University of Alberta

Transactional XML database benchmark

By

Jun Chen

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of

The requirements for the degree of Master of Science

Department of Computing Science

Edmonton, Alberta
Fall 2006

Your file Votre référence
ISBN: 978-0-494-22240-9
Our file Notre référence
ISBN: 978-0-494-22240-9

ABSTRACT

With increasing popularity of XML[1] database applications, the demand for good benchmarks for XML database application is also rising. A few XML benchmarks have been proposed recently, but few of them involve the transactions that are critical tools for maintaining data consistency in multiple user environments. In this thesis, we first extend the nested transaction [20] into the XML domain to specify the nested XML transactions. We then specify an XML benchmark, called TXMark. As a first XML benchmark with transactions as atomic units of the database operations, our benchmark provides a more accurate assessment of the performance and capabilities of XML system. We have also designed and implemented some interesting experiments. The experimental results demonstrate that the performance of XML application systems varies with respect to single or multiple user environments, and the ones adapted to our extended transactional model perform much better than those with the standard transaction model.

ACKNOWLEDGMENTS

I would like to thank Dr. Li-yan Yuan for his supervision and help through this research.

I would also like to thank the thesis defense committee members – Professor Scott Dick, Professor Davood Rafiei and Professor Ehab Elmallah for their valuable suggestions to improve the quality of this thesis.

To my wife Lisan Zhou

Table of Contents

Chapter 1

1. INTRODUCTION

In this chapter, we first introduce the basic concepts of the research, and then discuss the motivation of the proposed research. Finally, a list of contribution of the thesis research is given.

1.1 XML and XML benchmark

The eXtensible Markup Language (XML) [1] has emerged as the potential standard of exchanging data through the Internet. The nested and self-describing structure gives applications a powerful way to model data. Different types of data can be described in such a clear way that it becomes easier to exchange them. We need an XML database to store and manage the XML documents just like we have the Relational database management system for the traditional data process.

Many native XML databases (NXDs) have been proposed in the last few years. In the mean time, the RDBMS vendors also introduced their XML features in the database. We call them XML-enabled RDBMS. The differences between the NXD and XML-enabled RDBMS can summarized succinctly as:

1. NXD use a XML document as the basic storage unit while RDBMS use a row in tables as the fundamental unit.
2. NXD has its own logic model for XML documents such as XPath data model. This model is specialized for XML only. XML enabled RDBMS extends its existing model to support storing and retrieving XML document intact.
3. NXD can be implemented on any physical storage model while RDBMS has its particular model.

NXDs can manipulate the XML files very efficiently compared to XML-enabled RDBMS. The reason is very straightforward: NXDs are specially designed to handle the XML documents. Most business data is still stored in RDBMS using relational schema, so completely shifting to native XML database is currently not a very good choice for the commercial user . The reason is that native XML database cannot harness the sophisticated storage and query capability already provided by existing RDBMS [4]. So techniques that can efficiently store and query XML data using RDBMS are desperately needed. Research in this area has made a lot of progress and some techniques have been proposed in the last few years.

How to evaluate the performance of a computer system? E. O. Joslin proposed that an application benchmark is the key to meaningful computer evaluations [26] back in 1965. The practice of benchmarking has been an assessment tool in computer performance evaluation since the early 1960s [27]. A benchmark is a set of specifications, which consists of a typical application, a simulated workload and a performance metric. The essential of a benchmark is, "How long will it take this system to process the workload?"[26], the performance metric. For example in TPC-C Benchmark Version 5.7 [35], an OLTP System is used, and a bunch of queries/updates has been defined to simulate the real application workload. The primary performance measurement is defined as transactions per minute (tpmC) [35] along with the associated price-per-tpmC, and the availability date of the priced configuration. By comparing the benchmark results, we can assess the performance and evaluate the capacity and scalability of different systems. In TPC-C, the higher number comes out of the tpmC and the lower number comes out of the price-per-tpmC, the better the system is [35]. The benchmark result can also provide insights to potential bottlenecks and improvements for both users and developers. A few good benchmarks in this area have been proposed in the last few years, such as Xmark [5], Xmach-1 [6], X007 [7], Michigan Benchmark [8] and Xbench [9].

1.2 Transaction and Long Durational Transaction

The database consistency [19] under multiple user environments is usually guaranteed by properly executing concurrent transactions [29] using certain locking mechanism [19]. A database transaction is defined as an atomic unit of interactions with a DBMS. All of those interactions must be successfully completed or aborted [29]. For example, when withdrawing money from your checking account, there are three interactions with the database – account validation, balance checking, and balance updating. You can get your money only if all three interactions are completed.

A typical database transaction is assumed to last at most minutes not weeks [19]. So a transaction span considerably longer durations than their counterpart typical transaction is called long durational transaction [30] [31].

Because of its strict ACID property [19], the classical transactional theory may encounter some difficulties while dealing with applications of long durational transactions. For example, system will have to pay the high price of cascading rollback effect [40]. Cascading rollback effect can be summarized as, when one transaction abort, it will cause a lot of other transactions in the system to rollback to maintain system consistency and those transactions will cause more other transactions rollback recursively. The nested transaction theory has therefore been proposed to address such problems [18] [20] [21] [32]. We ported the nested transaction model [20] and incremental operation [18] into TXBench to solve the cascading rollback effect in the long durational application. Our experiment results verify that the incremental operation can reduce the cascading rollback effect [40].

1.3 Motivations and Transactional XML Benchmark

The proposed research is motivated by the following two facts.

First, based on the fact that both TPC-App[35] and TPC-C[36], the current industry standard database application benchmarks, address the performance of simultaneous execution of multiple transaction types that span a breadth of business functions in their specification[36]. We can assume that most real world XML database applications are

running under multiple user environments. And all proposed XML benchmarks so far run under single user environments except Xmach1 [6], and none involve the transactions.

Second, Many XML database applications are on-line applications [1], and consequently the long-durational transaction theory may provide an excellent tool for dealing with the XML applications.

Third, since a benchmark is used to assess the performance of application systems while the consistency of applications largely depends on their processing of transactions [40], we believe that any working XML benchmark must be specified based on the transaction theory.

In this thesis research, we first extend the nested transaction model [20] into the XML, re-define the XML transaction, and then define the XML transactional benchmark based on the extended transaction model.

In our benchmark, we use long durational nested transaction to simulate the real-world online transaction. After more than 10 years of rapid growth, electronic commerce [39] is everywhere. A lot of the big companies have their own web site and feature online shopping. Those online shopping transactions are very long duration transactions. People login to the website to browse and search for information about the goods they want to buy, for example like finding a vacuum made by Siemens and checking its rating and customer reviews. We put the satisfied items in a shopping cart and finally check out using a credit card, or we decide to logout without buying anything. This type of transaction takes more time compared to traditional database transaction and we can define sub-transactions for it. For example, login, search, placing item in the cart, and checkout are separated transactions within the shopping transaction. The most concerned issue about long durational transaction is a cascading abort effect [40], which rarely happens in a traditional transaction. We call it cascading abort when one of the sub transactions abort, a lot of related transactions have to abort as well. This can dramatically slow down the system . In our experiment, we proved that this effect could be greatly reduced by implementing an object database model [18].

1.4 Contribution of the thesis

Contributions of the thesis are listed below:

1. We extend the nested transaction [20] into the XML domain to specify the nested XML transactions.
2. We propose an XML benchmark, called TXMark, based on the XML transactional model. As a first XML benchmark with transactions as atomic units of the database operations, our benchmark provides a more accurate assessment of the performance and capabilities of XML systems, as well as better insights for both users and developers.
3. We have designed and implemented some interesting experiments. The experimental results prove that the incremental operation [18] can reduce the cascading rollback effect [40] in the long durational database model.

All existing XML benchmarks focus on testing individual XML queries in single user environments except Xmach1 [6]. None of them ever consider the transactions which are a critical concern in multiple user environments. Our benchmark, TX Mark, provides a solution to these problems.

1.5 Thesis Organization

In chapter 2, we present some background information on XML, some techniques in XML support of RDBMS and the reasons why they perform differently.

In chapter 3, we describe the related works in RDBMS and XML benchmark areas. We discuss the contemporary XML benchmarks one by one; reveal their design intention, test query scope, advantage and disadvantage. We also describe the existing nested transaction and object database models.

In chapter 4, we define a long durational nested transactional database model, which is used in our benchmark.

In chapter 5, we propose the first Transactional XML database benchmark.

In chapter 6, we present the benchmark results of two tested XML databases. Benchmark results are analyzed and some conclusions are made.

In chapter 7, we discuss the future works.

Chapter 2

2. XML REVISITED

This section presents some background information on XML, some techniques in XML support of RDBMS, and the reasons why they perform differently.

2.1 XML

XML stands for eXtensible Markup Language [1], which has emerged as the potential standard of exchanging data through the Internet. XML is a markup language much like HTML. One of the differences between XML and HTML is that XML tags are not predefined. You must define your own tags in the XML file. XML was designed to describe data. It is a cross-platform, software and hardware independent tool for transmitting information. The way to implement its nested and self-describing feature is using Document Type Definition (DTD) [1] or an XML Schema [1] to describe the data.

For example in the following XML file (the number in the front is the line number),

```
1.  <? Xml version="1.0"?>
2.  <!DOCTYPE email [
3.  <!ELEMENT email (from,to,heading,body)>
4.  <!ELEMENT from     (#PCDATA)>
5.  <!ELEMENT to    (#PCDATA)>
6.  <!ELEMENT heading (#PCDATA)>
7.  <!ELEMENT body    (#PCDATA)>
8.  ]>
9.  <email>
10. <from>Leo</from>
11. <to>Jun</to>
12. <heading>Reminder</heading>
13. <body>Call me</body>
14. </email>
```

DTD is wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root-element [element-declarations]>

It is interpreted like this:

!DOCTYPE email (in line 2) defines that this is a document of the type **email.**

!ELEMENT email (in line 3) defines the **email** element as having four elements: "from,to,heading,body".

!ELEMENT from (in line 4) defines the **from** element to be of the type "#PCDATA"[1].

!ELEMENT to (in line 5) defines the **to** element to be of the type "#PCDATA"[1]

And the format of this xml file can also be defined using following schema.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.cs.ualberta.ca"
xmlns="http:// www.cs.ualberta.ca "
elementFormDefault="qualified">
<xs:element name="email">
<xs:complexType>
<xs:sequence>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

This schema can interpret as the **email** element is a **complex type [1]** because it contains other elements. The other elements (from, to, heading, body) are **simple types [1]** because they do not contain other elements.

The nested and self-describing structure gives applications a powerful way to model data. Different types of data can be described in such a clear way that their exchange becomes easier. Many native XML databases have been proposed in the last few years. These database can manipulate the XML files very efficiently compared to XML enabled RDBMS. Even though more and more data is stored in XML files, most business data

is still stored in RDBMS using relational schema. Therefore completely switching to native XML databases is currently not a very good choice for commercial users. The reason is that native XML database cannot harness the sophisticated storage and query capability already provided by existing RDBMS [2]. So techniques which can store and query XML data using RDBMS efficiently are desperately needed. Research in this area has made a good deal of progress and some techniques have been proposed in the last few years.

2.2 Techniques of XML support in RDBMS

There are mainly two ways to store XML into the RDBMS. One is storing the XML as Binary Large Object and the other one is chopping up XML file into a lot of separate tables.

2.2.1 Storing the XML file as Binary Large Object

Binary Large Object is supported in most relational databases. It just stored a pointer, which points to the actual large object like an XML files in this case. The XML file locates in the operating file system separate from the database files. To query or update this XML file, it has to been parsed and manipulated using mainly either the DOM [10] or the SAX [11] interface. DOM stands for document object model. It is a tree-based API, and will construct an in-memory hierarchical tree to represent the XML document. For example for the following XML file,

```
<?xml version="1.0"?>
<Library>
<book category="Computing Science">
  <title>Oracle DBA</title>
  <author>Leo C</author>
  <year>2006</year>
  <price>45.00</price>
</book>
<book category="Computing Science">
  <title>Software Engineering</title>
  <author>John D</author>
  <year>1998</year>
  <price>50.00</price>
```

</book>
</Library>

DOM can parse it and construct a tree; following figure is a fragment of this tree.

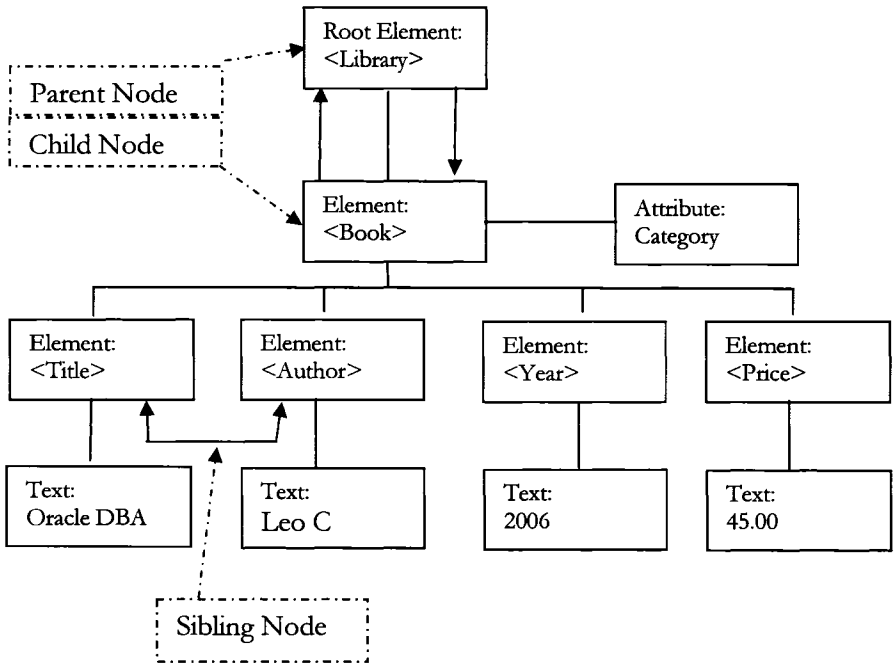


Figure 1: DOM tree

From this figure, we can see that every parent node can reach to its child node; every child node can reach to its sibling node. It is very convenient to navigate and manipulate (search and update) the tree using appropriate API. The limitation of using DOM is that the computer must have enough memory to hold the whole XML file.

SAX stands for simple API for XML; it is event-based API. It uses the callback to inform the application about the parsing event such as "start document, end document, start element, end element and etc." It is suitable for shredding the XML document into pieces and reassembling it. Using SAX, You may need to write some extra codes to manipulate (search and update) the XML file comparing to using DOM API. However SAX is much more memory efficient which is a big advantage over DOM in the multiple users' application environment.

