Efficient Query Routing in Distributed Spatial Databases[*]

Roger Zimmermann, Wei-Shinn Ku, and Wei-Cheng Chu
Computer Science Department
University of Southern California
Los Angeles. California 90089
USA
+1-213-740-7654
{rzimmerm, wku, chuw}@usc.edu

ABSTRACT

Spatial databases are prominently used in Geographic Information System (GIS) applications. However, many of the current architectures rely on a centralized data repository. The next evolution will be GIS applications that utilize and integrate a multitude of remotely accessible data sets, for example via Web services. Our involvement in a project where geotechnical borehole information is retrieved from a large number of repositories that are under different administrative control has motivated us to design an efficient distributed access structure and routing middleware for spatial queries. In this study we present our middleware design based on distributed R-tree and Quadtree index structures. Importantly, the framework supports both spatial range and k nearest neighbor queries. We have performed a theoretical analysis and simulations with synthetic and real data sets. The results show a large reduction in message traffic to a level only slightly above what is minimally necessary.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*distributed databases*; H.2.8 [**Database Management**]: Database Application—*spatial databases and GIS*

General Terms

Algorithms

Keywords

Query routing, distributed spatial databases, database middleware

1. INTRODUCTION

The combination of spatial database systems and Web services promises to form the foundation of powerful new applications that dynamically integrate data from multiple, distributed repositories. The use of a Web services infrastructure allows direct programmatic access to remote data and applications have the opportunity to obtain a plethora of data without storing and managing a lot of information locally. There are clearly a number of advantages to this concept. First, data is often maintained by specific entities or organizations. By allowing remote access to their data sets, these entities continue to have administrative control over their data. Second, the correct data set for a specific calculation can be downloaded automatically without manual user intervention, reduced data acquisition time. Third, updates and changes to the data are almost instantly available to remote applications. Here we report on our design of an efficient spatial indexing and query routing middleware where requests are sent only to the specific repositories that most likely have relevant data.

This research has been inspired by our involvement in a project to manage geotechnical borehole information [17, 1]. Local, state and federal agencies, universities and companies need geotechnical information on soil deposits for various applications in civil engineering and urban planning, e.g., mapping of natural hazards, soil liquefaction and earthquake ground motions. For instance, state agencies rely on available geotechnical data for producing hazard maps, from which insurance companies evaluate potential earthquake risks and calculate insurance premiums. We have implemented a distributed Geotechnical Information Management and Exchange (GIME) system based on a spatial database engine to maintain these geotechnical data sets [17].

The geotechnical borehole information is collected and managed by a multitude of private and public agencies, such as the U.S. Geological Survey (USGS), the California Department of Transportation (CalTrans), etc. Given such a federation of spatial database servers, our research focuses on the efficient querying of this distributed infrastructure and we report here on our proposed framework. We introduce a tree-based design, which is built upon distributed and replicated R-tree [4] and Quadtree structures [2, 12] in Section 2. We evaluate our design with simulations using both synthetic and real data and report on the results in Section 3. Section 4 contains a brief discussion and survey of related work. Finally, conclusions and future research directions are presented in Section 5.

2. SYSTEM DESIGN

Our distributed query access middleware is motivated by the design of our Geotechnical Information Management and Exchange

[*] We would like to thank Jean-Pierre Bardet and Jianping Hu for their assistance in conducting the experiments and providing real world data sets. This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0082826 and CMS-0219463, and unrestricted cash/equipment gifts from Intel, Hewlett-Packard and the Lord Foundation.

GIS'04, November 12–13, 2004, Washington, DC, USA.
Copyright 2004 ACM 1-58113-979-9/04/0011 ...$5.00.

(GIME) system. The GIME architecture as shown in Figure 1 is comprised of a number of geographically distributed spatial databases. Access to the data is public and available through a Web services interface. Specifically, the following features and goals have guided our design.

- *Autonomy*: Each of the archives contains data that is maintained by a specific organization, for example the U.S. Geological Survey (USGS). For organizational rather than technical reasons, it is undesirable to replicate or cache the data sets at other participating archives. Data sets may geographically overlap. For example, the USGS maintains data about all of the United States, while the California Geological Survey (CGS) collects data about California.
- *Cooperative and efficient query processing*: When presented with a query at any one of the participating database nodes, the overall system must cooperatively execute the request and return all relevant data. For this purpose, an efficient *access method* is required which can rapidly decide which other nodes contain potentially relevant data and which do not. The query must then be forwarded to the candidate nodes and the result returned expediently to the querying host. Both spatial range queries and k nearest neighbor queries (k-NN) must be supported.
- *Decentralization*: Because the data archives are geographically disbursed, we also expect the query access method and routing mechanism to be fully decentralized. Hence, a centralized indexing system is not considered a suitable solution. Furthermore, an archive that is temporarily unavailable should have minimal impact on the query execution capabilities of the overall system.

Note that our target application involves mostly large organizations, corporations, and educational institutions as service providers. Therefore, we assume a relatively stable environment[1]. Nonetheless, the system must be able to gracefully handle volatility.

