

# Resource Information Management in Grid Middleware: Evaluation of Multiple Platforms with a Benchmark/Workload

Beth Plale

Computer Science Department  
Indiana University  
Bloomington, Indiana

## Abstract

Grid computing is a new paradigm for wide area distributed computing that is based web services. A challenge in the distributed middleware that implements a grid envisioned to span the world, is the management of information about the resources available to the Grid. Management of grid resource information shares much in common with directory services, in that both serve client requests for resource information. Directory servers, however, primarily serve to map host name to IP addresses. As such, they are optimized for relatively static data, simple and small set of known queries, and extremely short query execution times. Grid resource information management is distinctly different in a number of challenging ways.

Though aspects of grid resource information management have been investigated, the contribution that an underlying database platform makes towards meeting system requirements and achieving overall performance is less well understood. This paper describes an experimental study we undertook to better understand these issues. We identify the key requirements of a grid resource information server and from the requirements derive a set of metrics for evaluating a platform. Using a database of realistic grid resource information implemented on three platforms of different data models, (*i.e.*, relational, native XML, and LDAP), we run a benchmark set of queries and workload scenarios. The queries test a broad range of database functionality through realistic questions asked on meaningful data. The scenarios are

short synthetic workloads that test query response time of a repository under a workload of concurrent query and update requests. Based on the results, we illuminate the strengths and weaknesses of a platform in its contribution to the requirements of a grid resource information manager.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Computer Systems Organization**]: Performance of Systems – Performance attributes; H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms: Experimentation, Performance

Additional Key Words and Phrases: grid resource information management architecture

## 1 Introduction

Grid computing [17] is a new paradigm for wide area distributed computing. The Grid itself is a distributed middleware layer organized as web services. Web services form the basis for intercommunication amongst services within the middleware layer and also define the interface to applications. The Grid's strength is open extensibility in that new services can be flexibly and dynamically added by third parties.

Critical functionality in a grid is management of information about resources existing on the grid. We define a *grid resource* as an entity that contributes value to the grid and must be managed. Grid resources are akin to the system resources of a computer that must be managed by an operating system. Grid resources include web services, hosts, file systems, databases, large scale instruments, libraries, and large-scale projects [13, 23]. The kinds of information describing a resource could include its owner, access policy, and current state. Resource information is used by schedulers [28, 19, 26], large data-set movers [10, 34], web portals, other grid services, and users for purposes of discovery, description, and status updates. A *grid information resource manager (GRIM)* is a software entity that manages information about grid resources.

We posit that a grid resource information manager has the following set of requirements:

- resources have aspects of their description that require continuous refreshing

at high rates,

- the Grid is worldwide, thus resources are distributed worldwide,
- providers of resource information will be in close proximity to the resource information manager that serves them,
- a client will often be a significant distance from the information manager it needs to contact, and
- client requests are on-demand
- the organizing paradigm of the Grid is web services, thus a resource information manager must be a web service.

A defining characteristic that distinguishes grid resource information management from, for example, a directory service, is the freshness demands on the data. Resources such as hosts (*i.e.*, workstation, cluster node) and network connections have aspects of their description that change at millisecond rates. For example, a host record might include current CPU load. That the UNIX `kstat` command samples current CPU load once per second implies that a provider might reasonably want refresh its description continuously at a once per second rate. For 30,000 hosts, the database could be subject to 30,000 updates/second. Because of the freshness demands on the data, providers of resource information, which will likely be the resources themselves, and will need to be in close spatial proximity to the information management service that manages the description. Additionally, individual providers are separated from each other, distributed throughout an academic department for instance. Thus updates to the resource descriptions at the grid resource information manager will likely come directly from the providers instead of through an intermediary. A favorable candidate database platform is one that can sustain high update rates and have minimal intrusion on processing of query requests from clients. The number of providers is fixed, known and small relative to the number of clients. Clients, on the other hand, make requests on demand, as the data is needed. Clients hold expectations that the data is the latest or at least the freshest data possible. client requests are therefore sporadic, irregular, large in number and potentially travel a great distance. A suitable platform is one where the query response time is minimized, number of requests are minimized, and the duration of each interaction is minimized.

Existing solutions for managing grid resource information generally deal with

one or a few aspects of the problem. UDDI [32] for instance is a tool and language for creating *registries* and enabling discovery of web services. The web services it represents are described by a modeling language. The language, however, has a strong business and web services focus which makes it less useful for the broader kinds of representation needed on the grid. Directory services, such as DNS [20], manage host descriptions (*i.e.*, workstations, servers). Their primary goal is host-name to IP address mapping and as such are optimized for relatively static data, simple and small set of known queries, and extremely fast response times. The first realization of a grid resource information management tool is MDS2 [13] of the Globus Toolkit [16]. MDS2 is built on top of openLDAP [21] and augments openLDAP with Globus compliant security and remote distributed index support for capturing the contents of multiple LDAP servers. As demonstrated in this paper, MDS2 only partially satisfies the needs of resource representation on the Grid. But all these solutions are partial. The problem needs to be better understood.

This paper describes an experimental study we undertook to better understand resource information management in grid middleware. We frame our study with the question “*How well does a particular database platform support the requirements of a grid resource information manager?*” That is, does a database platform, which embodies a data model, access interface, and query language, support the requirements of a GRIM server or detract from it? To address the question, we have developed a database of resource information that is realistic in size, and contains meaningful resource descriptions and relationships between resources. Resource descriptions are taken from the GLUE data model currently under development in the Global Grid Forum [1]. This data model has wide acceptance in the Grid community. We developed a set of realistic queries and workload scenarios. The *queries* test a broad range of database functionality through realistic questions asked on meaningful data. The *scenarios* are short synthetic workloads that test query response time of a repository under a workload of concurrent query and update requests. The platforms under consideration are MySQL 4.0, Xindice 1.1, and MDS2.