The advantage of this method is simple and easy to implement into the existing RDBMS. The common disadvantage of storing XML file as Binary Large Object and XML native database is that they cannot harness the sophisticated storage and query capability already provided by existing RDBMS [4]. They cannot take advantage of any research breakthrough in relational database area in the future.

2.2.2 Storing the XML file into multiple tables

Instead of processing the XML file natively, researchers have been working on how to take advantage of the existing state-of-the-art techniques of high-performance data storage and retrieval in relational database area. Some researches in this area have made a lot of progress and some techniques have been proposed in the last few years. The details of the techniques may vary from one to the other, but most of them are composed of the following steps: (a) create meta tables (b) shred the XML documents into those tables (c) convert XML query into SQL query (d) reconstruct XML document with SQL result set. (a) and (b) belong to the relational schema generation category; methods used in this category decide the methods used in (c) and (d).

In other words, relational schema generation serves to define how many tables are needed to store the pieces of the chopped up XML files. How to shred the XML file? The answer to this question decides issues like what is the structure of the table, how many tables are to be used, how much disk space is needed to store the documents and how efficient it is to query and reconstruct XML documents. So the algorithm to shred the XML file is very important compared to the other algorithms of manipulating XML files in RDBMS.

The following are well-known algorithms in this area.

Edge Mapping Algorithms: [12] [13]

In Naïve Edge mapping algorithm [12], XML file will be convert into a tree. Every node in the tree is assigned an id and an order number. Based on the id and order number, the whole XML file is chopped up into a lot of edges which is composed of two nodes where the parent node is source node and child node is target node. All the edges are stored in the same table "Edge" with following columns: (id, order_number, name, flag,

target). This table stores the order id of the source node, the target node order number and flag that tells if the edge is an inter-object reference [12] or just point to a leaf node. By properly joining the edge, we can easily reconstruct the original XML file.

Based on this algorithm, a few similar algorithms have been proposed because the different ways to assign the order to the nodes in the XML file.

1) We can assign the unique global order to every node like Global order & corresponding edge mapping algorithm [13]. Thus every node has the absolute order information in the whole document which makes it easy to reconstruct the original XML because the order of the elements in XML file has to be intact after reconstruction.

2) Even though above algorithms are good at query, it is very time consuming when it comes to update XML file, because every node with bigger order number than the updated node have to be updated. To address this issue, local order & corresponding edge mapping algorithm [13] has been proposed, the node will be assigned the order number among the sibling nodes. This algorithm can greatly speed up the XML update operation.

3) Dewey order & corresponding edge mapping algorithm has been proposed to combine above two approaches' advantage. It uses the Dewey Decimal Classification technique [34] which devised by Melvil Dewey in 1876, is a method of classifying and cataloging library materials by subject. Every node is assigned a vector that represents the path from root to this node. This approach can handle queries efficiently just like Global order if the overhead caused by the extra space storing the paths can be ignore. Fortunately, the UTF-8 encoding technique [14] can used to reduce this overhead. To update operation, the re-numbering only affects the sibling and their descendants. So this approach can perform better for the mix of update and query operations compare to Global and Local Order algorithm.

Path mapping algorithm (Monet Model) [15]

In the Edge mapping, the idea of binary approach is very good. But it is not suitable for the wild cast XML queries. It only spreads the data according to the tag name. There is still too much irrelevant data involved in the join-operation, it will slow down the system and consume extra disk space. The so call "Path mapping" is proposed in [15]. It

decomposes the XML documents into binary association to the path level. Path is defined as from root to the every reachable node including elements and attributes. Every node in the hierarchical tree is assigned a global order number id. In this approach, the number of tables will equal to the number of path. Also another table is needed to store the information of paths; we call it Meta table. By joining Meta table to path table, it can reconstruct the XML effectively. And much less redundant information is stored in the database.

Every algorithm has its own advantage and disadvantage. The fact is that different XML databases implement different algorithms. So it is obvious that different databases will show different results for any given benchmark.

Chapter 3

3. RELATED RESEARCH

In this chapter, we first introduce some existing benchmarks for RDBMS and XML Database systems and analyze the shortcomings of these XML benchmarks. Then we discuss the nested transaction model [20], which has been ported into our TXMark. At last, we talk about object-based database model [18], which has been implemented in our experiment to solve the cascading abort effect in the long durational transaction model.

3.1 TPC Benchmark [3]

The TPC is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry [3]. The current TPC benchmarks include TPC-App v1.1.1 [35], TPC-C v5.7 [36] and TPC-H.v2.5.0 [37]. They are all focus on the RDBMS system benchmarking.

3.1.1 TPC-App [35]

TPC-App is an application server and web services benchmark. It simulates a business-to-business [38] transactional application server. The workload is designed to measure that following aspects:

- Web data exchange
- Distributed transaction management
- Messaging
- Web service response

- Simultaneous execution of long durational multiple transaction
- Databases scale up drive test
- Transaction integrity (ACID properties)

The TPC-App benchmark's performance metrics include the SIPS[35] per Application Server SYSTEM, Total SIPS, the associated price per SIPS (e.g.,$USD/SIPS) and the Availability Date of the priced configuration. SIPS stands for Web Service Interactions per second. Because it is the most current benchmark in TPC, there is only one posted test result in [3].

3.1.2 TPC-C [36]

TPC-C is an on-line transaction processing (OLTP) benchmark. In the simulated business model, a company owns a number of warehouses. The system scale just as the company expands and new warehouses are created. Each warehouse serves a certain number of customers. The workload will be simulated to test the following:

- Multiple simultaneous on-line transactions
- System response with time constrain
- Disk input/output drive test
- Transaction integrity (ACID properties)
- Data access drive test
- Databases scale up drive test
- Data contention test

The primary performance measurement for TPC-C benchmark is defined as transactions per minute (tpmC) [36] along with the associated price-per-tpmC, and the availability date of the priced configuration. The price in the price-per-tpmC is the total three year pricing.

3.1.3 TPC-H [37]

The TPC-H benchmark is a decision support benchmark. It simulates a lot of business oriented ad-hoc queries and concurrent data modifications. This benchmark examines large volumes of data, test complex queries and come up business decision. The primary performance metric used in TPC-H is QphH@Size[37],the Composite Query-per-Hour along with the TPC-H Price/Performance metric is expressed as $/QphH@Size.

3.2 Existing XML database benchmark

3.2.1 XMach-1

The XMach-1 benchmark [6] is one of the first XML benchmarks designed for XML databases. It is the only XML benchmark to work under multiple user modes. And it is also the only one that presents the metric of performance for the whole system. The SUT of XMach-1 simulates a web application that uses XML database as a backbone storage system for XML documents. Beside the XML database, the SUT consists of an application server as well, see Figure 2.

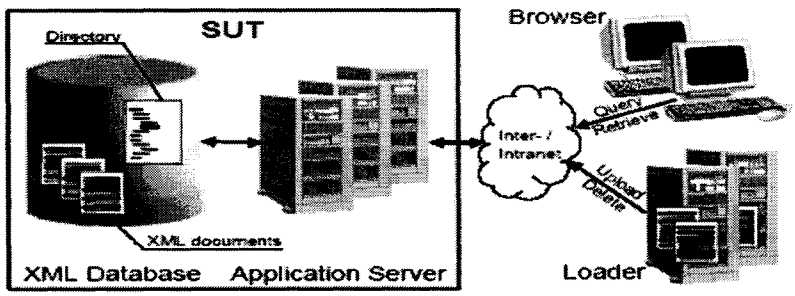


Figure 2: XMach-1 benchmark architecture [6]

Both types of XML documents-- document-centric [6] and data-centric [6] – are supported in the SUT while the document-centric XML data is the majority. Every document in the system is associated with a unique URL. There is a directory document that links all the references to the rest of the documents using this URL. The SUT

supports DTDs but not XML Schema, XML namespaces, CDATA sections and entities.

Database can scale up by increasing the number of documents in the system and by increasing the number of concurrent accesses to the SUT. The number of initial text documents is at least 1000 with a scale option by a factor of 10, 100, 1000, etc. The size of the document varies from 2K to 100K. The performance metric is measured in Xqps (XML queries per second).

3.2.2 XBench - A Family of Benchmarks for XML DBMSs

Xbench[9] is the most comprehensive XML benchmark in defining the benchmark input data source and workload which claims that it provides a family of benchmarks for XML DBMSs. The data source includes text-centric and data-centric documents. It also supports the choice of a single document or multiple documents. The data size of these documents may vary from 10M to 10G with limited options of 10 MB, 100 MB, 1 GB, and 10 GB. The number of documents may scale up to over thousands and more.

With the combination of data source type and document dimension, there are four possibilities for a database: data-centric/single-document (DC/SD), data-centric/multi-document (DC/MD), text-centric/single-document (TC/SD) and text-centric/multi-document (TC/MD). The following are the samples provided for each combination in [9]:

DC/SD: online shopping catalogs and internet movie database (IMDB);

DC/MD: e-commerce transactional data;

TC/SD: GCIDE dictionary and Oxford English dictionary ;

TC/MD: Reuter's news corpus, Springer digital library, Shakespeare's works , and DBLP data .

One of the main shortcomings of XBench is that the SUT works under single user mode. No concurrent access to the system is supported. The multiple users work mode is supposed to be implemented in the future. Another weakness of XBench is that only query workloads are supported in the first version, but the designer plans to include the update and bulk-loading workloads. The following are the queries classified by functionality:

1. Exact match.
2. Function application.
3. Ordered access.
4. Quantifier.
5. Regular path expressions.
6. Sorting.
7. Document construction.
8. Irregular data.
9. Retrieve Individual document.
10. Text search.
11. References and joins.
12. Data Type casts.

There are 20 queries in the workload collection which includes almost all the queries defined in XML Query Use Cases [2]. 18 out of 20 queries apply for both text-centric document and data-centric document data sources.

The lack of clear definition of the metric of performance of the SUT is obviously the biggest drawback in Xbench. The lack of definition of XML oriented transaction is another disadvantage, because the data derived from TPC-W[3] cannot reflect the real XML application data.

3.2.3 XMark

The XMark [5] is designed to evaluate the query processing ability of the XML database in the operation level. It intently designs a set of queries to challenge the query processor. The feature of bulk loading makes it unique among all those XML benchmarks. The document used in XMark benchmark is modeled after a database as deployed at an Internet auction site[5]. The document conforms to a specific DTD.

20 queries are proposed to cover the major aspects of XML query processing ability which expand to following 14 categories.

1. Exact Match.
2. Ordered Access.
3. Casting.
4. Regular Path Expressions.
5. Chasing Reference.
6. Construction of Complex Results.
7. Joins on Values.
8. Reconstruction.
9. Full Text.
10. Path Traversals.
11. Missing Elements.
12. Functions Application.
13. Sorting.
14. Aggregation.

Xmark can only scale up by increasing the size of the XML document. The optional size for the document can be 10 MB (tiny), 100 MB (standard) , 1 GB (large), and 10 GB

(huge). There is no overall system performance metric in Xmark, instead it shows the average response time on each of tested queries. We may assume it works under single user mode because it did not mention anything about this aspect.

3.2.4 XOO7

The XOO7[7] benchmark is an XML version of the OO7 benchmark [17] which was designed to evaluate the performance of object-oriented database management system (OODBMS). So the application model tested under OO7 was transformed to XML version to be tested in XOO7 benchmark. Three groups of queries are tested in XOO7 benchmark.

Queries in group I work like traditional database queries such as range searches and joins. Queries in Group II are navigational queries to test the tree traversal ability. Queries in Group III are document oriented queries to test the performance of retrieving data from the document while preserving the order.

The time and space needed to convert the XOO7 test data file to the tested XML management system are two system performance factors. The DTD is mandatory for the converted XML file. The size of test data file has three options: 4.2MB, 8.4MB and 12.8 MB. The main performance metric of XOO7 is the query's response time for each query group. Without practical SUT definition, we can also assume XOO7 is a single user benchmark.

3.2.5 Michigan Benchmark

Of all the existing XML benchmarks, the Michigan benchmark [8] is the only one focusing on the micro aspect of basic query operations while the rest concentrate on testing XML database working in a simulated SUT. The XML file tested using Michigan Benchmark must conform to a schema and has the most depth of 16 levels. The fan-out

of nodes at each level is used as scale up factor for the system. The types of elements in the tested XML document have only two options. The number of attributes for each element limits to seven. The queries tested in Michigan Benchmark range from returned structure query, simple selection, structural selection, value-based join, pointer-based join to aggregation query and update queries. The performance metric is the query response time of each tested query.

3.2.6 Disadvantage of Existing XML Benchmark

Most of existing XML benchmarks focus on testing XML queries. Only Xmach1 [6] works under multiple user modes. There is still no XML benchmark involved in the transaction concept like TPC benchmarks can do so far. So we propose the first transactional XML database benchmark to address the real world long durational transaction simulation issue in XML database benchmarks.

3.3 Nested transaction model [20]

In 1985, Moss [20] introduced a new transaction model – nested transaction model. The basic idea of nested transaction model is that all the transactions can be linked as a tree. The root of the tree is the main transaction, it can have a lot of sub transactions as its child node. Those sub transactions can also have their own sub transactions. The real database object access can only happen in leaves transaction. Moss also proposed a new two-phase locking mechanism to synchronize the nested transactions.

1. A transaction cannot hold a lock in WRITE mode for one object if there are any other transactions which are not the ancestors of the requesting transaction holding a lock (WRITE or READ) for that object.

2. A transaction cannot hold a lock in READ mode for one object if there are any other transactions which are not the ancestors of the requesting transaction holding a WRITE lock for that object.

3. The transaction release the locks (WRITE and READ) when it aborts. All its ancestors holding the same lock in the same mode have to do that recursively.

4. The parent transaction will inherit all the locks held by its child transaction after its child transaction commits.

We used this model in our benchmark. However, when this model was applied to long durational transaction like online shopping transaction, the recursive cascading rollback effect [40] and WRITE lock blocking [40] will hurt the system performance. To solve this, we need to port some ideas of object database model[18] proposed in 1996 into the nested model.

3.4 Incremental Operation in Object-based database model

In the nested database model, there are two operations defined: WRITE and READ. The two phase locking mechanisms [19] also have the same locking mode accordingly in the nested model. To solve the cascading rollback effect [40] and WRITE lock blocking [40] in the nested model, this object-based model [18] introduced another database operation – incremental operation and another locking model – incremental lock [18]. The incremental operation is also an update operation like WRITE operation, the only difference is that the value of the object accessed by the incremental operation can be described using this formula (Original value = new value + numeric constant) while the value of the object accessed by the write operation cannot.

