Augmenting the Performance of Existing OODBMS Benchmarks

V. Geetha
Department of Information Technology
Pondicherry Engineering College
Puducherry -605014.

N. Sreenath
Department of Computer Science & Engineering
Pondicherry Engineering College
Puducherry -605014.

ABSTRACT

Object oriented databases are widely popular for their complex data support and data relationships. Several object oriented database products are now coming to the market. Existing benchmarks are inadequate in testing the complex data support, concurrency control and performance of the database. They do not exploit the object oriented features native to the databases. This paper aims in highlighting the points where the benchmarks should focus, how they should be structured to test the performance of the databases satisfactorily.

General Terms: Object oriented databases, benchmarks, performance

Keywords: Object oriented databases, Benchmark, Concurrency control, Performance, Object oriented features

1. INTRODUCTION

Object oriented databases are very popular for their complex data support. They are widely used for applications like CAD, CAM. Several object oriented databases like E/Exodus, Objectivity, ObjectStore and Versant etc., are in the market. New object oriented database products are introduced in the market and the developers of these systems have made different choices for fundamental aspects of the system. The buyers of these object oriented database products should make certain whether these products will cater to their needs. Hence, a standard benchmark is needed to compare all the object oriented database products and choose one of them that fit best to their requirements. Benchmark also helps to identify problems with database systems. Standard benchmarks alleviate the design and development cost that are incurred in custom-made benchmarks.

Benchmarks are usually designed to measure one of the following objectives:

1. Performance in single user environment
2. Sufficiency of query support
3. Concurrency in multi user environment
4. Complex data type support

Though other objectives like disk I/O, object reference and inter process communication exist, they are given lesser importance.

In the case of relational databases, the first industry benchmark [1] by IBM purported to measure the performance of a system handling ATM transactions in batch mode. Both TPC-C and TPC-D [23] are popular benchmarks for OLTP and decision support. TPC-E is an enterprise benchmark.

Object databases differ from relational databases. They can be object - relational databases or object oriented databases. Lakey [18] has pointed out the following reasons for not adopting relational database benchmarks to test object databases. They are

1. Applications in object databases are characterized by complex structures and relationships. They follow conceptual paths rather than logical or physical paths.
2. In object database, a programming language acts as data definition language, data manipulation language and data control language. This is tightly integrated with database management.
3. Traditional metrics are different from metrics of object databases.

Several benchmarks [7, 8, 16, 19] are available for object-relational databases. Object-relational databases [24] are built on top of relational database by adding the key features namely inheritance, complex object support, an extensible type system and triggers. They emphasize on the features that are not covered in pure relational databases. The objectives of object-relational database benchmarks are different from object oriented databases. As object –relational databases are hybrid of objectives of object oriented concepts and relational databases; they have to cover features of both of them.

Gray [12] defines four primary criteria to specify a good benchmark. They are

(1) *Relevance:* it must concern aspects of performance that appeal to the largest number of potential users.

(2) *Portability:* it must be reusable to test the performances of different OODBMS.

(3) *Simplicity:* it must be feasible and must not require too many resources.

(4) *Scalability:* it must be able to be adapted to small or large computer systems or new architectures.

Based on the above factors, Darmont [9] has made a comparison of the four benchmarks namely 001, HyperModel, 007 and OCB (Object Clustering Benchmark) as given in table 1. However, the factors are too generic and can be applied to any type of benchmark.

Table 1. Comparison of existing benchmarks with Gray's criteria by Darmont [9]

	Relevance	Portability	Simplicity	Scalability
001[7]	– –	++	++	–
Hypermodel [2]	+	+	–	– –
007[5]	++	+	– –	–
OCB[10]	++	+	–	++

Strong point: + Very strong point: ++ Weak point: –
Very weak point: – –

Several object oriented database benchmarks [2, 9, 11, 18, 19] are proposed for object oriented databases. OO1 or "Cattell Benchmark"[7] was developed early in the nineties when there was no appropriate benchmark for engineering applications such as computer aided design (CAD), computer aided manufacturing (CAM), or software engineering (SE). OO1 is a simple benchmark that is very easy to implement. A major drawback of this tool is that its workload model is too elementary to measure the elaborate traversals that are common in many types of object-oriented applications.

The HyperModel Benchmark [2] or Tektronix Benchmark possesses a richer workload model than OO1. This renders it potentially more effective than OO1 in measuring the performance of engineering databases. However, this added complexity also makes HyperModel harder to implement.