We have designed an efficient distributed access method that fulfills the requirements and goals described above, based on replicating well-known spatial index structures. Before we detail our proposed techniques that leverage R-tree and Quadtree access methods, we will briefly introduce a baseline algorithm for comparison purposes.

2.1 Baseline: Exhaustive Query Routing

We assume a non-volatile environment where occasionally, but not very frequently, a node leaves or joins. The data sets that we consider are large and valuable, and hence they are usually professionally managed. As a result, we can compile a list of all the participating archives. This list may not be completely up-to-date at a specific time instance, but accurate enough to result in few disruptions.

If the list of database servers is known, it can be distributed to every archive. A query q that arrives at a specific node can then be forwarded to all other nodes for exhaustive processing, irrespective of the selectivity of q. We call this naive method *exhaustive query routing* (EQR). Even though EQR is inefficient, it is useful as a baseline mechanism to compare our more sophisticated models against.

The metric that we use to compare the different techniques is the total number of messages created in the system to execute a query q and collect the results. A lower number of messages reduces network traffic and indicates better scalability of the system. The number of messages generated by queries with EQR can be represented as shown in Equation 1.

$$M = 2 \times Q \times (N - 1) \qquad (1)$$

The overall number of messages M is the product of the total number of queries Q and the number of nodes N in the system. The total is doubled because an equal number of result messages are generated.

2.2 Query Routing with Spatial Indexing

The naive flooding method of EQR generates a lot of message traffic and scalability is poor. However, many established access methods exist that reduce the query space. The core idea of most access methods is to recursively partition the key space into a set of equivalence classes. An index is then constructed as a hierarchy of the class representatives at successive levels. The hierarchical index allows filtering out (i.e., dismiss without inspection) the irrelevant classes while the query is directed from the root of the hierarchy toward the similarity class of the query tree.

For spatial (and multi-dimensional) data indexing, the R-tree [4] and Quadtree algorithms [2, 12] are well established. Both generate tree-structure indices that partition the overall space into successively smaller areas at lower levels of the index hierarchy. R-trees and Quadtrees are very successfully used in the core engines of spatial database systems. We propose to use them in a novel way as index structures across multiple spatial databases to decrease the query forwarding traffic. Specifically, we insert the minimum bounding rectangle (MBR) of the data set of each archive into a global R-tree or Quadtree. Because we prefer to avoid a centralized index server we further distribute copies of this global index structure to each archive. As a result, an archive can intersect each query rectangle with the archive MBRs stored in the global index. The query is then only forwarded to archives whose MBR overlaps with query rectangle, immediately reducing inter-node message traffic significantly.

Very sparse and widely distributed data sets may be enclosed with large MBRs that result in intersections with query rectangles that produce zero matched and retrieved data objects (false positives). In this scenario, a physical archive can be decomposed into several clustered, logical data sets and hence mapped to a number of smaller, logical MBRs. Any query window which falls into the original MBR but does not intersect with any logical MBRs can be dismissed.

An additional cost is incurred with the above design because it requires the global index structures to be synchronized. Furthermore, the complexity of keeping a distributed data structure consistent is added. However, one characteristic of this technique greatly reduces the overhead and makes it an attractive solution. Because the global index structures manage bounding rectangles, changes to the data set of any individual archive only result in index updates if the MBR changes – and this is very infrequent. Consider the following example. An archive manages 1,000 two-dimensional spatial data objects. The MBR is defined by at most four of them[2]. Any insertion or deletion of data objects confined within the MBR do not affect the global index; only changes that either stretch or shrink the MBR need to be propagated. Equation 2 shows the estimation function of the number of messages when a global index is used.

[1] A more dynamic environment would generally be expected for peer-to-peer systems.

[2] More than four points may define a rectangle if some of them have the exact same x- or y-coordinate values.

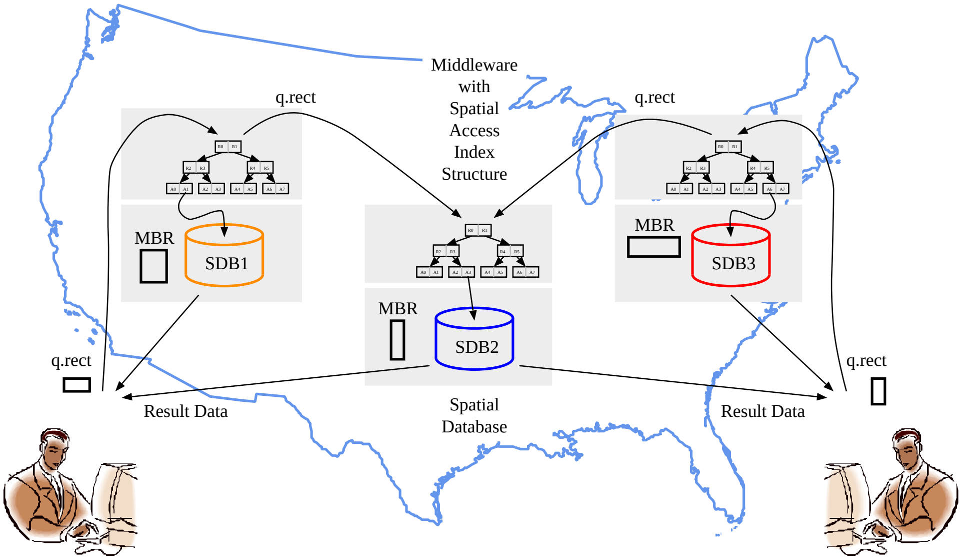

Figure 1: The proposed distributed spatial database infrastructure with middleware utilizing replicated spatial index structures (either R-trees or Quadtrees).

