Understanding Grid Resource Information Management through a Synthetic Database Benchmark/Workload

Beth Plale, Craig Jacobs, Scott Jensen, Ying Liu, Charlie Moad, Rupali Parab and Prajakta Vaidya

Computer Science Department
Indiana University
Bloomington, Indiana

Abstract

Management of grid resource information is a challenging and important area considering the potential size of the grid and wide range of resources that should be represented. Though example Grid Information Servers exist, behavior of these servers across different platforms is less well understood. This paper describes a study we undertook to compare the access language and platform capabilities for three different database platforms, relational, native XML, and LDAP, serving as a grid information server. Our study measures query response times for a range of queries and highlights sensitivities exhibited by the different platforms to variables such as result set size and collections size.

1 Introduction

The Grid [10] is a new paradigm for wide area distributed computing wherein services are organized as web services that can be flexibly and dynamically allocated and accessed, often to solve problems requiring resources that span multiple administrative domains. An important capability of grid middleware is access to and discovery of resources consisting of clusters, file systems, users, virtual organizations, web services, and databases [6, 13] that are geographically disbursed and under the administration of diverse groups. Descriptions of resources must be made available to clients wishing to discover services [20], data sources, and computational resources [6]. Data distribution of resource information can range from every resource being responsible for responding to client queries (a highly distributed model), to a host of small grid information servers and registries, to one central grid information server that maintains information on every service or resource for the grid (highly centralized model.) For purposes of this study, our model is of distributed grid information servers (GIS) where a single server stores and serves resource information for a single administrative domain. This localized control is highly desirable as evidenced by the huge success of DNS.

Grid information servers evolved out of directory services, but the GIS has requirements that make traditional directory service solutions inappropriate. These consist of the following:

- GIS servers are distributed,
- The organizing paradigm of the grid is web services, so the lingua franca is XML,
- Resources continually change state, and the descriptions of these resources must be kept up-to-date,
- Providers, those who make updates to resource information, are in close proximity to the GIS,
- Clients, those who seek resource information, will often be a significant distance away from the GIS they access,
- Client queries against a GIS can be complex, and
- Access to data must be controlled.

A grid contains geographically distributed resources. Thus it is reasonable to expect that information describing the resources is distributed as well. The web services paradigm organizes the grid into components conforming to the web services model of standardized portTypes, WSDL descriptions, support of the SOAP communication protocol, and message encoding using XML. In order to participate on the grid, grid information servers must conform to the web services paradigm as well.

The defining characteristics that distinguish a GIS from a directory service are the freshness demands on the data and the richness of the data model. Resources such as hosts (*i.e.*, workstation, cluster node) have aspects of their description that change at millisecond rates. For instance, the UNIX kstat command samples current CPU load and available virtual memory once per second. The update rates for these two attributes to a GIS holding information about 30,000 hosts is 30,000 updates/second. The

GIS data model more closely resembles a database management system than a directory service. The data model consists of several dozen entities and rich relationships between them, resulting in client queries that can be complex and time consuming.

In a widely distributed Grid spanning multiple administrative domains, it is a reasonable assumption that an GIS server will serve the resources within an administrative domain. Thus resource owners, providers who publish into the GIS, are located physically close to the GIS that serves them. The update requests issued by providers are generally simple (not complex transactions.) Consumers of resource information, on the other hand, are clients hunting for resources to use. These clients can often be remote to the GIS. The widely disbursed client population means that network latency is an important component of perceived response time of the server.

In order to better understand resource information representation and retrieval in grid computing, we have developed a grid-specific synthetic database benchmark/workload for grid information servers. The benchmark/workload is a set of queries and scenarios developed against a data model of grid resources that extends the Global Grid Forum [1] (GGF) proposed GLUE schema [7]. The database schemas for the three platforms are derived from this single data model using well-defined derivation rules. The databases are populated with synthetic but realistic data about grid resources. The data are realistic in terms of the types of entities represented, the proportions of the entities to one another, and attribute values that describe the entities. For instance, the database holds information about 60 clusters, 388 subclusters, 20 users, and 33605 hosts. The *queries* test a broad range of database functionality while asking realistic questions. The *scenarios* are short synthetic workloads that test query response time of a repository under a workload of concurrent query requests and update requests.

