Benchmark Evaluation of Xindice as a Grid Information Server

Prajakta Vaidya Beth Plale

Computer Science Dept.

Bloomington, IN 47405

{pvaidya,plale}@cs.indiana.edu

TR585

Abstract

A grid information server is a repository that stores information about resources on the Grid and forms an important part of the grid middleware. Our aim is to explore the use of Xindice 1.1, a hierarchical native XML database for the management of Grid resource information. This paper details the work done as part of the benchmark with particular emphasis on a strong and diverse query set, Xindice database generation and population, and Xindice performance.

1. Introduction

The objective of the project is to evaluate the performance of Xindice1.1 hierarchical native XML database. This work has been done as part of the Relational Grid Resource Project under the guidance of Dr. Beth Plale and parts of this report are based upon the work done by the team working on this project. This report gives an introduction to Xindice1.1, explains the database generated for testing the performance, describes the test suite used for benchmarking the database and details the results obtained from the tests.

2. Xindice Database

Xindice is an open source Native XML Database. It stores and indexes compressed XML documents in order to provide that data to a client application with very little server-side processing overhead. It also provides functionality that is unique to XML data, which can't easily be reproduced by relational databases.

The benchmarking was done on two different versions of Xindice. The first version was Xindice1.0. This is a stable release of Xindice. The other version used is Xindice 1.1, which was in its beta-release stage. Also, Xindice 1.1 is not backward compatible. For differences between Xindice 1.0 and Xindice 1.1 please refer to Appendix A.

3. Data Population

Xindice is a hierarchical database. The hierarchical structure is defined by collections (equivalent of folders) and documents contained in collections (equivalent of files contained in folders).

The data inserted in the Xindice database was taken from the existing MySQL database to maintain consistency during the benchmarking phase. Taking a dump of the MySQL database and parsing it to create XML documents out of every tuple maintained the data consistency across the databases. The tables in the MySQL database correspond to the collections in the Xindice database. So the set of tuples in a table were mapped to documents under the corresponding collection. The hierarchical structure for the collections is described by the UML diagram that was used as a foundation for modeling data. This hierarchical data structure can be in Appendix B.

The data population is a two-step process. It involves creation of the XML documents from the MySQL dump followed by population of the Xindice database with the created documents. The population script erases the previous data in the database before repopulating it. Also, indexes are created on elements corresponding to all indexes that exist in the MySQL database.

4. Benchmarking

4.1 Query Language Limitations

The query language supported by Xindice is XPath. XPath is an XML query language that was originally designed for querying a single XML document. So, to use for querying a native XML database, the basic specification of XPath was extended to support querying a collection which may contain more than one single document or even for querying across multiple collections. Xindice does support querying multiple documents within a single collection and hence extends the basic XPath specification. However, Xindice does not support querying across collections even though XPath has been extended to support it. Because of this limitation, queries issued to the Xindice database cannot span multiple collections. Similarly, it does not support querying sub-collections recursively. This is a limitation of Xindice, in contrast to other hierarchical databases like LDAP. Xindice may eventually support XQuery, which is a more powerful query mechanism. The update language used by Xindice is XUpdate.

The benchmarking involved implementing the set of queries defined as part of the benchmark. The implementation was programmed using Java and XMLDB API to interface with the Xindice database.

4.2 Xindice Efficiency and Ease of Use

For querying across multiple collections, the Java program has to contain the added logic required on top of the basic operations that are supported by Xindice. This can involve splitting a single query into multiple queries and then manipulating the results returned by the queries programmatically. Also, Xindice does not support attribute level selection. Hence, when attribute values are required, the element containing them has to be selected and the value of the attribute has to be parsed out of the returned data. Hence, the amount of data retrieved per query and the amount of computation that is required, increases.

4.3 Bench Mark Query Set

The set of queries that were used to estimate the efficiency of the database have been defined by Prof. Beth Plale. These queries test the performance of the database based on the following categories:

- *Scoped/NonScoped Queries*: The query set has 2 Scoped queries that search for objects starting one level above the required objects. These two queries differ in the size of the data that is queried. Also, there are 2 NonScoped queries that start the search at the same level as that of the required objects and that differ in the size of the data queried.
- *Indexed/NonIndexed Queries*: The query set includes one pair of queries that test the performance of Xindice in the presence/absence of indexes. One query in this

set queries on an indexed attribute. The other query queries on a non-indexed attribute.

- *Selectivity Queries*: The query set contains 3 queries that select exactly one tuple, one percent of total number of tuples, and ten percent of total number of tuples. This selectivity reveals the performance of the database under varying quantity of data returned.
- *Joins*: Table Join is a very common and highly optimized operation in relational databases. The query set contains 2 queries that test the performance of Xindice when performing a query that requires a table join operation to be performed. Xindice does not support the join operation and hence the logic is user defined.
- *Other*: The query set also tests the performance of Xindice for database connect and update operation.

The actual queries executed and their SQL equivalents have been described in detail in Appendix C.

5. Xindice Performance

5.1 *Scoped Queries*

The following figures show the time taken by the Scoped set of queries for both, Xindice 1.1 and Xindice 1.0.