ACOB (Altair Complex Object Benchmark) [11] is a study of three workstation-server or client –server object oriented database. It is designed to understand the system behavior of architecture in the execution of object operations.

Of all the object oriented database benchmarks, 007 benchmark [5] [6] is very popular and widely used. It tests the performance of object-oriented databases, by providing wide range of pointer traversals and a rich set of updates and queries. OO7 benchmark [5] reuses the structures of OO1 and HyperModel to propose a more complete benchmark and to simulate various transactions running on a diversified database. It has also been designed to be more generic than its predecessors do and to correct some of their known weaknesses. OO7 is even harder to implement than HyperModel.

However, it requires some more additions to make it a good object oriented database benchmark. Jun and Gruenwald [15] has identified the following lacuna in 007 benchmark.

1. Test cases to test queries that alter the schema are lacking.
2. The depth of the class hierarchy is not sufficient.
3. The depth of composition or nested hierarchy is very small for testing.

OO1, HyperModel, and OO7, though aimed at engineering applications are often viewed as general-purpose benchmarks. However, they feature relatively simple databases and are not well suited for other types of applications such as financial, telecommunication, and multimedia applications [3].

Hence, many benchmarks were developed to study particular domains such as client-server architectures, object clustering, active databases, workflow management, CAD applications or the study of views in an object-oriented context. A fair number of these benchmarks are more or less based on OO1, HyperModel, or OO7.

An alternative to very specific benchmarks resides in generic and tunable benchmarks such as OCB [9]. The flexibility and scalability of OCB is achieved through an extensive set of parameters that helps OCB to simulate the behavior of the *de facto* standards in object-oriented benchmarking. Furthermore, OCB's generic model can be implemented within an object-relational system easily and most of its operations are relevant for such a system. Hence, it can also be applied in an object-relational context with few adaptations. It is mainly meant for testing clustering performances of object oriented databases.

This paper highlights the lacuna in current object oriented database bench marks and indicate what could done to make them a good object oriented database benchmark. The main drawback in these benchmarks is they do not use exploit the object oriented aspects of object oriented databases to design their test cases. The performance of object oriented databases is mainly based on complex data support and concurrency. Hence, this paper focuses in highlighting how the test parameters can be chosen based on object-oriented aspects.

The paper is organized as follows. Chapter 2 gives the background needed to understand the need for new benchmark. It describes the structure of a typical object oriented database and identifies the lacuna in existing benchmarks. Chapter 3 describes the expectations from a good object oriented database benchmark. It highlights the expected benchmark objectives and lists the test parameters and test cases. Chapter 4 concludes the paper.

2. BACKGROUND

2.1 Structure of Object Oriented Databases

Object oriented databases are widely used for advanced applications like CAD, CAM etc., as they support representation of complex data and their complicated relationships. Object oriented database is a collection of objects. The objects are of two types - classes and instances. A class object consists of attributes and methods. It defines the structure and behavior of an entity. The domain is mapped on the instances and represented as database.

The clients can access the ODBMS in two modes - runtime mode and design time mode. In runtime mode, the domain data is mapped onto the attributes of instances and the associated member functions operate on them to satisfy client requests. The member function (also called as methods) might read / modify the attribute values. Design time mode is used to read or modify the schema to reflect the changes in the domain. It can be in one of these granularities: - class lattice level, class level and instance level. The operations allowed are to read and modify attribute definitions, method definitions, class definitions and class relationship definitions. In general, database access can be a read/write operation. The read operations can be executed in shared lock mode and write operations should be executed in exclusive lock mode. The transactions in object oriented databases are long duration transactions.

The schema of OODB is represented as a class diagram. The class diagram is a collection of classes related by inheritance, aggregation (composition) and association relationships. Group of classes related by inheritance (excluding *multiple inheritance*) is called *class hierarchy*. Group of classes related by a combination of all types of relationships mentioned above is called *class lattice*. Then class diagram can be viewed as a class lattice and represented as Directed Acyclic Graph (DAG). The classes are viewed as nodes and the relationship links connecting classes are viewed as edges. The design time transactions can do changes to schema in two ways as specified in Bannerjee et al. [4].