Symbol	Description
Q	Number of queries
U	Number of global index updates
N	Number of nodes
M	Number of messages
M_T	Total number of messages
M_Q	Number of messages generated by queries
M_U	Number of messages generated by updates
S_Q	Selectivity of queries
S_U	Selectivity of global index updates

Table 1: Parameters used to describe the query message traffic models.

$$M_T = [2 \times Q \times (N-1) \times S_Q] + [U \times (N-1) \times S_U] \quad (2)$$

The query traffic is reduced by a factor equal to the selectivity of the query, $0 \le S_Q \le 1$. Similarly, the update message traffic U is diminished by the factor $0 \le S_U \le 1$ that describes how many of the data updates (including insertions and deletions) actually result in global index changes. Consequently, the total number of messages is dependent on the frequency of data updates in the system. Table 2 shows an example of S_U values ranging from 0.001 to 0.058 with one of our experiments. Recall that the exhaustive query routing technique has no such dependency. As we will show in Section 3, global indexing generally results in a significant reduction of message traffic with the benefits being highest when few updates must be propagated. Next we will describe our R-tree and Quadtree global indexing techniques.

2.2.1 *Common Design Assumptions*

Every archive in the distributed environment hosts a database engine capable of storing, retrieving and querying spatial data. Furthermore, every archive holds a directory, termed the *server list*, with entries denoting the network location (e.g., IP address) and the minimum bounding rectangle (MBR) of every known spatial database member in the system. From the directory information, each server computes the corresponding R-tree (or Quadtree) global index data structure. If any MBR update messages are received, the local data structure is updated accordingly. Any local MBR changes due to data insertions or deletions initiate update messages to all the other database servers. In the R-tree based design, the upper bound and lower bound of the number of entries that will fit in one internal tree node are prespecified (e.g., M is the maximum number of entries that will fit in one node and $m \le M/2$ is the parameter specifying the minimum number of entries in a node).

2.2.2 *The R-Tree Based Design*

Index Initialization and Topology Maintenance. A new server A joins the spatial database collective by sending its information (IP address and MBR) as an update message to one of the existing servers, say B. Archive B updates its local R-tree index and replies with the current system information (i.e., all known server hostnames and MBRs). With this information A constructs its own R-tree index. In a final step, A broadcasts an update message with its hostname and MBR to all the other servers (except B), which in turn update their local R-tree indices according to the R-tree update algorithm [4].

In the event of an archive departure, it broadcasts an update message to announce that it is removing itself from the topology. Correspondingly, the other servers delete the leaving server's MBR from their R-tree. If an archive fails, the node that first detects the unresponsive system broadcasts a removal message to everyone. In the outlined protocol short term inconsistencies among the R-tree data structures on different servers can arise. In the described application this is tolerable. However, if stronger update semantics are desired, then existing algorithms to perform distributed updates can be used.

Query Routing. Clients do not need to contact all the servers in a distributed spatial database environment to obtain comprehensive query results. Queries sent to a server will automatically be forwarded and yield accurate spatial results from the complete data

Number of	R-tree Based Design				Quadtree Based Design			
Servers	Updates	MBR	Index Structure	S_U	Updates	MBR	Index Structure	S_U
		Changes	Changes			Changes	Changes	
10	1000	50	50	0.05	1000	50	5	0.005
100	1000	55	55	0.055	1000	55	4	0.004
200	1000	52	52	0.052	1000	52	5	0.005
300	1000	58	58	0.058	1000	58	2	0.002
400	1000	43	43	0.043	1000	43	4	0.004
500	1000	55	55	0.055	1000	55	1	0.001
600	1000	42	42	0.042	1000	42	2	0.002
700	1000	58	58	0.058	1000	58	1	0.001
800	1000	54	54	0.054	1000	54	2	0.002
900	1000	50	50	0.050	1000	50	3	0.003
1000	1000	52	52	0.052	1000	52	1	0.001

Table 2: The relationship between data updates, MBR updates, and index updates.

