Scoped and Approximate Queries in a Relational Grid Information Service

Dong Lu Peter A. Dinda Jason A. Skicewicz
{donglu,pdinda,jskitz}@cs.northwestern.edu
Department of Computer Science, Northwestern University

Abstract

We are developing a grid information service, RGIS, that is based on the relational data model. RGIS supports complex queries written in SQL that search for compositions (using joins) of resources. For example, we might ask it to find a Linux cluster with a certain bisection bandwidth and total memory. Such queries can be expensive to execute, however, and so we have developed several approaches that leverage our GIS schema to let us trade off between the number of results returned and the execution time. In this paper, we describe two of them: scoped queries and approximate queries. Scoped queries constrain search to a network neighborhood, returning all matching results in the neighborhood. Approximate queries reduce the number of joins done by replacing collective constraints with constraints on individual resources, returning a subset of all the possible results in the grid. Scoping, approximation, and nondeterminism (described elsewhere), can be combined. In this paper, we describe scoped and approximate queries, how they are implemented, and present performance evaluations for two examples. The evaluation suggests that scoping and approximation can greatly reduce query times while still returning a useful number of results.

1 Introduction

Grid technologies strive to enable large-scale sharing of computing resources and services. A central role is that of Grid Information Services (GIS), which provide resource discovery and monitoring. GISes store information about the resources available within a wide area distributed computing environment, such as hosts, clusters, switches, routers, links, services, sensors, available software, and services. A GIS consists of a set of objects representing the resources and their relationships, and programs that query and update the objects. Each object has a unique identifier, a timestamp, and a set of attributes. The GIS makes updates (inserts, deletes, and changes of objects) available to queries as soon as possible. It also manages access to the objects, making sure that they are updated and read by authorized users only. As grids become larger and more powerful, the number of objects grows, as does the complexity of the queries over them. A more detailed description of this view of a GIS is available elsewhere [9].

We are developing a distributed GIS system, RGIS, that is based on the relational data model. RGIS servers are implemented on top of a RDBMS and use SQL as their query language. They push updates to each other using a content delivery network. The RGIS schema focuses on modeling the hardware and software resources of a distributed computing environment. While most computational grids today are relatively small, we intend RGIS to scale to very large grids, and possibly even to the scale of the Internet. An in-depth description of the merits of a relational approach to GIS systems is available elsewhere [2], as is a more detailed description of the system [3].

A powerful feature of RGIS is that users can write queries in SQL to search for complex compositions of resources. However, these queries can be extremely expensive to execute because they often involve large numbers of joins. In response, we have introduced the concept of nondeterministic queries [3], which allow the user (and RGIS) to trade off between the running time of a query (and the load it places on an RGIS server) and the number of results returned. The result set returned is a random subset of the full result set.

In this paper, we describe two additional techniques for making complex RGIS queries fast: scoped queries and approximate queries. In a scoped query, the query, with all its joins, is limited to a neighborhood in the network, exploiting the network topology captured in the RGIS schema. In approximate queries, the number of joins is reduced by replacing them with constraints on individual objects and the simplified query is run against the entire network. In both cases a deterministic subset of the full result set is returned. In essence, both involve a more tightly constrained version of the original query. Scoping, approximation, and nonde-

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

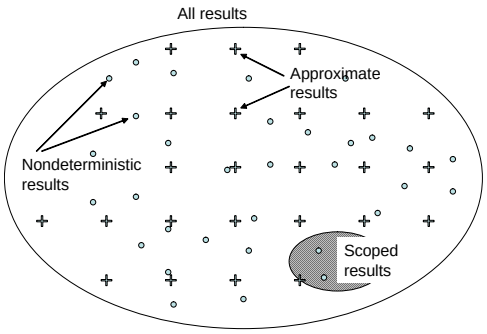

Figure 1. Schematic of query semantics.

terminism are orthogonal and can be combined. Figure 1 schematically illustrates the nature of these techniques.

