Nondeterministic Queries in a Relational Grid Information Service

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

Abstract

A Grid Information Service (GIS) stores information about the resources of a distributed computing environment and answers questions about it. We are developing RGIS, a GIS system based on the relational data model. RGIS users can write SQL queries that search for complex compositions of resources that meet collective requirements. Executing these queries can be very expensive, however. In response, we introduce the nondeterministic query, an extension to the SELECT statement, which allows the user (and RGIS) to trade off between the query's running time and the number of results. The results are a random sample of the deterministic results, which we argue is sufficient and appropriate. Herein we describe RGIS, the nondeterministic query extension, and its implementation. Our evaluation shows that a meaningful tradeoff between query time and results returned is achievable, and that the tradeoff can be used to keep query time largely independent of query complexity.

1 Introduction

As the scale and diversity of the resources, applications, and users involved in Grid computing [12, 15] continues to explode, the amount of information needed to keep track of them grows commensurately. Simultaneously, applications need to pose and answer increasingly powerful queries over this information in order to exploit Grid resources well and satisfy users. Grid Information Service (GIS) systems provide this functionality. The possible models and the design space for GIS systems is large.

Our view of a GIS is that it is a database (in the generic sense of the word) of information about the entities within a wide area distributed high performance computing environment. Examples of Grid entities include organizations, people, computational resources (hosts, clusters), communications resources (switches, routers, topologies), services, benchmarks, software, event channels, sensors, scientific instruments, and others. A GIS consists of a set of objects that represent these entities, relationships between objects, and systems needed to query and update the objects and relationships. Each object has a unique identifier, a timestamp, and a set of attributes. Updates to the database take the form of additions or deletions of objects and of changes to the attributes of existing objects. The GIS makes updates available to queries as soon as possible. It also manages access to the objects, making sure that they are updated and read only by valid users. It may present different views of the objects to different users. A more detailed description of this view of a GIS is available elsewhere [25].

We are developing a GIS system, RGIS, that is based on the relational data model. Specifically, RGIS servers are implemented on top of the Oracle RDBMS and use SQL as their query language. Oracle is not a requirement of our approach—other RDBMSes could also be used. RGIS focuses on modeling the hardware and software resources of a distributed computing environment. Information streams from dynamic resource monitoring and prediction tools such as RPS [7] are currently outside of the scope of RGIS, although RGIS does model the existence and location of such tools. We plan to extend RGIS to provide a unified query model over resource and monitoring information. 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 unified relational approach to GIS systems is available elsewhere [6].

A powerful feature of RGIS is that users can write queries in SQL that search for complex compositions of re-

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).

SC'03, November 15-21, 2003, Phoenix, Arizona, USA

sources, such as groups of hosts and network resources, that meet collective requirements. These queries can be very expensive to execute, however. In response, we have introduced an extension to the SQL select statement that we call the nondeterministic query. In essence, the nondeterministic query extension allows 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 is a random sample of the result set of the deterministic version of the query, which we argue is sufficient and appropriate for a GIS. We implement nondeterministic queries using a combination of query rewriting, schema extensions, indices, and randomness. No changes to the RDBMS are needed. Combined with two other query extensions, scoped queries and approximate queries, which are described elsewhere [22], nondeterministic queries make it possible to ask complex questions of RGIS and get useful responses quickly.

In this paper, we describe RGIS, the nondeterministic query extension, and its implementation. We also present a performance evaluation of our implementation, populating our database with networks as large as five million hosts using our GridG [21] grid generator tool. The evaluation shows that a meaningful tradeoff between query processing time and result set size is possible using nondeterministic queries, and that we can use that tradeoff to keep query running time largely independent of query complexity. These results suggest that we can deliver powerful relational queries to users.

2 Related work

The data model, query language, and implementation of GISes and similar services that store the information about networked resources has been evolving for some time.

Today, many sites that provide externally accessible computing resources make a description of those machines available as web pages. Web search engines are often used to find appropriate resources. This has been aided significantly by the advent of highly discriminating search algorithms for arbitrary documents, such as PageRank [3]. By providing highly structured data, most GIS systems aim to provide more sophisticated queries.

Within the networking community, SLP [36] has been proposed as a standard for discovering services. The DNS name service [2] is universally used. DNS maps a hierarchical name (a path) to a blob of information and is typically used to resolve hostnames to IP addresses. Protocols for constructing and querying hierarchical distributed databases such as X.500 [17] and LDAP [16] can be viewed as extensions to this idea, although hierarchical databases predate DNS. Each node on an LDAP tree can have multiple typed attributes associated with it. An LDAP query is a traversal of a subtree that returns nodes whose attributes satisfy the query constraints. Each subtree can be serviced by a different LDAP server, making it straightforward to partition responsibility and security over multiple sites. In contrast to these approaches, RGIS builds on a relational data model instead of a hierarchical data model.

Within the distributed systems community, service location and naming services are basic needs. DCE [32], CORBA [23], and Java's Jini Framework [37] include these services. In DCE and CORBA, the service and the query consists of a type specification for a procedure or object (the interface) and the result is matching instances. Jini uses a more general tuple of attribute-value pairs as the service descriptor, and a tuple of attribute constraints as the query. One strand of recent research [1, 35, 18, 33] has focused on timelines of updates and on how services can push updates to users. Another strand has focused on distributing data throughout the network and then routing queries to likely nodes where matching data may reside using distributed hash tables [30, 28]. In contrast to these systems, RGIS attempts to provide compositional queries (joins) where collections of objects are needed to satisfy the query.

The Grid computing community has seen an explosion of work on GIS systems. The most relevant systems are Globus MDS2 [5], the Condor Matchmaker [26], and R-GMA [11]. MDS2 is based on LDAP and defines a schema (the attribute types) that can be associated with nodes in the tree. In contrast, RGIS uses a relational data model.

In the Condor Matchmaker, *both* resources and queries are collections of attributes and constraints. This enables bilateral matchmaking, where both the resource owner and the querier can constrain which results are returned on a query. Bilateral matching is a very fast process. Condor Matchmaker was later extended to support gang-matching, meaning that a query can be written that requires more than one resource to be satisfied [27]. Gang-matching is implemented using prioritized search with backtracking, which is more expensive than the search for bilateral matchmaking. 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) [19]. CSPs are NP-Hard and are solved using the heuristic techniques implemented in an underlying CSP solver.