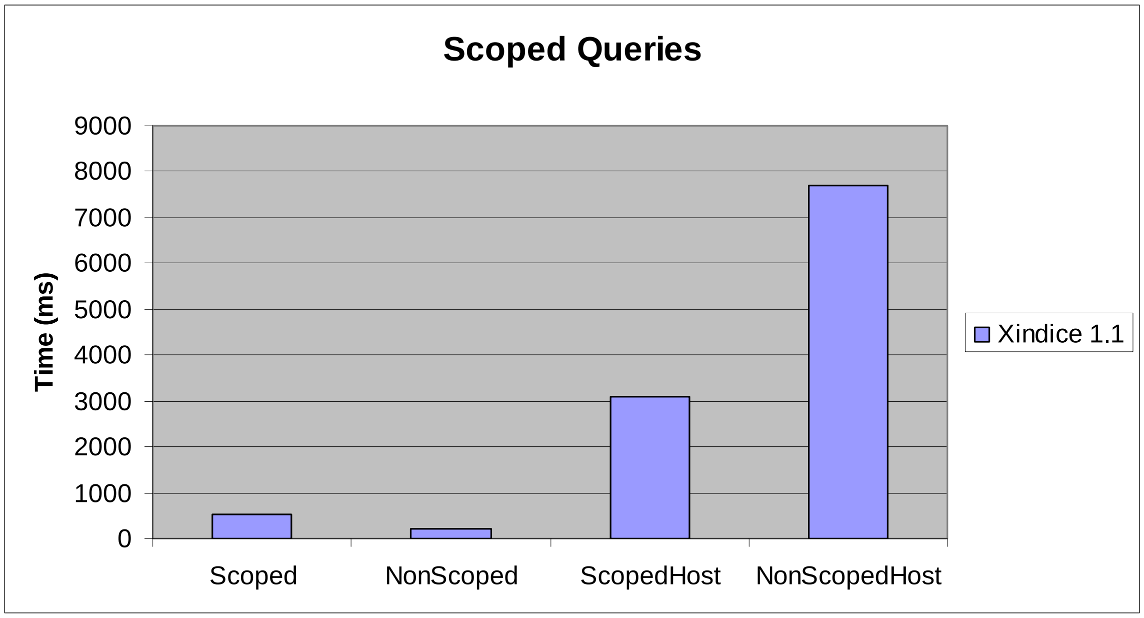


Figure 1: The above graph shows the performance of Xindice 1.1 on the 'Scoped' set of queries.

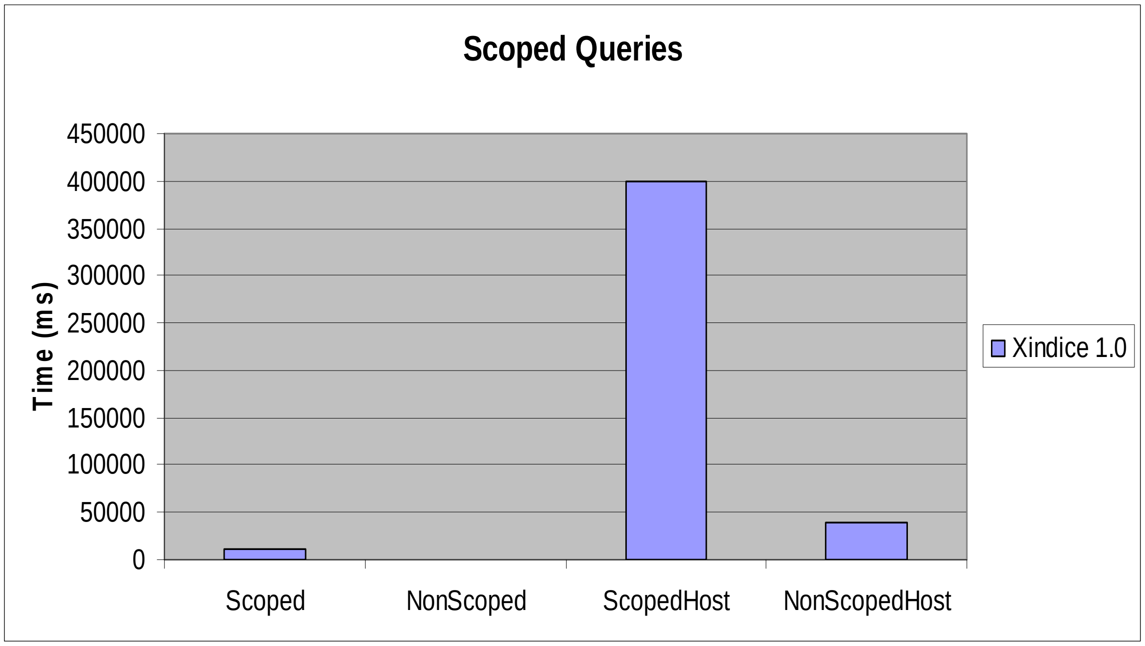


Figure 2: The above graph shows the time taken by Xindice 1.0 for the 'Scoped' set of queries.

The following conclusions are drawn based upon the Figure 1 and Figure 2:

- Xindice 1.1 has much better performance than that of Xindice1.0.
- It can be noted from the above figures that the NonScoped queries perform better than the Scoped queries. The exception to this in Figure 1 can be explained by the fact that the NonScopedHost query returns much more data as compared to the ScopedHost query.

5.2 Indexed/NonIndexed Queries

Indexing on attributes that are part of the search term improves the performance of a query in Xindice1.1 by a large factor.