The contribution of this paper is the identification and assessment of the contribution that a database platform makes to the requirements of a grid resource information manager. We identify the key requirements of a grid resource infor-

mation manager and from the requirements, derive a set of metrics used to evaluate a platform. In order to compare the strengths and weaknesses of each platform, we have created a database and set of use cases that are virtually mirrored across the three platforms. Through a measure of response times under a set of queries and scenarios common across the three platforms, we illuminate the performance sensitivities exhibited by a platform in response to features in the benchmark and workload. Based on the study results, we draw conclusions as to how well a platform contributes to meeting the requirements of a grid resource information manager. The benchmark/workload scripts and data population scripts used in the study are available to the community under the name IU RGRbench benchmark, <http://www.cs.indiana.edu/plale/RGRbench>.

The study is focused on understanding grid resource information management, and as such is not a performance comparison of database management systems. Our metrics assess the performance of a database only in its role as the supporting platform for a grid resource information manager. Also, this is a study of a single instance of a grid resource information manager. Distributed resource information management is the topic of ongoing research. Our selection of platforms is based on several criteria. First, all are in current use for some aspect of resource information management in the Grid community. MySQL is used in R-GMA [15]; Xindice is used in MDS3 [36], and MDS2 is used in the Globus Toolkit 2.x [16]. Second, all platforms are open source so are more amenable to wide spread adoption. Finally, the three platforms embody useful diversity: MySQL is a relational database, Xindice is a native XML database, and MDS is built on top of an LDAP database.

The paper is organized as follows: immediately following is a discussion of the data model underlying the study and the experiment controls applied throughout the experiment. Section 3 introduces the benchmark queries and scenarios. In Section 4 we discuss the experiment and make observations drawn from the results. Section 5 is a discussion of related work. Section 6 offers conclusions and a glimpse at future work.

## 2 Data Model

The data model defines the representation of resource descriptions in the database, and the relationships between them. In this section we discuss the data model and its representation on the different platforms, the experiment controls applied during the first stage of GRIM setup on the platforms, and talk about motivations underlying the choices of database size and resource counts.

The starting point of our study is a data model of the resource information manager, an abstract representation of resource entities and their relationships. The UML [30] model and Extended Entity Relationship model are the two most widely used data modeling notations. Our data model starting point is an October 2002 snapshot of the GLUE data model [14] defined by the Global Grid Forum [1]. The goal of the GLUE effort is to bring together predominately US and European grid middleware researchers to reach agreement on a single data model that will ensure interoperability between efforts.

Terminology referring 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*.

	<i>Relational</i>	<i>LDAP</i>	<i>Xindice</i>
collection level	table	entry	<i>collection</i>
member level	tuple	<i>object</i>	document

Table 1: Terminology used in different data models. Italicized terms are the ones used in this study.

A Xindice database does not have a schema, per se. Xindice stores collections of documents, and it is the documents that are described by schemas through a language such as XMLSchema. However, we use the term "schema" uniformly across platforms to mean the implementation of the data model on a platform.

Each of the three database platforms hold the same kinds and numbers of entities and relationships. The number of objects 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 objects among collections for the major collections is shown in Table 2.

<i>Collection</i>	<i>Number of Objects</i>
Cluster	20
User accounts	60
Computing element	106
Subcluster	345
Application	600
Active network connection	12200
Host	29743

Table 2: Object counts for a sampling of collection types.

## 2.1 Experiment Controls

The validity of the study is contingent upon experiment controls that are applied consistently across the three platforms at every stage of the experiment. Experiment controls are applied beginning with the derivation of the schemas for the different platforms from a common data model, and are applied consistently thereafter including 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 resource information manager.

Schemas are derived from the data model for the native XML and relational platforms using well defined mapping rules [33]. The data schema of the LDAP plat-

form, on the other hand, already existed [14]. We strove for consistent and faithful representation of entities across all platforms. If a resource description were implemented as multiple separate entities in the LDAP schema, we maintained that separation across the other platforms despite query performance gains that might be gained by other organizations. For instance, storing memory and CPU information in the Host object directly, would likely yield improved performance across all platforms because the number of joins required to access host, CPU, and memory information would be reduced.

The databases are populated with data derived from dumps of small to medium-scale MDS2 servers gathered between November 2000 and January 2002.

The three platforms contain identical data, made possible by 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 bulk loads through a file in LDIF format, is also populated directly from the MySQL dump.

**Meaningful data.** The data in the database is meaningful and the cardinalities reflect meaningful relationships. 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 looks for a path of distance four (edges) between hosts. This query finds roughly 13000 paths at that distance in a connection table of roughly 12000 entries.

## 2.2 Information Distribution Model

For purposes of this study, we assume a particular information distribution model, and that assumption guides our selection of size of database and number of resources for which the grid resource information manager maintains descriptions. We argue that this assumption is realistic.

The information distribution model used in the study assumes a distribution of resources across distributed grid resource information managers, and the affiliation of grid resource information managers to administrative domains. Specifically, a resource description is managed by a grid resource information manager that resides in the same administrative domain as the resource and has primary responsibility for the description. The decomposition of resources into administrative domains follows the decomposition lines used by Domain Name Service (DNS) [20]. An *administrative domain* is a management entity and a DNS server manages resource descriptions of the hosts within that domain. The services, servers, and datasets within a Computer Science department, for instance, are under the control of a single administrative domain. A university’s computing services controls additional resource descriptions but since it also represents the university as a whole, it maintains a description of the CS department administrative domain as one of its resources. In our model of data distribution, a resource information server maintains information about all the resources in a single administrative domain.

We argue that under this assumption, a grid resource information manager representing 350 subclusters, 30,000 hosts, and 12,000 active network connections is realistic.

### **3 Benchmark/Workload**

The kinds of queries and updates that might be issued against a grid resource information manager can be broad, limited only by the user’s knowledge of the query language and data, and the expressiveness of the query language. Our goal in constructing the benchmark is a set of queries that taken as a whole, exercise several orthogonal axes: simple queries versus complex, queries that test specific search support, realistic job submission requests, sequential versus concurrent access, small return set versus large. This section introduces the benchmark/workload.