R-GMA [11] is close to our work in that it also proposes a relational data model for GIS systems. RGIS differs from R-GMA in two ways that are relevant to this paper. First, R-GMA focuses currently 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. The success of R-GMA suggests that effectively incorporating dynamic information into RGIS is feasible. Our second

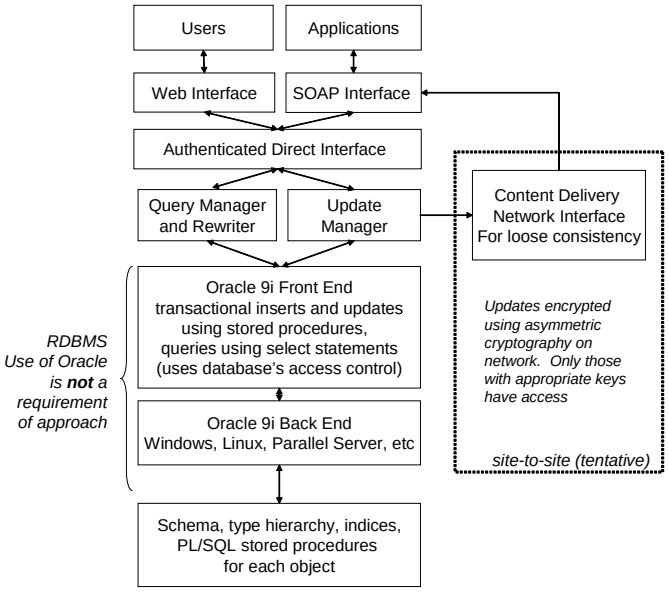


Figure 1. RGIS Structure.

difference is RGIS's support for nondeterministic queries, as described here.

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 RGIS's approach to this problem, which is random sampling. While there has has been considerable work in how to build random sampling directly into database systems (Olken's dissertation [24] is a good introduction to this), reducing cardinality of results [4], and incremental queries [31], current database systems do not support these features. RGIS builds its random sampling on top of unmodified ordinary database systems using query rewriting, schema extensions, indices, and randomness.

Efforts to define the broad structure of the computational grids [14] and standardize the specifics of interaction among components [13] suggest that there are roles for multiple kinds of GIS systems, and that different systems can and will interoperate.

3 Design of the RGIS system

Figure 1 illustrates the structure of the RGIS system, focusing on a single RGIS server. We expect that each site within a computational grid will run one such server, although multiple servers per site can also be supported. The goal of the RGIS server for a site is to provide a view of the computational grid appropriate to that site's users. A site's RGIS server is responsible for all queries issued by users on the site.

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 such as DB2, MS SQL Server, Postgres, and MySQL. A early implementation of our work used MySQL. Like most serious database systems, Oracle provides a single front-end interface to multiple back-end implementations. In particular, this provides platform independence (Oracle runs on many operating systems and platforms) and intra-site scalability (Oracle has a variety of implementations, including a Parallel Server product that scales over clusters). All components of RGIS above the database front end are written in Perl for portability.

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 idea of modeling networks at layer 2 and below, which can be quite useful in mapping applications, is inspired by the Remos system's bridge discovery collector [20]. Typed objects are inserted into the database by updating one or more tables. An object is also identified in a special table (insertids) by a unique insertion identifier, a timestamp (NTP is assumed), and ancillary information to support our specialized needs, such as nondeterministic queries, and to link virtual resources with physical resources [10]. Every other table that is updated includes this insertion id, hence making it easy to find all elements of an object, no matter what tables it spans.

The RGIS schema includes the sequences, tables, constraints, triggers, and indices that represent our grid modeling efforts. Figure 2 shows a high-level view of the RGIS schema, focusing on the representation of a host computer. Figure 3 gives details of the current implementation of the host-specific tables.

Given transactional updates, the constraints and triggers are designed to keep the database in a consistent state. We use Oracle's access control mechanisms to assure that insertions, updates, and deletions occur only via stored procedures that force transactions. For every type of object, there is an associated PL/SQL package (essentially, a class) that provides this functionality. Each package also includes non-transactional versions of the operations which very privileged users can use to batch multiple updates together into a single transaction. The code to implement each package is generated automatically using templates written in Perl.

Layered on top of the RDBMS front-end is a query manager/rewriter, and an update manager. Together, these two provide the core application interface to an RGIS server. This interface is exported through a layer that provides au-

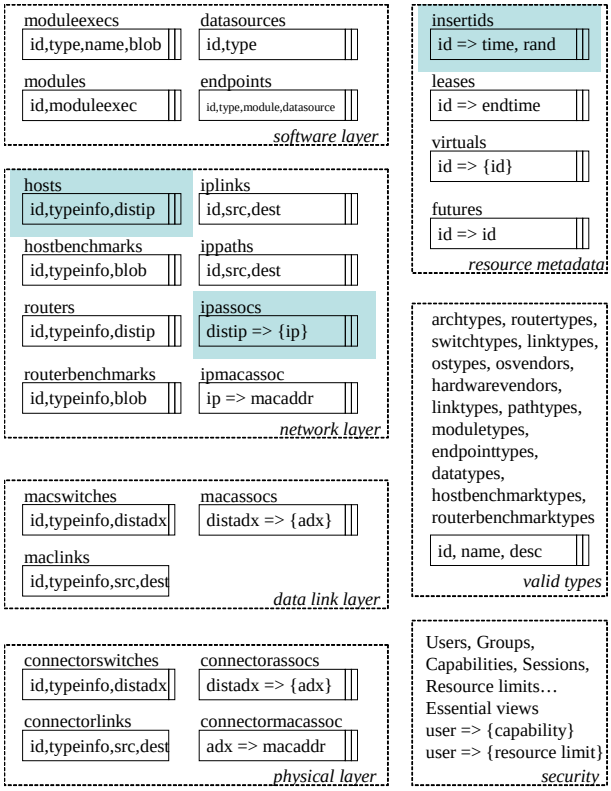


Figure 2. Overview of the RGIS Schema. Highlighted are the minimum tables used to represent a host. A host may also be represented in the leases table if it may leave the system, the virtuals table if it is a virtual machine, and the futures table, if it is not yet instantiated.

thentication of the user and check of his capabilities for each request, mapping from an external notion of user to a database-local notion of user. This interface in turn is made visible to the outside world via a web interface (Figure 4), and a SOAP interface, both running over the encrypted HTTPS protocol.

The update manager aggregates updates (insertions of new objects, deletions of existing objects, and changes to the properties of existing objects) coming from local sources and remote sources and batches them into transactions for the RDBMS. In this role, it can prioritize updates and also control the rate of updates and the update latencies going into the database.