The contribution of this paper is a comparison of the abilities of three different platforms to support the unique needs of a grid information server. Through a measure of response times under a standard set of queries and scenarios, we illuminate platform sensitivities, or weaknesses. A second dimension that our study attempts to capture is ease of use, that is, a quantification of the performance impact of access language on the client, the network, and the GIS server itself.

The paper is organized as follows: immediately following is a discussion of related work. Section 3 introduces the data model underlying the synthetic database benchmark/workload and discusses the study controls applied starting with a single UML data model. Section 4 introduces the benchmark/workload. In Section 5 we discuss the performance evaluation and observations that can be drawn from the experiment. Section 6 offers concluding remarks and a glimpse at future work.

2 Related Work

A number of studies have examined aspects of performance of the Grid Information Server. The definitive reference implementation has been Globus MDS [6]. MDS2, the version of MDS used in this study, is organized as a hierarchical architecture of lower level Grid Resource Information Servers (GRIS), connected to one or more higher-level index servers (GIIS). Multi-layer hierarchies of 4-5 levels are not uncommon [2]. MDS2 and earlier versions employed the LDAP data model and protocol. The recent release of the Globus Toolkit, GT3, a reference implementation for the Global Grid Forum OGSA [19] grid services architecture, distributes management of grid resource information between MDS3, the registries, and discovery services like UDDI [20]. The burden of Information management is also shifted onto the resource itself, which is responsible for providing a web service interface that exports the information about itself.

Smith [17] examined MDS2 performance for different versions of LDAP. The work predates MDS' hierarchical GRIS/GIIS architecture so the main result of the paper is exposing LDAP's poor performance under even minor update loads. Aloisio *et. al* [3] studied the MDS2 grid information server and conducted experiments focused on simple tests of the Grid Index Information Service (GIIS). Schopf [23] examined the scalability of three information servers, MDS2, RGMA [9], and Hawkeye of the Condor system [4], all three of which are tightly coupled to monitoring systems from which they obtain input data. These systems were subject to scalability testing under increasing user loads and used a single, simple query used is throughout. Schopf's work complements ours; ours is focused on a broad set of queries and scenarios issued against a rich data set.

The University of Wisconsin benchmark [5] evaluates the performance of off-the-shelf relational database management systems. It was applied against a relatively small database of synthetic tables and data, and queries were designed solely to test features of the database. The Transaction Processing Council's decision support benchmark (TPC-H) [15] is an application specific benchmark for decision support. It is closest to ours in spirit, but does not capture the unique dynamic needs of a grid information service. The XMark project [16] provides a framework to assess the abilities of an XML database.

3 Data Model

Our study compares the strengths and weaknesses of GIS servers implemented on various platforms. The experiment controls are applied from the first step, with derivation of the schemas from a common data model, and are applied consistently thereafter. Controls include the following:

- Single UML data model – schemas for the three platforms derived from single data model,
- Replicated data – data population of the three databases daisy chained from single script,
- Reproducible database population process – reproducible number of tuples created per object and across platforms, and
- Meaningful data - associations between objects and cardinalities of the associations are meaningful in the context of a GIS.

The starting point of our study is a data model of the GIS. A data model is an abstract representation of entities and their relationships. The UML [18] model and Extended Entity Relationship model are the two most widely used data modeling notations. Our data model is an October 2002 snapshot of the GLUE data model [7] defined by the Global Grid Forum [1]. We derive the schemas for the native XML and relational platforms using well defined mapping rules [21]. The data schema of the LDAP platform, on the other hand, was already provided by the GLUE group. We strove for consistent and faithful representation of the data model across all platforms. If entities were represented separately in the LDAP schema, we kept them separate across the other platforms despite query performance gains that might result from other organizations. For instance, rolling memory and CPU information into the Host table would likely yield improved performance across all platforms because it would reduce the number of joins.