Notice that with each technique it is possible that the query returns no results although results do exist for the full query. Scoped and approximate queries will always return the same results, while the nondeterministic query returns different results each time with high probability. Of course, any query involving scoping can also be iterated with randomly chosen scopes. Scoping is easy to implement, either by the system or the user, since it involves only the addition of constraints to the given query. Nondeterminism is relatively straightforward to implement by equijoining additional tables and placing constraints on values on those tables. Approximation involves a much more complex query rewrite because aggregate constraints must be translated into constraints on individual objects. However, because it eliminates joins, it has very dramatic effects on query time. RGIS uses methods as appropriate. RGIS also can hard-limit the running time of any query simply by killing it.

In the following, we first briefly describe the RGIS architecture and work related to it. Then we describe scoped and approximate queries in greater detail. Finally, we evaluate the performance of scoped, approximate, and scoped+approximate queries on two example queries.

2 Related work

The grid computing community has seen an explosion of work on GIS systems. The most relevant systems to this work are Globus MDS2 [1], the Condor Matchmaker [11], Redline [7], and R-GMA [6].

MDS2 is based on LDAP and defines a schema (the attribute types) that can be associated with nodes in an LDAP tree. Queries are scoped to subtrees. In contrast, RGIS uses a relational data model with scoping possible with respect to the network topology that is modeled.

The Condor Matchmaker provides bi-partite matching of requests and offers, and was recently extended to support gang-matching [12] or more complex queries. Recently, Liu and Foster have proposed a matchmaking scheme and developed a system, Redline, in which the language for constraints enables the definition of constraint satisfaction problems (CSPs) [7].

R-GMA [6] is closest to our work in that it also proposes a relational data model for GIS systems. However, R-GMA currently focuses on dynamic properties of resources (e.g., load), while RGIS focuses currently on relatively static properties (e.g., memory). Both systems are evolving to unify static and dynamic information, however. Our second difference is RGIS's support for nondeterministic, scoped, and approximate queries, as described here and in an earlier paper [3].

Interestingly, by enabling what we call compositional queries, the Condor Matchmaker with gang-matching, Redline, R-GMA, and RGIS run into the same problem: the exploding cost of query execution. Each system deals with this problem in a different, heuristic manner. This paper describes and evaluates scoping and approximation, two of RGIS's approaches to this problem.

3 RGIS system architecture

An RGIS server is built around an RDBMS system. At the present time, we use Oracle 9i Enterprise Edition, but our system could also be based on other RDBMS systems. RGIS includes a type system to identify a wide range of components including hosts, routers, switches and hubs at layers 2 and 1, links at layers 3 through 1, paths at layer 3, benchmarks, operating systems, operating system vendors and versions, switches, switch vendors, software modules, running software, and communication endpoints. The RGIS schema includes the sequences, tables, constraints, triggers, and indices that represent our grid modeling efforts. Given transactional updates, the constraints and triggers are designed to keep the database in a consistent state.

Layered on top of the RDBMS front-end is a query manager/rewriter, and an update manager, which provide the core interfaces to the system. The goal of the query manager/rewriter is to shape the query workload so that it can be effectively executed by the RDBMS. It is here that nondeterministic, scoped, and approximate queries are implemented.

An RGIS server serves a site and can be scaled by using more powerful implementations of the underlying RDBMS and the hardware it runs on. A site's RGIS server is responsible for the query workload produced by the site and for maintaining a replica of information about friendly remote sites so that queries can also be posed against those sites. To make this this possible, each RGIS server pushes locally occurring updates to friendly remote RGIS servers, providing weak consistency among mutually friendly servers. A

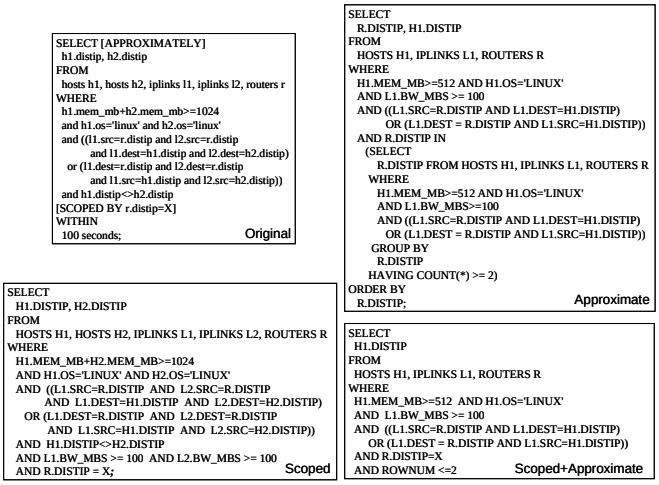

Figure 2. Cluster finder query and its implementation as a scoped, approximate, and scoped approximate queries.

more detailed description of the distributed RGIS system architecture is available [3].