RGIS servers do not talk directly with each other, but indirectly via a content delivery network (CDN), which is used solely to propagate updates to friendly RGIS servers. There is no implicit notion of trust among RGIS servers. If a site is interested in receiving updates from a remote RGIS server, it must arrange with the remote administrator

```
CREATE table hosts (
  distip      varchar2(15)     not null primary key,
  name        varchar2(256)    default('UNKNOWN'),
  numproc     number           default(1),
  mhz         number           default(0),
  constraint good_mhz_hosts check (mhz>=0),
  arch        varchar2(32)     default('UNKNOWN'),
  constraint good_arch_hosts foreign key (arch)
     references archtypes(name),
  hwvendor    VARCHAR2(32)     DEFAULT('UNKNOWN'),
  constraint good_hw_hosts foreign key (hwvendor)
     references hardwarevendors(name),
  os          varchar2(32)     default('UNKNOWN'),
  constraint good_os_hosts foreign key (os)
     references ostypes(name),
  osvendor    varchar2(32)     default('UNKNOWN'),
  constraint good_osv_hosts foreign key (osvendor)
     references osvendors(name),
  osver       varchar2(256)    default('UNKNOWN'),
  kernelver varchar2(256)      default('UNKNOWN'),
  mem_mb      number           default(0),
  constraint good_mem_hosts check (mem_mb>=0),
  vmem_mb     number           default(0),
  constraint good_vmem_hosts check (vmem_mb>=0),
  disk_gb     number           default(0),
  constraint good_disk_hosts check (disk_gb>=0),
  location    varchar2(256)    default('UNKNOWN'),
  owner       varchar2(256)    default('UNKNOWN'),
  description varchar2(256),
  insertid    number           not null unique,
  constraint good_insert_hosts foreign key
     (insertid) references insertids(insertid)
     ON DELETE cascade
);
CREATE table insertids (
  note      varchar2(256),
  time      timestamp  not null,
  insertid  number             not null primary key,
  rand      number             not null
);
CREATE table ipassocs (
  distip      varchar2(15)     not null,
  ip          varchar2(15)     not null primary key,
  insertid    number           not null,
  constraint good_insert_ipassocs foreign key
      (insertid) references insertids(insertid)
       ON DELETE cascade
);
```

Figure 3. Specific SQL representation of a host. Definitions of indices elided.

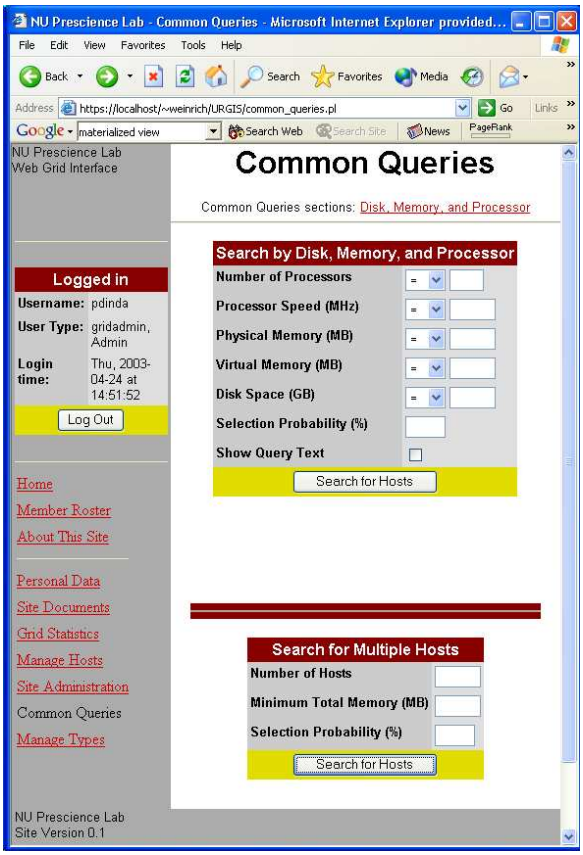


Figure 4. RGIS web interface.

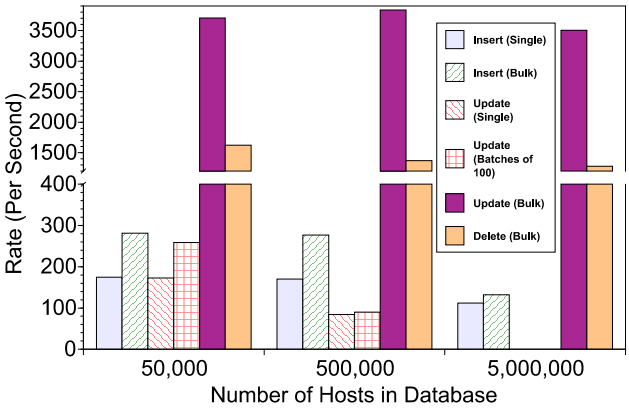


Figure 5. Insert, update, and delete rates.

to create a key pair. Each update to the local RGIS serer is then encrypted in a manner similar to PGP multiple destination messages, making it readable only to those RGIS servers that hold one of the perhaps many keys used in the encryption process. The CDN is used to send the update to the group of all those holding keys, using, for example, application-layer multicast for effi ciency. If a higher level of trust between the two RGIS servers exists, fi ner grain information control is also possible: the update can contain a list of user keys, one of which must be matched before an RGIS server will use the update to answer a query. An RGIS server combines local updates and remote updates to create a view of the computational grid that corresponds to that which its users have access.

In the limit, each RGIS server could contain data about all resources on a wide area network, although we expect this will rarely happen. Although this is clearly asymptotically unscalable, it is not unreasonable for computational grids of likely size. Consider a computational grid of one billion hosts and routers (about fi ve times the current size of the Internet). With two kilobytes of information per host, about 2 TB of data storage would be necessary in the RGIS server. This requires less than $10,000 of disk storage using a modern direct-attach RAID box, making it clearly within the realm of possibility. Furthermore, the $/MB of disk capacity is shrinking much faster than the Internet is growing. Update rates can be an issue, but three things ameliorate this. First, we can achieve quite high update rates on off-the-shelf RDBMS systems such as Oracle. Figure 5 shows the rates for insertions, updates, and deletions in RGIS with different size databases running on our hardware (See Section 5 for details about the hardware and software confi guration). Here an insert means adding a host to the database, which involves a transactional modifi cation of a sequence and three tables, an update means modifying the memory attribute of a host already in the database, which is a transactional modifi cation of two tables, and delete means to remove a host from the database, transactionally modifying three tables. Second, bandwidth into a site grows with the update rate, since the update rate grows with the number of hosts and routers. Third, RDBMS systems such as Oracle and DB2 can scale over clustered servers to support very high update rates. In effect, we can leverage the existing TPC-C online transaction processing benchmark competition [34] to address the updates.

A site sends queries only to its RGIS server. This ties the resources a site is willing to commit to its RGIS server to the number and kind of queries it wants to make. This is vital because the nature of many RGIS queries is similar to decision support queries (TPC-H [34]) in relational database systems. Such queries can be very expensive to execute and so are unlikely to be welcome on foreign RGIS servers.