The three databases contain identical data derived from data dumps of MDS2 servers gathered between November 2000 and January 2002. This is accomplished through a daisy chained data population process. Specifically, the relational database is populated from a script, then Xindice is populated by a program that reads a dump of the mySQL database. Similarly, MDS2 which requires population by means of a file in LDIF format, is also populated by a program that parses the mySQL dump.

The data in the database is meaningful and the cardinalities reflect relationships that could exist in a real grid. For instance, a host belongs to a small number of subclusters and a subcluster has one or more member hosts. Hosts also have one or more endpoints (*i.e.*, network interface cards (NICs)), and endpoints can have one or more active network connections. We configured the associations so that a smaller number of network connections exist within the subcluster while a larger number of connections exist between subclusters and clusters. One of the benchmark queries is a transitive query, that is, it looks for a path of distance three (edges) between hosts. This query finds roughly 13000 paths at that distance in a connection table of roughly 12000 entries.

Terminology used to refer to entities in a data model is very dependent on the implementation. As shown in Table 1, what is known as a 'table' in a relational database is referred to as an 'entry' in LDAP and a 'collection' in Xindice. Whereas an individual instance or member of a relational table is called a 'tuple', in LDAP it is called an 'object', and in Xindice it is called a 'document'. For purposes of this paper, we use the term 'collection' to refer to a collection of instances, and 'object' to refer to a data element. These are shown italicized in the table. We refer to the fields that describe a member as *attributes.*

	Relational	*LDAP*	*Xindice*
collection level	table	entry	*collection*
member level	tuple	*object*	document

Table 1: Terminology used in different data models.

Each of the three database platforms hold the same entities and relationships. The number of instances of entities and relationships is also held constant across the three platforms. A database platform contains 34 entities/relationships and 81684 instances. In following with the standard adopted by the GLUE schema, a relation between two entities is represented as a separate collection. The distribution of instances among the entities is shown in Table 2 for the major collections.

Collection	*Number of Objects*
Cluster	20
UserAccounts	60
ComputingElement	106
Subcluster	345
Application	600
Connection	12200
Host	29743

Table 2: Object distribution for the major collections.

4 Benchmark/Workload

The kinds of queries and updates issued against a grid resource repository can vary widely, limited only by the user's knowledge of the query language, the expressiveness of the query language, and limitations of the underlying implementation. Our goal is a set of queries that when

taken as a whole exercise several orthogonal axes: simple queries versus complex, queries that test specific search support versus those that could be posed by a user, sequential versus concurrent access; small return set versus large. The intent is to provide a synthetic database workload that is both broad and representative of actual workloads so as to accurately assess the strengths and weaknesses of different grid resource information repositories. This section introduces the benchmark/workload.

The synthetic database workload consists of 13 queries, one update, and four scenarios. The queries/updates are grouped into five major categories: scoping, index, join, selectivity, and base operations; the grouping is for purposes of ease of understanding. Over half the queries are paired for purposes of testing the presence or absence of a feature (*e.g.*, an index). Variables not subject to testing are tightly controlled across the pairs. Because of constraints on paper length, the queries are summarized briefly below. Full details of the benchmark can be found in [14].

Scoping. We use the term *scoping* to define queries that limit their search to a particular subtree. Intuitively scoped queries should yield better response times than their non-scoped counterparts in hierarchical databases.

Indexing. Query response times are often dramatically improved when indexes are used. Indexes provide fast access for queries that request indexed attributes. Our query set includes one index pair, that is a pair of queries wherein the independent variable is whether or not the requested value is indexed.

Selectivity. The selectivity of a query is the number of objects returned. According to DeWitt [5], coverage of the performance domain can be achieved with queries that return 1 tuple, 1% of tuples, and 10% of tuples. These queries execute over the Connection table which contains information about 12200 active network connections.

