims are correct, which say both of hierarchical indexing and replication help improving the performance. However we can not tell a particular indexing scheme is better than the others since centralized indexing works better with more replicas and hierarchical indexing works better if there are large number of data servers. For a relatively small number of servers, full replication of centralized index can be a good choice, especially when the frequency of index search operations is much greater than the frequency of index updates. However, as the number of data servers increases, index updates would become a performance problem in replicated centralized indexing.

6 Conclusion

We have compared two approaches to creating and searching a multidimensional index for distributed datasets - a centralized indexing scheme and a two-level hierarchical indexing scheme. Our performance evaluation demonstrates that two-level indexing scales well for both insertion and searching, but there is still some potential to improve the performance of centralized index by replication. Although replication of the centralized index does not cause very high additional overhead, insertion operations into the single centralized index without replication is very expensive compared to insertions into a two-level hierarchical index. In order to make the replicated centralized indexing more scalable, we need to investigate how to improve the index creation performance.

As in many scientific applications, if search operations are much more frequent than insertions into the index, then the replicated centralized indexing scheme may be an attractive choice. However we still need to investigate how to select the number of replicas dynamically, based on the load on each index server.

References

[1] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Proceedings of CASCON'98 Conference*, Dec. 1998. *http://www.cas.ibm.com/archives/1998/index.html*.

[2] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of SIGMOD'84*, pages 47–57. ACM Press, May 1984.

[3] HPSS: High performance storage system. *http:/www.hpss-collaboration.org/hpss*.

[4] I. Kamel and C. Faloutsos. Parallel R-trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD92)*, pages 195–204. ACM Press, 1992.

[5] M. Kornacker, C. Mohan, and J. M. Hellerstein. Concurrency and recovery in generalized search trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD97)*, pages 62–72. ACM Press, 1997.

[6] N. Koudas, C. Faloutsos, and I. Kamel. Declustering spatial databases on a multi-computer architecture. In *Proceedings of Extended Database Technologies*. Springer Verlag, 1996.

[7] J. Liebeherr, E. Omiecinski, and I. Akyildiz. The effect of index partitioning schemes on the performance of distributed query processing. In *IEEE Transactions on Knowledge and Data Engineering, vol. 5, No. 3*, 1993.

[8] W. Litwin, M.-A. Neimat, and D. A. Schneider. LH^*: Linear hashing for distributed files. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD93)*, pages 327–336. ACM Press, 1993.

[9] A. Mondal, M. Kitsuregawa, B. C. Ooi, and K. L. Tan. R-tree-based data migration and self-tuning strategies in shared-nothing spatial databases. In *ACM Proceedings of the 9th international symposium on Advances in Geographic Information Systems (GIS)*, pages 28–33, 2001.

[10] A. Mondal, Yilifu, and M. Kitsuregawa. P2PR-tree: An R-tree-based spatial index for peer-to-peer environments. In *Proceedings of the 1st international workshop on P2P Computing and Databases*, 2004.

[11] B. Nam and A. Sussman. Improving access to multi-dimensional self-describing scientific datasets. In *Proceedings of CCGrid2003: IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society Press, May 2003.

[12] B. Nam and A. Sussman. A comparative study of spatial indexing techniques for multidimensional scientific datasets.

In *Proceedings of 16th International Conference on Scientific and Statistical Database Management (SSDBM)*. IEEE Computer Society Press, June 2004.

[13] A. Rajasekar, M. Wan, and R. Moore. MySRB & SRB – components of a data grid. In *Proceedings of the 11th International Symposium on High Performance Distributed Computing (HPDC-11)*. IEEE Computer Society Press, July 2002.

[14] B. Schnitzer and S. T. Leutenegger. Master-Client R-Trees: A new parallel R-tree architecture. In *Proceedings of the 11th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 68–77, 1999.

[15] S. Song, Y. Kim, and J. Yoo. An enhanced concurrency control scheme for multidimensional index structure. In *IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 1*, pages 97–111, Jan. 2004.

[16] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Unerstanding replication in databases and distributed systems. In *20th International Conference on Distributed Computing Systems (ICDCS2000)*, 2000.