The goal of the query manager/rewriter is to shape the query workload so that it can be effectively executed by the RDBMS, by which we mean that the load on the RDBMS is kept below one and individual queries fi nish quickly. Queries take the form of select statements written in a slightly extended form of SQL. The query manager/rewriter translates queries into the underlying SQL dialect, *modify-*

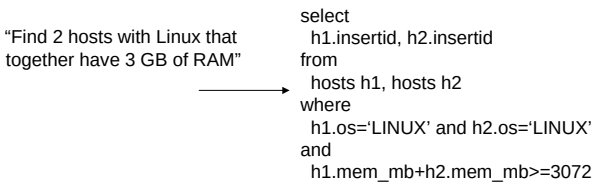


Figure 6. An RGIS Query.

ing the query semantics to balance between the needs of the query and the needs of the system. In essence, the query manager/rewriter can trade off between the result set size for a query and the resources the query requires to execute. The adaptation mechanism it uses is the selection probability for inputs of the query. The next section describes this idea in detail.

4 Nondeterministic queries

Parallel and distributed applications are 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 four processors. At startup, it will want to ask questions such as "find me a set of four unique hosts which in total have between 0.5 and 1 GB of memory and which are connected by network paths that can provide at least 2 MB/s of bandwidth with no more than 100 milliseconds of latency." Such questions can be readily posed to RGIS using the SQL language. Indeed, SQL lets the application or user combine multiple resources in arbitrary ways. Figure 6 shows a simple RGIS query that is searching for all pairs of hosts that both run the Linux operating system and together have at least 3 GB of RAM. For clarity in this example query, we omit the constraint that the two machines be distinct. In our evaluation, where in part we use similar queries, we introduce this constraint.

Because the query is declarative, there is significant room for the query optimizer in the RDBMS to make the query efficient. It also means that the query is independent of the underlying RDBMS implementation that is being used. The same query may run today on a basic Windows implementation of Oracle, while tomorrow it may be run by a parallel implementation of Oracle on a cluster or SMP. The query is also independent of the indices created by the database or by the database administrator. Hence, if this form of query becomes common, the administrator can create indices to speed it up. Finally, if queries are written in ANSI standard SQL, the underlying RDBMS can be changed without changing the query. Common queries can also be provided as materialized (i.e., precomputed) views on the database.

Unfortunately, queries such as the one in Figure 6 can be very expensive to execute, especially as the number of joins (number of hosts in the query in this example) grows. In the worst case, the query cost can grow exponentially with the number of joins. Not only must individual queries not take long periods of time to execute, an RGIS server must also be able to handle the query workload of a whole site. If we supported such queries directly, we would very soon begin disappointing users and overloading the RGIS server.

Deficiencies of limited deterministic queries

One approach to reducing the work involved in answering a query is to limit the size of the result set that is returned (using "rownum<N" as part of the where clause in Oracle, or MySQL's "limit" clause, for example). The query would then only run until the specified number or rows was returned. We'll refer to this as a limited deterministic query. It is intuitive why a limited deterministic query would be reasonable from an application's perspective. The application making the query of Figure 6 is not interested in *all* pairs of hosts that meet its requirements. It is merely trying to find *some* pairs that do.

Limiting result set size has two serious problems, however. First, the computational time for the query is not directly proportional to the result set size—it depends on the data distribution in the input tables. Continuing the example, the rarer that pairs of hosts that meet the requirements are, the longer a limited query will run. In the worst case, the RDBMS may have to scan the cross product of the hosts table to the very end to find a single match, making this query as expensive as one without limits. The other problem with the limited deterministic query is that the query returns exactly the same results each time it is run. Suppose there are 10 pairs of hosts that are appropriate, but the query is limited to one pair. Different applications making the same query would end up choosing the same pair, leading to contention. In general, limited deterministic queries can lead to certain resources suffering contention hotspots merely due to where they happen to be placed in the database.

Implementing nondeterministic queries

RGIS limits query running time (and load) and avoids contention through the use of nondeterministic queries. The left-hand side of Figure 7 shows a nondeterministic, time-bounded version of the earlier query. The additions to the query are shown in italics. A nondeterministic query returns a random subset of the full set of query results. The computational cost of the query is controlled by the *selection probability*, which is derived from the time limit of the "within" clause and the current load on the RGIS server. The selec-

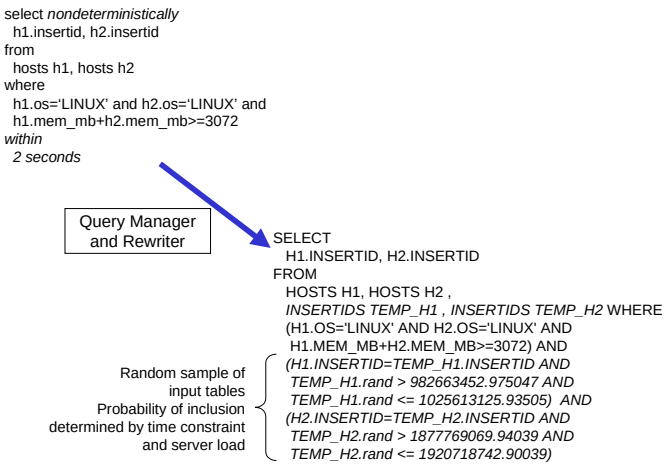


Figure 7. An RGIS nondeterministic query and its implementation.

tion probability is the probability that a row of an input table will be included in the join. Intuitively, as the load increases or the time limit shrinks, the selection probability shrinks. As the selection probability shrinks, so does the amount of work needed to perform the query and the expected number of rows returned by it. Each time the query is run, the rows returned are different while the computational cost of getting them stays roughly the same.

We implement nondeterministic queries using a combination of query rewriting, schema extensions, indices, and randomness. No changes to the RDBMS are needed. When a nondeterministic query is posed to the query manager/rewriter, it determines a selection probability, p, for the query. Associated with each object inserted into the database (in the insertids table), at insert time, is a random number, ranging from R_{min} to R_{max}, for a range, $R = R_{max} - R_{min}$. We translate the p into a subrange, $r = pR$. Next, for each input table T_i in the query, we add a where clause that constrains rows in that table to have associated random numbers in the range $[s_i, s_i + r)$, where s_i is chosen from a uniform random distribution over $[R_{min}, R_{max}]$ at query translation time.

It is important to note that this approach has a grouping effect that should ideally not occur in an implementation of random sampling. Two objects inserted into the database may be assigned nearby random numbers. Hence, if one is chosen, there is a greater likelihood that the other will also be chosen. With small selection probabilities, the effect is negligible. However, to ameliorate it, we regularly "reshuffle" the insertids table, assigning new random numbers to objects.