Joins. Joins occur when a user requests information that resides in more than one table. Our query benchmark includes three non-paired join queries: the first queries over six collections; the second query is a realistic job submission query, specifically, a user is seeking a subcluster wherein the needed software environment exists, the job owner has an account, and the binary is resident. This latter query might be posed by a scientist desiring to find a specific set of nodes on which her binary can and is allowed to execute. The final query is a recursive and transitive query. The user searches for all network connection endpoints reachable from a starting network connection endpoint in exactly three hops.

Other. The two final queries test connection speed and row modification times. request and update request. The former connects to the database and immediately disconnects. The latter updates a single attribute in a set of objects that match a particular condition.

Scenarios. A scenario is a synthetic workload of queries and updates issued over a controlled time duration. The purpose of scenarios is to expose the sensitivity of query response time to update rates. We noted earlier that the dynamic nature of grid resource data is a key characteristic of grid information servers. A scenario is scripted as follows: in Phase I, a number of concurrent clients are started that repeatedly issue a blocking query request to the database. In Phase II, update clients are added, these execute concurrently with the query clients for the duration of phase II. The query clients then continue alone in Phase III. This final phase captures any lingering effects the updates may have.

The total scenario execution time is configurable. For the results presented here, a scenario runs for a duration of 10 minutes. Query response time is sampled every 20 seconds, at time t_i where $0 < i < 1200$ and $i \ mod \ 20 = 0$. Query response time is defined as:

t_0: query response time at t_0 is 0 ($t_0 = 0$)

$t_1 - t_n$: query response time for t_i, i ! = 0, is the average of query responses received in the interval $(t_{i-1}, t_1]$. If no query responses received in the interval, then

$$\mathsf{queryResponseTime}(\mathsf{t_i}) = \mathsf{queryResponseTime}(\mathsf{t_{i-1}})$$

When no query responses are received over an interval, QRT at the interval end is the QRT of the previous interval. This case occurs in complex queries over Xindice and MDS2 where a 20 second interval may not be large enough to capture even a single query response.

The benchmark consists of a set of scripts for each database. The mySQL scripts are issue SQL queries and are written in Perl. They communicate with the database using the mySQL C API. The Xindice queries are written in XPath. The scripts are written in Java and interface with Xindice using XML:DB. The MDS2 queries are written in the LDAP query language. The scripts are written in C and interface to MDS2 using the LDAP protocol.

5 Performance Evaluation

Our comparative assessment of different platforms is done by means of executing the query benchmark/workload on realistic GIS servers implemented on relational, native XML, and LDAP platforms.

The purpose of the benchmark is to enable comparison of three popular database platforms as host environments for a GIS. Through a measure of response times under a standard set of queries and scenarios, we illuminate platform sensitivities where *sensitivity* is a measure of change in QRT in response to change in a variable such as collection size. We expose sensitivities as a guideline to the community in the design of such systems. Sensitivities

exposed by the queries are in collection size, database size, and size of result set. The scenarios expose sensitivities to concurrent activity, particularly the impact on query response time of insertion and modification activity. This behavior is important to understand since data freshness is a key requirement in a GIS.

The performance evaluation described in this paper is of one GIS server. We are looking at distributed performance in ongoing work. Because we are comparing vastly different platforms, we do not microbenchmark the platforms. Microbenchmarking of a database platform is outside the scope of our study. Our focus is on an apples-to-apples performance comparison of GIS servers implemented on different platforms. The results we obtain across the platforms differ by several minutes or greater. Thus though we carefully configured each database to its best advantage to take advantage of the hardware resources and index support, we do not optimize the platforms further because with that large a difference, a few microseconds performance improvement would have no effect on the final outcome.