The synthetic database workload consists of 13 queries, one update, and four scenarios. For descriptive purposes, we partition the benchmark into five major categories: scoping, index, join, selectivity, and base operations; some queries are paired. That is, a single variable is tested, say the presence of an index. All other variables (*e.g.*, size of return set, collection size, string matching operators,

collection size) are controlled. This pairing attempts to quantify the benefit of a variable. Full details of the queries including the different language representations can be found in [24].

**Scoping.** We use the term *scoping* to define queries that limit their search to a particular subtree by means of specifying a scope for the search. Intuitively scoped queries should yield better response times than their non-scoped counterparts in hierarchical databases (*i.e.*, Xindice, MDS2).

**Indexing.** Query response times are often dramatically improved when the attributes on which a search is done are indexed. Our benchmark contains 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. Based on DeWitt [7], representative results of the impact of selectivity can be obtained by 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 collection. 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 four hops.

**Other.** The two final queries test connection speed and row modification times. 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 the scenarios is to expose platform behavior under multiuser workloads. The synthetic workload consists of six streams, three of which perform repeated updates and three which perform queries. A stream contains the same request issued repeatedly. The workload was designed

to quantify the effects of concurrency at a platform, and as such does capture realistic use scenarios [5, 3]. It is expected that the scenario response times reflect a high level of cache hits at the database. The server used in the study has 2000MB of RAM whereas the total footprint of the mySQL database is estimated at only 10MB.

Since continually changing resource descriptions are a key characteristic of resource information managers, the impact of update streams on user perceived performance is a key metric. A scenario executes as three phases: 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, but for our study a scenario runs for a duration of 20 minutes. Average query response time is calculated every 10 seconds as

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

$t_1 - t_n$ : query response time for  $t_i$ ,  $i \neq 0$ , is the average of query responses received in the interval  $(t_{i-1}, t_i]$ .

If no query responses are received in an interval,  $i$ , then average QRT for the interval is taken as the QRT of the previous interval.

## 4 Performance Evaluation

To quantify how well a particular platform supports the requirements of a grid resource information manager, we developed a benchmark of fourteen queries and four workload scenarios. We developed metrics for evaluating the outcome of the benchmark and the workloads. We examine the metrics in light of the requirements to determine how well a platform supports the requirements of a GRIM manager.

The model used in this study is a client-server model. That is, the database exists as an independent task. Client interaction is through the APIs exported by the platforms. The benchmark consists of a set of scripts for each database, one per query. The mySQL scripts issue SQL queries, are written in Perl, and communicate with the database using the mySQL C API. The Xindice queries are written in

XPath, the scripts written in Java, and the interface to Xindice is through XML:DB. The MDS2 queries are written in the LDAP query language, the scripts are written in C, and the interface to MDS2 uses the LDAP protocol. The benchmark cost metric is *Query Response Time (QRT)*. QRT is the elapsed time as measured at the client from the issuance of the first request of a benchmark query to the receipt at the client of the last data from the last request of the benchmark query. In the case where a benchmark query corresponds to a single request, QRT is the elapsed time between when the request is sent by the client and when the response is received. When the benchmark query is implemented as multiple requests, elapsed time includes multiple interactions with the database and often client processing as well. For instance, the following query “*For a cluster x, give me a list of subclusters and their operating system type*” can be satisfied in SQL with a single request/response sequence to the database. XPath on Xindice and LDAP, which do not permit traverses across collections, require multiple requests plus client processing. The first request retrieves a list of subclusters. The client must then loop through the list of subclusters to retrieve OS type.

The scenarios are described in Section 3. The scenario cost metric for the scenario employs average query response time as described above except that average QRT is computed over a 40 second interval and repeated over a 20 minute duration. During that time, the database is subject to concurrent activity from other clients and from providers and the impact of that activity is measured.

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 8.0. Each database is implemented as a stand-alone 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 which provides 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 issued against the GIIS are as anonymous queries to eliminate authorization that would normally occur.

The client and server are co-located on a dual processor machine. This allows us to focus our examination on query processing times since network latency is a

very small constant that is uniform across all measurements and platforms. The dual processor schedules the client on one processor and server on another, so interference is minimized.

Many of the queries in the benchmark require the issuance of multiple requests to the database in order to be satisfied. The performance evaluation described in this paper is of a local resource information manager server. We are looking at distributed performance in ongoing work. Because we are comparing vastly different platforms, we do not microbenchmark the platforms. That is, microbenchmarking of a database platform is outside the scope of our study. Our focus is to provide an apples-to-apples performance comparison of resource information manager 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.

#### 4.1 Benchmark Queries

The results of applying the benchmark on the three platforms is shown in Figures 1 and 2. On the X-axis are the individual queries and their results for each platform. Query pairs appear next to each other. A pair is denoted by the presence of a query having the same name but prefaced with *non*. For instance, *scoped* and *nonScoped* are a pair. The Y-axis shows average query response time in milliseconds. Note that the Y-axis scale is logarithmic.

The leftmost four queries in Figure 1 test scoping, whereby a search is narrowed by the specification of a starting scope. In comparing the results on the hierarchical platforms (*i.e.*, Xindice and MDS2) for “*scopedHosts*” and “*nonScopedHosts*”, one can see a benefit to scoping. In this case the collection accessed by the queries is large. Contrary results are exhibited for Xindice by “*scoped*” and “*nonScoped*” where the collection accessed is small. The apparent contradiction exists because “*scoped*” is implemented as three XPath queries whereas “*nonScoped*” is implemented as two. The gain realized by scoping over a small collection in

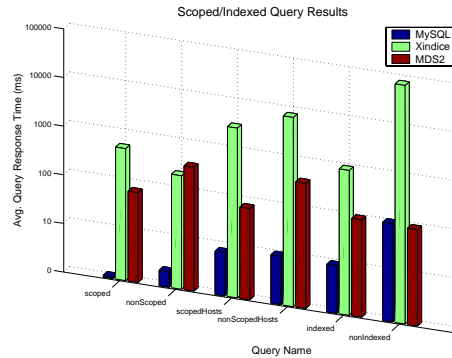


Figure 1: Query response times for scoped and indexed queries.