In addition to the random numbers associated with each object in the database, the database also includes indices on these random numbers and on their associations with other attributes. These indices help to make the random sampling fast. In the next section, we will evaluate the effectiveness of nondeterministic queries for limiting query time and load.

Scoped and approximate queries

RGIS also supports two other methods for speeding up queries by modifying their semantics, scoping and approximation [22]. 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, either by prefix-matching against IP addresses or by rooting the query at a router or switch. 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 nondeterministic queries.

Common queries

One issue with relational queries is that their power comes with significant complexity. Indeed, in industry, the development of relational queries and their optimization is often allotted to a specialist. To address this, we are developing a set of "common queries", which are essentially Perl scripts that generate queries for what we believe will be common forms of questions. However, it is possible that the role of a "grid query developer" may also need to exist.

5 Evaluating nondeterministic queries

Our evaluation of nondeterministic queries examines how the query run time and the result set size depends on the database size, the selection probability, and the complexity of the query. We use two different queries. The first looks for groups of hosts that together have a given amount of memory. The second looks for two hosts of the same operating system that are directly connected.

Unless otherwise noted, our experimental infrastructure is based on Oracle 9i Enterprise Edition running on 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.

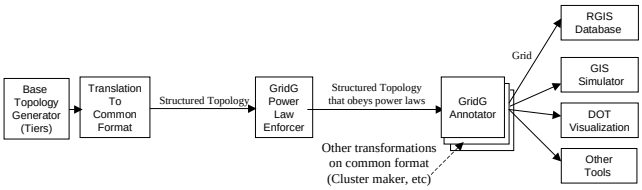


Figure 8. GridG Architecture.

GridG: realistic synthetic grids

To evaluate queries, we must first populate our database. We did this using GridG [21], our tool kit for generating realistic computing grids. GridG generates a Grid as an annotated topology network graph in which hosts, routers and other network devices are represented as nodes. The topology has a hierarchical structure and also conforms to the power laws of Internet topology [9]. Annotations include memory, clock speed, CPU type, number of CPUs, operating system type, link bandwidths, router bandwidths, etc. Figure 8 illustrates the architecture of GridG.

In the following, we will separately describe the GridG parameters used in evaluating each form of query. In general, we populated our database using power law distributions found for routers in the Internet [9] and the memory distribution found in Smith, et al's study of MDS server contents [29].

"Find n hosts with at least 3 GB of memory"

This query is a generalization of our running example from Section 4, but parameterized to find n different hosts. A four host example is shown in Figure 9. What is returned are the distinguished IP addresses of the hosts. Note that while such queries can become quite complex as the number of hosts grows, they can generally be automatically generated very easily.

To evaluate the performance of queries like this, we needed to populate the database with large numbers of hosts whose memory sizes were distributed in meaningful ways. To do this, we studied (anonymized) data dumps from the MDS servers running on several large grids, and data provided by the BOINC project at Berkeley. The largest dataset was one collected by Smith, et al [29]. Smith's dataset contains fewer than a thousand hosts. We extracted memory sizes from Smith dataset and then configured GridG to generate hosts with the same memory size distribution. Figure 10 shows the memory size distribution.

Using the memory size distribution, we generated grids of 50,000, 500,000, and 5,000,000 hosts with GridG. For each grid, we evaluated 2, 4, 8, and 16 host versions (using

```
select nondeterministically
  h1.distip, h2.distip, h3.distip, h4.distip
from
  hosts h1, hosts h2, hosts h3, hosts h4
where
 h1.mem_mb+h2.mem_mb
    +h3.mem_mb+h4.mem_mb>3072 and
 h1.insertid<>h2.insertid and
 h1.insertid<>h3.insertid and
 h1.insertid<>h4.insertid and
 h2.insertid<>h3.insertid and
 h2.insertid<>h4.insertid and
 h3.insertid<>h4.insertid
within
 1 seconds;
```

Figure 9. Sample query to find 4 hosts with minimum memory over 3GB.

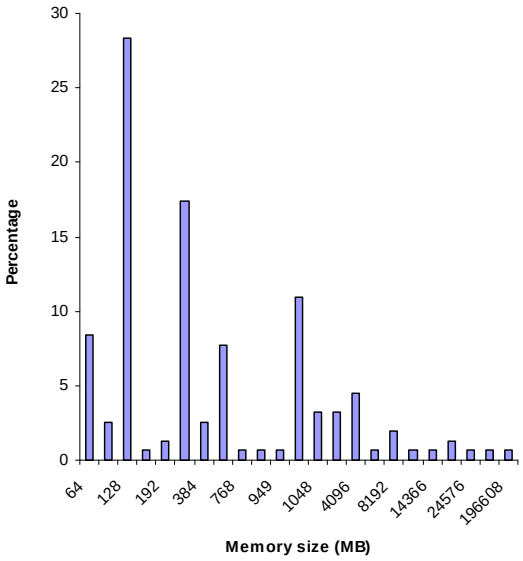


Figure 10. MDS memory size distribution.

2, 4, 8, and 16-way joins) of the query, varying selection probabilities. We ran each query five times, measuring the query running time and the number of rows returned by the query. We report the mean, minimum, and maximum of the five runs. Figure 11 shows all the performance data.

In addition to the nondeterministic queries, we also evaluated the performance of deterministic and limited deterministic versions of the simplest query (2 way join) on the smallest number of hosts (50K) , data that occupies the first

			Number of rows selected			Query Time (seconds)		
Number of Hosts	Number of joins	Selection Probability	Average	Min	Max	Average	Min	Max
50K	2 way	deterministic	-	-	-	> 1 hour	> 1 hour	> 1 hour
50K	2 way	limited deterministic 1 or 10 rows	1,10	1,10	1,10	0.13	-	-
50K	2 way	0.001	562	261	933	0.42	0.376	0.463
		0.01	55729	50093	66803	17.8	17.3	18.7
	4 way	0.0001	527	111	1343	0.35	0.3	0.45
		0.0005	131791	69357	181139	26.3	12.1	34.2
	8 way	0.00005	1156	0	3103	0.72	0.53	0.99
		0.0001	178853	1920	597911	29.3	0.83	102
	16 way	0.00001	0	0	0	6.67	6.64	6.7
		0.00005	298598	0	1492992	81.3	6.64	380
500K	2 way	0.0001	566	299	802	0.81	0.53	0.94
		0.0005	13048	10620	16168	5.34	4.42	6.73
		0.001	57524	13048	62340	18.3	16.3	19.43
		0.002	216382	210290	220030	73.0	70.0	76.0
	4 way	0.00001	541	0	1293	0.36	0.26	0.48
		0.00005	143226	62853	219366	26	18.2	32.9
	8 way	0.000005	1231	0	6848	0.69	0.53	1.3
		0.00001	54127	9368	130971	9.1	2.2	19.9
	16 way	0.000001	0	0	0	6.65	6.63	6.7
		0.000005	804533	0	3930540	115.6	6.63	523.6
5000K	2 way	0.00001	507	380	635	1.1	0.96	1.13
		0.0001	60315	52707	70613	22.4	20.9	23.9
	4 way	0.000001	235	20	624	0.55	0.46	0.65
		0.000005	189920	109533	322668	23.2	17.5	35.4
	8 way	0.0000005	551	138	1296	0.77	0.71	0.87
		0.000001	272704	110614	674554	28.9	13.9	68.2
	16 way	0.0000001	0	0	0	6.7	6.69	6.71
		0.0000005	121473	0	330884	31	6.7	78.1