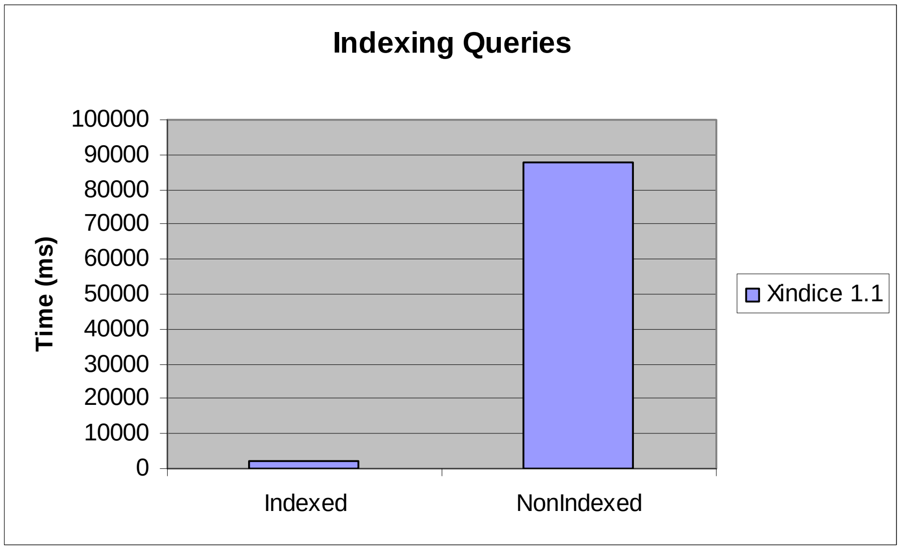


Figure 3: The above figure shows the performance of indexed versus non-indexed queries.

5.3 Selectivity Queries

The following figure illustrates the effect of selecting different number of tuples.

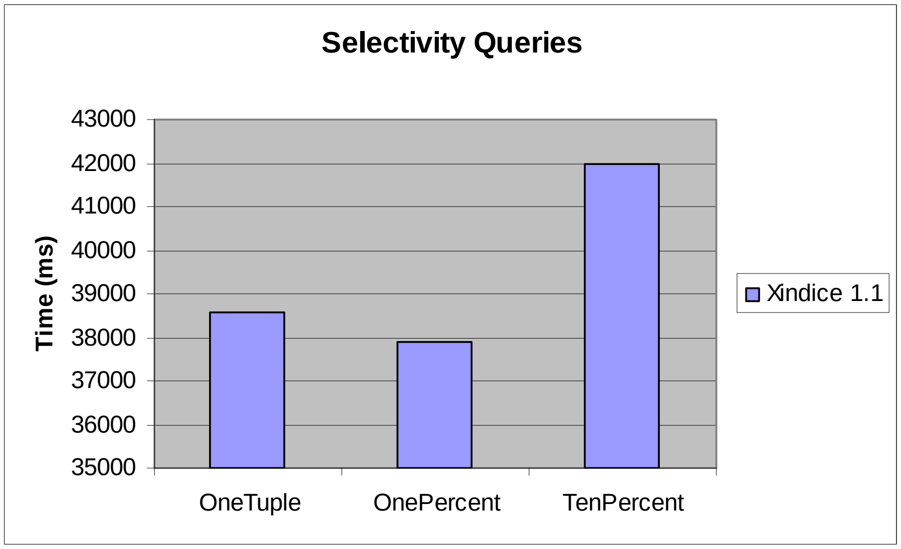


Figure 4: The above figure displays the effect of varying number of tuples.

5.4 Join Queries

Join operations are not supported by the XPath query language and hence additional processing is performed by the client when performing a join operation.

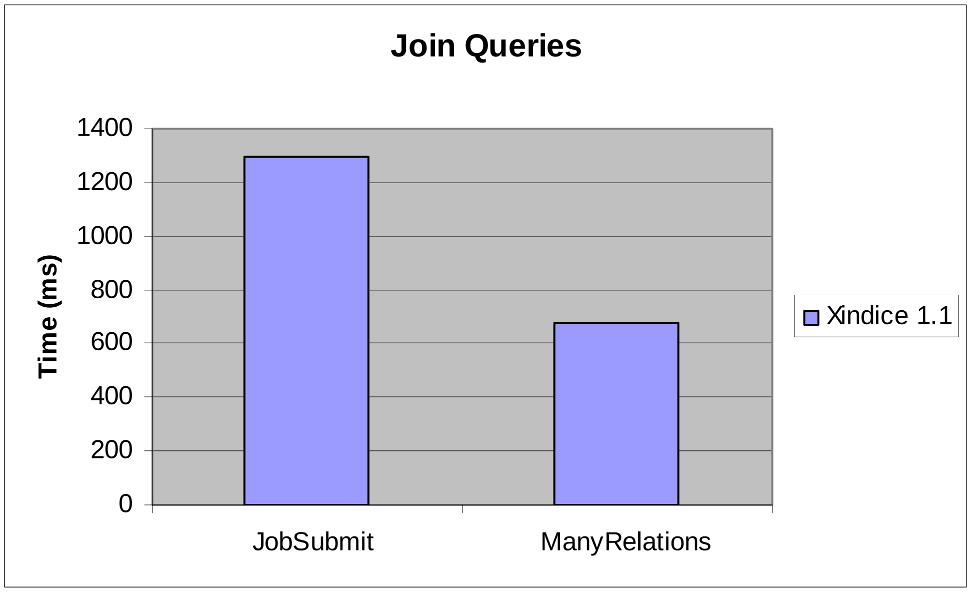


Figure 5: The above figure shows the performance of join operations using Xindice1.1.