Number of	R-tree Based Design				Quadtree Based Design			
Servers	Updates	MBR	Index Structure	S_U	Updates	MBR	Index Structure	S_U
		Changes	Changes			Changes	Changes	
10	1000	45	45	0.045	1000	45	3	0.003
100	10000	398	398	0.0398	1000	398	20	0.002
200	20000	806	806	0.0403	1000	806	54	0.0027
300	30000	1207	1207	0.0402	1000	1207	65	0.0022
400	40000	1582	1582	0.0396	1000	1582	91	0.0023
500	50000	2008	2008	0.0402	1000	2008	101	0.0020
600	60000	2457	2457	0.0410	1000	2457	143	0.0024
700	70000	2788	2788	0.0398	1000	2788	162	0.0023
800	80000	3359	3359	0.0420	1000	3359	186	0.0023
900	90000	3644	3644	0.0405	1000	3644	169	0.0019
1000	100000	4083	4083	0.0408	1000	4083	207	0.0021

Table 3: The relationship between data updates, MBR updates, and index updates with a linear increase of the server and update numbers.

set. The queried server determines through its local R-tree whether any of the other archives in the collective potentially have relevant data (i.e., the query rectangle and the archives' MBR intersect). Forwarded queries are flagged to show that they originated from a server rather than a client to avoid query loops. The results of forwarded queries are returned to the initially contacted server which aggregates them and returns the set to the client.

R-Tree Index Update. Each spatial database server must also process data object update requests from local users. We categorize update requests into (1) data insertions and (2) data deletions. If an object insertion or deletion does not change the server MBR, then the local R-tree index remains structurally unchanged. On the other hand, if the insertion or deletion results in a variation of the MBR boundary, then the local R-tree is updated [4]. Subsequently the new MBR is broadcast to all the other servers in the system for tree index synchronization.

2.2.3 The Quadtree Based Design

Quadtree Index Update. The operation of the Quadtree based design is very similar to the R-tree based design. However, some slight differences for tree index updates arise as follows. If an object insertion or deletion results in changes to the MBR boundary, the Quadtree model checks whether the MBR variation affects the Quadtree *structure*. If the structure is changed, then the update is propagated to all the other servers as usual for tree index synchronization.

However, if the Quadtree structure is unchanged, then the MBR update is not broadcast. The reason is that the Quadtree structure determines the routing of queries and in this case the correct results are still achieved. Table 2 illustrates this effect. In our simulation environment approximately 4.2% to 5.8% of all insert or delete operations result in an MBR change. For the R-tree based design, every MBR change translates to an index structure update. However, in the Quadtree design the index structure updates are reduced by an order of magnitude. Hence we can conclude that the update message traffic to synchronize distributed Quadtrees is much lower than for R-trees. Table 3 shows the experimental results obtained when increasing both the server and update numbers linearly. This illustrates the case where the activity per server is relatively constant. Again, most updates do not generate any changes to the archive MBRs and therefore the selectivity S_U of the global index remains a fixed fraction of the number of updates. Based on these results, we expect our techniques to scale well.

2.2.4 Nearest Neighbor Queries

In geotechnical data applications, nearest neighbor (NN) queries are very important. NN queries with spatial data have been investigated in previous research [15, 8, 14]. The most relevant work that applies to our system is a branch-and-bound R-tree traversal algorithm that efficiently answers both NN and k-NN queries [10]. We slightly modified the R-tree traversal algorithm to adapt it to our distributed spatial database environment.

In the branch-and-bound R-tree traversal algorithm two metrics, the minimum distance (MINDIST) and the minimum of the maximum possible distance (MINMAXDIST), are proposed for ordering the NN search. The MINDIST describes the minimum Euclidean distance between the query point and the nearest edge of an MBR; it is the optimistic choice (see Figure 2). The MINMAXDIST value defines the minimum of all the maximum distances between the query point and points on each of the axes of an MBR and it is the pessimistic option (Figure 2). Because of the characteristics of MBRs, the MINDIST metric cannot always provide the optimum tree traversals, and therefore MINMAXDIST is indispensable. The NN search algorithm implements an ordered depth first traversal based on the values of MINDIST and MINMAXDIST. It begins from the R-tree root node and proceeds down the tree hierarchy. At a leaf node, a distance computation function is invoked to decide the actual distance between the query point and the candidate DB objects. The algorithm iterates with three search-pruning strategies until it finds the NN object.

In the distributed design, every server maintains a local R-tree and the NN search algorithm is executed on this local R-tree to compute both the MINDIST and MINMAXDIST values. To access these distance values across multiple archives, we have created a Web service interface at each node to remotely obtain the distance between the search point and a candidate nearest data point. To answer a NN query, a server needs to send several distance query messages to other servers in the system during the branch-and-bound process. With the three search pruning strategies proposed in [10] and a slightly modified search algorithm, NN queries can be efficiently executed.

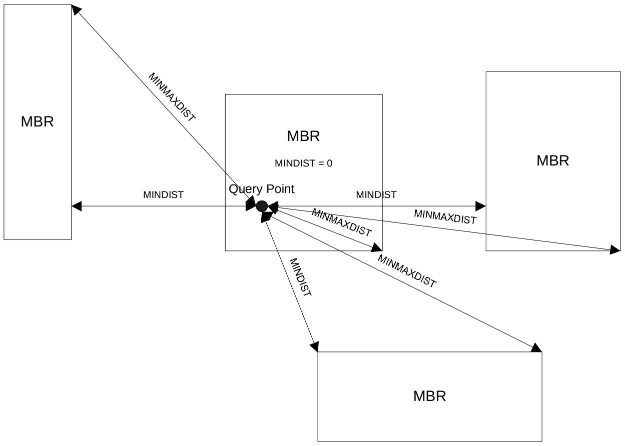

Figure 2: The MINMAXDIST and MINDIST distances in two dimensional space.