This model can greatly boost the system performance by allowing maximum parallel transactions. This can be explained using a very simple example. Say, in two traditional long durational nested transactions, there is a withdraw sub transaction in each main transaction. These two withdraw sub transactions happen need to update the same object (like updating the balance of a bank account). In the nested database model, one of the main transactions has to wait until the other one commits or aborts. Or the waiting transaction aborts because of time out setting.

If this update operation can be interpreted as incremental operations, we can make those two transactions working concurrently with a proper locking and rollback mechanism

[18]. If the balance update operation can be interpreted as an incremental operation (balance = balance – withdraw). As long as the balance is greater than zero, those sub transactions can both get the incremental lock without waiting the other main transaction to rollback or commit even though the actual update operations cannot happen at the same time. If one of the transaction chooses to rollback, it only need to do a reverse operation (balance = balance + withdraw) to reimburse the difference caused by the incremental operation to the object. It will not cause the other transaction rollback while the database consistency preserved [18]. We have ported the above nested and object-based database model into our TXMark.

Chapter 4

4. DATABASE MODEL

Database is a collection of abstract data objects which can only be accessed through the operations defined by the specifications of the objects [18]. By combining the nested transaction model and object-based database model, we introduce our database model used in TXMark. We discuss three operations in this model and their conflict table. We also look into the detail of the atomic transaction, nested transaction definition and nested serializability.

4.1 Operations

There are three operations for every object in our model. They are increment [18], read and write operations and they are all atomic [19]. For any accessed data objects, if the new value of the objects can be represented by original values using this formula , new value = original value + a numeric constant, we say that this object is accessed by an increment operation. The object is called being accessed by read operation if the value of the object doesn't change after the operation; any operations other than the above two are called write operations which not only can update existing objects but also can add new objects and remove existing objects.

4.2 Atomic Transaction Definition

Users can interact with the database by executing transactions [19] which are composed of at least one operation defined for the objects. All the operations in a transaction can only be performed sequentially even though the transactions can be performed concurrently. If two operations in different transactions try to access the same object consecutively, it may lead to potential conflict situation where different execution order

of these two operations may affect the result of the operations or affect the result of their following operations. Figure 3 is the conflict table. We say that two operations conflict if both operations access the same data object and one of these operations is a write operation. And two operations also conflict if both operations access the same data object and one of these operations is increment operation while the other one is not.

Operation/Operation	Read	Write	Increment
Read	No conflict	Conflict	Conflict
Write	Conflict	Conflict	Conflict
Increment	Conflict	Conflict	No Conflict

Figure 3: conflict table for operations

A transaction is atomic transaction if it is only composed of operations. The following is the commit, rollback and reverse rule of atomic transaction:

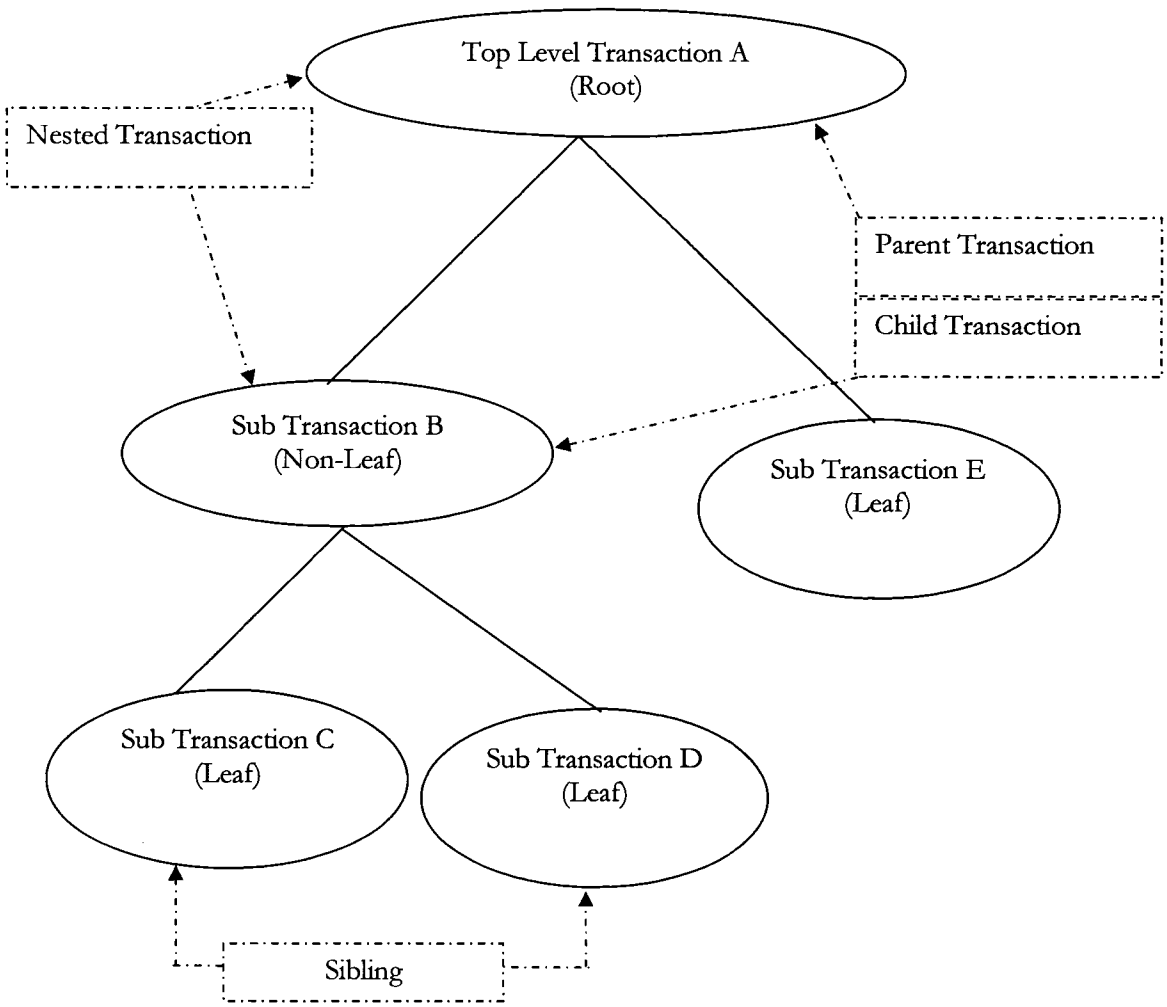
1. An Atomic transaction is committed only if all the operations finished successfully, otherwise the transaction needs to rollback.
2. Changes to the database made by the atomic transaction cannot be seen outside this transaction before it commits. After committed, all those changes can be seen by any other atomic transactions in the system.
3. The committed atomic transaction can be reversed.
4. After an atomic transaction has been reversed, the objects added by this transaction should have been removed; the objects deleted by this transaction should have been recovered. The objects updated by this transaction should have been recovered using rule 5.
5. If the object was updated by increment operation, we reverse the update by reimbursing the difference to the current values of the updated object. Otherwise, we reverse the update by replacing the current values with the original value of the updated object.

A transaction can be called nested transaction [20] if it applies to the following definition.

4.3 Nested Transaction Model Definition

1. A nested transaction is an upside down tree which consists of at least one atomic transaction. Every leaf node is an atomic transaction. Every non-leaf node is a nested transaction.
2. The root of the tree is the top-level nested transaction; all the other transactions are sub-transactions. A transaction predecessor in the tree is a parent transaction [21]; a sub-transaction at the next lower level is a child transaction [21]. Any children transactions, which share the same parent transaction, are sibling transactions to each other.
3. Only the atomic transaction performs the actual database operations.
4. The children transactions with the same parent transaction can only be executed sequentially.
5. The parent transaction can demand its children transactions to execute in certain order. Otherwise, the sibling transactions can execute without any priority order.
6. A nested transaction is terminated only if all its children transactions committed or rollback

The simulated tree graph is shown below.



Nested transaction diagram

We need to govern the behavior of the nested transaction by introducing some commit and rollback rules, which is quite different from the rules used in traditional transactions [19]. The purpose of these rules is to make sure that the nested transaction can preserve the satisfaction of ACID properties [21]. Based on these rules, sub-transaction can be implemented in such a way that it can notify the parent transaction to commit or

rollback. And parent transaction can also decide when and how to terminate the sub-transaction. [21]

The following is the commit and rollback rule of nested transaction:

1. A failed child transaction can be re-executed by its parent transaction.
2. A nested transaction commits only after all its children transactions are committed if its children transactions have to be executed in certain order.
3. A nested transaction commits after all its children transactions committed or aborted if its children transactions can be executed without any priority order.
4. Changes to database made by a nested transaction can be accessed by any transactions in the system after it was committed.
5. When a nested transaction rolls back , it reverses all its committed children transaction recursively with the rule that the last one committed the first one to be reversed.
6. To reverse an atomic transaction please refers to reverse rule of atomic transaction.
7. Changes to database made by the committed top-level transaction are final.

4.4 Nested Serializability

Serializability Definitions Revisit

We need to revisit some useful definitions related to database serializability before we specify our definition and theorem.

Conflicting operations:

Two operations from different transactions are considered to be *conflict* to each other if the execution order of these operations may change the consistency status of the database. [40] For example, write and increment operations in our model are a pair of conflicting operations.

History

A history is a sequence of operations from a set of transactions that preserves the order of operations from each individual transaction in the set. [40]

History conflict equivalent

According to the original definition in [40], two histories are conflict equivalent, if

1. the number of operations in the histories are the same; and
2. the order of any pair of conflicting operations in the histories is the same.

Serial history

A history is *serial* if, for any two transactions, either all operations in one transaction appear before the operations in the other or vice versa. [40]

Cascading rollback effect

Cascading rollback effect can be summarized as, when one transaction aborts, it may cause other transactions in the system to rollback to maintain system consistency and those transactions may cause other transactions rollback recursively until system reach a consistency state. [40] The cascading rollback effect is always caused by improper execution order of some conflicting operations. [40]

We say a history is conflict serializable if it is conflict equivalent to a serial history [40]. Traditional notion of conflict serializability [40] can be redefined to be more stringent to apply to our nested transaction model. We are using the following definition of the serializability.

Definition 4.1 [18]

A history is SR-ACA serializable if

(1) the history is conflict serializable; and

(2) it remains conflict serializable after the operations of any subset of uncommitted transactions are removed from the history.

The second condition of the SR-ACA Serializability is to avoid cascading rollback effect. [18] Which means the removal of the aborted transactions will not affect the uncommitted transaction in the history.

Man Hon Wong also specifies a sufficient condition for this SR-ACA Serializability (without proof) [18]. For our purpose, a different sufficient condition for the serializability is given below.

First, some useful notations are needed.

Let T_n be a top-level nested transaction where n can be any number.

Let O be an Object in database.

Let $R(T_n, O)$ be a Read operation accessed the Object O in T_n.

Let $W(T_n, O)$ be a Write operation accessed the Object O in T_n.

Let $I(T_n, O)$ be an Increment operation accessed the Object O in T_n.

Let $First(W(T_n,O))$ be the time when the first Write operation accessed the Object O in T_n.

Let $First(I(T_n,O))$ be the time when the first Increment operation accessed the Object O in T_n.

Let $CT(T_n)$ be the time when T_n finished (committed or roll backed) or be the time when the history ended, whichever is earlier.

Theorem:

A history is SR-ACA serializable if the following two conditions are satisfied:

Condition 1: For any object O and any operation $W(T_i,O)$, there is no $R(T_j,O)$, $W(T_j,O)$ or $I(T_j,O)$ during the period from the time $First(W(T_i,O))$ to the time $CT(T_i)$.

Condition 2: For any object O and any operation $I(T_i,O)$, there is no $R(T_j,O)$ or $W(T_j,O)$ during the period from the time $First(I(T_i,O))$ to the time $CT(T_i)$.

(Both T_i and T_j are any transactions in the history and $i \neq j$)

Proof:

Let T_i and T_j be two transactions in the concerned history Hc.

We define $T_i < T_j$ if there exists an object O such that First($W(T_n,O)$) or First($I(T_n,O)$) before T_j access O with conflicting operation.

We first show that if $T_i < T_j$, then $T_j < T_i$ is not true. Assume not, then we have $T_i < T_j$ and $T_j < T_i$. By the two conditions in the theorem, if $T_i < T_j$ for an Object O, there are no transactions shall access O with conflicting operation before T_i commits or the history ends, which contradicts to $T_j < T_i$.

Similarly, we can show that < is transitive, that is if $T_i < T_j$ and $T_j < T_k$, then $T_i < T_k$.

Now, we define a relationship $\leq$ on the set of all transactions as:

$T_i \leq T_i$, for all T_i

and $T_i \leq T_j$ if $T_i < T_i$, for $i \neq j$

Then, $\leq$ is reflexive, anti-symmetric and transitive. Thus $\leq$ specifies a partial order. Let Hs be a serial history compatible with the partial order. Now it is straight forward to show Hc is conflict equivalent to Hs.

Here we are going to show that Hc is conflict equivalent to Hs after all operations of any subset of uncommitted transactions are removed from the history.

All we need is to show that the removal does not affect the partial order on the remaining transactions. This follows that fact that for any removed transaction Tr.

Tr < Tk is not true for any Tk where $r \neq k$.

Assume not. Since Tr is uncommitted transaction, by the two conditions in the theorem, no transactions shall access (with conflicting operation) any objects which updated (Write/Incremental) by Tr. This contradicts to Tr < Tk.

This completes the proof.

Chapter 5

5. TRANSACTIONAL XML DATABASE BENCHMARK

In this chapter, we discuss the simulated system in our benchmark and show the entities and relationships diagram. We also define the transactions in the system, the detail of workload, system properties, database population, and performance metric used in TXMark.

5.1 The Simulated Business and Application Environment

After more than 10 years of rapid growth, electronic commerce [39] is everywhere. And eleven business models for electronic markets [39] have been classified to define the electronic counterpart of traditional forms of doing business [39]. The business-to-business (B2B) and business-to-consumer (B2C) models are two typical forms of electronic commerce [39]. Business-to-business model is a model to describe the behaviors about how to do business between companies, while business-to-consumer model provides the guide to the companies about how to serve their online customers.

We use the commodities supply chain management system model [38] in figure 4 as the simulated business application environment. This model does not represent any particular business activity. The company in the model can be any industrial company that sells and purchases products online. It has many warehouses in which keep the products for sale. The customers can be interpreted as anybody who shops online and any terminals within the company. Suppliers here stand for any commodity manufacturers which provide products to the company. B2B (The company restocks items from suppliers) [38] and B2C (The company provides goods to Customers) [38] models are combined in this simulated environment. This application may not cover all the operations found in any particular real world production applications, but it keeps

those essential activities like category searching, order placing and restocking which are common and important in all supply chain management systems [38]. The company in figure 4 can have a lot of warehouses whose number increase as the company's business expands. Each warehouse serves a certain number of customers. All warehouses maintain stocks for all the items sold by the Company.