5.5 Basic Queries

The bench mark set included the evaluation of performance on certain basic queries such as creating a database connection, creating a collection, dropping a collection and updating a document.

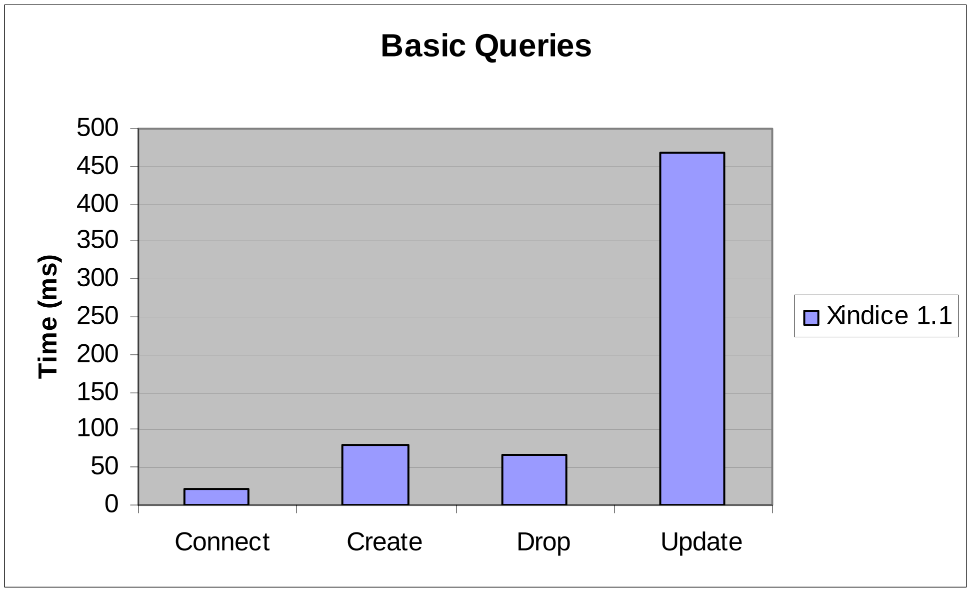


Figure 6: The above figure shows the performance of Xindice1.1 on a basic set of queries.

5.6 Higher Level Scoping

To further test the behavior of Xindice1.1 on 'Scoped' queries, a query set was defined that tested the performance of sets of scoped queries executed sequentially. The overall performance of these queries was plotted for 1000 successive iterations. The following graphs represent the results obtained.

Figure7: Performance of Xindice1.1 for a 'NonScoped' query for 1000 successive iterations.

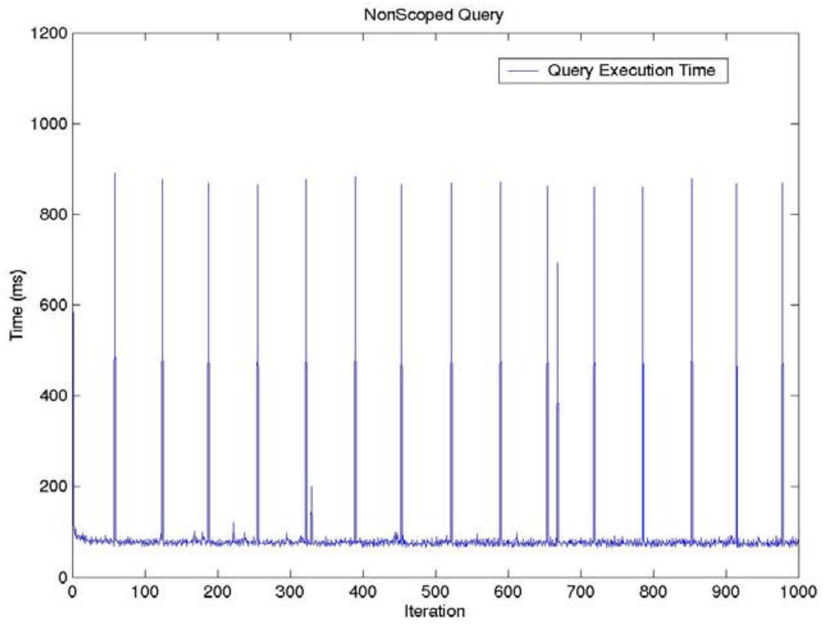


Figure 8: Performance of Xindice1.1 for a single 'Scoped' query for 1000 successive iterations.