Xindice is overshadowed by the additional processing time required for the additional query. The impact on Xindice performance of multi-request queries is explored more deeply in [35].

A comparison of the indexed and nonIndexed results in Figure 1 for relational and XML platforms confirms the widely held belief that indexes greatly enhance performance. Note however that indexed Xindice results are still an order of magnitude slower than non-indexed mySQL results. MDS2 displays no sensitivity to indexes because MDS2 has overridden the native openLDAP [21] index support and replaced it with a caching scheme deployed in the GIIS.

The selectivity of a query is the number of returned objects that satisfy the query. In Figure 2 we see that in mySQL and Xindice QRT shows no sensitivity to the number of objects returned. MDS2, on the other hand, shows considerable sensitivity to return set size. The high overall response times for all three platforms are attributable to the fact that the three queries access a large collection (*i.e.*, connections) on a non-indexed attribute. The “manyRelations”, “networkHop”, and “jobSubmit” queries measure a repository’s ability to assemble a result from numerous collections. The “manyRelation” and “jobSubmit” queries both touch six collections. “JobSubmit” asks the following realistic question that returns a single object as a result set:

”Of the machines in cluster xx, give me a list of subclusters running Linux and their total RAM, but only where I have an active account

and the binary yy is resident.”

“NetworkHops” is a transitive query of length four on an indexed attribute over the large Connections table. On Xindice, the network hop query was unable to complete even a single iteration before the 100,000 ms cutoff.

In comparing the Xindice “manyRelation” join query with the Xindice selectivity results one can see that the join query is significantly faster (1000 ms versus roughly 38,000 ms). This belies the commonly held belief that joins are too expensive and should be somehow disallowed in grid resource information management. In fact, for Xindice, collection size is the biggest determinant in predicting QRT than is number of joins.

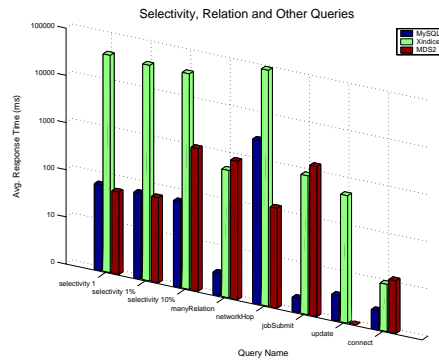


Figure 2: Query response time for selectivity, join, and other queries.

The “update” results are for a single one-attribute updates. As such, they give an indication of the 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 exhibited by Xindice is the single largest factor making Xindice poorly suited as a platform for a grid resource information manager by failing to meet the requirement of frequent updates to attributes. The update rate for mySQL is disturbingly low as well, but as can be seen from the connect results, much of the update time is attributed to the connection cost. Thus significantly better rates could be achieved through use of persistent connections. In MDS2, traditional LDAP updates have been disabled and replaced by the 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.

The MDS2 results in this paper are for fully cached data, and taken from a database that is 5% of the full size used on the other platforms. Instabilities in MDS2 caching in the GIIS foiled our assiduous efforts to capture non-cached results. We observed an exponential relationship between query response time and database size in MDS2, based on running the benchmark against MDS2 databases of size 5%, 10%, 25%, 50%, and 100% of the full size. Further, for all queries in the benchmark, in order for a query to successfully complete one iteration on the full MDS2 database, the minimum timeout (queries time out if successful completion is not achieved within a certain amount of time) had to be set at 4 hours. Compare this to MySQL where the maximum time needed for a single iteration of the most time-consuming query in the benchmark is 100ms.

## 4.2 Workload Scenarios

Scenarios are synthetic workloads that simulate multiuser access. They are an important part of the benchmark because they capture effects of concurrent query and update requests on user perceived performance. A requirement of the GRIM manager is support for very rapid updates to a small number of select attributes. The synthetic workloads address this requirement by means of a workload consisting of simultaneous query and update threads that continuously issue requests to a database platform. The goal of the scenarios is to expose the impact of various update workloads on user perceived response time.

Updates to a database can take different forms ranging from an update to a single attribute in an object to a transaction. The types of updates relevant to a grid resource information manager depend exclusively on the nature and location of the resource providers. A requirement of a GRIM is that resources are located in geographic proximity to the resource information manager that serves them but are distributed (*i.e.*, scattered) from each other within that localized space. Additionally, selected attributes of the resource description must be updated at millisecond rates, driven for example by grid schedulers that depend on timely information for the accuracy of the schedules they generate. Because of these timeliness demands on certain attributes, it is likely that a provider of the state information about the re-

source will be the resource itself, and not some intermediate agent. Given these requirements, updates are most likely to be in the form of individual update requests rather than complex transactions. Motivated by these assumptions, our scenarios test the following:

- update to multiple attributes within a single record,
- insert a new record, and
- overlap in collections accessed, so that every update thread accesses a table that is simultaneously being accessed by a query thread.

A scenario is a stand-alone client program consisting of six threads, three of which execute a query request, and three an update request. A thread when activated, repeatedly issues a request to the database and blocks awaiting a response. As explained in Section 3, a scenario executes in three phases over a 20 minute duration. In Phases I and III, only the query threads are active, while in Phase II both query and update threads are active.

The phases are easily discernable in the mySQL scenario results shown in Figure 5. Plotted are query response times for each of the three query threads that repeatedly over a 20 minute duration issue their individual request. A data point is the average QRT taken over all requests satisfied in the time interval since the last data point. During Phase I, which begins at time 0, only query threads are executing. After 360 seconds (6 minutes) into the execution, the update threads are launched and execute simultaneously with the query threads for 300 seconds (5 minutes). During Phase III only the query threads remain active.

Within a scenario graph, the differences between the query plots are differences in the queries. For instance, in Figure 5 Query1 searches for a particular software service, such as a library, on a non-indexed field over a collection that is 640 objects in size. Query2 searches the large Host table on an indexed attribute. Query3 searches a tiny GlueSE collection, 60 objects, on a non-indexed attribute. The update threads, not shown, all add records to the Policy collection. This access patterns are depicted graphically in Figure 8.