Like nondeterministic queries, scoped queries return a subset of all possible results. We also exploit the network structure by approximating large joins with complex constraints with smaller joins and simpler constraints. Approximate queries return a set of results that overlaps with the set of all possible results. The remainder of this paper focuses on scoped and approximated queries.

4 Scoped and approximate queries

Parallel and distributed applications are often not interested in individual resources *per se*, but rather in compositions of them. For example, suppose a data parallel program has been compiled to run on two processors. At startup, it will want to ask questions such as "find me a Linux cluster of two machines with a total memory of at least 1 GB, all with at least 100 mbit links to a common router". Such questions can be readily posed to RGIS using SQL SELECT statements, as can be seen by the upper left query shown in Figure 2. This *cluster finder query* answers the above question and it can be mechanically extended to search for N-host clusters.

In general, SQL lets the application or user combine multiple resources in arbitrary ways because it essentially provides set (bag) operations. Unfortunately, such queries can be very expensive to execute. In the worst case, the query cost can grow exponentially with the number of joins. Notice that the two-host cluster finder query involves a five-way join. In general, an N-host version of the query would involve $(2N + 1)$-way join. Not only must individual queries execute in short periods of time, an RGIS server must also be able to handle the query workload of a whole site. If we support such queries directly, we would quickly start to disappoint users and would also overload the RGIS server.

Scoped queries

It is a common misconception that, unlike hierarchical data models, it is impossible to scope queries in the relational data model. In actuality, it is schema-dependent. Because the RGIS schema models the network, it is possible to scope queries with respect to the network of the grid, either by prefix-matching against IP addresses or by rooting the query at a router or switch. As shown in the query in the lower left of Figure 2, scoping is achieved by constraining the choice of router to one particular router.

A scoped query may return no results if the router does not have a set of machines that meets the constraints. The user may issue the query iteratively with a list of routers known to him, or have the system issue it repeatedly with randomly chosen routers until a match is found.

Approximate queries

The main idea behind approximate queries is to minimize the number of joins by rewriting SQL queries with tighter constraints which return a result set that is a deterministic subset of the full result set. Consider an example query in which we seek to find N machines with total memory greater than or equal to m MB. In this simple example the original SQL query will execute a N way join. A valid approximation of this query is to find N machines each with memory greater than or equal to m/N MB. The approximation is valid in that those N machines must also be a row of the result set of the original query. It is an approximation in that the constraints are stronger. A solution in which $N/2$ machines have $4m/3N$ MB and $N/2$ have $2m/3N$ would be found by the original query but not by the approximate query. Also note that we must know the semantics of the particular resource (hosts and their memory) to know whether the transformed query makes sense. The upshot is that we can quickly compute a result set that is a deterministic subset of that of the original query without any join operations. This makes the query much faster and improves the scalability of the system.

The upper right hand query in Figure 2 shows an example of an approximate query. Notice that the number of tables joined is three (with an additional 3-way join in the sub select). This is fixed regardless of the number of hosts in size of the cluster. A cluster finder query that looks for 64 machines would also involve the same joins. In the query,

the sub select returns those routers that have at least $N = 2$ appropriate machines attached to them, and the main select returns all those machines ordered by their router.

Scoped approximate queries

Scoping and approximation are orthogonal and can be combined, as they are in the lower right query of Figure 2. In this query, the entire sub select of the approximate query is replaced by the router to which the query is scoped. The query now involves only a three-way join. Instead of looking for two machines with total memory of 1GB, the approximation query looks for two machines that each have memory bigger than half of a GB, which is a deterministic subset of the original query results. This query can be extended to N host clusters in the following way.