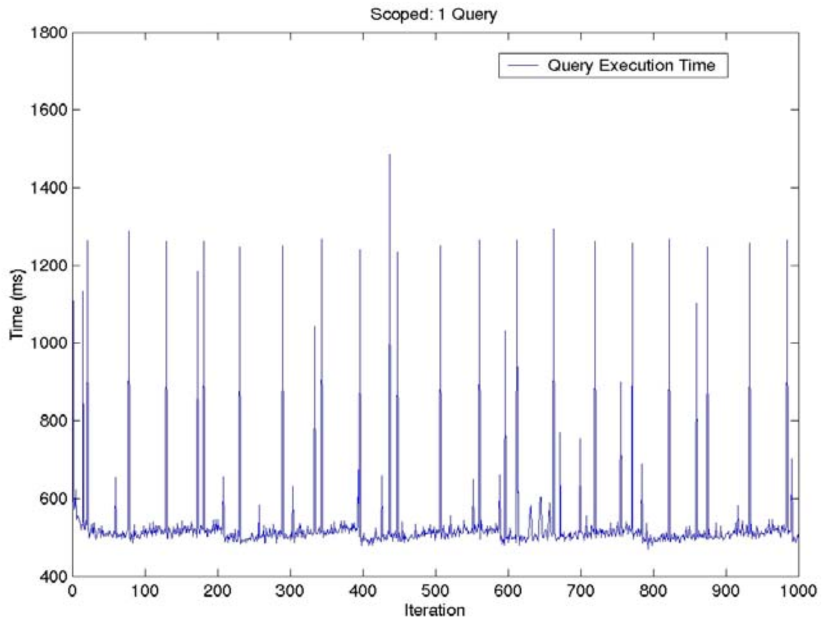


Figure 9: Performance of Xindice1.1 for two 'Scoped' queries for 1000 iterations.

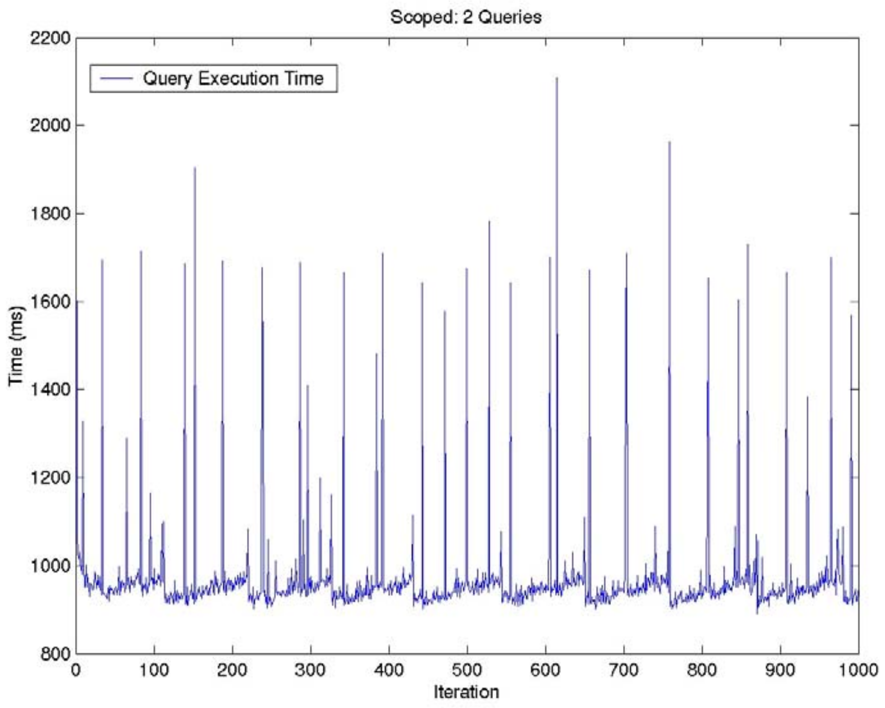
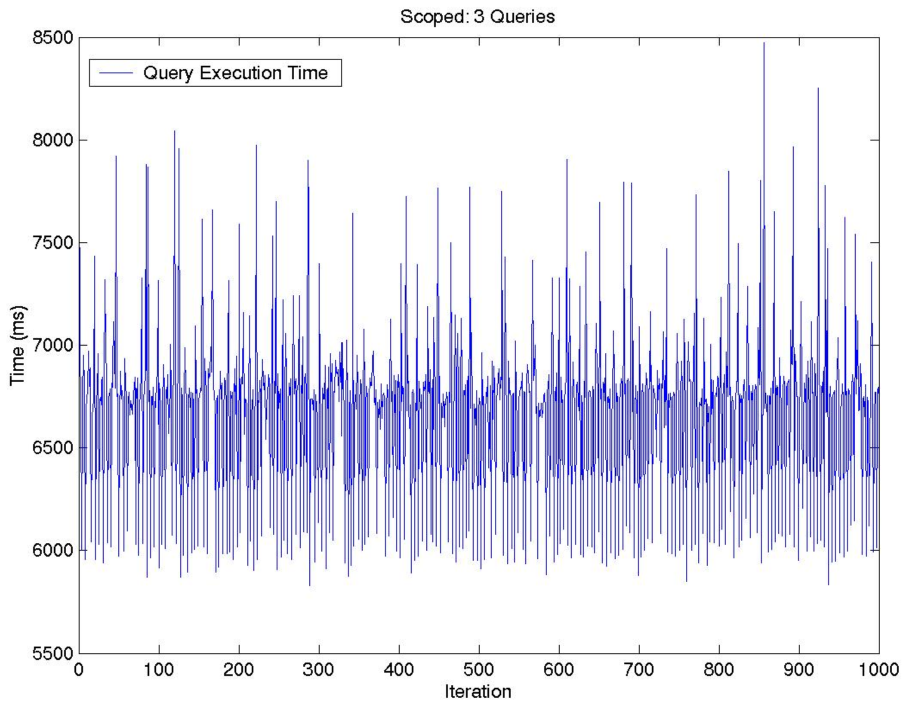


Figure 10: Performance of Xindice1.1 for three 'Scoped' queries for 1000 iterations.