Across the scenario graphs shown in Figure 3, 4, and 5, the differences are in type of update being performed. In Figures 4 and 5, updates are to the Policy table. In Figure 4 a 340 byte object is added and in In Figure 5 multiple attributes

in an object are changed. This access pattern is depicted in Figure 8. In Figure 3, however, updates overlap with queries in collections accessed. This access pattern is depicted in Figure 9.

Comparing the three scenarios of Figures 3, 4, and 5, one can conclude that object insertion has a larger impact on QRT than does update to multiple attributes within an object, but updates that compete for collection access with queries have a much larger impact on QRT. These results were obtained with the InnoDB back end which provides row level locking.

Xindice performance for the concurrent read/write overlap scenario, shown in Figure 6, further substantiates our observation that Xindice's performance sensitivity to collection size detracts from its viability as a GRIM platform. We argued earlier that our database is of a realistic size. Since the query is over an indexed field, latencies in the 13 minute range, such as are shown in Figure 6, can be expected to be a regular occurrence in practice. Suppose a query arrives at the database the same instant as an update request that updates a fast-changing variable. A query that takes 200 ms to process under conditions of concurrency on MySQL, will take over 13 minutes on Xindice. This latency is clearly unacceptable for a GRIM server. The remaining scenarios are not shown for Xindice because they resemble each other.

MDS2 performance in the overlap scenario is shown in Figure 7. The MDS2 numbers while accurate in capturing overheads attributed to the various concurrencies, as discussed next, cannot be directly compared to Xindice and MySQL. As discussed earlier, the MDS workloads are run on a database that is only 5% of the database size used in the rest of the study. This is because it was only at the small database sizes that we were able to achieve cache and query stability. It is our observation from working extensively with larger database sizes that QRT scales exponentially with overall database size. The updates in MDS require clarification. Updates for the scenarios had to be simulated because the LDAP add mechanism is disabled in the MDS2 API. We simulated updates by issuing a client query that queried an object that had a very 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 4. MDS2 may be a viable platform for a GRIM in the case where up-

dates are rare and the size of the database is small. Otherwise it is not acceptable as QRT is extremely sensitive to database size as we observe, and performs very poorly under update loads [29].

A summarization of the scenario study is captured in Figures 10 and 11. Figure 10 captures results from a simple baseline scenario. As such, it can give an intuitive feel for upper performance. Specifically, the query threads issue a single predicate query on an indexed attribute against a small collection (*i.e.*, 60 objects) that returns a null set. The update threads select on an indexed attribute and update a single attribute. There is no overlap in collections accessed. At the top of the pyramid is the query response time (QRT) for the *nonScoped* query in Figure 1. This QRT reflects no other client activity. One level down, labeled “concurrent queries”, is overhead attributed to concurrent query threads (2 others). The further additional overhead due to concurrent update threads appears at the next level. The bottom level is the summed, or total QRT. The figure shows the impact of concurrency on the platforms. Since a high degree of concurrency is expected because of freshness demands on the GRIM manager, this behavior is important to understand.

Figure 11 conveys the same information but for a more complex case. The query being depicted is the *indexed* query (see Figure 1) which accesses the large Host table on an indexed field. The update threads add new objects to the database, but there is no overlap between query and update threads in the collections that they access. In this more complex case, MDS2 performance remains the same as for the simpler case. Xindice performance, on the other hand, incurs significant additional overhead as a result of the more complex concurrent activities.

### 4.3 Additional Metrics

The characteristic that clients of a GRIM server are geographically dispersed over a wide region, and are often physically distant from the server that they access, implies that communication will be through a WAN. Hence an important metric of a GRIM server is the network bandwidth consumed in communicating with the GRIM server. Table 3 shows the total number of bytes requested by one client (one query thread) over a 20 minute scenario. The bandwidth consumption is shown to the right of the total bytes. Evident from the table is the impact that query language

has on the connection scalability of a server.

<i>Query</i>	<i>mySQL 4.0</i> <i>(MB/Mbps)</i>	<i>Xindice 1.1</i> <i>(MB/Mbps)</i>
jobSubmit	580MB/3.87Mbps	7,450MB/49.7Mbps
indexed	1.8MB/.012Mbps	697MB/4.64Mbps

Table 3: Number of bytes returned to one client thread during the course of a 20 minute scenario. Reflects the difference between richer languages such as SQL that enable a query to be satisfied with a single-request, versus a simpler language where a query requires multiple requests, and partial results are returned to the client for processing and refinement.

The maximum number of requests that must be issued per high-level benchmark query differs depending on query language. As is shown in Table 4, the less expressive query languages, such as LDAP and the subset of XPath implemented by Xindice require more interaction with a grid resource information manager, and must process intermediate data to obtain the desired results. In essence, these simpler languages cast the task of processing intermediate results onto the client. In the jobSubmit query, for instance, the user must issue 5 requests and write 315 lines of code in five loops in order to obtain the essential data sought. Imposing this kind of obligation on the client can and should be avoided.

<i>Description</i>	<i>mySQL 4.0</i>	<i>Xindice 1.1</i>	<i>MDS2</i>
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.

## 5 Related Work

Grid resource information managers are one of many types of systems that manage and serve information. Databases, web servers, information retrieval systems, directory services, and file systems all share the common characteristic that they

store and serve information on behalf of multiple users. Benchmarks and workloads are well known methods for the experimental evaluation of systems. In this section we examine related work in performance evaluation of information systems

The first implementation of the grid resource information manager, and one that is still in widespread use, is MDS [13] of the Globus Toolkit [16]. MDS2, the version of MDS that we used in this study, has a hierarchical organization of Grid Resource Information Servers (GRIS) that connect to one or more higher-level index servers (GIIS). The new version of MDS, MDS-3, is part of the Globus Toolkit, GT3, and is a reference implementation for the Global Grid Forum Open Grid Services Infrastructure (OGSI) [31] proposed standard. MDS-3, like MDS-2 is hierarchically organized, and distributes grid resource information between MDS3 and discovery services like UDDI [32]. UDDI [32] is a tool and language for creating registries and enabling discovery of web services. The web services it represents are described by a modeling language. The language, however, has a strong business and web services focus which makes it less useful for the broader kinds of representation needed on the grid.