To find a cluster of N qualified machines that are connected to one router, we first randomly choose a router (or iterate through a list provided by the user). We then perform the three-way join query, with `ROWNUM<=`N, to find possible combinations of hosts in the LAN. This process repeats until we find a router with N appropriate hosts attached, we run out of routers with no results found, or we run out of time. Notice that for any arbitrary N, the query remains a fixed three-way join. With the original SQL query, we would need a $2N + 1$-way join to find a N machine cluster, which is of course extremely expensive. If random routers are chosen, then we essentially randomly sample the result set of the approximate query (upper right), which is itself a deterministic subset of the full result set.

It is important to note that query time is largely dependent on the number of routers (or LANs) that we must search before we find one with enough appropriate hosts, which in turn is affected by the distribution of machine and network characteristics in the database. The query itself is also a factor. Query times will also vary based on the size of the cluster requested and the required resources on each of the hosts of the cluster. In what follows, we attempt to estimate the expected number of LANs that must be searched in order to obtain an appropriate result.

We would like to know the expected number of LANs that must be searched in order to find a cluster of N hosts in an M host LAN. As an example, suppose that we want each host to have at least 512 MB of memory, run the operating system Linux, and have a link bandwidth of at least 100 Mbps. To find the expected number of LANs searched, we must first know the a-priori probabilities of matching the memory size, operating systems and link bandwidth constraints. We can perform online queries to estimate these probabilities. We assume that these probabilities are independent, an assumption that is basically true in our database but may not be valid in all cases. We can find the probability that a host meets the constraints by

$$P_{mo} = P(mem \geq 512)P(OS = LINUX)P(BW \geq 100) \quad (1)$$

To find N qualified hosts in an M host LAN, we use the Bernoulli process:

$$P(num = N) = \binom{M}{N} P_{mo}^{N} (1 - P_{mo})^{M-N} \quad (2)$$

To find the probability that there are more than N qualified hosts in the LAN we sum up the probabilities from N to M as these satisfy our query:

$$P_{lan} = \sum_{num=N}^{M} P(num = N) \quad (3)$$

The expected number of LANs we need to search is then given by

$$ExpectNum = \frac{1}{P_{lan}} \quad (4)$$

We believe that the assumptions and model are reasonable in most cases, but some tests show discrepancy. The discrepancy is likely caused by the assumption that the probabilities of meeting constraints are independent. There can be weak dependences among them. We are currently in the process of improving the model.

Nondeterministic scoped approximate queries

In addition to scoping and approximation, nondeterminism can also be used orthogonal. Our proposed extended query syntax is as follows:

```
select
 [nondeterministically]
 [approximately]
  result_spec
from
  join_spec
where
  where_clause
[scoped by scope_clause]
[within x seconds]
```

In Section 5, we evaluate the performance of scoped, approximate, and scoped approximate queries and show that they can quickly return useful results in situations where the full query cannot be executed in a reasonable amount of time.

Time-bounded queries

In our current implementation, we can time-bound queries posed to the system. The query manager/rewriter

```
SELECT APPROXIMATELY
  h1.distip, h2.distip
FROM
  hosts h1, hosts h2
WHERE
  h1.mem_mb+h2.mem_mb>=1024 and
  h1.disk_gb+ h2.disk_gb>=160 and
  h1.insertid<>h2.insertid
WITHIN
  1 seconds;
```



```
SELECT
  H1.DISTIP
FROM
  HOSTS H1
WHERE
  H1.MEM_MB>=512 AND
  H1.DISK_GB>=80 AND
  ROWNUM<=2;
```

Figure 3. Non-network query and implementation as a scoped approximate query.

starts the query as a child process, and is allowed to run until the deadline is exceeded. If the query completes before that time, it returns the result set to the parent which then forwards the results to the caller. If it runs out of time, it is killed and no result set is returned.

Non-network example

Figure 3 shows a second example. Here we search for two machines with a total memory size greater than or equal to 1 GB and a total disk size greater than or equal to 160 GB. Since there is no network constraint, the scoped approximate query boils down to finding the first two hosts that meet the individual constraints. No joins at all are involved.