Clients of GIS servers are other grid services, portals, and agents acting on behalf of a client. Unlike data providers, clients are often remote, potentially on other continents. A second dimension of performance that our study attempts to capture arises out of this remoteness of clients. Retrieving the right information from a GIS can be thought of as a dialog between a client and a GIS. The more interaction required to satisfy a request, the higher the load on the GIS. In the case of database queries, it also means higher network bandwidth needs because the GIS returns data with every request. When the query language of the database is rich, in general the dialog is shorter, in fact often a single query, and the result set is exactly and only the data that meets the user's needs. When the query language is more restricted, what results is a series of questions being asked of the database, with the database returning a result set for each question asked, and the client having the responsibility of filtering and manipulating that data to get the final results set. For instance, the jobSubmit query is formulated in SQL as a single query. The result set is 34 byes of precisely the data the user wants. That same query written in XPath and executed on Xindice takes 6 queries to the Xindice database. In that time Xindice responds with a total of 13KB of data from which the client must have 157 lines of triple-nested looping code to process [14].

To capture this less tangible distinction in platforms, we define an *ease of use* metric that attempts to quantify the total impact of access language of a GIS server on the client, the network, and the GIS server itself.

The architecture of the study consists of three database platforms: mySQL 4.0, Xindice 1.1, and MDS2 2 (GT 2.2) on a dual processor Dell Poweredge 6400 Xeon server, 2GB RAM, 100 GB Raid 5, RedHat 7.3. The client is co-located on the dual processor server. Each database is implemented as a standalone server; client access to the server is through the C API for mySQL and LDAP, and XML:DB for Xindice. MySQL is configured with the InnoDB back end for row-level locking. Xindice 1.1 is a native XML database that is bootstrapped from an Apache Tomcat server. MDS2 is configured as a single GRIS talking to a single GIIS with GRIS and GIIS co-located on the same dual processor server. All queries are issued as against the GIIS by means of the anonymous query mechanism. This mechanism bypasses the security checking overhead that would be incurred otherwise. The MDS2 results in this paper are for fully cached data, and interpolated from measurements taken on database sizes of 5%, 10%, and 25% of the full size. The interpolation assumes a linear relationship from which an average slope is obtained. Instabilities in MDS2 caching in the GIIS precluded our taking measurements on the full sized database and also foiled our assiduous efforts at capturing non-cached results.

5.1 Query Response Time

Query response time is a measure of the amount of time it takes for a server to complete a query request and return the result set. For the queries in the benchmark (not scenarios), since query scripts are non-threaded and blocking, a script executes one query at at time. The query response times, captured in Figure 1, show results organized by the query groups described in Section 4. Listed across the X-axis of each are the individual queries and their results for the three different platforms. If a query is part of a binary pair, its pair resides to its right and is prefaced with 'non'. The Y-axis plots query response time in milliseconds. It is important to note that the Y-axis scale is logarithmic.

Scoping appears to achieve its effect when collection size is large, as it is for scopedHosts and nonScopedHosts. But this result is not achieved for small collections, as when comparing "scoped" and "nonscoped" queries for Xindice. Any gain from scoping is overshadowed by additional response time overhead of answering multi-request queries. "Nonscoped" is implemented as three separate XPath queries while "Scoped" is implemented as two separate queries. The impact on Xindice performance of multi-request queries is explored more deeply in [22].

A comparison of indexed and nonIndexed queries for relational and XML platforms reveals the obvious conclusion that indexes greatly improve performance. The interesting results however are comparative across platforms. Indexed Xindice results are still nearly an order of magni-

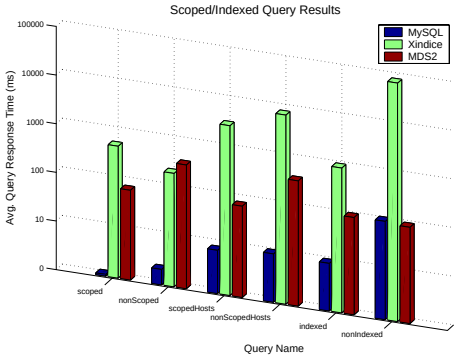

Figure 1: Query response times for six of the benchmark queries. Grouping is by pairs: scoped/nonScoped, scopedHosts/nonScopedHosts, indexed/nonindexed.