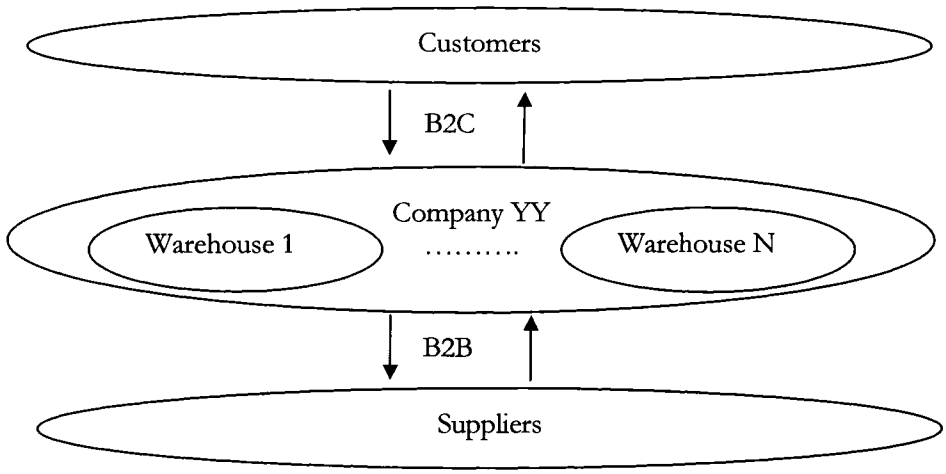


Figure 4: supply chain management system model [38]

There are two work flows (shopping and restocking) in the system.

Shopping:

The customer searches online to find the goods and put them into to the shopping cart. A customer can put any items into his cart whether or not they are available. If the item is not in-stock at the nearest warehouse from customer's shipping address then it will be supplied by another warehouse where the item is available.

Prior to check out, the system checks the warehouse to see if the items in the cart are available and return a list of unavailable items if there is any. Customer can choose to continue to do more shopping or choose to pay for the available goods.

After check out, the system updates the customer's account balance, warehouse stock, shopping history and shopping cart.

Restocking:

The system checks the warehouse stock every hour to get a list of under-stocked goods and sends this list to the corresponding suppliers. The warehouse receives the shipped goods from suppliers and updates the warehouse stock.

5.2 Database Entities, Relationships and Characteristics:

The components of the database are defined to consist of six separate and individual XML files. The relationships among these XML files are defined in the entity-relationship diagram shown in Figure 5.

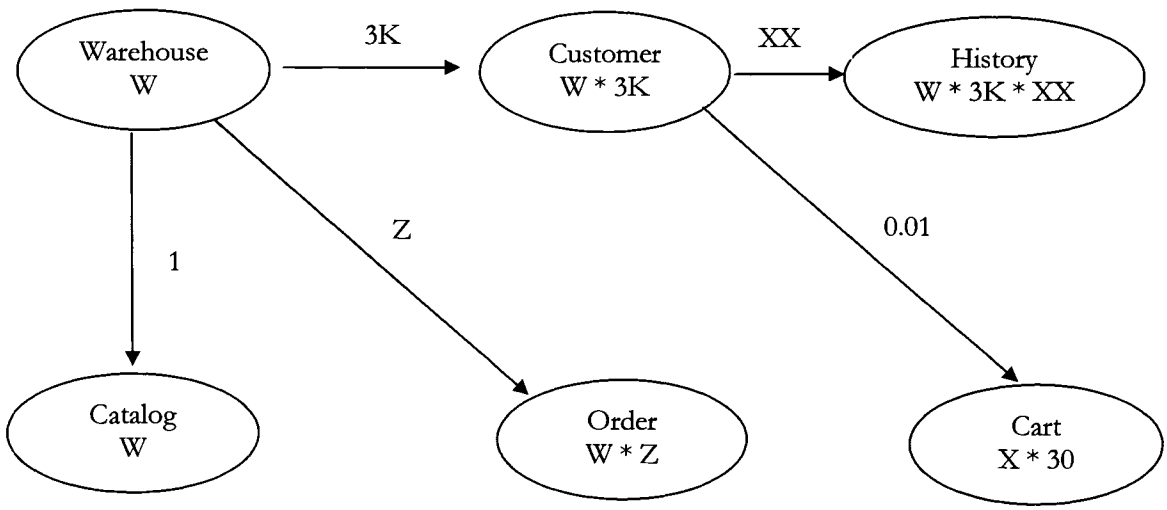


Figure 5: E-R Diagram

Legend:

• All numbers shown illustrate the database population requirements (see 5.6).

• The numbers in the entity blocks represent the cardinality of the tables (number of elements). These numbers are factored by W, the number of Warehouses, to illustrate the database scaling. (See 5.6).

• The numbers next to the relationship arrows represent the cardinality of the relationships (average number of children per parent). The Z and XX are the positive number which varies depends on the system performance.

5.3 XML Files Layout

Following are six XML documents used in this benchmark.

1. Catalog: The catalog document is used for user online search. It stores all the information of the products such as name, price, available number etc.

Partial DTD: Please refer to 5.6.3 for detail description of each element

<!ELEMENT CATALOG (CD *, PLANT *, FOOD *)>

<!ELEMENT CD (NAME, ARTIST, COUNTRY, SUPPLIER, PRICE, DESCRIPTION ,YEAR, BARCODE,AMOUNT)>

<!ELEMENT PLANT (NAME, BOTANICAL, ZONE, LIGHT, PRICE, DESCRIPTION , COUNTRY, SUPPLIER, BARCODE,AMOUNT)>

<!ELEMENT FOOD (NAME, PRICE, DESCRIPTION , CALORIES , COUNTRY, SUPPLIER, BARCODE,AMOUNT)>

2. Customer: The customer document keeps customer information like customer account numbers, the shipping addresses and account balances etc. It is used for user login and payment transaction. Also the account number can be used to keep track of the goods in the shopping cart.

Partial DTD: Please refer to 5.6.3 for detail description of each element

<!ELEMENT CUSTOMER_LIST (CUSTOMER *)>

<!ELEMENT CUSTOMER (C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE, C_ACCTBAL, C_COMMENT)>

3. Cart: This file stores all the goods in every customer's shopping cart before the payment.

Partial DTD: Please refer to 5.6.3 for detail description of each element

<!ELEMENT CART (C_CUSTOMER *)>

<!ELEMENT C_CUSTOMER (C_CUSTKEY, C_GOOD*,)>

<!ELEMENT C_GOOD (C_BARCODE,C_PRICE, C_REQUEST_AMOUNT)>

4. Warehouses: There is a separate warehouse document for each warehouse. Those files are used to store all the warehouse stock information. They need to be checked and updated in each payment transaction. The system generates a list of under-stocked items from this file every hour. This file needs to be updated again when stock is replenished.

Partial DTD: Please refer to 5.6.3 for detail description of each element

<!ELEMENT WAREHOUSE_LIST (WAREHOUSE *)>

<!ELEMENT WAREHOUSE (W_WAREHOUSE_ID, W_WAREHOUSE_NAME, W_SUPPLIER*)>

<!ELEMENT W_SUPPLIER (W_SUPPLIER_ID, W_SUPPLIER_NAME , W_GOOD*)>

<!ELEMENT W_GOOD (W_CATEGORY, W_NAME , W_BARCODE, W_ARTIST*, W_PRICE, W_YEAR*, W_DESCRIPTION* , W_CALORIES*, W_COUNTRY* , W_BOTANICAL*, W_ZONE*, W_LIGHT* ,W_AMOUNT, W_THRESHOLD, W_NEXTORDER)>

5. Order: This file stores all the orders which are placed after the hourly stock-checking process. This file is supposed to be sent to product suppliers in the real world.

Partial DTD: Please refer to 5.6.3 for detail description of each element

<!ELEMENT ORDER _LIST (ORDER *)>

<!ELEMENT ORDER (ORDER_NUMBER, O_SUPPLIER*, O_WAREHOUSE*, O_GOOD*,O_DATE,O_ARRIVAL)>

<!ELEMENT O_SUPPLIER (O_SUPPLIER_ID, O_SUPPLIER_NAME)>

<!ELEMENT O_WAREHOUSE (O_WAREHOUSE_ID, O_WAREHOUSE_NAME) >

<!ELEMENT O_GOOD (O_CATEGORY, O_NAME , O_BARCODE, O_ARTIST*, O_PRICE, O_YEAR*, O_DESCRIPTION* , O_CALORIES*, O_COUNTRY* , O_BOTANICAL*, O_ZONE*, O_LIGHT* ,O_AMOUNT)>

6. Transaction history: This file keeps all the transaction history information for all successfully committed transactions. It can be used for online top-K/Ranked [24, 25] search such as "to search for the top10 best sellers in 2002".

Partial DTD: Please refer to 5.6.3 for detail description of each element

<!ELEMENT TRANSACTION_HISTORY (TRANSACTION *)>

<!ELEMENT TRANSACTION (T_CUSTKEY, T_DATE, T_GOOD*,T_TOTAL) >

<!ELEMENT T_GOOD (T_WAREHOUSE_ID, T_CATEGORY, T_NAME, T_BARCODE, T_PRICE, T_COUNTRY, T_SUPPLIER, T_AMOUNT, T_SUB_TOTAL) >

5.4 The Nested Transaction Definition Used In TXMark:

5.4.1 The components of the simulated application

There are 3 main transactions in this system. They are named T1, T2 and T3.

Shopping transaction (T1): Search online, add goods in a shopping cart and pay for those goods. It is a long duration nested transaction which includes sub-transactions T11 and T12 defined below.

Stock-Checking transaction (T2): Check the stock level every hour and generate a list under stocked goods.

Restocking Transaction (T3): Update goods information when ordered goods arrived.

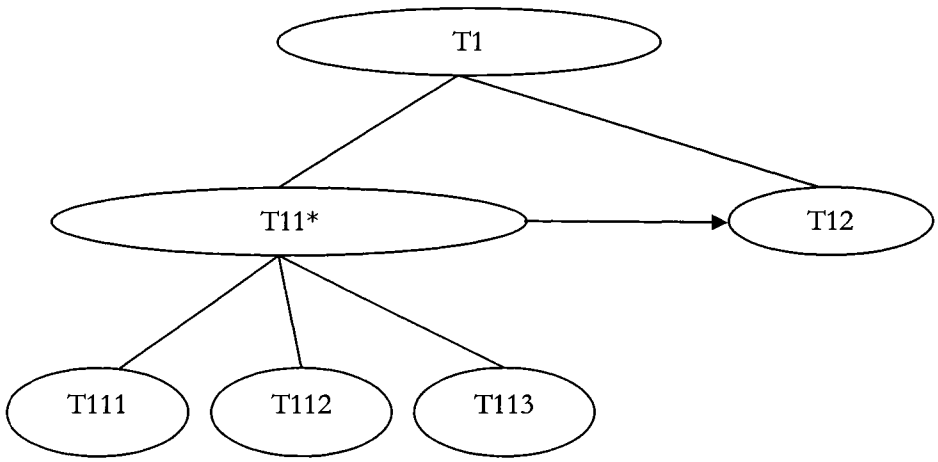


Figure 1: Shopping transaction diagram:

Searching & Carting Transaction (T11): Search online and add the searched goods into a shopping cart or remove goods from the shopping cart. It is also a nested transaction which includes sub-transactions T111, T112 and T113 defined below.

Paying Transaction (T12): Pay for the items in the cart.

Searching transaction (T111): Online search based on catalog or transaction history files.

Carting Transaction (T112): Add goods into a shopping cart or remove goods from the shopping cart.

Order-Checking Transaction (T113): Check if the goods in the cart are available and to get a list of unavailable goods.

Figure 5 is the shopping transaction diagram which can be interpreted as following:

T1 = {T11 → T12} means T1 is the parent transaction with two child-transactions T11 and T12. T11 and T12 have to be carried out in order from T11 to T12.

*T11 = {T111, T112, T113} means T11 has three child-transactions (T111, T112 and T113). All of those child-transactions can be carried out more than once without any order restrained.

5.4.2 The Detail description of transactions used in the benchmark:

T1: Shopping transaction ----- Search online, add goods in a shopping cart and pay for those goods. Involved XML documents are catalog, transaction history, customer, and warehouse and cart files. T1 is a long duration nested transaction which has two sub-transactions T11 (Searching & Carting Transaction) and T12 (Paying Transaction). These two sub-transactions have to be performed in order (T11→ T12). T1 commits only if T12 committed, otherwise T1 rollbacks.

T11: Searching & Carting Transaction -- Search online and add the searched goods into a shopping cart or delete goods from the shopping cart. It is also a nested transaction which includes sub-transactions T111 (Search transaction), T112 (Carting Transaction) and T113 (Order-Checking Transaction). T11 can commit at any time without restriction from its children transaction. These sub-transactions can be carried out more than once without any order required. Involved XML documents are catalog, customer, warehouse and cart files.

T12: Paying Transaction – Pay for every available item in the shopping cart. Involved XML documents: warehouse, transaction history, customer and cart files. The query workloads are light read on warehouse file, medium write on warehouse, transaction history, customer and cart files. The following are transaction detail descriptions:

1. The system updates the customer's account balance element in the customer file. It goes to step 2 if the money can be deducted successfully. Otherwise the transaction fails.
2. The system updates warehouses file for each available item in the cart before going to step 3.
3. The system adds all items purchased by the customer into the transaction history file with the same order as they were in the cart document.
4. The system clears the related information in the cart files.
5. The system needs to update the amount information in the catalog for unavailable items if there are any.

The transaction will be considered successful only if all the above operations done without any problems. Otherwise if it failed it leads to a T1 roll back.

T111: Searching transaction – Searching is the most frequent operation when customers shop online. The backbones of shopping transaction are searches which include catalog search and Top-K search [24, 25]. Following are the detailed descriptions of these two search operations:

The catalog search is based on catalog file. 90% of searches are catalog searches in our benchmark. Among catalog searches, 30% of them are specific search and 70% are the wild card searches. Usually Less than 10 goods return for each specific search. There will be a lot of search results returned for wild card searches which are more time consuming than specific searching. Figure 6 is the sample search screen snapshot.