3. EXPERIMENTAL VALIDATION

We implemented our tree-based design in a simulator to evaluate the performance of our approach. For the index structures, we used the R-tree and MX-CIF Quadtree algorithms [6, 13]. Hence the index tree search complexity is the same as with these algorithms. In a distributed environment, the search complexity is dominated by the communication overhead between servers. Therefore, the focus of our simulation is on quantifying the query routing traffic generated by processing a sequence of spatial range queries and updates. We performed our experiments with both synthetic and real-world spatial data sets.

3.1 Simulator Implementation

We implemented our middleware in the simulator with plug-in modules for both the R-tree and the Quadtree algorithms. Either structure was used to index all the MBRs of the servers present in the distributed system. The leaf nodes represent specific server MBRs and contain forwarding pointers (i.e., the host names and IP addresses) to the remote servers. The leaf node of the MBR of the local data set directly points to the local database. If a query window intersects with several server MBRs, then the query is forwarded to each. The simulator counts all the messages that are generated through the query forwarding mechanism and also accounts for all the return messages containing result data sets. Additionally, tree update information must be broadcast to all servers. Recall that the tree index update frequency of the Quadtree based design is much lower as compared with the R-tree implementation (as discussed in Section 2).

3.1.1 Event Generation.

The simulator was designed to process two types of user events: (1) queries and (2) updates. Both types were generated according to a Poisson distribution, with the inter-arrival rate λ_Q and λ_U being specified independently. For example, one might specify five query requests and one update request per minute. Data updates could either be insertion or deletion requests. The total simulated time for all our experiments was ten hours (for parameters see Table 4). To enable direct performance comparisons, all techniques (EQR, R-tree and Quadtree) were executed with the exact same sequence of events for a specific experiment.

3.1.2 Query Parameter Generation.

Each query was dynamically created based on the mean query window size (*QWS*-μ) and its deviation *QWS*-σ (see Table 4). The *QWS*-μ parameter defined the mean percentage of the global geographical area that was used for the query window based on a normal distribution. In addition, *QWS*-σ provided a variation range bound by one *QWS*-σ deviation such that the query window area was different for each query event. With these parameters, the simulator first chose the query window size. It then randomly selected a point (x_1, y_1) as one corner coordinate and a value x_2 inside the global boundary as the x-value of the other coordinate across the diagonal of the query window. Based on the window size, the value of y_2 was calculated and hence the final position of the query window determined.

3.1.3 Synthetic Data Generation.

Each borehole data item has a spatial location attribute with longitude and latitude, which map it to a point in a two dimensional space. We created a synthetic data set by randomly generating N data center points, C_0, C_1, ..., C_{N-1}, where each C_i = (x_i, y_i) is the geographical center of all the borehole data managed by an individual spatial database server. All the data centers were located inside a global boundary, which encloses the geotechnical data stored in the system. For each data center location C_i, B associated boreholes p_j are generated according to a normal distribution. Consequently, the borehole points are more dense in the vicinity of the center point and become more and more sparse as the distance to the center point increases. The generator limited the maximal distance of a borehole from its center to the value of two standard deviations. After all the borehole points were created, the MBR of each database server was computed. Figure 3 illustrates the boreholes managed by ten servers and their respective MBRs.

3.2 Experiments

The simulator executed each event as it was generated. For each query event, a server was randomly choosen as the injection point to receive the query from a client. Update events were randomly decided to be either insertions or deletions. For an insertion event, the

Parameter	Value	Description
N	1,000	The number of database servers in the system
B	200	The number of boreholes managed by each server
σ	2	The maximum distance from a data center to it's borehole points
λ_Q	5	The number of queries per ten minutes
λ_U	2	The number of updates per ten minutes
T_{exec}	10 hours	The length of a simulation run
QWS-μ	10%	Query window size: the percentage of the whole system area, which is occupied by a query window
QWS-σ	2	The standard deviation for generating the query window size range

Table 4: Parameters for the simulation environment.

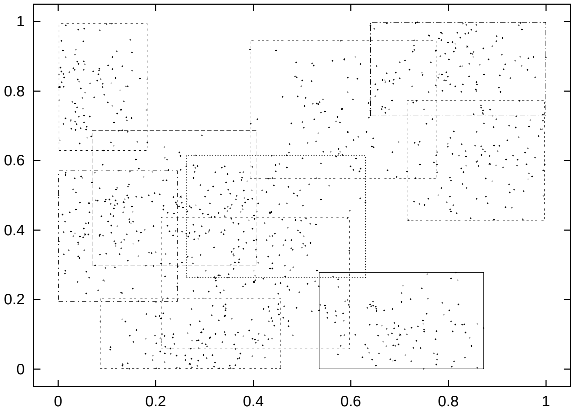

Figure 3: The synthetic server data centers, their MBRs, and the individual borehole data sets.

simulator randomly chose a server which received the new borehole point. The position of the new borehole was generated as described in the 'Synthetic Data Generation' paragraph, but its distance from the server center was bound by four standard deviations instead of two. Therefore, the MBR of each server could expand. For a deletion event, the simulator simply deleted a randomly selected borehole point from a randomly chosen server. If the deletion resulted in a tree index change (i.e., the MBR shrinks), the simulator counted its related update synchronization traffic cost.