Smith [29] 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* [2] studied the MDS2 grid information server and conducted experiments focused on simple tests of the Grid Index Information Service (GIIS). Schopf [38] examined the scalability of three information servers, MDS2 [13], RGMA [15], and Hawkeye of the Condor system [6], 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. The testing was done under conditions of a small simple database and a trivial query. Schopf's connection scalability study complements ours, which is focused on a broad set of queries and scenarios issued against a rich data set in a database of a real world size.

Directory services, such as DNS [20], manage host descriptions (*i.e.*, workstations, servers). Their primary goal is hostname to IP address mapping and as such are optimized for relatively static data, simple and small set of known queries, and extremely fast response times. The proactive directory service [8] discusses an ap-

proach to increasing scalability in a directory service that keeps clients up to date by means of pushing changed information to clients. We do not believe that a push model to clients is appropriate for grid resource information management. Client requests in a GRIM are on-demand. The bottleneck to scalability is not in keeping clients up to date, but in supporting good query response times while allowing rapid refreshes to happen simultaneously and at good rates.

The University of Wisconsin benchmark [7] evaluates the performance of off-the-shelf relational database management systems. The benchmark is 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) [25] 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 [27] provides a framework to assess the abilities of an XML database.

In [9], the authors evaluate the performance of a distributed information retrieval system. Grid resource information management systems differ from IR systems in the timeliness demands on the data that drive unique access and update requirements. Additionally, the authors constructed and measured a distributed IR system that holds large distributed collections of up to 128GB of data. The workload differs from ours in that it strives for realism through varying the rate at which clients send commands, and a user workload which uses both query and document retrieval operations.

Web server and proxy server performance evaluation by means of workloads has been an active area over the past seven years. Crovella and Bestavros have investigated self similarity in [12]. Realistic workloads have been used to evaluate the Flash [22] web server and others [5, 3].

## 6 Conclusions

We draw a number of conclusions from the study, which in this section we summarize and tie back to the requirements of a grid resource information manager that are listed in Section 1.

*Resources have aspects of their description that require continuous refreshing*

*at high rates. This requirement implies that a platform support sufficient update rates for multiple providers and that the response time for clients issuing queries to the system not be significantly impacted by the presence of active updates.* The platforms we measured separate most distinctly from one another on the requirement of minimal impact to queries. Xindice has an unacceptably low update rate of 1.3 updates per second. The MySQL update rate is better at 200 updates/second, but could be higher. Our results show, however, that a significant percentage of update time is consumed by connection time to the database. Hence update rate would benefit significantly by use of persistent connections. MDS2 performs poorly under update loads. Though for reasons explained in Section 4 we were unable to determine a precise update rate, the problem of poor performance of LDAP servers under nontrivial update loads is well known [29].

Xindice QRT displays the most sensitivity to concurrent queries and updates on the two axes of increased level of concurrency and increased complexity of workload. Specifically, QRT of a query doubles in the presence of simultaneous simple queries and quadruples under updates. Under a more complex workload, QRT increases by two orders of magnitude. That is, a 1 second query running alone finishes in 7 minutes under the influence of two concurrent query threads and three concurrent update threads. MDS2 is less affected by concurrency than is Xindice, but still is several orders of magnitude worse than MySQL. But MDS2 is more stable than Xindice across workloads. MySQL QRT times are minimally affected by the relatively light workloads we were required to use in order to get repeatable performance from the other platforms.

*The requirement that clients issue requests on demand is satisfied by a platform that minimizes query response time. Additionally, a client is often located a long distance away from the grid resource information manager it needs to contact. Hence a platform that minimizes the number of requests and amount of data that must be transferred in order to satisfy a request is preferred.* Of the three platforms, MySQL overall has substantially lower QRT for all queries in the benchmark. Xindice query response times are 2-3 orders of magnitude greater than that of MySQL; concurrency adds another order or two of magnitude. Additionally, Xindice query response time degrades quickly as collection size increases. MDS2 query response times grow exponentially relative to overall database size. At rel-

atively small database sizes, MDS2 query response times are good. For instance at a database size of 5% of the size used for mySQL and Xindice, MCS2 query response times are better than Xindice. The exponential growth of QRT at larger sizes makes MDS2 a poorly suited platform for realistically sized databases. Additionally, it is our observation that MDS2 caching is highly unstable. In our study, either the entire database had to be cache resident or none of it was. This instability impacted our ability to evaluate the impacts of caching in the GIIS.

The less expressive query languages, LDAP and the subset of XPath supported by Xindice, require multiple requests be issued to the grid resource information manager to satisfy a single query. Each request requires code exist at the client that can process the intermediate results. These simpler languages impose additional bandwidth demands on the network, increase the workload at the database, and cast upon the client the real work of processing intermediate results. For instance for the jobSubmit query on Xindice, 13 times more data was transferred from the Xindice database to satisfy the query than for mySQL. Additionally, what was accomplished with one SQL query to the database and a few lines of code to process the results, on Xindice took 5 requests and 315 lines of code in five loop in order to filter and transform the data. Imposing this kind of obligation on the client is not necessary and should be avoided.