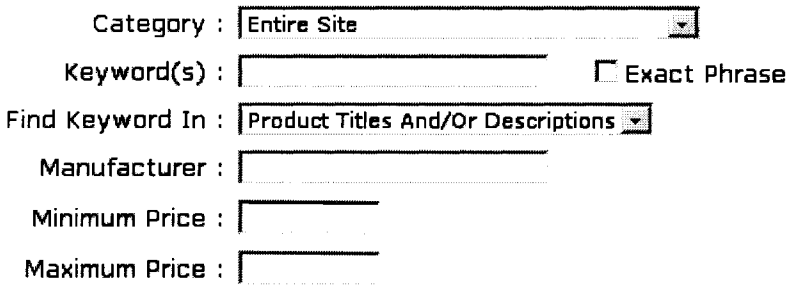


Figure 2: Catalog search criteria screen snapshot

The following are some catalog search samples.

- Search for the product description given the name of the product
- Search for the product description given the name of the product and price range
- Search for the product description given the name of the product and name of the manufacturer.
- Search for the product description given the name of the product and the name of category

Top-K/Ranked [24, 25] search is based on transaction history file. 10% of searches are Top-K/Ranked searches in our benchmark. Among Top-K/Ranked searches, 90% of them are looking for best sellers. 10% of them may look for best company item for certain goods. Figure 7 is the sample search screen definition.

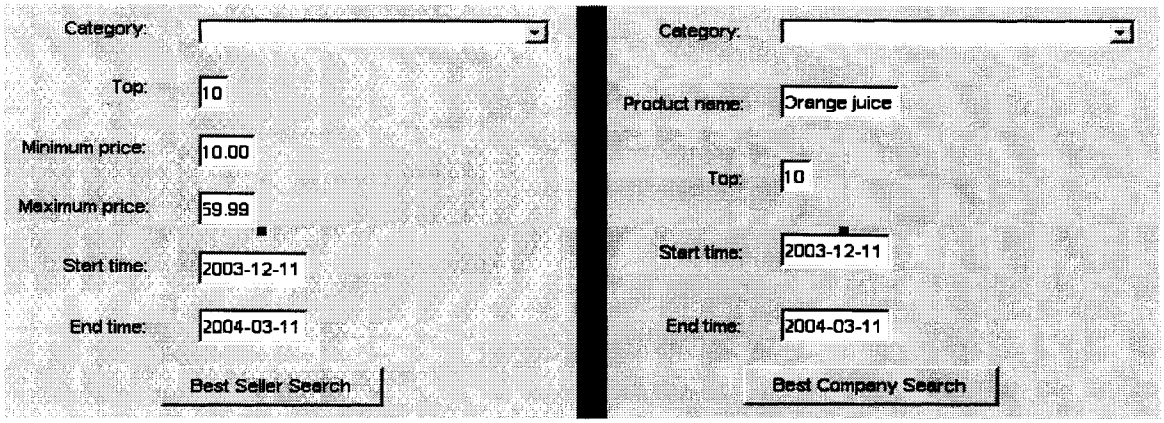


Figure 3: Data mining search criteria screen

The following are some data mining search samples.

- The top 10 sellers in the last year for each category.
- The top 5 sellers in the last month for MP3 players.
- The best seller whose price is between $5.00 and $20 in the last year for category CD.
- The top 10 company items (also sold in the same transaction) in the last year for the book <<Oracle DBA tips>>.

Most searches require quick responses. The timeout setting for a catalog search is 30 seconds and the setting for a data mining search is 60 seconds.

T112: Carting Transaction -- Add the searched items into a shopping cart or remove items from the shopping cart. The involved XML documents include catalogue and cart files. The query workload is light write on cart and catalogue file. The following is the transaction's detailed description:

1. Whenever an item is put into the shopping cart, the amount of this item in the catalog needs to be reduced.
2. Whenever an item is removed from the shopping cart, the amount of this item in the catalog needs to be increased.

3. Whenever a new item is added into the cart, the item is added as the last element with the following sub-element information: category, name, barcode, and request_amount.

4. Whenever a new item is removed from the cart, the responding element is removed from the cart document.

T113: Order-Checking Transaction ---- Search warehouses to make sure the goods in the cart are available. The involved XML documents include warehouse and cart files. The query workloads are heavy query upon warehouse and light write on cart file. Usually the system will do the order-checking transaction automatically before customer pays for it, because the items in the cart may not available. Customers can choose to do this transaction manually to see how many items in the cart are in stock. The following is the transaction's detailed description:

1. There should be a warehouse search order for each customer according to his/her delivery address because of the financial reason that the company always ships the goods to the customer from the nearest available warehouse. The Order-checking will be performed item by item.

2. The system updates corresponding sub-elements: available_amount and subtotal in cart file for each available item.

3. The whole transaction should be rolled back if any operations fail within this transaction. The total amount for all available goods should be updated in the cart file if the transaction is committed successfully.

T7: Stock-Checking transaction – Check the stock level every hour. The involved XML documents include warehouse and Order files. The query workloads are heavy read on warehouse file, mid-weight write on Order file. The following is the transaction's detailed description:

1. The system searches warehouse files for every item whose stock level is under the minimum threshold. If this item has been put into order file before and the supplier has not delivered it yet then ignore it. Otherwise put it into the order file.

2. Transaction should be rolled back if there is any problem encountered.

T8: Restocking transaction --Update warehouse, catalogue and order files when ordered goods arrive. The involved XML documents include warehouse, catalogue and Order files. The query workloads are light reads on order file and mid-weight writes on warehouse and catalogue. The following is the detailed description:

1. The system updates the amount element in warehouse documents for arrived goods.
2. The system updates the amount element in catalogue documents for arrived goods.
3. The system updates the arrival element in order document as "YES".
4. If any above updates failed, the transaction is rolled back.

5.5 System ACID Properties Requirements

5.5.1 Atomicity Requirements:

Atomicity property definition

SUT must promise that either all actions of transactions be executed completely, or no partially-completed actions leave any permanent effects in the database in case of some failure.

Atomicity Tests:

1. Carry out a shopping transaction successfully and verify that all the related elements in the catalog, order, warehouses and history files have been updated appropriately.
2. Perform a shopping transaction, go through the sub-transaction searching & carting transaction successfully and fail the paying transaction. Then verify that all the related elements in the catalog, order, warehouses and history files have NOT been changed at all.

5.5.2 Consistency Requirements:

Consistency property definition

Given the assumption that the database is consistent before an execution of transaction, the database must keep its consistency after the execution of the transaction whether or no it was committed or rolled back.

Consistency Conditions:

Consistency condition 1:

AMOUNT(barcode XXX, catalog) = SUM_AMOUNT(barcode XXX, warehouse)

Given the barcode of an item, the sum of the amount in every warehouse file must equal to the amount in the catalog file. AMOUNT(barcode XXX, catalog) stands for the amount of goods whose barcode is XXX in catalog file. SUM_AMOUNT(barcode XXX, warehouse) stands for the sum of the amount of goods whose barcode is XXX in the warehouse file.

Consistency condition 2:

AMOUNT(barcode XXX, warehouseID YYY ,warehouse) + SUM_AMOUNT(barcode XXX, warehouseID YYY ,transaction history) = SUM_AMOUNT(barcode XXX, warehouseID YYY ,order, delivered)

Given the barcode of an item and the warehouse id, the sum of the amount in the warehouse and the sum of the amount in the transaction history file must equal to the sum of the delivered amount in the order file. AMOUNT(barcode XXX, warehouseID YYY ,warehouse) stands for the amount of goods whose barcode is XXX in warehouse YYY. SUM_AMOUNT(barcode XXX, warehouseID YYY ,transaction history) stands for the sum of the amount of goods of warehouse YYY whose barcode is XXX in transaction history file. SUM_AMOUNT(barcode XXX, warehouseID YYY, order, delivered) stands for the sum of the delivered amount of goods of warehouse YYY whose barcode is XXX in order file.

Consistency condition 3:

BALANCE(customerKEY XXX, customer,initial) - BALANCE(customerKEY XXX, customer) = SUM_TOTAL(customerKEY XXX, transaction history).

Given the customer key of a customer, the initial balance minus the present balance in the customer file must equal to the sum of the total payment in the transaction history file. BALANCE(customerKEY XXX, customer) stands for the present balance of a customer whose customer key is XXX in customer file. BALANCE(customerKEY XXX, customer,initial) stands for the initial balance of customer whose customer key is XXX in customer file. SUM_TOTAL(customerKEY XXX, transaction history) stands for the sum of the total payment of the customer whose customer key is XXX in transaction history file.

Consistency Tests

Verify the database is consistent before testing three conditions above. Then,

1. Submit 10 transactions that all purchase at least one of the same items to the SUT simultaneously. Make sure some of them commit and some rollback. The test should last long enough to simulate at least a 2 hour transaction in real world.
2. Stop submitting transactions to the SUT and retest the three conditions above to verify the consistency.

5.5.3 Isolation Property Definition

We use almost the same definition which is used in TPC-C [3] like the following. Isolation can be defined in terms of phenomena that can occur during the execution of concurrent database transactions. The following phenomena are considered, given two atomic database transactions, T1 and T11:

- P0 ("Dirty Write"): Database transaction T1 reads a data element and modifies it. Database transaction T11 then modifies or deletes that data element, and performs a COMMIT. If T1 were to attempt to re-read the data element, it may receive the modified value from T11 or discover that the data element has been deleted.
- P1 ("Dirty Read"): Database transaction T1 modifies a data element. Database transaction T11 then reads that data element before T1 performs a COMMIT. If T1

were to perform a ROLLBACK, T11 will have read a value that was never committed and that may thus be considered to have never existed.

- P2 ("Non-repeatable Read"): Database transaction T1 reads a data element. Database transaction T11 then modifies or deletes that data element, and performs a COMMIT. If T1 were to attempt to re-read the data element, it may receive the modified value or discover that the data element has been deleted.

- P3 ("Phantom"): Database transaction T1 reads a set of values N that satisfy some <search condition>. Database transaction T11 then executes statements that generate one or more data elements that satisfy the <search condition> used by database transaction T1. If database transaction T1 were to repeat the initial read with the same <search condition>, it obtains a different set of values.

The following table defines four isolation levels with respect to the phenomena P0, P1, P2, and P3.

Isolation Level	P0	P1	P2	P3
0	Not Possible	Possible	Possible	Possible
1	Not Possible	Not Possible	Possible	Possible
2	Not Possible	Not Possible	Not Possible	Possible
3	Not Possible	Not Possible	Not Possible	Not Possible

The database transactions are represented as following:

- Tr = Any read-only database transaction used to implement SUT
- Tu = Any update database transaction used to implement SUT
- Tn = Any arbitrary transaction (Although arbitrary, this transaction may not do dirty writes)

Unless otherwise specified, the system being tested will ensure that the isolation requirements defined in the table below are met by all database transactions.

Req. #	For transactions in this set:	These phenomena:	must NOT be seen by this transaction:	Description:
1	{Tu, Tu}	P0, P1, P2, P3	Tu	Level 3 isolation between any two flat update transactions.
2	{Tu, Tn}	P0, P1, P2	Tu	Level 2 isolation for any update transactions relative to any arbitrary transaction.
3	{Tr, Tn}	P0, P1	Tn	Level 1 isolation for any read-only transaction relative to any arbitrary transactions.

Isolation Tests

The isolation tests require that transaction implementation needs to be modified so that an operation to the database may be halted while in progress, while not affect the result of the transaction.

Isolation test 1:

To verify the isolation between two update transactions, carry out following steps.

1. Perform paying transaction A which pays for item X and Y in warehouse 1. Interrupt transaction A after it pays for X but before it pays for Y.
2. Perform paying transaction B which pays for item W and X in warehouse 1.

3. Verify that transaction B has to wait for transaction A commit.
4. Resume transaction A that was interrupted in step 1.
5. Verify that the amount element of X in warehouse 1 has been updated properly for transaction A.
6. Verify transaction B resumes after transaction A is committed.
7. Verify that the amount element of X in warehouse 1 has been updated properly for transaction B.

Isolation test 2:

To verify the isolation between an update transaction and an arbitrary transaction, carry out following steps.

1. Perform paying transaction A which pays for item X and Y in warehouse 1. Interrupt transaction A after it pays for X but before it pays for Y.
2. Carry out an update for the amount of the X in warehouse 1 using any database utilities. Commit this update as soon as possible.
3. Resume transaction A interrupted in step 1.
4. Verify that the amount element of X in warehouse 1 has been updated properly for transaction A.
5. Verify that the amount element of X in warehouse 1 has been updated properly in step 2.

Isolation test 3:

To verify the isolation between a read only transaction and an arbitrary transaction, carry out following steps.

1. Let the amount of the X in every warehouse be more than 10. Perform a Carting transaction which adds item X into shopping cart with request amount 1.

2. Perform order-checking transaction A which checks if the item X in shopping cart is still available.
3. Verify that the order-checking transaction returned the available result.
4. Carry out an update operation which set the amount of the X in every warehouse as 0 using any database utilities. But do not commit this update.
5. Rerun step 2.
6. Verify that the order-checking transaction returned the available result.
7. Commit the update operation in step 4.
8. Rerun step 2.
9. Verify that the order-checking transaction returned the unavailable result.

5.5.4 Durability Requirements

System failure may happen no matter what precautious measures are taken. To provide durability, SUT must guarantee that committed transactions must be preserved after the recovery from system failure; also the database consistency must be maintained. Durable Medium can be any non-volatile storage medium such as hard disk, magnetic tape and optical disk. It can also be a volatile storage medium which can transfer the data to a non-volatile medium automatically without any data loss before the system failure. With the definition of durable medium, we define committed transaction as "a transaction which all the updated data has been stored in durable medium". That means a good system should have a way to recovery the committed transaction from durable medium whenever a system failure may happen. The following failure cases can be used as durability test scenarios.

Case 1: Single durable medium malfunction leads to irrecoverable data lost in that medium.

Case 2: System malfunction leads to non-resumable interruption of all executing transactions.

Case 3: Memory or memory related hardware malfunction leads to data lost in memory.

Case 4: Power malfunction leads to invalidity of SUT.

Durability test procedures:

SUT must implement another logging mechanism to record all the executed transactions into a file named "log" whether or not they are committed or rolled back. This log will be used to verify the durability test result. Make sure the system satisfies the consistency requirement before performing a durability test. Then

1. Start submitting shopping transactions.
2. Raise the failure case.
3. Recover the SUT.
4. Compare the "log" file and the "transaction history" file. For every committed transaction in "log" file, there must be a corresponding entry in "transaction history" file. There must be no entry in "transaction history" file for any rolled backed transaction in "log" file.
5. Verify the consistency requirement test.

5.6 Benchmark SUT Scaling and Database Population

The throughput of the SUT is related to the simultaneous access to the database. If there are more simulated customers in the SUT, there will be more accesses to the database at the same time. Also more and more storage space will be consumed while the benchmark is carried on. So the definition of scaling and database population is necessary.