Figure 11. Performance of nondeterministic queries with different sizes of grid, different numbers of hosts, and different selection probabilities.

two rows of Figure 11. Notice that the purely deterministic query, which will eventually return all possible results, requires over an hour of running time. The limited deterministic query, which returns the first result, or the first 10 results, finished very quickly (0.13 s), but always returns the same results. The nondeterministic version of the same query executes more slowly, taking about twice as long even with a very low selection probability. However, following the discussion of Section 4, we now get a different set of results each time we run the query, and the query will do a fixed amount of work each time it is run. It is slower because there are two additional equijoins with the insertids table, as can be seen in Figure 7. This overhead is the cost for implementing random sampling above the database. A database engine that supported random sampling would not pay this penalty.

To better illustrate our results for nondeterministic queries, we show two slices through the table. We use the 500,000 host grid. Figure 12 shows the average number of results and the average query time for the two host version of the query as a function of the selection probability. The left hand scale corresponds to the query time, while the right hand scale shows the number of results. Note that all scales are logarithmic. These results show that it is possible to meaningfully trade off between query processing time and result set size. We can vary the query time and the result set size over several orders of magnitude by modulating the selection probability.

Figure 13 shows a second slice through our data. Once again, we have used the 500,000 host grid and show the average query time and result set size, but here we vary the complexity of query (the number of hosts asked for) and choose selection probabilities to try to keep the query time as constant as possible. The point here is that it is possible to use the selection probability to control the query time largely independent of query complexity.

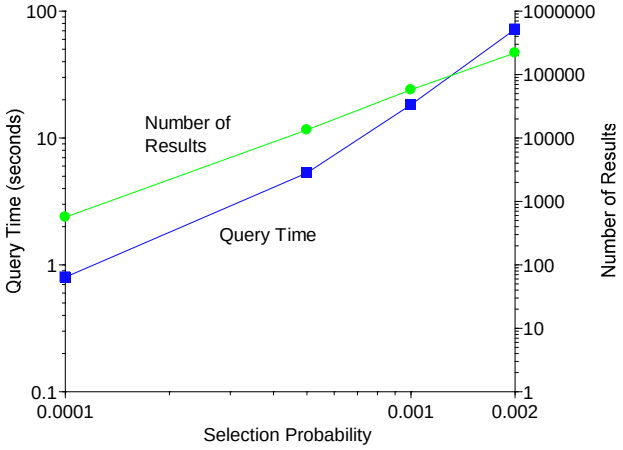


Figure 12. Query time and number of selected rows versus selection probability.

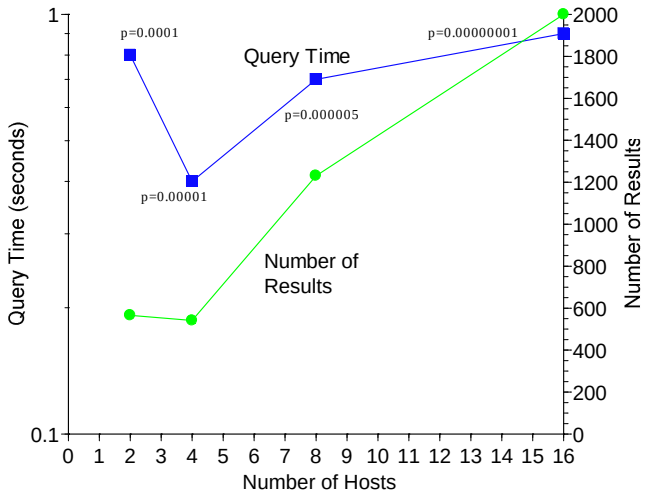


Figure 13. Query time and number of selected rows versus number of hosts.

Grid topology-related query

Because RGIS also stores information about the network topology of a grid, we can include topology in our queries. For example, we can find the shortest paths between a pair of hosts, or all pairs shortest paths, or a group of hosts that are tightly connected. For example, "find n hosts with the same operating system and with a total memory of b bytes, all attached to the same router" would find tightly coupled groups of machines that could be used as clusters.

```
select nondeterministically
  h1.distip, h2.distip
from
  hosts h1, hosts h2, iplinks links
where
  h1.mem_mb+h2.mem_mb>1024 and
  h1.os=h2.os and h1.insertid<>h2.insertid and
   ((h1.distip=links.src and
     h2.distip=links.dest) or
    (h1.distip=links.dest and
     h2.distip=links.src))
within 1 seconds;
```

Figure 14. Sample query to find pairs of directly connected hosts.

Hosts	α	O	NW	NM	NL	SW	SM	SL
10K	8.915	-2.49	1	20	10	10	10	50
50K	8.915	-2.49	1	100	10	10	10	50
100K	8.915	-2.49	1	100	20	10	10	50

Figure 15. Parameters passed to GridG in Topology related query datasets.

Figure 14 illustrates a very simple such query which tries to find all pairs of hosts that are directly attached (at layer 3). The machines must have the same operating system and must have a total memory of at least 1 GB.

To evaluate such a query, we must first have a network topology. 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 [8]) and the remaining two determine the parameters of outdegree power law of Internet topology (our extension). The eight parameters are:

- α : constant in outdegree law
- O : outdegree exponent
- NW : maximum number of WANs (currently only 1 supported)
- 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
- SL : maximum number of nodes per LAN

Figure 15 shows the parameters that were used to generate the topologies used for this section. The hierarchy parameters are based on requirements for 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 [9].