We also created a simulation module to analyze the message traffic generated by the exhaustive query routing (EQR) mechanism with the same event sequence. Consequently, we were able to measure the performance differences between the two approaches.

3.2.1 Synthetic Data Experiment

We performed simulations with the R-tree, Quadtree and exhaustive query routing designs based on our synthetic data sets. The same event sequence was executed over a simulation period of ten hours with the parameter set shown in Table 4. Figure 4 shows the accumulated traffic of queries and updates of the tree-based designs and the exhaustive query routing mechanism.

3.2.2 Kobe Data Experiment

To verify the efficiency of our design with real-world data, we ran our simulation with geotechnical data provided by Kobe University, Japan. The data set includes four thousand boreholes of Kobe county, and we will refer to it as the Kobe data set.

We used the popular K-means algorithm [7] to cluster the Kobe data points in Euclidean space and assign them to database servers. We divided the Kobe data set into ten clusters (illustrated in Figure 5). MBRs were generated based on all the borehole coordinates in each cluster.

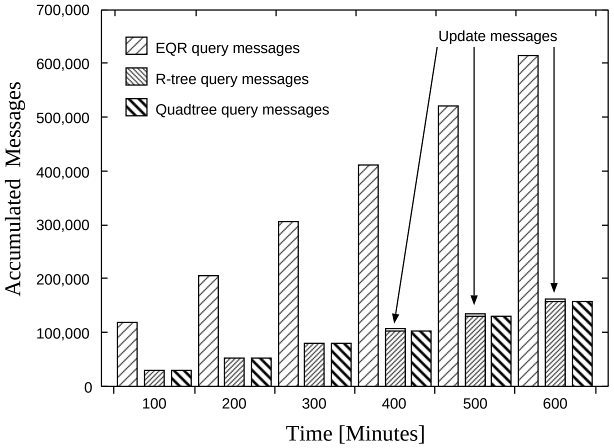

Figure 4: A comparison of the query message traffic between the exhaustive query routing mechanism and the tree-based designs over a ten hour period. The update message traffic is negligible.

We experimented with both the R-tree and Quadtree index structures with different query window sizes (ranging from 1% to 50%). For comparison purposes, we also generated a synthetic data set with the same parameters (10 servers, 400 boreholes per server) and the simulation results are shown in Figure 6. We discuss the performance results next.

3.3 Discussion of Results

Figures 4 and 6 illustrates that our design improves the query routing performance significantly with both synthetic and real-world data sets. The tree-based designs result in a decrease of approximately 60% to 70% of inter-server message traffic compared with exhaustive query routing (with query window sizes of 10% to 20%). Since the EQR scheme represents the worst case message routing traffic we also defined the *optimum query routing* (OQR) traffic representing the best case. With OQR, a query that arrives at a specific server is forwarded to precisely the set of servers which maintain relevant data. Therefore, OQR is the optimal baseline against which we can compare the performance of our designs. Figure 7 illustrates the relationship between EQR (upper bound), the two tree-based designs, and OQR (lower bound) with different query window sizes. We divided the accumulated message count

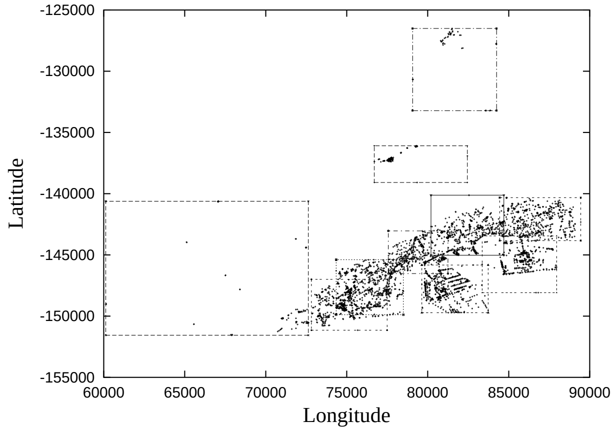

Figure 5: The borehole clusters of Kobe county, Japan (this figure uses the Japanese coordinate system).

of EQR and the tree-based designs by the message count of OQR to normalize the y-scale. Hence a value of one indicates optimal performance. Remarkably, the traffic generated by our tree-based design is very close to this optimum.

Symbol	Description
$NPIR$	Network performance improvement rate
T_{EQR}	Total of accumulated messages for EQR
T_{tree}	Total of accumulated messages for the tree based designs

Table 5: Parameters used to describe the network performance improvement rate.