5.6.1 Scaling Rule

The more customers in the system, the heavier load applied to the system because 1 % of the customers will always doing shopping transactions in our benchmark. Instead of using the customer number as base unit, the system uses the warehouse number. We define that if there is one more warehouse; there will be 3K more customers in the system and there will be 30 more concurrent database accesses to the system. To simulate the real world environment, we demand that 90% of the shopping transactions

must be finished within 60 minutes. The following table is the cardinality of initial population per warehouse.

File name	Cardinality (in elements)	Typical element size (in bytes)	Typical file size (in 1,000 bytes)
Warehouse	1	125,000K	125,000
Customer	3K	350	1050
History *	X	2500	2.5*X
Catalog **	1	100,000K	100,000
Order ***	Z	350	0.35*Z
Cart ****	30	3500	105

* The number depends on the system performance.

** Catalog file will not change because goods are the same.

*** The number depends on the system performance.

**** 1% of customers are shopping online.

5.6.2 60 day space computation

The database will grow during the test period which may extend to 60 days. It is the vendors' responsibility to calculate the disk consuming base on scaling rules because the different implementation of XML support.

5.6.3 Database Population

The following are some terms used in our database population:

- "random string[x,y]" stands for a string with minimum x and maximum y random alphanumeric characters. The mean length of this string is (x+y)/2.
- "random number[x,y]" stands for a string with minimum x and maximum y random numeric characters. The mean length of this number is (x+y)/2.

- unique within[x] stands for any one number randomly selected between 0 to x, which is unique amongst those that have been generated.
- random within[x,y] stands for any one number randomly and independently selected between x and y, which is uniformly distributed with a mean of (x+y)/2.
- [CD/PLANT/FOOD] stands for one of those which are separated by "/".

XML document population requirements

The initial XML document population must be following:

1. Catalog file:

100,000 CD elements in the Catalogue documents with following format.

<NAME> **random string[20,30]** </ NAME >

<ARTIST> **random string[20,30]**</ARTIST>

<COUNTRY> **random string[10,30]**</COUNTRY>

<SUPPLIER> **random string[20,50]**</ SUPPLIER >

<PRICE>**random within[1.00,100.00]**</PRICE>

<DESCRIPTION>**random string[10,200]**</ DESCRIPTION >

<YEAR>**random within[1950,2004]**</YEAR>

<BARCODE>**unique within[9999999999]**</BARCODE>

<AMOUNT>**3000**</ AMOUNT >

100,000 PLANT elements in the Catalogue documents with following format:

< NAME > **random string[20,30]**</ NAME >

<BOTANICAL> **random string[20,30]**</BOTANICAL>

<ZONE> **random within[0,10]**</ZONE>

<LIGHT> **random string[20,30]**</LIGHT>

<PRICE> **random within[1.00,100.00]**</PRICE>

<DESCRIPTION> **random string[10,200]**</ DESCRIPTION >

<COUNTRY> **random string[10,30]**</COUNTRY>

<SUPPLIER> **random string[20,50]**</ SUPPLIER >

<BARCODE> **unique within[9999999999]**</BARCODE>

<AMOUNT>**3000**</ AMOUNT >

100,000 FOOD elements in the Catalogue documents with following format:

<NAME> **random string[20,30]**</ NAME >

< PRICE > **random within[1.00,100.00]**</ PRICE >

<DESCRIPTION> **random string[10,200]**</ DESCRIPTION >

<CALORIES> **random within[10,10000]**</ CALORIES >

<COUNTRY> **random string[10,30]**</COUNTRY>

<SUPPLIER> **random string[20,50]**</ SUPPLIER >

<BARCODE> **unique within[9999999999]**</BARCODE>

<AMOUNT>**3000**</ AMOUNT >

2. Warehouse file

There are 10 warehouse elements in a warehouse document. For every item listed in Catalogue documents there must be a corresponding good element for each warehouse element in WAREHOUSE document like following.

- <WAREHOUSE>

<W_WAREHOUSE_ID> **unique within[999]**

</W_WAREHOUSE_ID>

<W_WAREHOUSE_NAME> **random string[20,30]**

</W_WAREHOUSE_NAME>

<W_SUPPLIER>

<W_SUPPLIER_NAME> **random string[20,50]** </W_SUPPLIER_NAME>

<W_GOOD>

<W_CATEGORY>**CD** </W_CATEGORY>

<W_NAME> **random string[20,30]** </W_NAME>

<W_BARCODE> **unique within[9999999999]** </W_BARCODE>

<W_ARTIST> **random string[20,30]**</W_ARTIST>

<W_PRICE> **random within[1.00,100.00]** </W_PRICE>

<W_YEAR> **random within[1950,2004]**</W_YEAR>

<W_AMOUNT>**100**</W_AMOUNT>

<W_THRESHOLD>**10**</W_THRESHOLD>

<W_NEXTORDER>**90**</W_NEXTORDER>

</W_GOOD>

</W_SUPPLIER>

</ WAREHOUSE >

3. Customer file

30,000 customer elements in the Customer documents with.

<C_CUSTKEY> **unique within[3000000]** </C_CUSTKEY>

<C_NAME> **random string[10,30]**</C_NAME>

<C_ADDRESS> **random string[20,80]**</C_ADDRESS>

<C_NATIONKEY> **random string[10,30]**</C_NATIONKEY>

<C_PHONE> **unique within [1111111111,9999999999]** </C_PHONE>

<C_ACCTBAL> **random within[1000.00,10000.00]** </C_ACCTBAL>

<C_COMMENT> **random string[10,200]**</C_COMMENT>

4. The order, transaction history and carting documents are empty before the execution of our benchmark.

5.7 Workload Definition And Performance Metrics:

The workload used in our SUT includes three parts: queries, updates, and bulk loads. The update workload is mainly in the paying transaction which updates the customer, catalogue, warehouses and carting files. When the ordered products are restocked, the system need to bulk load the information into the warehouse and catalogue files. The queries workload is the focus in our SUT workload definition, because most of the workload in real world online transaction is still information search. Above, we divide the queries into catalogue search and data mining searching based on the files the query executed on. Now we present the queries by functionality.

5.7.1 Queries used in SUT by Functionality

5.7.1.1 Exact Match

This type of queries tests the capability of the engines in dealing with string matching. The sample queries are:

- Find the names of the books published by (University of Alberta).
- Find the names of food produced in (Canada)

5.7.1.2 Function Query

Just like in the application using traditional RDBMS, aggregate functions such as count, average, max, min and sum are frequently used in our XML files based SUT. The sample queries are:

- Group the CDs by singer and return the total selling amount in each group
- Find the selling amount in last week for the given CD.

5.7.1.3 Sorting

The sorted search result is commonly desired by the online users. This type of query tests not only the string sorting ability but also other casted type sorting strength. The sample queries are:

- List the CDs of (Michael Jackson) sorted by CD name
- List the CDs of (Michael Jackson) sorted by published date
- List the CDs of (Michael Jackson) sorted by selling amount

5.7.1.4 Regular Path Expressions

A good XML database should at least support one of these XML query languages like XPath, XQuery and XSLT. Those languages all support the regular path expression query. The sample queries are:

- List the name of all the products in the warehouse
- List the name of all the products purchased last week

5.7.1.5 Ordered Access

The ordered feature is very important in XML document. So we use this type of query to test the system's ability to preserve the order of elements. The sample queries are:

- Find the second transaction since 2004-01-01 for customer Jun Chen
- Find the 100001st customer who shopped since 2004-01-01

5.7.1.6 Document Construction

Even though the XML documents are data-centric in our SUT. We still can test the document construction ability of the database. The sample queries are:

- Construct a brief information document? ?? on warehouse one, including name and stock amount for each product in this warehouse
- Retrieve all the products' information in warehouse one.

5.7.1.7 Retrieve Individual Documents

This is the basic query that every database should support. The sample queries are:

- List the customer information for the customer Jun Chen
- List the products' information in the customer Jun Chen's shopping cart.

5.7.1.8 Text Search

This is also a very commonly used the search. The sample queries are:

- List the name of customers whose address contains text "South Gate"
- List the name of products whose description contains text "allergy".

5.7.1.9 Irregular Data

Missing some elements and having some empty elements is very common in XML documents. The sample queries are:

- List the name of the plants that do not have BOTANICAL element
- List the name of the plants for which the BOTANICAL element is empty.

5.7.1.10 References and Joins

Like in RDBMS, join operation is also very powerful to retrieve information in XML database. The sample queries are:

- List the name of all the customers who bought the same products in the same day as the customer Jun Chen did in 2004-01-01.

5.7.2 Workload distribution and restriction :

1. 90% of the shopping transactions need to be done within 60 minutes
2. 5% of all shopping transactions will be rolled back.
3. There are 22 searching sub-transactions, 12 carting sub-transactions, 2 Order-Checking sub-transactions and one paying sub-transaction in each successful shopping transaction on average.
4. One of the carting transactions is removing one item from the shopping cart, the rest of the carting transactions add items into the shopping cart and they always follow one of the searching transactions.

5. Paying transaction is always the last transaction. The average waiting time between any sub-transactions is 10 seconds.

Workload Definition for stock-checking transaction:

The bulk load operation in this transaction must finish within one hour because it runs every hour.

Workload Definition for restocking transaction:

The system makes the 24 hour-old order available automatically.

Workload Definition for searching transactions:

The timeout setting for catalog search is 30 seconds and the setting for data mining search is 60 seconds. The percentage of different search will be:

Exact Match 40%

Function Query 20%

Sorting 5%

Regular Path Expressions 5%

Ordered Access 5%

Document Construction 5%

Retrieve Individual Documents 5%

Text Search 5%

Irregular Data 5%

References and Joins 5%

5.7.3 Performance Metrics:

1% of the customers in SUT will submit the shopping transaction requests constantly. SUT will keep running for 60 days. The performance will be measured using stpm which is "The number of successful shopping transactions per minute".

Measurement requirement:

SUT is required to reach a steady state after running 24 hours. Measurement began from the beginning of the 25th hour. Duration of the measurement is 24 hours. All the

transactions that finished successfully within the second 24 hours are legitimate transactions to be used in performance calculation.

Chapter 6

6 BENCHMARK EXPERIMENTS

This chapter describes the experiment conducted in the thesis research. The first purpose of this experiment is to find out if the XML systems will perform differently when it works on single user and multi-user environments. The second goal is to see if the incremental operation indeed solves cascading abort effect in the long durational transaction model. We also wish to demo, through these experiments, that our proposed benchmark is also be able to capture the performance difference of various systems that other benchmarks may fail to do so.

6.1 Experiment System Module Structure

There are 4 modules (figure 6) in our experiment system.

1. XML files generator.

This module will generate 6 xml files to be tested. The size of the file varies from the input parameter –number of warehouse.

2. Database populator.

This module will read the xml files generated from module 1, and then populated it into the tested database.

3. Transaction generator and submitter.

This module will create a lot of threads according to the input scale factor parameter – concurrent online users (defined in 6.3.1). These threads randomly generate the simulated transactions at the same time simulating the multi-users environment.

4. Transaction Manager.

This module will create a strong dataguide [23] by analyzed the tested database. Dataguide is a graphic tree to represent the XML structure. It is used to implement the locking and recovery mechanism in our experiment. Then it will execute the transactions from the queue which is populated by module 3.

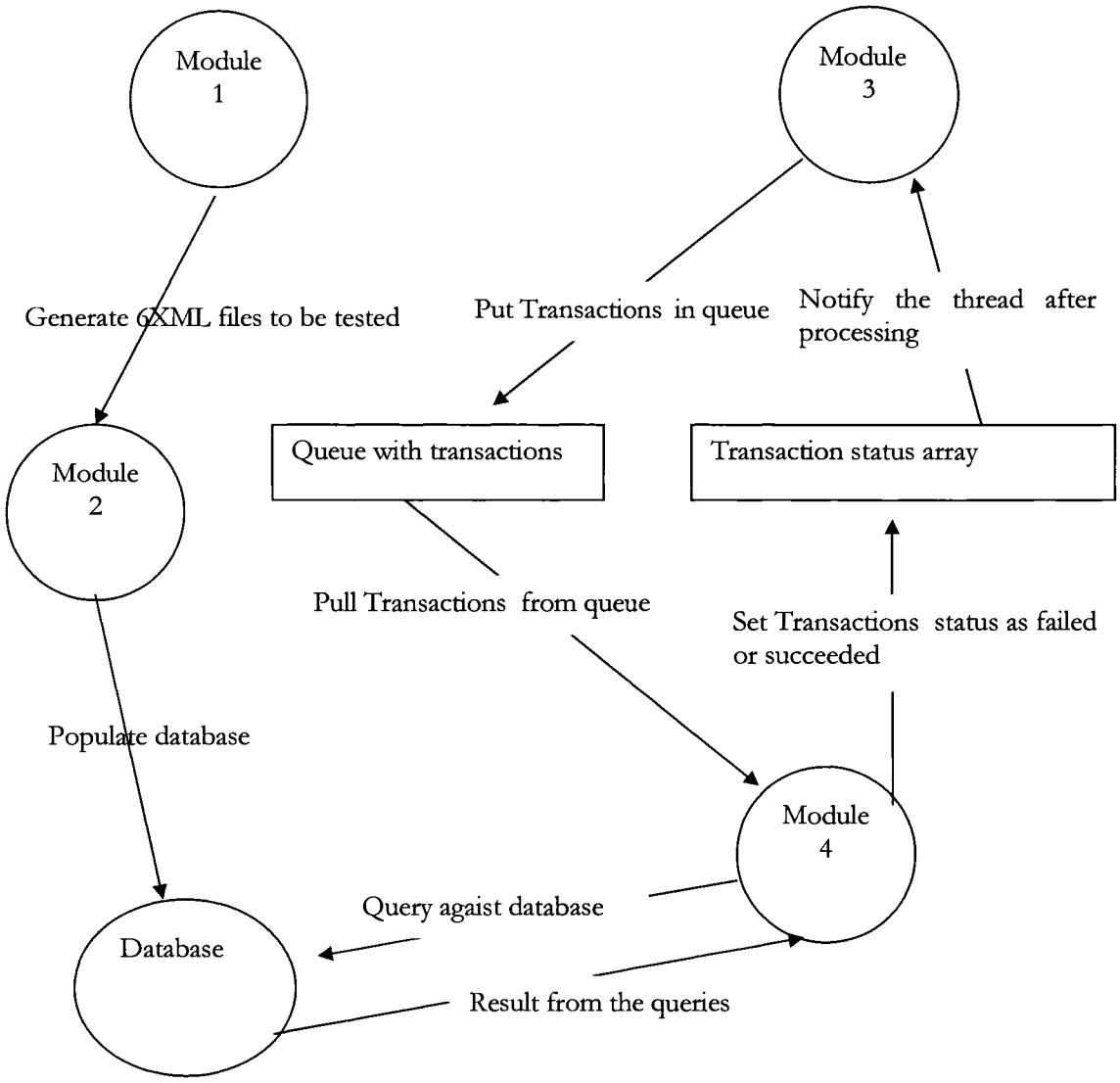


Figure 6: Transaction model

6.2 Experiment Setup:

The TXMark benchmark tests are conducted on two different XML application servers: one is the SQL Server 2000, and the other is Berkeley DB's DBXML 2.1.8, both are well-known XML system. For SQL Server 2000, to support XML, we have to install

SQLXML. SQLXML is free downloaded program from www.microsoft.com, which enables XML support for SQL Server 2000. After populating the XML file and examining the table structure, we found out that SQL 2000 will shred the XML file and create corresponding table to store the shredded information. For DBXML, they also chopped up the file into a lot of small binary files. The noticed difference between them is that we can clearly pin point which table has been created to store XML data in SQL 2000, while we can only locate the file system directory created by DBXML without knowing the detail meaning of the chopped up files inside it.