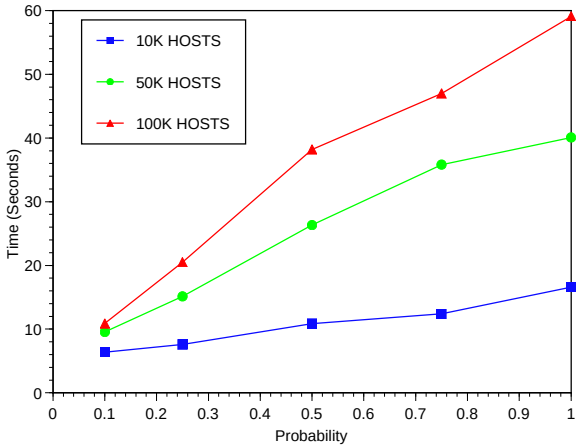


Figure 16. Query running time versus selection probability.

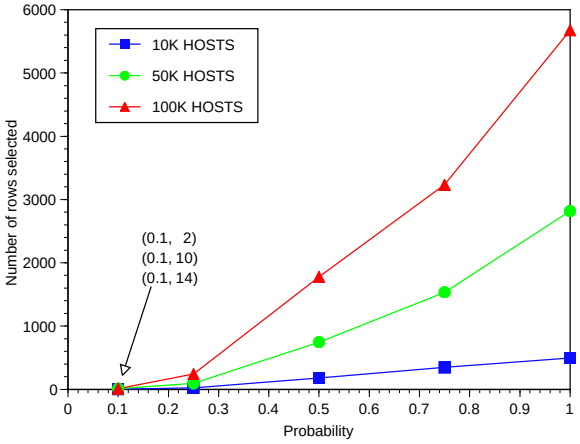


Figure 17. Number of selected rows versus selection probability.

Figure 16 shows the average query time versus selection probability for different size grids, namely, with 10,000, 50,000 and 100,000 hosts. Figure 17 shows the average result set size for the same parameters. For this range of selection probabilities, query time increases approximately linearly with selection probability, while the result set size increases slightly faster.

These figures provide more evidence that it is possible to use selection probability to trade off between result set size and query time for queries about grids with typical topological and memory size distributions.

6 Time-bounded queries

The evaluation of the previous section showed that selection probability can be used to trade off between the running time of a query and the number of results returned, and that the result set would vary from query to query. However, as described in Section 4, nondeterministic queries in RGIS are also time-bounded. RGIS implements these deadlines using three techniques, although this part of the system continues to evolve.

The first technique is *hard-limiting*. The query manager/rewriter starts the query as a child process or thread. The child is then allowed to run until the deadline is exceeded. If it completes before that time, it returns the result set to the parent which returns them to the caller. If it runs out of time, it is killed and no result set is returned. Hard-limiting can be used in conjunction with the other techniques.

The second technique is *climbing*. Here, we initially run the query with a very small selection probability. If no results are returned, the probability is doubled and the query is run again. This happens iteratively until either the deadline is exceeded or a non-null result set is available. Notice that because climbing always issues another query if there is time left, it may overshoot the deadline.

The third technique is *estimation*. Estimation is similar to climbing except that we predict the next query time from the previous query times and then only issue the next query if there is sufficient time remaining. Hence, it is far less likely to overshoot the deadline. Predicting query time in general is a complicated problem that could involve hardware configuration modeling, scheduler modeling, modeling of the database engine, and query analysis. Surprisingly, predicting query time from previous instances of a nondeterministic query run with lower selection probabilities appears to be easier to solve.

We studied several functions (linear, power, polynomial, exponential) for mapping from selection probability to running time. Degree two polynomials worked best for the queries described in the previous section. In our implementation, we monitor each query's time and selection probability. After the first query, we estimate the second query time to be the same as the first. After the first two queries, we do a degree one Lagrange interpolation to estimate the third query time. For the fourth and further queries, we estimate the next query time by applying a degree two Lagrange interpolation polynomial to the previous three query times. Hence, after the first query, we have some model that maps from selection probability to query time. We then use that model to predict if we still have enough time to do next query. If we don't, we terminate. Figure 18 shows this process for the two host query described in the next paragraph, comparing the predicted and actual iterative query times.

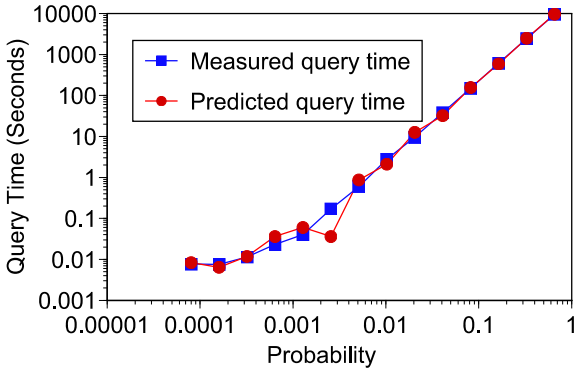
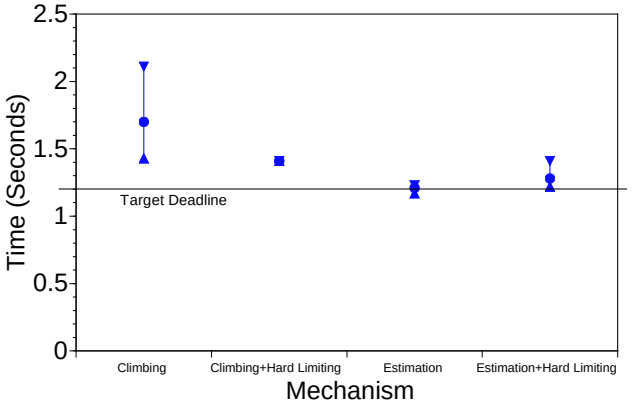


Figure 18. Prediction accuracy in estimation.

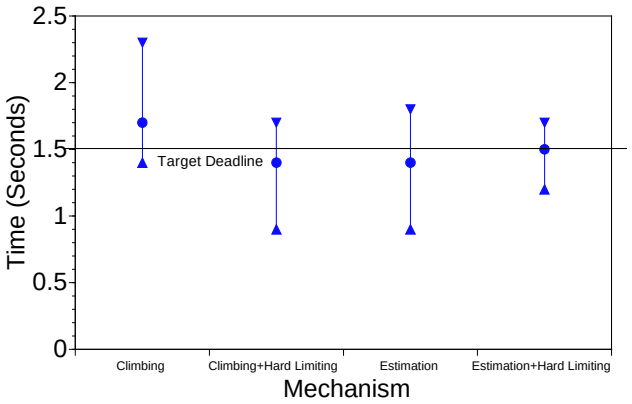
Because no time limit is supplied, the query runs until the selection probability approaches one. This prediction algorithm works very well for the two and four host queries and other similar joins, but has difficulties with larger joins.