Limitations of scoped and approximate queries

There are mainly two limitations of scoped and approximate queries. The first is that the returned results are a subset of the original query. Thus, it is possible to return *no* results while in fact results exist for the given query. This possibility depends heavily on the query itself and the distribution of host attributes inside the database. Second, not all queries can be approximated or scoped. In other words, there certainly are situations in which there is no way to avoid or minimize the use of joins. Nonetheless, many queries can have their running times dramatically reduced through combinations of scoping, approximation, nondeterminism, and, of course, time-bounding. One proposed strategy is to first apply scoping, if possible, then rewrite the query to an approximate form, if possible. If the approximate query runs for too long or returns no results, step back and apply nondeterminism to it. If there are no results, apply nondeterminism to the original or scoped query.

5 Experimental Results and Evaluation

The goal of the experiments is to evaluate how the performance of scoped, approximate, and scoped approximate

Hosts	α	O	NW	NM	NL	SW	SM	SL
9.8K	8.915	-2.49	1	7	7	40	30	200
101.2K	8.915	-2.49	1	23	22	40	30	200
980K	8.915	-2.49	1	70	70	40	30	200

Figure 4. Parameters passed to GridG to generate 3 different sizes of synthetic grids.

queries on RGIS depends on the database size, the complexity of the query, and the load on the server. We use two different queries in our evaluation, the cluster-finder and non-network queries in the previous section. The first looks for sets of Linux hosts meeting total memory and individual link bandwidth requirements that are all connected to a common router. The second query looks for groups of hosts that meet a total memory and disk requirement. Similar to earlier GIS evaluation work [14], our evaluation metric is the average response time from the users perspective.

Our first step is to populate the database. We do this with GridG [8], a tool for generating realistic synthetic computing grids. GridG generates a grid as an annotated graph in which hosts, routers and other network devices are represented as nodes. The topology has a hierarchical structure but also conforms to the power laws that have been found in the Internet topology [5]. Annotations include memory, clock speed, cpu type, number of CPUs, operating system type, link bandwidths, router bandwidths, etc.

GridG network topologies are configured using eight parameters. Six determine the hierarchical structure of the generated grid (these are passed to the underlying Tiers generator [4]) and the remaining two determine the parameters of outdegree power law of the Internet topology (our extension). The eight parameters are: α (constant in outdegree law), O (outdegree exponent), NW (maximum number of WANs), NM (maximum number of MANs per WAN), NL (maximum number of LANs per MAN), SW (maximum number of nodes per WAN), SM(maximum number of nodes per MAN), and SL (maximum number of nodes per LAN). Figure 4 shows the specific parameters used to generate the three synthetic grids used in our experiments. The hierarchical parameters are based on requirements for the total number of hosts. The values of α and O in the table are from a measured router-level Internet topology discussed by Faloutsos, et al [5]. Memory size distributions are determine from Smith, et al's MDS data [13].

Unless otherwise noted, our experimental infrastructure is based on Oracle 9i Enterprise Edition running Red Hat Linux 7.1 on a dedicated Dell PowerEdge 4400 server. The server has two 1 GHz Xeon processors, 2 GB of main memory, and a PERC3DI RAID controller producing about 240 GB of RAID 5 storage over eight 36 GB U3 SCSI disks. Each test is performed 25 or 100 times and we provide the

Cluster Size	*Technique* Standard SQL query	Scoped query	Approx query	Scoped + Approx query
2	21.44	2.27	7.62	1.16
4	>7200	2047.93	7.48	1.32
8	>9000	>3600	7.46	1.43
16	N/A	>3600	7.51	1.45
32	N/A	>3600	7.65	5.96
64	N/A	>3600	>120	9.58

Figure 5. Cluster finder query times in seconds for the four query techniques for a database populated with 9.8K hosts. In the figure, *N/A* represents those tests that were not run due to expected extremely long query times.

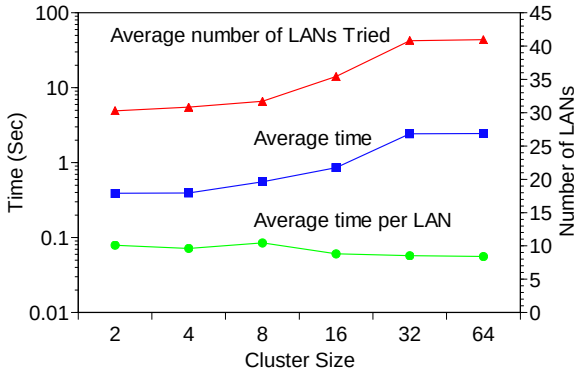

Figure 6. Cluster finder query time vs. cluster size with 9.8K hosts (scoped approximate).