All the experiments are conducted under the specifications of the TXMark, that is,

a) All testing data are randomly generated according to 5.6.3.

b) All the work loads are simulated according to 5.7

c) Even though we defined the performance metric as "The number of successful shopping transactions per minute" in TXmark, the experiment result will be shown as how many transactions succeeded within 45 minutes. The reason for that is that the server's performance is very slow. There are not too many transactions finished within one minute, and it is hard for us to show a good figure using small performance output.

All the experiments are run on a Xeon dual CPUs Server. This server's CPU frequency is 733M HZ. It has 2G RAM. The operating system is Windows 2000 professional.

6.3 Experiment Result and Analysis:

6.3.1 Terms used:

Percentage of Incremental Query --- Update Query consists of incremental and writes queries. Percentage of Incremental query means the percentage of incremental query out of all the update queries.

Transaction abort rate – The percentage of user voluntarily aborted transactions out of the total transactions.

XML files size: -- The size of the catalog file. Because most of the operations are placed on this file, we use its size as a measure factor.

Concurrent users: -- Threads that carry out the experimental queries.

6.3.2 Databases tested and experiment result presentation:

There are two databases used in this experiment. First one is SQL Server 2000 from Microsoft company, the second one is DBXML from the sleepy cat company. Even though we defined the performance metric as "The number of successful shopping transactions per minute" in our benchmark, the experiment result will be shown as how many transactions succeeded within 45 minutes.

6.3.3 Experiment description and analysis

Test one – scalability drive test varied by concurrent users

The first test is to find out how well the system behaves in the multi-user environment. So given this condition that all the sub-transactions must be finished within one minute, we increase the number of concurrent shopping users while the XML files size, percentage of Incremental Query and transaction abort rate stay still.

For SQL Server, the result shows that our system gets the maximum transaction throughput while the number of concurrent users is around 20. After that even the concurrent users increase, the throughput doesn't go up anymore. On the contrary, it declines after the user count reaches more than 25. The reason may be that the locking probability increases while the user increases. For DBXML, it gets the best performance when there is only one thread. This phenomenon shows that XMLDB is not very good at handling multi-thread request. It shows us the need for a benchmark that can work on multi-user mode.

SQL Server: XML files size: 730M; Percentage of Incremental Query: 90%. transaction abort rate: 5%

Figure 4: SQL Server Test One Result

DBXML: XML file size: 730M; Percentage of Incremental Query: 90%. transaction abort rate: 5%

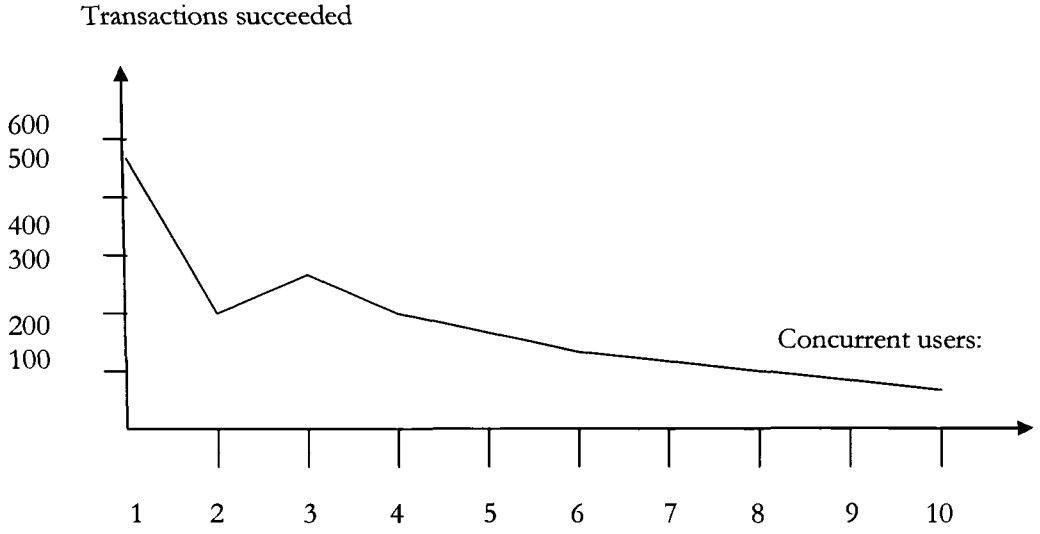


Figure 5: DBXML Test One Result

Test two – Incremental operation drive test

The second point we need to prove in our experiment one is that incremental operation concept is very good to handle the long durational transactions situation. In this experiment we change the percentage of incremental queries. The incremental queries percentage increases from 0% to 100% while the transaction abort rate, size of xml files and concurrent user number all stay still. They are 5% transaction abort rate, 730M XML files and 100 concurrent users for SQL Server and 10 concurrent users for DBXML. We can see that we can get a higher transaction success rate with higher percentage of incremental queries. For online shopping transaction, where most updates can be represented as incremental operations, our database model is very good at solving the locking problem in a traditional database model.

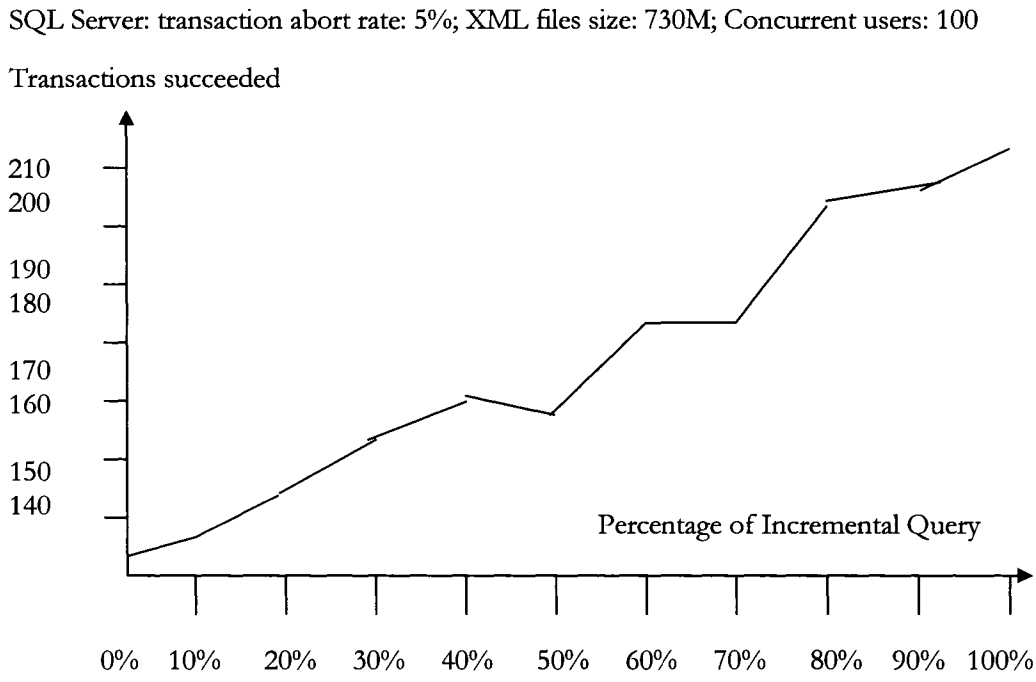


Figure 6: SQL Server Test Two Result

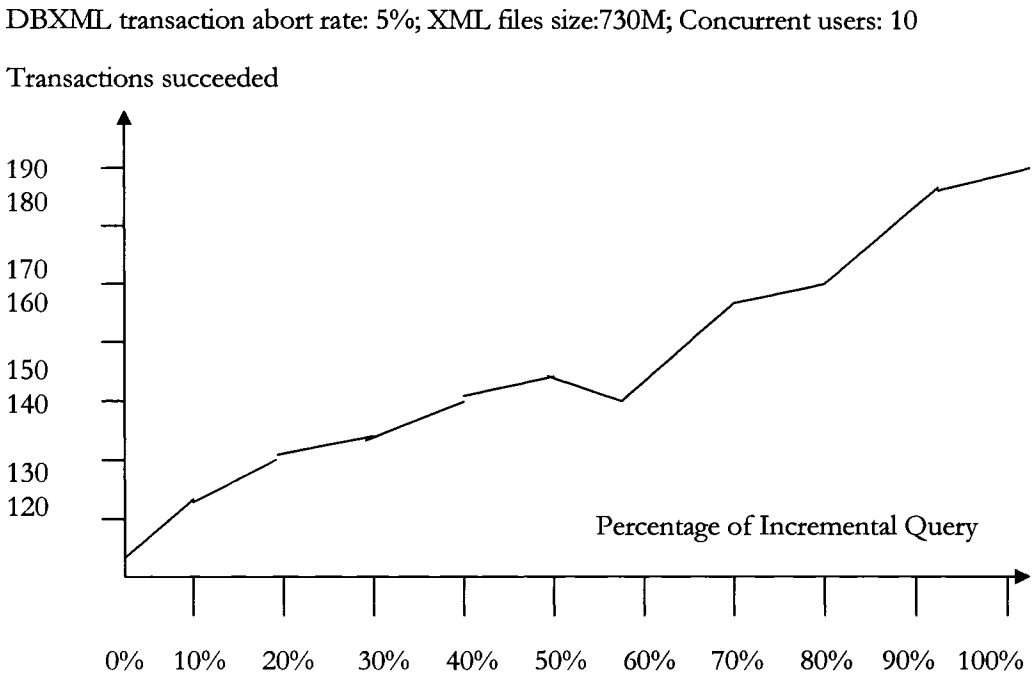


Figure 7: DBXML Test Two Result

Test three – abort rate drive test

As we already know, the most concerning issue about long durational transaction is recursive rollback effect. The locking system in our model will guarantee there will be no recursive rollback effect. The downside of our model will be blocking potential successful transactions. The following figure is the benchmark result we performed to test the effect of transaction abort rate to the transaction success rate. The transaction abort rate will vary from 2% to 20%. The size of the XML files are 730M and concurrent user number are 100 for SQL Server and 10 for DBXML. The percentage of incremental operation out of the entire update operation is 90%.

XML files size: 730M; Concurrent users: 100; Percentage of Incremental Query: 90%

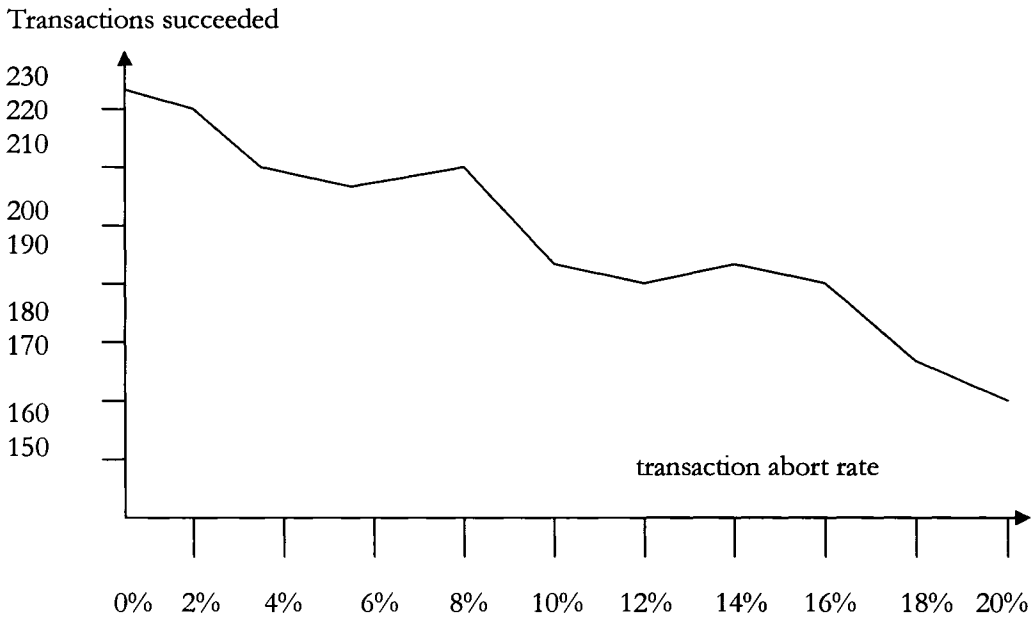


Figure 8: SQL Server Test three Result

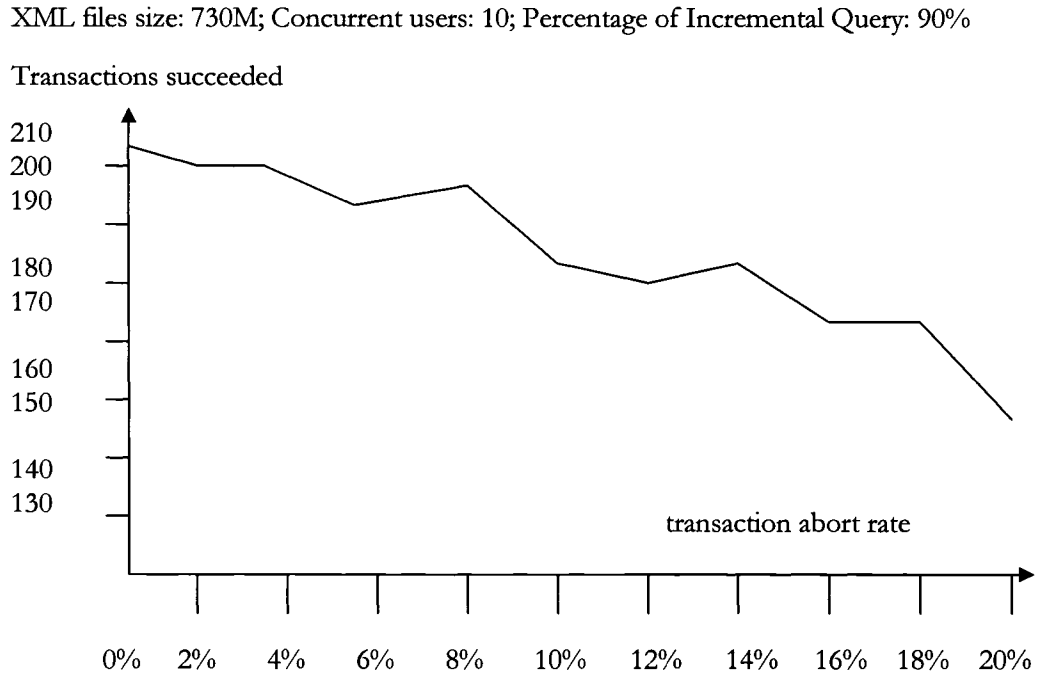


Figure 9: DBXML Test three Result

We can see that with the transaction abort rate climbing, the success rate of our transaction does not decline too much. For the SQL Server, there are 225 successful Transactions when the transaction abort rate is 0% and 170 successful when the transaction abort rate up to 20%. The successful transaction rate declined 24.4% after the transaction abort rate increased 20%. For DBXML, the successful transaction dropped 28.2%. This proves our model is very good at dealing with the recursive rollback situation.