Next we defined a comparison metric termed the *network performance improvement rate* ($NPIR$) shown in Equation 3.

$$NPIR = \frac{T_{EQR} - T_{tree}}{T_{EQR}} \qquad (3)$$

Figure 8 illustrates the results of our scalability experiments where we increased the number of servers from ten to one thousand while keeping the other parameters constant. As illustrated, the $NPIR$ stays remarkably constant over the full spectrum of configurations and we conclude that the tree based designs scale well to large distributed systems.

The query window size affects the performance of our design since with large window sizes the query must be forwarded to almost all servers and less traffic pruning can be achieved. Figure 9 shows how the $NPIR$ declines when the query window size increases from 1% to 50%. Additionally, the overall data arrangement influences to the performance of systems with tree-based query routing. The best condition exists when there is no overlap between any server MBRs and conversely the worst scenario consists of significant MBR overlap. Under good conditions the tree-based designs can reduce inter-server traffic by up to 90%. In the worst case, the performance decreases to the same level or slighly worse (because of the update costs) than ERQ. Therefore, a system designer needs to consider the charactristics of the data set before opting for the tree-based query routing algorithms.

4. RELATED WORK

Recently, considerable attention has focused on an emerging class of large scale distributed data management systems classified as

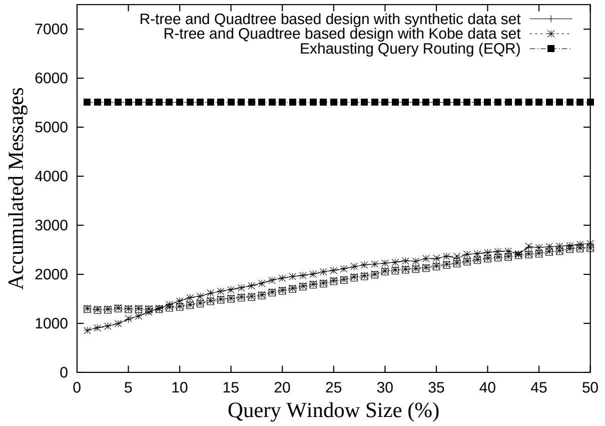

Figure 6: Performance comparison between the tree-based design and exhaustive query routing with the synthetic and Kobe data sets.

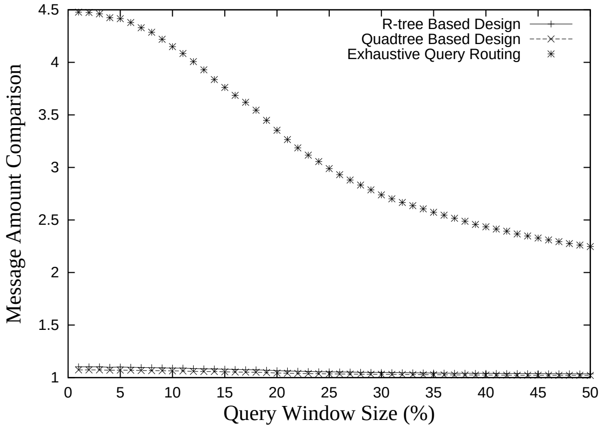

Figure 7: Message count comparison between exhaustive query routing (EQR), the tree-based designs, and optimum query routing (OQR).

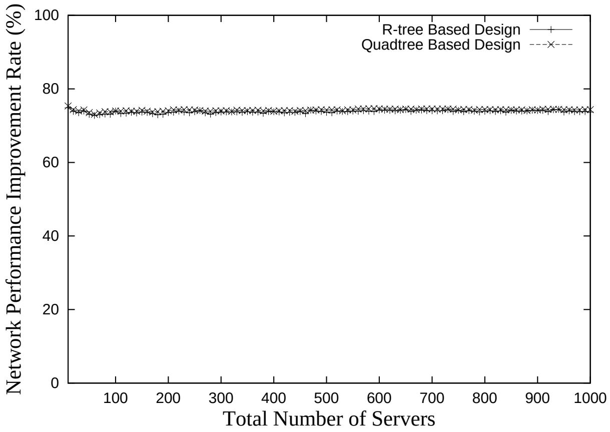

Figure 8: The performance of the tree-based designs remains stable when the number of servers increases from ten to one thousand.

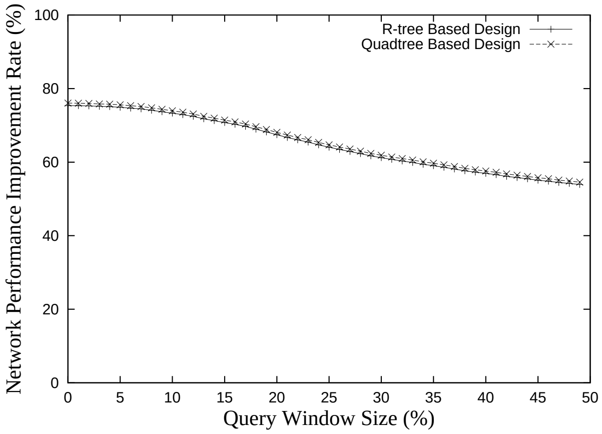

Figure 9: The network performance improvement rate (NPIR) as a function of the query window size.