It is concluded from these graphs that as the number of scoped queries executed is increased, a large amount of variation is seen in the query execution time with successive iterations. Hence the performance of Xindice1.1 with multiple scoped queries is not stable.

5.7 Byte Count

The following table shows the number of bytes returned for each query executed:

QUERY	BYTE COUNT
Scoped	7511
ScopedHost	229996
NonScoped	15969
NonScopedHost	549543
OneTuple	484
OnePercent	633397
TenPercent	691095
JobSubmit	13177
ManyRelations	40082
Indexed	140918
NonIndexed	139790
Update	3491

Since Xindice does not support attribute level querying, hence the data returned contains the entire element containing the attribute and the attribute value has to be parsed out of this data. Hence, the amount of data received from the server is much larger than the amount of data required.

6. Acknowledgements

I am extremely grateful to Dr. Beth Plale for her support and guidance during the course of this project. I also thank the team members of the Relational Grid Resource Project, Ying Liu, Rupali Parab, Charlie Moad and Craig Jacobs for their contribution to the project.

7. Appendix A

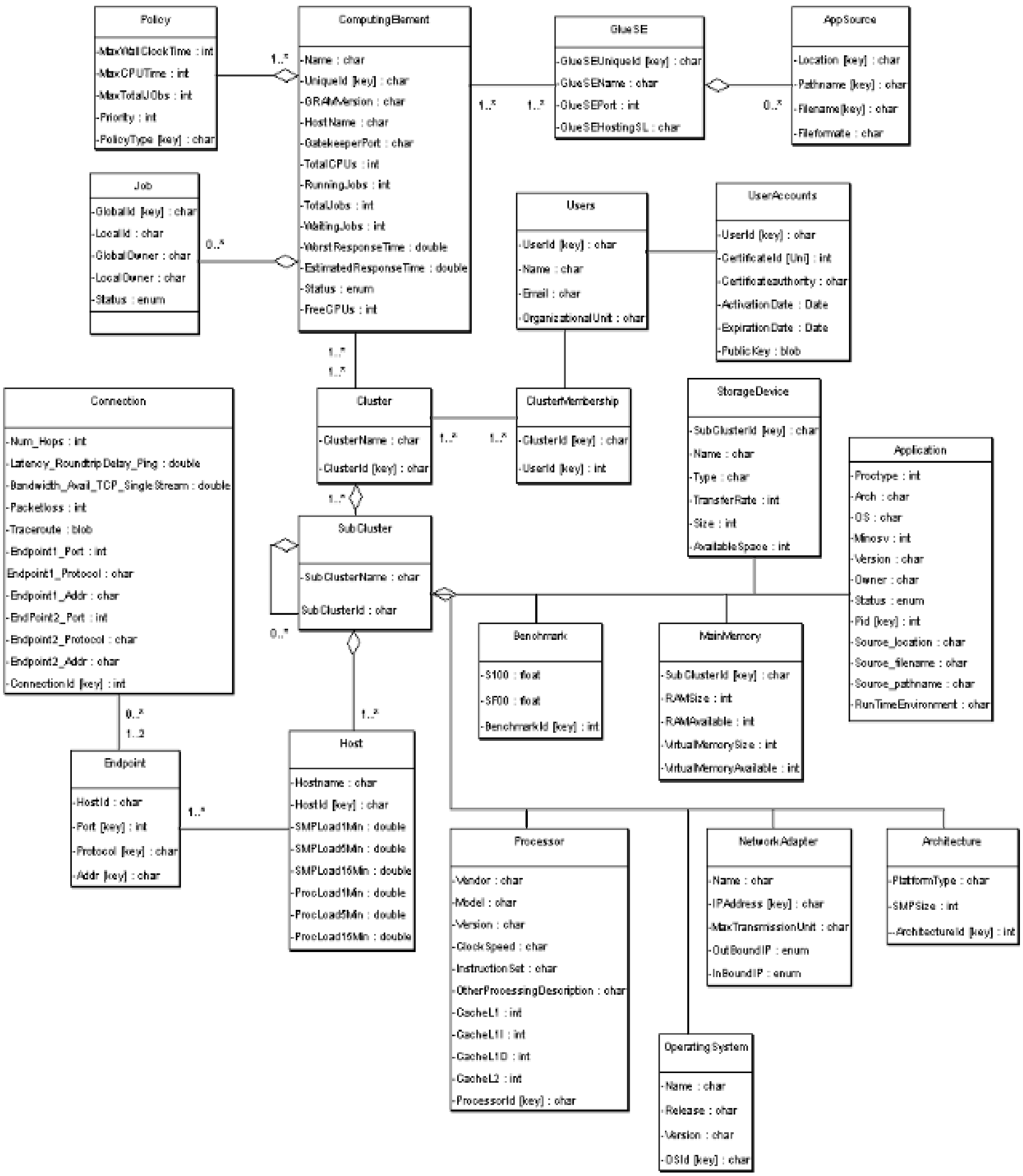
Following is a list of changes going from Xindice 1.0 to Xindice 1.1 as mentioned in the "readme" document for Xindice 1.1:

- Server side installation is now done by deploying a WAR archive within a Java servlet engine.
- The database is now deployed within a servlet engine to enable network access. This is different from in 1.0 where Xindice had its own server framework. This change was made because the custom Xindice server framework just duplicated much of the functionality provided by servlet engines. This has the nice side effect of creating more flexibility in deployment options.
- The network access API is now based on XML-RPC rather then CORBA. This was done for simplification and to eliminate the problems with the CORBA ORB and consumption of resources. It also was done to address UTF-8 encoding issues that were present with CORBA. Initial tests show a minimal performance impact from this change.
- All CORBA related code has been removed from the system.
- The server now fully supports the storage and retrieval of documents encoded as UTF-8.
- There is now an embedded version of the XML:DB API. This allows you to build Xindice applications that access the database without using the network. The API should be fully compatible with the network enabled XML:DB API implementation. An embedded database can be used by simply changing the XML:DB URI from xmldb:xindice:// to xmldb:xindice-embed://.
- The xindiceadmin tool has been removed. All commands that were previously only accessible through xindiceadmin are now available through the xindice command. This should make working with the server a little simpler.
- An option was added to the command line tools to allow the specification of namespaces to be used with XPath queries.

- On the command line tools the confirmation during deletion has been removed. Along with this the -y option that would force automatic deletion has also been removed.
- The command line tools can be run against a network version of Xindice or against a local database. See the -l and -d options to learn more about local database access.
- XMLObjects have been removed.

8. Appendix B

8.1 Hierarchical data Structure



9. Appendix C

9.1 The Query Set Details

Following is the set of queries that was used for evaluating the efficiency of the Xindice database. This description contains the SQL and XPath equivalents of the queries executed. The XPath equivalent consists of multiple queries corresponding to each SQL query and user level processing is required to perform the equivalent operation using the XPath specific queries.

In the details that follow, queryXindice() function is used for performing a select operation on an entire tuple and EvaluateQuery() is used to perform a query and extract a certain attribute.

9.1.1 Scoped Query

Result Set: 10 Tuples

SQL Query:

```
SELECT CSC.ClusterId, CSC.SubClusterId, SCO.OSId, SCP.ProcessorID
FROM Cluster_SubCluster as CSC, SubCluster_Processor as SCP,
        SubCluster_OperatingSystem as SCO
WHERE
        CSC.ClusterId = "mds.sdsc.edu" and
        CSC.SubClusterId = SCO.SubClusterId
         and  CSC.SubClusterId = SCP.SubClusterId;
```

XPath Query

This involves the following sequence of queries:

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/Cluster_SubCluster","//Cluster_SubCluster[@ClusterID='mds.sdsc.edu']");
- EvaluateQuery("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/SubCluster_Processor","//SubCluster_Processor[@SubClusterId='"+subClusterId+"']","ProcessorID");
- EvaluateQuery("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/SubCluster_OperatingSystem","//SubCluster_OperatingSystem[@SubClusterId='"+subClusterId+"']","OSId");

9.1.2 NonScoped Query

Result Set: 60 Tuples

SQL Query

```
SELECT SCO.OSId, SCO.SubClusterId
FROM SubCluster_OperatingSystem as SCO
WHERE SCO.OSId = "Mac OS";
```

XPath Query

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/SubCluster_OperatingSystem","//SubCluster_OperatingSystem[@OSId= 'Mac OS']");

9.1.3 ScopedHost Query

Result Set: 914 Tuples

SQL Query

```
SELECT SCH.HostId, CSC.SubClusterId, CSC.ClusterId
FROM SubCluster_Host as SCH, Cluster_SubCluster as CSC
WHERE
 CSC.ClusterId = "mds.sdsc.edu"  and  CSC.SubClusterId = SCH.SubClusterId;
```

XPath Query

This involved the following sequence of queries:

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/Cluster_SubCluster","//Cluster_SubCluster[@ClusterID='mds.sdsc.edu']");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/SubCluster_Host","//SubCluster_Host[@SubClusterId='"+subClusterId+"']");

9.1.4 NonScopedHost Query

Result Set: 1554 Tuples

SQL Query

```
SELECT H.HostId
FROM Host as H
WHERE H.HostId > "sharkestra50";
```

XPath Query

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/Host/","//Host[starts-with(@HostId,'tamarack')]");

9.1.5 Indexed Query

Result Set: 405 Tuples

SQL Query

```
SELECT H.HostId
FROM Host as H
WHERE H.ProcLoad1Min = .40;
```

XPath Query

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/Host","//Host[@ProcLoad1Min = 0.4]");

9.1.6 NonIndexed Query

Result Set: 356 Tuples

SQL Query

```
SELECT H.HostId
FROM Host as H
WHERE H.SMPLoad15Min = .50 or H.SMPLoad15Min = .40;
```

XPath Query

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/Host","//Host[@SMPLoad1Min=0.5 or@SMPLoad1Min = 0.4]");