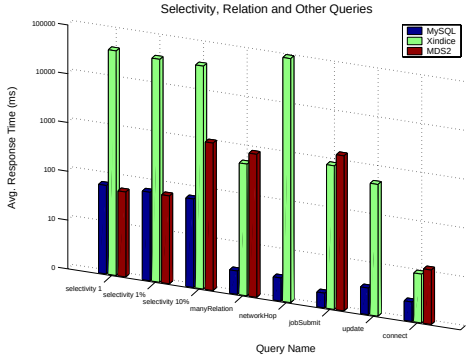

Figure 2: Query response time for remaining eight queries in the benchmark.

tude slower than non-indexed mySQL results. MDS2 displays no sensitivity to indexes because MDS2 has overridden the native openLDAP [12] index support and replaced it with a caching scheme deployed in the GIIS. Due to instabilities in the caching scheme wherein either the entire database had to be cache resident or none of it was, we were unable to evaluate the impact of cached versus non-cached data on QRT.

Selectivity is defined as the number of objects satisfying a query. As shown in Figure 2 we see that for neither mySQL nor Xindice is QRT sensitive to the number of objects returned. MDS2, on the other hand, shows sensitivity to return set size. The "manyRelations" and "jobSubmit" join queries measure a repository's ability to assemble a result from numerous collections. The results of the two queries are roughly the same, which is reasonable since both queries touch the same number of collections. The transitive query, "networkHops", on Xindice was unable to complete a single iteration in the 100,000 ms cutoff. Transitive queries are known to consume significant resources even over indexed fields such as this. *The MDS2 result for this query is forthcoming.*

In comparing the join and selectivity results for Xindice one can see that the joins finish in approximately 1000 ms whereas selectivity queries complete in roughly 38,000 ms. The reason for this is that the collection over which the selectivity queries is large. These results substantiate our observation that Xindice is quite sensitive to collection size.

The "update" times shown are for simple, one-attribute updates. They thus establish an upper bound on the rate at which a database can accept updates. For mySQL this rate is 200 updates per second whereas for Xindice the rate is 1.3 updates per second. The low update rate for Xindice is an overriding factor our conclusion that Xindice is generally inappropriate as a grid information server. The udpate rate for mySQL is disturbingly low as well. In MDS2, traditional LDAP updates have been disabled and replaced by a periodic triggering of scripts that pipe results to the MDS2 GRIS through standard output. Thus our query is unable to capture update rate for MDS2.

Scenarios. The *scenarios* are scripted synthetic workloads designed to capture the sensitivity of query response time to update load. We discuss the results of one of the four scenarios in this paper and refer the reader to [14] for the remainder. A scenario, shown in Figure 3 begins in Phase I with the execution of three concurrent query streams. Phase II begins at minute minutes into the run. Update threads are started up and run for the duration of Phase II. For mySQL ten to fifty update threads are started, whereas for Xindice, given its exhibited poor update rate, only three update threads are run. In Phase III the query threads allowed to run through until the end of the run. For MDS2, we forced updates for purposes of the scenarios by issuing a client query that queried a single attribute that has a short time to live. In that way, every time the query is executed, it forces the value to be updated through the provider mechanism (*i.e.*, LDIF file piped to standard output) described in paragraph three of Section 5.

The impact of update streams is observable in all three cases. MySQL average query response times are 2-10 millisecond range when no update threads are running, and increase to 10-100 milliseconds under the update load. Xindice varies between 1000-2000 milliseconds when no update threads are running and 2000-4000 milliseconds when they are. MDS2 results fall between that of mySQL and Xindice, but the reader is asked to note that the MDS2 scenario is run on a database that is significantly smaller (95% smaller) than the database size used for the other platforms. The failure of the MDS2 curve to taper down at the end of the scenario could be due to a lingering effect of rapid cache refresh being triggered for the

particular attribute after the query request has terminated.