peer-to-peer (P2P) systems. Some of the key characteristics of P2P systems are their very dynamic topology, their heterogeneity, and their self-organization. The query processing and routing approaches of some of the initial P2P systems focused on either a centralized index server (e.g., Napster) or a flooding mechanism (e.g., Gnutella). These techniques are considered either not very scalable or inefficient. More recently, distributed hash tables (DHT) have been proposed to achieve massive scalability and efficient query forwarding. The most prominent representatives are Pasty [11], Chord [16], and CAN [9]. DHTs provide a mechanism to perform object location within a potentially very large overlay network of nodes connected to the Internet. The distributed hash mechanisms work by transforming a key value into a number that is then mapped to a node whose identifier is numerically closest to the key. Key location is very efficient and the expected number of routing steps is $O(\log N)$, where N is the number of nodes in the system. An important property of a good hash function (e.g., SHA-1) is a close to uniformly random distribution of keys to numeric values. Hence, key values that are close to each other (or otherwise related) can be arbitrarily far apart in the generated index space. This property makes standard DHTs unsuitable for range queries.

A number of techniques have been proposed to adapt DHT mechanisms for range queries. Harwood and Tanin [5] introduce a method to hash spatial content over P2P networks. Space is divided in a Quadtree-like manner and the central points of each square, denoted *control points*, are hashed to a Chord ring. Spatial objects and queries are resolved to spatial regions whose control points are then hashed onto the DHT ring. Galanis et al. [3] propose a distributed catalog service that can efficiently locate XML path data. Range queries are supported via wildcards in XML strings (i.e., “*”), however, they may require a scan of some of the data.

5. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

We have presented an architecture to efficiently route and execute spatial queries based either one of two globally distributed and replicated index structures, namely R-trees and Quadtrees. We have performed extensive simulations with both synthetic and real data sets and several major results could be observed. First, the update message traffic to keep the replicated indices synchronized is negligible. This reduces the probability of temporarily inconsistent index structures and greatly increases scalability. Second, the overall query message traffic is significantly reduced to a level only slightly higher than what an optimal distribution algorithm with global knowledge could achieve.

We plan to extend our work as follows. The current performance metric – the number of messages – does not capture the parallelism that is achieved within the system. Hence we intend to measure the response time and the query throughput. For this purpose, we plan to integrate the tree-based design into our current GIME architecture and perform experiments with this real system.

6. REFERENCES

[1] Jean-Pierre Bardet, Roger Zimmermann, Wei-Shinn Ku, and Jianping Hu. Web Services for Exchange and Utilization of Geotechnical Information. *Computers & Geosciences*, 2004. Currently under review.

[2] R.A. Finkel and J.L. Bentley. Quadtree: A data structure for retrieval on composite keys. *ACTA Informatica*, 4(1):1–9, 1974.

[3] Leonidas Galanis, Yuan Wang, Shawn R. Jeffery, and David J. DeWitt. Locating Data Sources in Large Distributed Systems. In *Proceedings of the 29th International Conference on Very Large Databases*, Berlin, Germany, September 9-12, 2003.

[4] Antomn Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, Massachusetts, June 18-21, 1984.

[5] Aaron Harwood and Egemen Tanin. Hashing Spatial Content over Peer-to-Peer Networks. In *Australian Telecommunications, Networks and Applications Conference (ATNAC)*, Melbourne, Australia, December 8-10, 2003.

[6] G. Kedem. The quad-CIF tree: A data structure for hierarchical on-line algorithms. In *Proceedings of the* 19^{th} *Design Automation Conference*, pages 352–357, June 1982.

[7] J.B. McQueen. Some methods of classification and analysis of multivariate observations. In *5th Berkeley Symposium in Mathematics, Statistics and Probability*, pages 281–297, 1967.

[8] A. Papadopoulos and Y. Manolopoulos. Performance of nearest neighbor queries in r-trees. In *6th International Conference on Database Theory*, pages 394–408, 1997.

[9] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM*, San Diego, CA, August 27-31, 2001.

[10] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 71–79, 1995.

[11] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[12] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[13] Hanan Samet, editor. *The Design and Analysis of Spatial Data Structures*, pages 200–211. Addison-Wesley, 1990.

[14] Cyrus Shahabi, Mohammad R. Kolahdouzan, and Mehdi Sharifzadeh. A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases. In *Proceedings of the Tenth ACM International Symposium on Advances in Geographic Information Systems*, McLean, Virginia, November 2002.

[15] R.L. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. In *Algorithmica*, 6, 1991.

[16] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160. ACM Press, 2001.

[17] Roger Zimmermann, Jean-Pierre Bardet, Wei-Shinn Ku, Jianping Hu, and Jennifer Swift. Design of a Geotechnical Information Architecture Using Web Services. In *Proceedings of the Seventh World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2003)*, Orlando, Florida, July 27–30, 2003.