*Finally, the organizing paradigm of web services means that grid resource information managers that communicate using XML are better suited platforms.* The absence of XML support is the largest argument against mySQL. This shortcoming could be remedied by the OGSA-DAI [4] data access and integration framework which provides XML enabled communication and the framework enabling databases to become services on the grid. That is, to state it simply, OGSA-DAI enables a mySQL database to communicate with other grid services by means of XML documents, the SOAP protocol [37], and WSDL [11] descriptions of interfaces. But in work reported elsewhere [18], we ran the IU RGRbench benchmark on an OGSA-DAI enabled mySQL database and observed that OGSA-DAI adds an additional 100-700 ms overhead onto QRT for small return dataset sizes (under 10K). For larger return set sizes, third party transfer mechanisms are required that drive QRT into the range of minutes. A useful constraint on the grid resource information manager might be to restrict the size of the return set to enable the use of

the fast response mechanism for all requests. A viable solution would be an XML enabled relational platform that supports XQuery. XQuery is a more expressive query language, hence does not suffer the network bandwidth and server load consumption problems of XPath or LDAP. It handles XML data efficiently and uses a relational back end for the good performance expected from relational platforms.

**Future Work.** Future work is in exploring issues in distributed GRIM managers and in managing scalability that minimize some of the overheads found. For instance, we are looking at approaches to reduce the load on tables that are subject to simultaneous queries and updates. It is likely that most queries will be issued to the tables that are updated most frequently. Not all queries need the most up-to-date information, however, but are instead interested in historical trends. Thus it is possible to offload some of the queries onto other collections, for instance, that provide historical data instead. The benchmark code is available at <http://www.cs.indiana.edu/plale/RGRbench>.

## 7 Acknowledgments

We are grateful to the following people who have contributed ideas and probing insight to this work: Peter Dinda, Northwestern and Dennis Groth, Indiana University. Ying Liu, Craig Jacobs, Scott Jensen, and Charlie Moad were instrumental in obtaining the measurements that form the basis of this study.

## References

- [1] Global grid forum. <http://www.gridforum.org>, 2003.
- [2] 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](http://www.gridlab.org/Resources/Deliverables/D10.1.pdf).
- [3] Steven D. Gribble and Eric A. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

- [4] Mario Antonioletti, Neil Chue Hong, Ally Hume, Mike Jackson, Amy Krause, Jeremy Nowell, charaka Palansuriya, Tom Sugden, and Martin Westhead. Experiences of designing and implementing grid database services in the ogsa-dai project. In *Global Grid Forum Workshop on Designing and Building Grid Services*, September 2003.
- [5] Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: The search for invariants (extended version). In *ACM SIGMETRICS*, May 1996.
- [6] Jim Basney and Miron Livny. *High Performance Cluster Computing*. Prentice Hall PTR, 1999.
- [7] 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.
- [8] Fabian E. Bustamante, Patrick Widener, and Karsten Schwan. Scalable directory services using proactivity. In *ACM/IEEE Supercomputing 2002 (SC 2002)*, Los Alamitos, 2002. IEEE Computer Society.
- [9] Brendon Cahoon, Kathryn S. McKinley, and Zhihong Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transactions on Information Systems (TOIS)*, 18(1):1 – 43, January 2000.
- [10] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific data sets. *J. Network and Comput. Appl.* (to appear).
- [11] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. [www.w3c.org/TR/wsdl](http://www.w3c.org/TR/wsdl), 2001.
- [12] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *IEEE/ACM Transactions on Networking*, volume 5, pages 835–846, 1997.
- [13] 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.

- [14] DataTAG. Glue schema: common conceptual data model for grid resources monitoring and discovery. <http://www.cnaf.infn.it/sergio/datatag/glue>, 2003.
- [15] Steve Fisher. Relational model for information and monitoring. In *Global Grid Forum, GWD-Perf-7-1*, 2001.
- [16] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.
- [17] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.
- [18] 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.
- [19] Daniela Rosu Michael B. Jones and Marcel Rosu. Cpu reservations and time constraints: Efficient, predictable scheduling of independent activities. In *ACM Symposium on Operating Systems Principles (SOSP '97)*.
- [20] P. Mockapetris. Domain names - concepts and facilities. In *RFC 1034, Network Working Group*, November 1987.
- [21] OpenLDAP Organization. OpenLDAP. <http://www.openldap.org>, 2003.
- [22] Vivek Pai, Peter Druschel, and Willy Zwaenepoel. Flash: An efficient and portable web server. In *1999 Annual Usenix Technical Conference*. Usenix, 1999.
- [23] 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.
- [24] Beth Plale, Craig Jacobs, Ying Liu, Charles Moad, Rupail Parab, and Prajakta Vaidya. Benchmark details of a synthetic database benchmark/workload for grid resource information. Technical Report Technical Report TR-583, Computer Science Dept., Indiana University, 2003.
- [25] M. Poess and C. Floyd. New TPC benchmarks for decision support and web commerce. *ACM SIGMOD Record*, 29, December 2000.
- [26] Rajesh Raman, Miron Livny, and Marvin Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *Proc. Twelveth IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, pages 80–89, Seattle, WA, June 2003.

- [27] 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.
- [28] Shava Smallen, Henri Casanova, and Francine Berman. Applying scheduling and tuning to on-line parallel tomography. In *ACM/IEEE Supercomputing 2001*, Los Alamitos, CA, 2001. IEEE Computer Society.
- [29] 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.
- [30] Rational Software. Conceptual, logical, and physical design of persistent data using UML. [www.rational.com/uml/resources/whitepapers/index.jsp](http://www.rational.com/uml/resources/whitepapers/index.jsp), 1998.
- [31] S. Tuecke, K. Czajkowski, 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.
- [32] UDDI.org. Universal description, discovery, and integration (UDDI). <http://www.uddi.org>, 2003.
- [33] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice Hall, 1997.
- [34] Mustafa Uysal, Tahsin Kurc, Alan Sussman, and Joel Saltz. A performance prediction framework for data intensive applications on large scale parallel machines. In *Fourth Workshop on Languages, Compilers, and Run-time systems for scalable computers (LCR98)*, 1998.
- [35] 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.
- [36] Von Velch, Frank Siebenlist, Ian Foster, John Bresnahan, Karl Czajkowski, Jarek Gawor, Carl Kesselman, Sam Meder, Luara Pearlman, and Steven Tuecke. Security for grid services. In *Proc. Twelveth IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, pages 48–57, Seattle, WA, June 2003.
- [37] W3C. Simple object access protocol (SOAP) 1.1. [www.w3c.org/2000/xp/Group](http://www.w3c.org/2000/xp/Group), 2000.