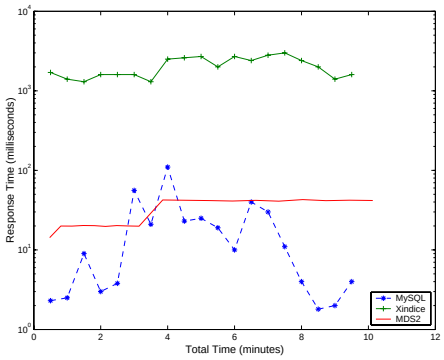

Figure 3: Scenario1: Average query response time over 10 minute duration subject to concurrent update streams during minutes three through seven.

Discussion. Based on our observations, overall mySQL outperforms the other platforms. It also exhibits the least perturbation to QRT in the face of a mild update load. But mySQL suffers in that it does not have an XML interface. It could gain an XML interface and a grid services interface as well by means of the OGSA-DAI [8] data access and integration framework from the UK e-Science Institute. In accessing mySQL through OGSA-DAI, one then incurs an additional overhead of 100-700 ms [11]. OGSA-DAI introduces an additional sensitivity in return set size. Specifically, when the size of the return set exceeds about 10K, due to limitations in the SOAP protocol the client must use the asynchronous delivery mechanism provided in OGSA-DAI to get the result set back from the server. Asynchronous delivery adds a couple orders of magnitude additional overhead to QRT. Xindice has a built-in XML interface, which is its strongest advantage. But Xindice generally performed very poorly relative to the other platforms, and is highly sensitive to collection size and the presence or absence of indexes. MDS2 results on a small database and fully cached results is generally good. LDAP, however, performs extremely poorly under high update loads [17]. MDS2 exhibits sensitivity to database size and return set size. QRT response time appears to grow exponentially with respect to database size. We also observed caching instabilities at all database sizes.

5.2 Ease of Use

Ease of Use attempts to quantify the impact of access language for a GIS server on the client, on the network, and on the GIS server itself. The metrics for quantifying ease of use are total number of bytes returned to the client and number of queries required to retrieve the requested data. These numbers, shown in Tables 3 and 4, are not independent. Database platforms requiring more queries to obtain data correspondingly return a larger number of bytes.

Description	*mySQL 4.0 (KB)*	*Xindice 1.1 (KB)*	*MDS2 (KB)*
scoping	0.4 - 46.0	7.5 - 549.5	*5.6 - 47.3*
indexing	9.5 - 11.0	139.8 - 140.9	*9.6 - 24.0*
selectivity	0.04 - 52.9	0.48 - 691.0	*0.03 - 267.0*
joins	0.03 - 0.03	40.1 - 131.8	*0.98 - 1.9*

Table 3: Minimum and maximum number of bytes returned per query group. The MDS2 numbers are estimated from smaller database sizes so are shown in italics.

Description	*mySQL 4.0*	*Xindice 1.1*	*MDS2*
scoping	1	3	3
indexing	1	2	1
selectivity	1	1	1
joins	1	6	5

Table 4: Maximum number of queries issued to database per higher-level query.

Discussion. Less expressive query languages, such as LDAP and the subset of XPath implemented by Xindice, have several drawbacks. They impose an additional load on the GIS server by forcing multiple message exchanges with a remote client to satisfy a request that can be written in the more expressive languages as a single query. They impose additional load on the network by larger bandwidth consumption between server and remote client. Finally, they impose query processing obligations on the client in the form of code needed to process the partial results returned through the multiple message exchanges. Our ease of use results indicate that a richer access language, such as SQL or XQuery, is a better choice for a grid information server. A rich language, however, introduces the challenge of excess resource consumption by a long running query in a multi-user setting.