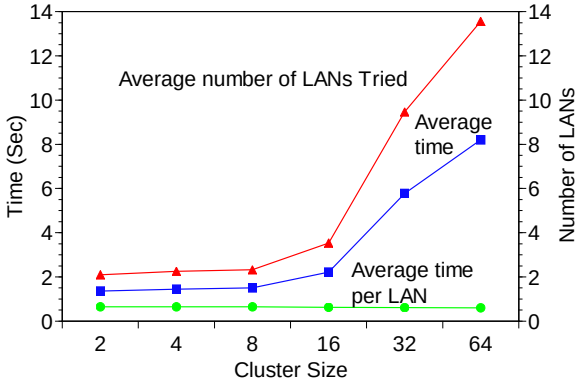

Figure 7. Cluster finder query time vs. cluster size with 101K hosts (scoped approximate).

average behavior. We limit all the tests to return one set of results to keep conformity between each of the experiments.

Cluster finder

In this experiment, we measured the response time of original, scoped, approximate, and scoped approximate versions of the cluster-finder query as a function of database size and query complexity (number of hosts). Because of time limitations, our data compares all versions on the small grid (9.8K hosts), but only the original (slowest) and scoped approximate (fastest) versions on the larger grids.

For each database size, cluster finder searches for an N host Linux cluster with link bandwidths $\geq$ 100 Mbps, and total memory $\geq 512N$. In our database each LAN is represented as a star with a router located at the center and the LAN links are full duplex. Given this, the link constraint is equivalent to a bisection bandwidth constraint of $\geq 100N$ Mbps.

Figure 5 shows a table of the average response times for each of the different versions of the query run on the small grid. Each query type that produce results, either an average or a limit, was run twenty-five times and an average value computed over the runs. When going from standard SQL queries to scoping, there is not much performance gain in terms of response time due to the presence of N-way joins in each. When moving to the approximate-only query without scoping, the queries scale well with increased cluster size. Only after the 32 node cluster search does the response time become unscalable. When mixing the scoped and approximate techniques, as shown in the last column of the table, it is seen that this provides not only the greatest gain in terms of response time, but scales extremely well as the cluster size increases. We can readily see that as cluster size grows, the spread between the techniques grows dramatically, with the ranking from slowest to fastest being: original, scoped, approximate, and scoped+approximate. In the remainder of this Section, we concentrate on the performance of the scoped approximate query (fastest).

Figures 6, 7 and 8 show the average response time, average number of LANs searched, and average time to search a LAN versus the size of cluster requested for the three different grid sizes. The scoped approximate queries always perform a three-way join, while the original queries perform a $2N + 1$-way join. The bigger the cluster size, the more benefit we get from scoped approximate queries.

An interesting trend in the Figures is that as the grid grows in size, the expected number of LANs that must be searched decreases. In addition, the average request time per LAN scales relatively linearly as the cluster size increases. In the worst case we studied (64 node cluster, 980K hosts) the average time of the request is less than 70 seconds, which is remarkably faster than the $>>$ 2000 second times seen by the original query.

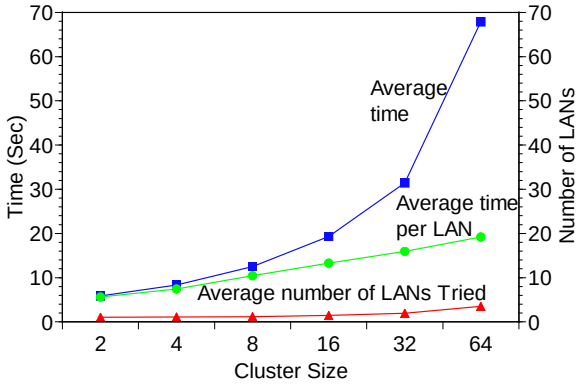

Figure 8. Cluster finder query time vs. cluster size with 980K hosts (scoped approximate).