Test four – scalability drive test varied by database size

At last, we experiment to see how the size of xml file will affect the benchmark performance in our system. The following is the benchmark result we got when we varied the xml file size from 75M to 730M with 5% transaction abort rate, 90% incremental operations and 100 concurrent users.

SQL Server: Percentage of Incremental Query: 90%. transaction abort rate: 5%; Concurrent users: 100

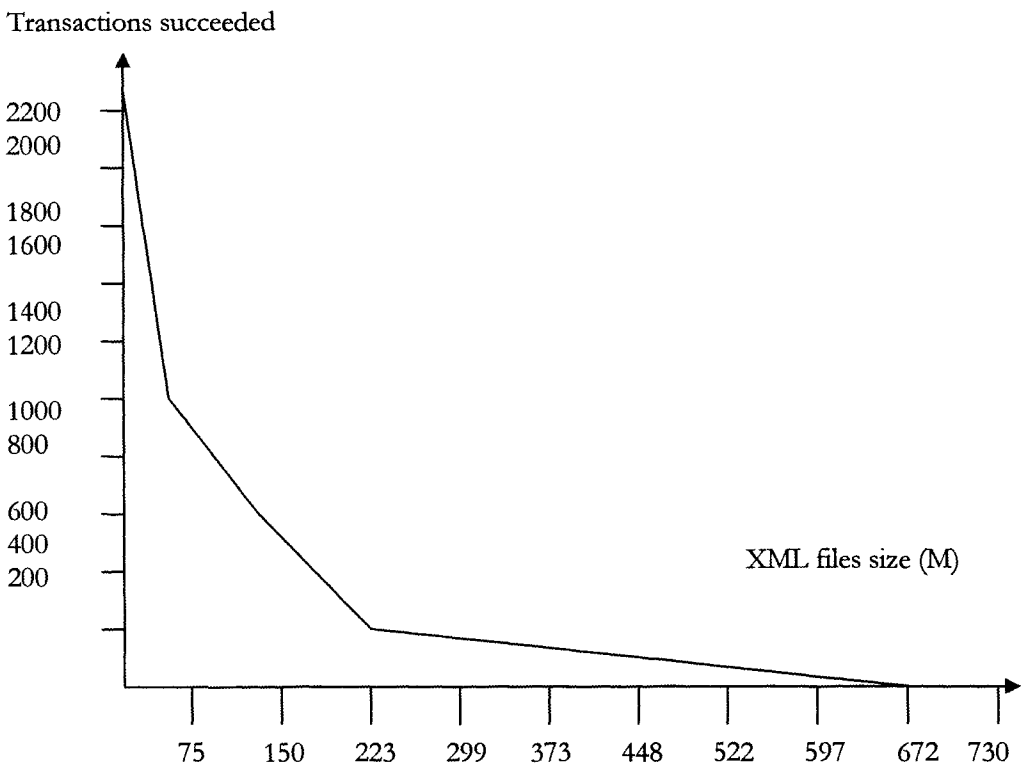


Figure 10: SQL Server Test four Result

Percentage of Incremental Query: 90%. transaction abort rate: 5%; Concurrent users: 10

Transactions succeeded

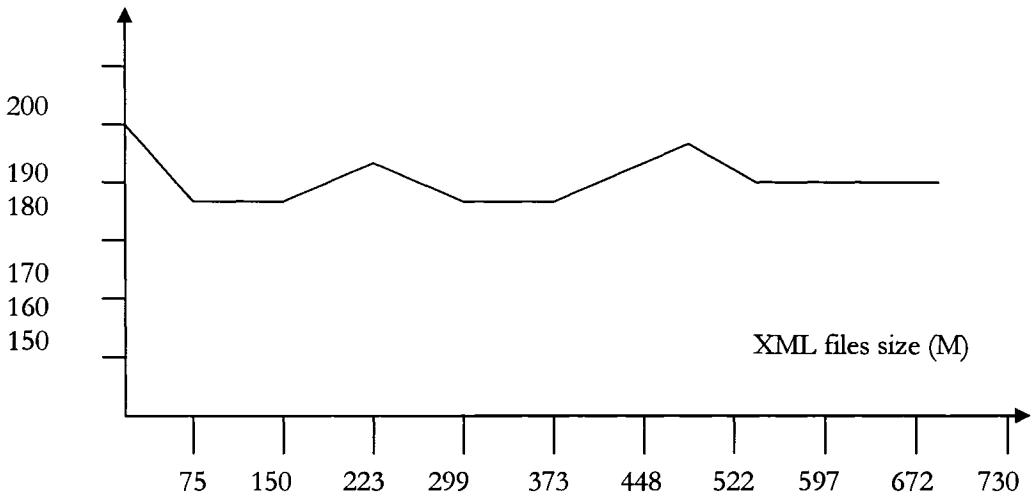


Figure 11: DBXML Test four Result

For SQL Server, we can see the transaction rate dropped greatly with the growth of the file sizes. When we increase the database size up to 730M, the throughput is merely over 200. But for DBXML, it doesn't vary too much with the changing of the XML files. The reason for that is because XML enabled databases like SQL Server, they use relational database mechanism to handle the search like using join to search and assemble the result. The join operation requires a lot of memory and the bigger the size of the file becomes, the more memory it consumes. Native XMLDB uses other index method, which consumes less memory. I observed the memory usage while I did the test; the SQL Server application consumed more than 2G memories while the XMLDB only took 200M memory.

6.4 Conclusion:

Our experiment result shows that to evaluate a database's performance, it has to be tested under a multi-user environment. The experiment also shows that the database model implemented is very good at dealing with long durational transactions. It can endure the recursive roll back effect much better than the traditional database model. We can see that these two systems behave differently in all tests. Based on that fact that all the existing XML benchmarks only measure system's query performance and all but Xmach1 [6] run on single user environment. I am very confident that our benchmark defined here will be a good one to measure an XML database's performance in a real application.

Chapter 7

FUTURE WORK

The work presented here should be seen as a first step towards a well designed XML database benchmark. We do not have very much reality information about how the XML database works in a long durational transaction application. However we would not be able to get such feedback without presenting our idea to the world. So one future aim will be to try and spread our benchmark, receive feedback, and improve it.

The benchmark experiment is a time consuming process. We can only test two databases here. It is not good enough to justify our proposal without testing more XML databases. Another possible feature work is to test more updated XML databases. To make it more persuasive, we need to work with a database vendor. They are the ones who design their databases; they have much more skills needed to tune the performance of their database. For example, without the help of the sleepycat's tech support, I would not be able to finish the test of XMLDB.

Chapter 8

REFERENCES

[1] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation, 16 August 2006.

[2] XML Query Use Cases W3C Working Draft 8 June 2006. http://www.w3.org/TR/xquery-use-cases/

[3] TPC BENCHMARK™ Transaction Processing Performance Council (TPC) http://www.tpc.org/information/benchmarks.asp.

[4] J. Shanmugasundaram et al. A General Technique for Querying XML Documents using a Relational Database System. SIGMOD Record, September 2001

[5] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu, Ralph Busse. XMark: A Benchmark for XML Data Management Proceedings of the 28th VLDB Conference, 2002

[6] Timo Böhme, Erhard Rahm. XMach-1: A Benchmark for XML Data Management. In Proceedings of BTW2001, 2001.

[7] Stéphane Bressan, Mong Li Lee, Ying Guang Li, Zoé Lacroix and Ullas Nambiar: The XOO7 Benchmark, in proceedings of the first VLDB Workshop on Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT), Hong Kong, China, August 2002, published by Springer-Verlag in the Lecture Notes in Computer Science series (ISBN 3-540-00736-9), pp146-147.

[8] Kanda Runapongsa, Jignesh M. Patel, H. V. Jagadish, Yun Chen, Shurug Al-Khalifa. The Michigan Benchmark: Towards XML Query Performance Diagnostics, Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[9] B. B. Yao, M. T. Özsu, and J. Keenleyside, "XBench -- A Family of Benchmarks for XML DBMSs", In Proceedings of EEXTT 2002 and DiWeb 2002, Lecture Notes in

Computer Science Volume 2590, S. Bressan, A. B. Chaudhri, M. L. Lee, J. Yu, Z. Lacroix (eds), Springer-Verlag, pages 162-164

[10]David Brownell. Official website of Simple API for XML (SAX) http://www.saxproject.org/

[11] W3C Activity Statement on the Document Object Model (DOM) http://www.w3.org/DOM/

[12] D. Florescu, D. Kossmann, Storing and Querying XML Data using an RDBMS. IEEE Data Engineering Bulletin 22(3), 1999

[13] I. Tatarinov, et al. Storing and Querying Ordered XML using a Relational DBMS. Tech Report, Univ. of Washington, 2002

[14] F. Yergeau, UTF-8, A Transformation Format of ISO 10646. Request for Comments 2279, January 1998. http://www.faqs.org/rfcs/rfc2279.html

[15] Albrecht Schmidt, Martin Kersten, Menzo Windhouwer, Florian Waas, Efficient Relational Storage and Retrieval of XML Documents, The World Wide Web and Databases: Third International Workshop WebDB 2000

[16] J. Shanmugasundaram et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. VLDB 1999.

[17] Michael J. Carey, David J. DeWitt, Jeffrey F. Naughton. The OO7 Benchmark. ACM SIGMOD International Conference of Management of Data 1993.

[18] Man Hon Wong. Recovery for Transaction Failures in Object-Based Database. PODS '1996.

[19] Eswaran, K.P., Gray, J., Lorie, R. A., Traiger, I.L., "The Notions of Consistency and Predicate Locks in a Database System." CACM 19(11), pp. 624-633, 1976.

[20] MOSS, J. E. B. Nested Transactions: an Approach to Reliable Distributed Computing. MIT Press, Cambridge, MA. 1985.

[21] ELISA BERTINO, BARBARA CATANIA, ELENA FERRARI. A Nested Transaction Model for Multilevel Secure Database Management Systems: ACM Transactions on Information and System Security (TISSEC) 2001

[22] Torsten Grabs, Klemens Bohm, Hans-jorg Schek. XMLTM: Efficient Transaction Management for XML Documents: CIKM'02 November 4-9,2002.

[23] Roy Goldman, Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Pcoceeding of the 23rd VLDB Conference Athens, Greece, 1997.

[24] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. RankSQL: Query Algebra and Optimization for Relational Top-k Queries In Proceedings of the 2005 ACM SIGMOD Conference (SIGMOD 2005), pages 131-142, Baltimore, Maryland, June 2005.

[25] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In ICDE, 2002.

[26] R. F. Hitti, E. O. Joslin. Evaluation and performance of computers: application benchmarks: the key to meaningful computer evaluations. In Proceedings of the 1965 20th national conference table of contents Cleveland, Ohio, United States Pages: 27 - 37.

[27] Byron C. Lewis, Albert E. Crews .The Evolution of Benchmarking as a Computer Performance Evaluation Technique. MIS Quarterly, Vol. 9, No. 1 (Mar., 1985), pp. 7-16

[28] Theo HErder and Kurt Rothermel. Concurrency Control Issues in Nested Transactions VLDB July, 1992.

[29] J. Gray, The Transaction Concept: Virtues and Limitations, Proceedings of the VLDB Conference, 1981.

[30] D. S. Yadav, Rajeev Agrawal, D. S. Chauhan, R. C. Saraswat, A. K. Majumdar, "Modeling Long Duration Transactions with Time Constraints in Active Database," itcc , p. 497, 2004.

[31] Henry F. Korth, Won Kim, and Francois Bancilhon. On Long-Duration CAD Transactions. In [ZM90], pages 408--432. Morgan Kaufman, 1990. Also in Information Science, Oct. 1988.

[32] Hossam S. Hassanein, Mohamed E. El-Sharkawi. Performance modeling of nested transactions in database systems. Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research

[33] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, M. J. Carey, I. Manolescu, and R. Busse. Why and How to Benchmark XML Databases. ACM SIGMOD Record, 3(30):27--32, September 2001

[34] Melvil Dewey, John P Comaromi, Julianne Beall, Winton E Matthews, Gregory R New, Forest Press. Dewey decimal classification and relative index. ISBN: 0910608377

[35] TPC BENCHMARKTM App Specification Version 1.1.1 August 11, 2005 http://www.tpc.org/information/benchmarks.asp.

[36] TPC BENCHMARKTM C Standard Specification Revision 5.7 April 2006

http://www.tpc.org/information/benchmarks.asp.

[37] TPC BENCHMARKTM H Standard Specification Revision 2.5.0 April 2006

http://www.tpc.org/information/benchmarks.asp.

[38] Scott Anderson, Visuale, Inc. Martin Chapman, Oracle, Marc Goodner, SAP, Paul Mackinaw, Accenture, Rimas Rekasius, IBM. Supply Chain Management Use Case Model ,Final Specification Version: 1.0 December 1, 2003. Web services interoperability organization, http://ws-i.org/.

[39] In: Gadient, Yves; Schmid, Beat F.; Selz, Dorian: EM - Electronic Commerce in Europe. EM - Electronic Markets, Vol. 8, No. 2, 07/98

[40] P. A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison Wesley, Reading, Massachusetts, 1987,