Figure 19 illustrates the performance of these different techniques for a sample query. The query looks for two hosts with a combined total of 600 GB of main memory in a 50,000 host database. Such a combination is very rare, but possible, hence the running time would be quite high for a deterministic version of the query. The only difference between (a), (b), and (c) is the deadline, 1.2, 1.5, and 60 seconds, respectively. Each query is run five times. The Figure illustrates the average, minimum, and maximum running time. Clearly, it is possible to keep the running time close to the deadline using the three techniques. This is also the case for queries that are allowed to run longer, and for queries involving a larger number of hosts. Surprisingly, hard-limiting can lead to missing the deadline by about 0.2 seconds in our system, and this delay is constant in all our experiments. This is largely because terminating a query process can be expensive. Estimation proves effective for four host queries, but has greater difficulty for eight host queries. Currently, we combine hard-limiting with estimation for eight-way and higher joins.

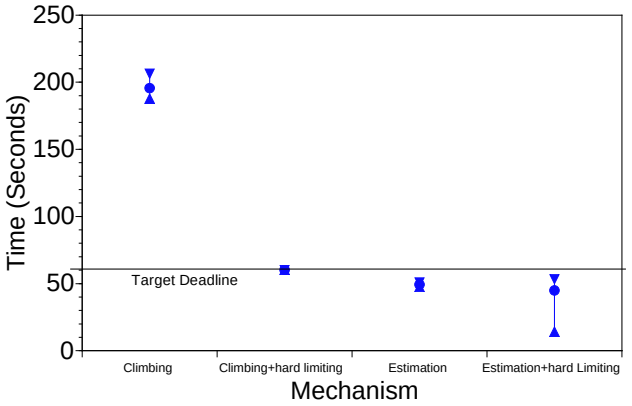
We are also considering a more complex technique in which we derive an analysis query from the user query. The analysis query essentially tests the distribution of significant attributes (e.g., the host memory for our examples) in the input tables, and then estimates the likelihood of these attributes coinciding, assuming that they are randomly distributed. The selection probability would then be set sufficiently higher than this number to insure that a row will be returned with high likelihood. The query would then be run until what is left of the time limit is past.



(a) 1.2 second deadline



(b) 1.5 second deadline



(c) 60 second deadline

Figure 19. Initial evaluation of techniques for time-bounding nondeterministic queries.

7 Conclusions

We described the RGIS relational grid information service system, focusing on nondeterministic queries, the RGIS mechanism for limiting the running time of queries and their load on the RGIS server. Nondeterministic queries

are implemented using a combination of query rewriting, schema extensions, indices, and randomness. No changes to the RDBMS are needed. We evaluated the performance of our implementation, populating our database with networks as large as five million hosts. The evaluation showed that a meaningful tradeoff between query processing time and result set size is possible using nondeterministic queries, and that we can use that tradeoff to keep query running time largely independent of query complexity. We then discussed three techniques that we use to time-bound nondeterministic queries and evaluated their performance.

The next major step for RGIS is the integration of the content delivery network scheme for loose replication of RGIS servers described in Section 3 and an evaluation of its effectiveness. We hope to have a release of RGIS available soon at the following URL: http://www.cs.northwestern.edu/∼urgis.

Acknowledgements

We would like thank Andrew Weinreich, who took the lead in developing the RGIS web and SOAP interfaces, and did a fantastic job.

References

[1] ADJIE-WINOTO, W., SCHWARTZ, E., BALAKRISHNAN, H., AND LILLEY, J. The design and implementation of an intentional naming system. In *Proceedings of the 17th ACM Symposium on Operating System Principles* (December 199).

[2] ALBITZ, P., AND LIU, C. *DNS and BIND*. O'Reilly and Associates, Inc., Sebastopol, California, 1992.

[3] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 107–117.

[4] CAREY, M., AND KOSSMANN, D. On saying "enough already!" in sql. In *ACM SIGMOD Conference* (1997).

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

[6] 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.

[7] DINDA, P. A., AND O'HALLARON, D. R. An extensible toolkit for resource prediction in distributed systems. Tech. Rep. CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

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

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

[10] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003). To Appear.

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

[12] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[13] FOSTER, I., KESSELMAN, C., NICK, J., AND TUECKE, S. Grid services for distributed system integration. *Computer 35*, 6 (2002).

[14] FOSTER, I., KESSELMAN, C., AND TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications 15* (2001).

[15] GLOBAL GRID FORUM. Global grid forum web site. http://www.gridforum.org.

[16] IBM INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION. *Understanding LDAP*. IBM Corporation, 1998.

[17] INTERNATIONAL TELECOMMUNICATION UNION. Information technology – open systems interconnection – the directory: Overview of concepts, models, and services, August 1997.

[18] LAMPSON, B. W. Designing a global name service. In *4th ACM Symposium on Principles of Distributed Computing* (August 1986).

[19] 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.

[20] LOWEKAMP, B., O'HALLARON, D. R., AND GROSS, T. R. Topology discovery for large ethernet networks. In *Proceedings of SIGCOMM 2001* (August 2001).

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

[22] LU, D., DINDA, P. A., AND SKICEWICZ, J. A. Scoped and approximate queries in a relational grid information service. In *Proceedings of the 4th International Workshop on Grid Computing (Grid 2003)* (November 2003). To Appear.

[23] OBJECT MANAGEMENT GROUP. The common object request broker: Architecture and specification (version 2.3.1). Tech. rep., Object Management Group, 1999.

[24] OLKEN, F. *Random Sampling from Databases*. PhD thesis, University of California, Berkeley, 1993.

[25] 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).

[26] 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.

[27] 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.

[28] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* (2001).

[29] 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.

[30] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001* (2001), pp. 149–160.

[31] TAN, K.-L., GOH, C. H., AND OOI, B. C. Query rewriting for SWIFT (first) answers. *IEEE Transactions on Knowledge and Data Engineering 12*, 5 (Sept/Oct 2000).

[32] THE OPEN GROUP. *DCE 1.2.2: Introduction to OSF DCE*. The Open Group, September 1997. http://www.opengroup.org/pubs/catalog/f201.htm.

[33] THEIMER, M., AND JONES, M. B. Overlook: Scalable name service on an overlay network. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)* (July 2002).

[34] TRANSACTION PROCESSING COUNCIL. Tpc benchmarks. http://www.tpc.org.

[35] VAHDAT, A., DAHLIN, M., ANDERSON, T., AND AGGARWAL, A. Active names: flexible location and transport of wide-area resources. In *USENIX Symposium on Internet Technology and Systems* (October 1999).

[36] VEIZADES, J., GUTTMAN, E., PERKINS, C., AND KAPLAN, S. Service location protocol. Internet RFC 2165, June 1997.

[37] WALDO, J. The Jini architecture for network-centric computing. *Communications of the ACM 42*, 7 (1999), 76–82.