6 Conclusion

This study contributes toward a deeper understanding of the impact of query language and database platforms on the performance of a grid information server. Through a synthetic benchmark set of queries and synthetic workloads, we expose the sensitivities of the individual platforms with the goal of influencing future design decisions.

There is more to the assessment of a grid resource information repository than that which is conveyed by our metrics, however. The performance implications of distributed GIS servers, and controls on excess resource con-

sumption of long running queries are interesting issues that form the basis for future work. The IU RGRbench benchmark is slated for release November 2003. See http://www.cs.indiana.edu/ plale/projects/RGR/.

References

[1] Global grid forum. http://www.gridforum.org, 2003.

[2] Rob Allen. CLRC HPCGrid services portal. http://esc.dl.ac.uk/GridWG, 2003.

[3] G. Aloisio, M. Cafaro, I. Epicoco, and S. Fiore. Analysis of the globus toolkit grid information service. Technical Report Technical Report GridLab-10-D.1-001-GIS_Analysis, GridLab Project, 2001. www.gridlab.org/Resources/Deliverables/D10.1.pdf.

[4] Jim Basney and Miron Livny. *High Performance Cluster Computing*. Prentice Hall PTR, 1999.

[5] Dina Bitton, David J. DeWitt, and Carolyn Turbyfill. Benchmarking database systems: A systematic approach. Technical report, University of Wisconsin, Computer Sciences Dept., Madison, Wisconsin, 1983.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, August 2001.

[7] DataTAG. Glue schema: common conceptual data model for grid resources monitoring and discovery. http://www.cnaf.infn.it/ sergio/datatag/glue, 2003.

[8] e Science Institute. Open grid services architecture data access and integration (OGSA-DAI). http://www.ogsadai.org, 2003.

[9] Steve Fisher. Relational model for information and monitoring. In *Global Grid Forum, GWD-Perf-7-1*, 2001.

[10] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[11] Deepti Kodeboyina and Beth Plale. Experiences with ogsa-dai: Portlet access and benchmark. In *Global Grid Forum Workshop on Designing and Building Grid Services*, September 2003.

[12] OpenLDAP Organization. OpenLDAP. http://www.openldap.org, 2003.

[13] Beth Plale, Peter Dinda, and Gregor von Laszewski. Key concepts and services of a grid information service. In *ICSA 15th International Parallel and Distributed Computing Systems*, September 2002.

[14] Beth Plale, Craig Jacobs, Ying Liu, Charles Moad, Rupail Parab, and Prajakta Vaidya. Benchmark details of a synthetic database benchmark/workload for grid resource infrmation. Technical Report Technical Report TR-583, Computer Science Dept., Indiana University, 2003.

[15] M. Poess and C. Floyd. New TPC benchmarks for decision support and web commerce. *ACM SIGMOD Record*, 29, December 2000.

[16] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A benchmark for XML data management. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 974–985, Hong Kong, China, August 2002.

[17] Warren Smith, Abdul Waheel, David Meyers, and Jerry Yan. An evaluation of alternative designs for a grid information service. In *IEEE International High Performance Distributed Computing (HPDC)*, August 2000.

[18] Rational Software. Conceptual, logical, and physical design of persistent data using UML. www.rational.com/uml/resources/whitepapers/index.jsp, 1998.

[19] S. Tuecke, K. Czaijkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, and P. Vanderbilt. Open grid services infrastructure, version 1.0. In *Global Grid Forum GWD-R*, March 2003.

[20] UDDI.org. Universal description, discovery, and integration (UDDI). http://www.uddi.org, 2003.

[21] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice Hall, 1997.

[22] Prajakta Vaidya and Beth Plale. Benchmark of xindice as a grid information server. Technical Report Technical Report TR-585, Computer Science Dept., Indiana University, 2003.

[23] Xuehai Zhang, Jeffrey L. Freschl, and Jennifer M. Schopf. A peformance study of monitoring and information services for distributed systems. In *IEEE International High Performance Distributed Computing (HPDC)*, 2003.