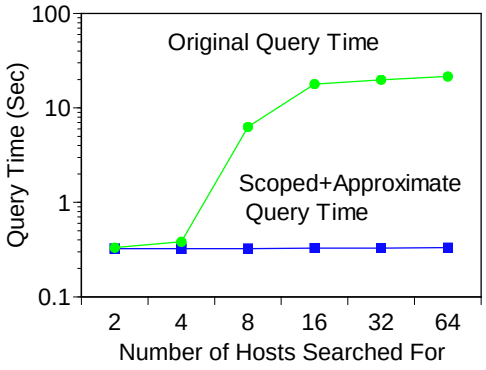

Figure 9. Non-network query response time vs. number of hosts with 9.8K hosts (scoped approximate).

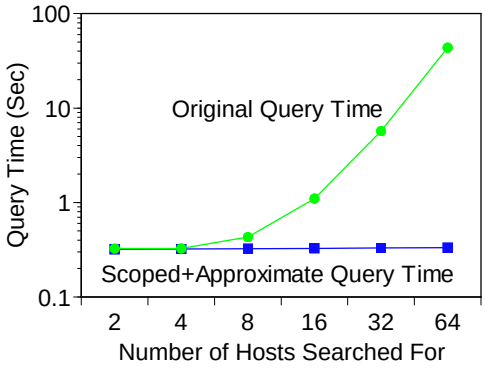

Figure 10. Non-network query response time vs. number of hosts with 101K hosts (scoped approximate).

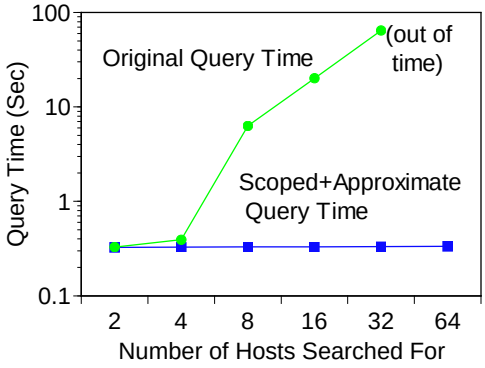

Figure 11. Non-network query response time vs. number of hosts with 980K hosts (scoped approximate).

Non-network example

Here we evaluate the performance of original and scoped approximate versions of a query that searchs for N hosts with a total memory $\geq 512N$ MB and total disk $\geq 80N$, Figure 3. The same grids are used as before. Figures 9, 10 and 11 show the average response time of both versions of the query for all three grid sizes. With the scoped approximate query, we have entirely elliminated all joins for this query, hence the performance is virtually constant with increasing numbers of hosts. In contrast, the original query can potentially scale exponentially.

Queries with multiple users and update load

GIS systems must scale in the presence of multiple concurrent users and under update load [10]. Hence we seek here to understand how multiple scoped approximate queries interact with each other and with updates. To study this we used the small grid (9.8K hosts) with update load that consists of a process continusly doing transactional updates of the disk size of randomly selected hosts.

In Figure 12, multiple users are iteratively issuing scoped approximate cluster finder queries that are each searching for Linux clusters with 64 nodes, link bandwidths of ≥ 100 Mbps, and a total memory of $512 * 64$ MB. We plot their average response time as a function of the number of concurrent users. Figure 13 is identical except here we run the non-network query trying to find N hosts with a total memory $\geq 64 * 512$ MB. We observe that the updating process can slow down the information server, but the effect is stable and no worse than a 10 percent degradation in most circumstances. How well multiple queries scale depends very much on their nature. The case of the non-network query, no joins are being done, which we expect dramati-

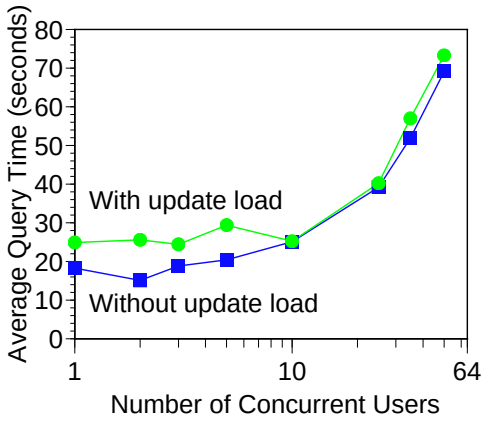

Figure 12. Cluster finder with multiple concurrent users and update load (scoped approximate).