9.1.7 Many Relations Query

Result Set: 1 Tuple

SQL Query

```
SELECT SCO.OSId, SCP.ProcessorID, CSC.SubClusterId, AP.Pid
FROM SubCluster_Processor as SCP, SubCluster_OperatingSystem as SCO,
Cluster_SubCluster as CSC,  SubCluster_Application as SCA, Application as AP
WHERE
CSC.ClusterId = "mds.sdsc.edu"  and  CSC.SubClusterId = SCO.SubClusterId
and CSC.SubClusterId = SCP.SubClusterId  and CSC.SubClusterId = SCA.SubClusterID
and SCA.Pid = AP.Pid  and SCO.OSId = "Linux" and SCP.ProcessorId  = "PENTIUM"
and AP.RunTimeEnvironment = "Globus 2.2" and AP.Status = "NOT RUNNING";
```

XPath Query

This involves the following sequence of queries:

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/Cluster_SubCluster","//Cluster_SubCluster[@ClusterID='mds.sdsc.edu']");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/SubCluster_Processor","//SubCluster_Processor[@ProcessorId = 'PENTIUM']");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/SubCluster_OperatingSystem","//SubCluster_OperatingSystem[@OSId = 'Linux']");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/SubCluster_Application","//SubCluster_Application[@SubClusterId = '"+SC.get(j)+"']");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/Application","//Application[@Pid = '"+ pid + "' and @Status = 'NOT RUNNING' and @RunTimeEnvironment='Globus 2.2']");

9.1.8 Job Submit Query

Result Set: 1 Tuple

SQL Query

```
SELECT CSC.SubClusterId, M.RAMSize, SCA.Pid, App.Owner, App.OS
FROM Cluster_SubCluster as CSC, SubCluster_Application as SCA,
Application as App, UserAccounts as UA, ClusterMembership as CM,
MainMemory as M
WHERE
CSC.ClusterId = "mds.sdsc.edu"  and CSC.SubClusterId = SCA.SubClusterId and
SCA.Pid = App.Pid and App.Owner = "aksharma" and App.OS = "Linux" and
App.Source_filename = "/u" and CM.UserId = "aksharma" and  UA.ActivationDate <
now() and UA.ExpirationDate >= now() and  UA.UserId = CM.UserId and
CSC.ClusterId = CM.ClusterId and M.SubClusterId = CSC.SubClusterId;
```

XPath Query

This involves the following sequence of queries along with user defined processing logic.

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/Users/UserAccounts","//UserAccounts[@UserId = 'aksharma']");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/ClusterMembership","//ClusterMembership[@ClusterId='mds.sdsc.edu' and @UserId = 'aksharma']");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/Cluster_SubCluster","//Cluster_SubCluster[@ClusterID='mds.sdsc.edu']");
- EvaluateQuery("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/MainMemory","//MainMemory[@SubClusterId = '" + subClusterId + "']","RAMSize");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/SubCluster_Application","//SubCluster_Application[@SubClusterId = '" + subClusterId + "']");
- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/Application","//Application[@Pid = '" + pid + "' and @Owner = 'aksharma' and @OS = 'Linux' and @Source_filename = '/u']");

9.1.9 One Tuple Selection Query

Result Set: 1 Tuple

SQL Query

```
SELECT Endpoint1_Addr, Endpoint1_Port, Endpoint2_Addr, Endpoint2_Port
FROM Connection
WHERE Num_Hops = 20 and Bandwidth_Avail_TCP_SingleStream = 65.62;
```

XPath Query

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/Connection","//Connection[@Num_Hops = 20 and @Bandwidth_Avail_TCP_SingleStream = 65.62]");

9.1.10 One Percent Selection Query

Result Set: 131 Tuples

SQL Query

```
SELECT Endpoint1_Addr, Endpoint1_Port, Endpoint2_Addr, Endpoint2_Port
FROM Connection
WHERE Num_Hops = 33;
```

XPath Query

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/Connection","//Connection[@Num_Hops = 33]");

9.1.11 Ten Percent Selection Query

Result Set: 1428 Tuples

SQL Query

```
SELECT Endpoint1_Addr,Endpoint1_Port,Endpoint2_Addr, Endpoint2_Port,Num_Hops
FROM Connection
WHERE Num_Hops > 89;
```

XPath Query

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/Connection","//Connection[@Num_Hops > 89]");

9.1.12 Update Query

Result Set: Updates 11 Tuples

SQL Query

```
UPDATE MainMemory
SET VirtualMemoryAvailable = (VirtualMemoryAvailable * 1.05)
WHERE
  MainMemory.SubClusterId > "sharkestra01" and
  MainMemory.SubClusterId < "sharkestra12";
```

XPath Query

This involves the following sequence of queries

Select the required data:

- queryXindice("xmldb:xindice://bitternut.cs.indiana.edu:8080/db/GlueGeneralTop/ComputingElement/Cluster/SubCluster/MainMemory","//MainMemory[starts-with(@SubClusterId,'sharkestra')]");

Perform update on each document returned in above select operation using following update string:

```
"<xu:modifications version=\"1.0\"" +"xmlns:xu=\"http://www.xmldb.org/xupdate\">" +
 "<xu:update select=\"/MainMemory[@SubClusterId = '" +
key+"']/@VirtualMemoryAvailable\">" +
Double.toString(MemAvail) +
"</xu:update>" +
"</xu:modifications>";
```

References

[1] Xindice1.1 Reference Manuals.

[2] Xindice1.1 Users List.

[3] Relational Grid Resource Project: *http://www.cs.indiana.edu/~plale/projects/RGR/*

[4] Synthetic Database Benchmark/Workload for Grid Information Servers: http://www.cs.indiana.edu/~plale/documents/Plale_Bench.pdf