The schema changes are categorized into

1. Changes to the contents of node or class
   1.1 Changes to instances
   - Add a new instance to a class
   - Delete an existing instance from a class
   - Modify the definition of an instance
   - Move an instance from one class to another class
   - Read the definition of an instance

   Changes to attributes
   - Add a new attribute to a class
   - Delete an existing attribute from a class
   - Modify the definition of an attribute
   - Move an attribute from one class to another class
   - Read the definition of an attribute

   1.3 Changes to methods
   - Add a new method to a class
   - Delete an existing method from a class
   - Modify the definition of an method
   - Move a method from one class to another class
   - Read the definition of an method
2. Changes to an edge
   - Make a class S as superclass of class C
   - Delete a class S from the super class list of class C.
   - Modify the order of superclasses of class C
   - Read the superclass list of class C.
3. Changes to a node or class
   - Add a new class
   - Delete an existing class
   - Modify the definition of a class
   - Move a class from one location to another position
   - Read the definition of a class

From the above group of operations, certain semantic aspects can be inferred. During runtime transactions, the values of attributes are read or modified by executing the associated methods in a class. The attribute values are locked in read or write lock mode. In design time transactions, the attribute definitions are read or modified. Thus, attribute has two facets and they are chosen depending on the type of transaction.

During runtime transactions, the methods are locked in read mode, as their contents are not modified by execution. In design time transactions, the method definitions are read or modified. When any attribute or method definition is modified, runtime transactions accessing them should not be allowed.

A runtime transaction can have attribute, instance or class level of granularity. It is based on the property of the method as to whether the method is 1. *Primitive* or C*omposed* and 2. *Instance* or C*lass* level as defined by Reihle and Beczuck [20].

2.2 Lacuna in existing object oriented database benchmarks

After analyzing all the benchmarks for object oriented databases, while testing the concurrency control techniques [15, 14], the following lacuna were identified:

- All the benchmarks focus on database aspects namely pointer traversal, indexing, support of scan, selection and range queries, join complexity, buffer management etc. The focus on object-oriented aspects is shallow. This is very important because all the features offered by object-oriented databases are offered through object-oriented aspects.
- Object oriented databases are adopted for several domains. The requirements of each of these domains are different. So testing the product with single application is not sufficient. A very good application in each domain has to be identified. For example, implementing ATM application can be considered as benchmark application for finance domain.
- The query support is inadequate in all the benchmarks. Hence, test cases should be framed in such a way as to cover all possible operations done at runtime and design time in the domain. For example, 007 benchmark does not support queries altering the schema. In object oriented databases, the schema is represented by class diagram as pointed earlier in section 2.1. 007 benchmarks allow usage of existing schema. It does not provide any test case to modify the structure of the schema.
- Complex data support is the main attraction of object oriented databases. It is possible only by the relationships namely inheritance, aggregation and association. Some of the benchmarks provide test cases to test the support of these relationships only individually. But in order to implement any domain, a mixture of all the relationships in any order is required. For example, object of an inherited subclass can be component of a composite class or class of a composite object may be a base class that is inherited to one or more sub classes. Hence, complexity of data support depends on the depth to which the combinations of these relationships are supported.
- To test the concurrency of any product, the conflicts among runtime transactions, among design time transactions and between runtime and design time transactions are to be taken as test cases. While testing the product in multi user environment, apart from measuring the response time of transactions, the smallest granularity supported should also be noted.

3. DESIGN EXPECTATIONS FROM A GOOD OBJECT ORIENTED DATABASE BENCHMARK

The expectations from a good benchmark are discussed in this section. The factors to be considered for benchmarking are first identified. Then associated testing parameters and test cases are listed.

3.1 Factors for benchmarking

1. Complex data support:

Data is represented as attributes in object oriented databases. The attributes and associated member functions compose the objects. Attributes map to the underlying database. The data type of attributes can be ADT (Abstract Data Type) or object. ADT is an atomic data type defining primitive data types. The support of ADT is available in all databases. Object data type is a user defined complex data type. It allows users to define one object as part of another object. The object that is defined inside another object is called component object and the object holding it is called composite object.

```
Class Person {
String name;          \* attributes*\
String address;
:
}
Class faculty: class person {
int empID;            \* attributes*\
String designation;
      :
}
Class department {
int deptID;
faculty staff [50];   \* attributes*\
       :
     }
```