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

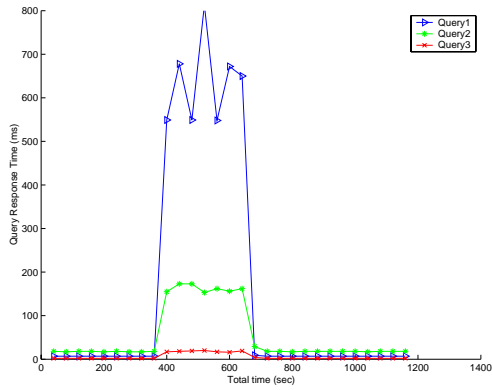


Figure 3: Scenario 2: SQL concurrent read/write overlap

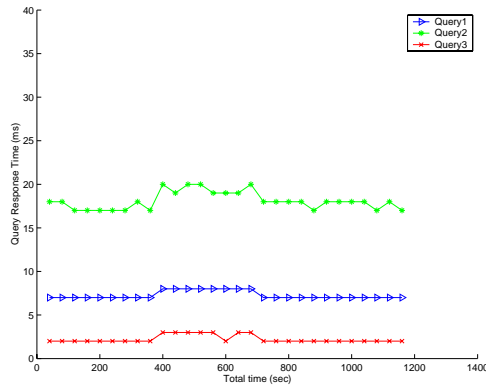


Figure 4: Scenario 3: SQL multiple attribute update

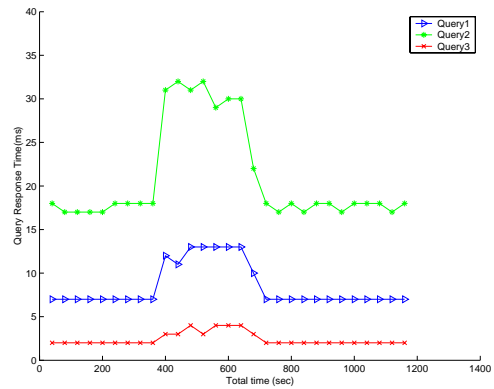


Figure 5: Scenario 4: SQL record insert

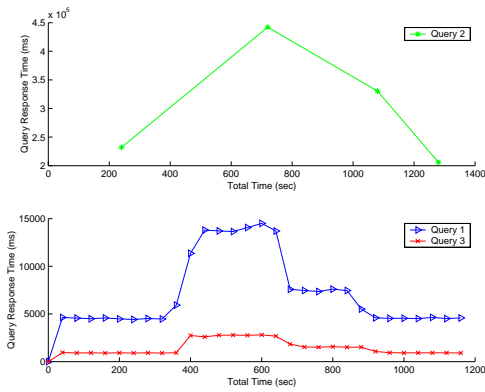


Figure 6: Scenario 2: XML concurrent read/write overlap

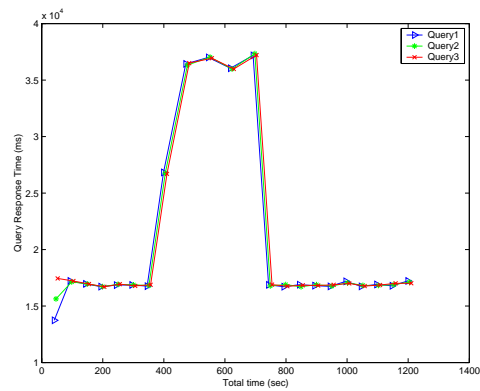


Figure 7: Scenario 2: LDAP concurrent read/write overlap

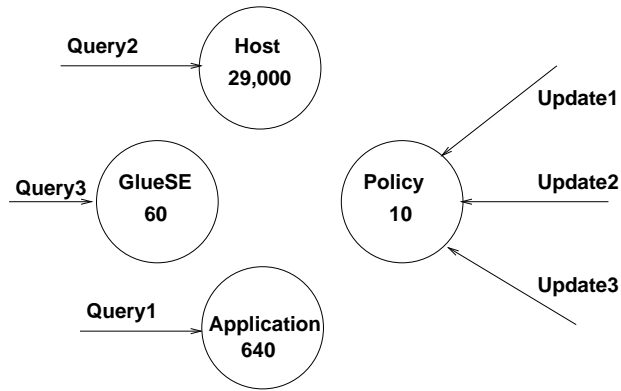


Figure 8: Access pattern for “Object add” and “multiple attribute update” scenarios.

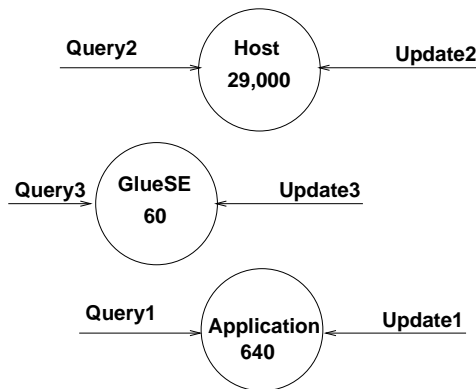


Figure 9: Access pattern for “Overlap” scenario.

single query	3 ms	400 ms	500 ms
+ concurrent queries	+ 0 ms	+ 340 ms	+ 16,300 ms
+ concurrent updates	+ 3 ms	+ 935 ms	+ 20,000 ms
= total QRT	= 6 ms	= 1675 ms	= 36,800 ms
	mysql	Xindice	MDS2

Figure 10: Impact of concurrency on QRT of the 'nonScoped' query under multiuser conditions of simple query and update requests.

single query	9 ms	1000 ms	100 ms
+ concurrent queries	+ 9 ms	+ 269,000 ms	+ 16,900 ms
+ concurrent updates	+ 12 ms	+ 170,000 ms	+ 20,000 ms
= total QRT	= 30 ms	= 440,000 ms	= 37,000 ms
	mysql	Xindice	MDS2

Figure 11: Indexed query response time under conditions of more complex concurrent queries and update requests.