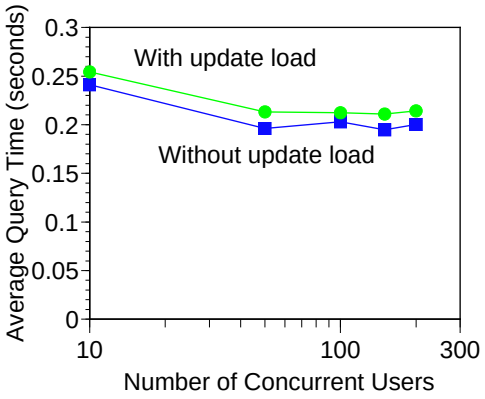

Figure 13. Non-network query with multiple concurrent users and update load (scoped approximate).

cally increases performance because the entire result table can probably be cached.

6 Conclusions

We have described and evaluated scoping and approximation, two techniques to reduce the running time of queries in the RGIS system. Scoping adds additional constraints to queries relative to the network topology and other relationships in the schema, while approximation replaces joins with tighter constraints on individual objects. In both cases there is a tradeoff: a subset of the full result set is quickly returned. Response time and server load can be dramatically reduced. These techniques are complementary to each other and to RGIS's nondeterministic queries. The three techniques help to make a relational approach to GIS feasible.

References

[1] CZAJKOWSKI, K., FITZGERALD, S., FOSTER, I., AND KESSELMAN, C. Grid information services for distributed resource sharing. In *Proceedings of HPDC 2001* (August 2001).

[2] DINDA, P., AND PLALE, B. A unified relational approach to grid information services. Tech. Rep. GWD-GIS-012-1, Global Grid Forum, February. Informational Draft.

[3] DINDA, P. A., AND LU, D. Nondeterministic queries in a relational grid information service. In *Proceedings of ACM/IEEE SC 2003 (Supercomputing 2003)* (2003). To Appear.

[4] DOAR, M. A better model for generating test networks. In *Proceedings of GLOBECOM '96* (November 1996).

[5] FALOUTSOS, M., FALOUTSOS, P., AND FALOUTSOS, C. On power-law relationships of the internet topology. In *SIGCOMM* (1999), pp. 251–262.

[6] FISHER, S. Relational model for information and monitoring. Tech. Rep. Informational Draft GWD-GP-7-1, Grid Forum, 2001.

[7] LIU, C., AND FOSTER, I. A constraint language approach to grid resource selection. Tech. Rep. TR-2003-07, Department of Computer Science, University of Chicago, March 2003.

[8] LU, D., AND DINDA, P. A. Synthesizing realistic computational grids. In *Proceedings of ACM/IEEE SC 2003 (Supercomputing)* (November 2003). To Appear.

[9] PLALE, B., DINDA, P., AND VON LASZEWSKI, G. Key concepts and services of a grid information service. In *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systmes (PDCS 2002)* (2002).

[10] PLALE, B., JACOBS, C., MOAD, C., PARAB, R., AND VAIDYA, P. Synthetic database benchmark/workload for grid information servers. In *http://www.cs.indiana.edu/ plale/projects/RGR/* (2003).

[11] RAMAN, R., LIVNY, M., AND SOLOMON, M. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC '98)*, (July 1998), pp. 140–146.

[12] RAMAN, R., LIVNY, M., AND SOLOMON, M. Resource management through multilateral matchmaking. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 2000)*, (July 2000), pp. 290–291.

[13] SMITH, W., WAHEED, A., MEYERS, D., AND YAN, J. C. An evaluation of alternative designs for a grid information service. *Cluster Computing 4* (2001), 29–37.

[14] ZHANG, X., FRESCHL, J. L., AND SCHOPF, J. M. A performance study of monitoring and information services for distributed systems. In *Proceedings of the International High Performance Distributed Computing (HPDC)* (2003).