In the above example, all attributes of class 'person' and 'faculty' belongs to ADT. Class 'faculty' is inherited as sub class from base class 'person' by inheritance. In class 'department', attribute 'deptID' is of abstract data type. Attribute 'staff' is of object data type. Object of 'faculty' is included as attribute of department. Then 'faculty' is called as component object and object of 'department' is composite object. This can be nested to any level. Kim [17] has defined this "part of "relationship also called as composition or aggregation to be of two types namely shared or dedicated composition. The dedicated composition does not allow an object to be part of more than one object. On the other hand, shared composition allows this. Same component object can be part of more than one composite object. When a component object is accessed in one composite object, it should not be simultaneously accessed by other composite objects sharing this component object. This is needed to preserve consistency of the component object. Composition is also classified into dependent and independent composition. A dependent composite object depends on component object for its life. If the component object is destructed, the associated composite object is also destructed. However an independent composite object's life is independent of its components. The composition is thus classified into shared dependent composition, shared independent composition, dedicated dependent composition and dedicated independent composition.

In the above example, inheritance is followed by composition. Inheritance is classified into single inheritance, multilevel inheritance, multiple inheritance and hybrid inheritance. Similarly, association is classified into independent association and dependent association. In the schema of a domain, these relationships can be defined any number of times and in any combination. Then the benchmark has to be adequate enough to test the support of the product to represent them.

2. Support for completeness of operations

The domain operations at database level are listed. Basically the operations can be classified into two types: operations possible at runtime and operations possible at design time. At runtime, the data from database copied on attributes are accessed. The attribute values are read or updated. At design time, the operations for modifying the schema are allowed. The operations allowed at design time are listed in section 2.1. So a good benchmark has to verify whether the domain operations relating to all these operations are supported in the product. Even if a database product does not individually support all these operations, it might provide all these operations in few groups. Then operation granularity becomes coarse and performance will be poor.

3. Support for concurrency

In single user scenario, performance is measured by response time. In multi user environment, performance is measured by concurrency. In multi user environment, three different test cases are to be included to test

[1] Conflicts among runtime transactions
[2] Conflicts among design time transactions
[3] Conflicts between runtime and design time transactions.

The database product has to be tested for concurrency support by applying all the above three cases.

3.2 Testing parameters

After analyzing the lacuna in the existing benchmarks, the following parameters are recommended to be added to make them a good benchmark.

- *Application* – An application has to be identified by for each of the domain namely CAD, CAM, Software Engineering, Finance etc. This is needed to test the versatility of the product.
- *Database Size* - Three database sizes namely small, medium and large as in 007 can be maintained to test the scalability of the product.
- *Lattice Depth* – This defines the total depth of the class diagram. This is defined by the number of levels in the class diagram. It is the maximum number of combinations of inheritance, composition and association relationships between the classes on the top of the class diagram and the bottom most class. Values for this parameter can be fixed for all the three database sizes.
- *Hierarchy Depth* – It is the maximum number of inheritance hierarchy levels between any two classes in the class diagram. It can be less than or equal to lattice depth. Values for this parameter can be fixed for all the three database sizes.
- *Composition Depth*- It is the maximum number of nesting levels of composition. It can also be less than or equal to lattice depth. Values for this parameter have to be fixed for all the three database sizes.
- *Association Depth*- It is the maximum number of associations chained among group of classes in the class diagram. It can also be less than or equal to lattice depth. Values for this parameter have to be fixed for all the three database sizes.
- *Maximum degree class* – It is the class which has maximum number of relationships with other classes. Values for this parameter have to be fixed for all the three database sizes.
- *Number of instances for maximum degree class*- This is the maximum number of instances that can be created for

class that is maximum related to other classes. Parameter values for all three database sizes are to be fixed.

- *Maximum in-degree class*- It is the class which is derived from maximum number of classes. Value is to be fixed for all database sizes.
- *Maximum out-degree class* - It is the class from which from maximum number of classes is derived.. Value is to be fixed for all database sizes.

3.3 Test cases

The test cases are to be defined separately for single user and multi user environment. In single user environment, the sufficiency of operations or query support is tested. For this queries for all the operations mentioned in section 2.1 are tested. The runtime operations are tested on the classes which are inherited or composed or associated maximum from other classes. When runtime transaction requests a base class instance, it is enough to lock the base class instance alone. But when a runtime transaction requests a sub class object, it is required to lock the associated base class objects that access the same record also to preserve database consistency. So consistency bugs of the database can be checked. This is applicable to aggregation and association also. In aggregation, the component objects are locked with composite objects. In association, associated objects are locked with associative objects.

The design time operations are tested on the base classes or component classes or associative classes as any change in these classes will affect all the derived classes.

For multi user environment, concurrency is tested for all the three cases mentioned in 3.1. The design time transactions can be tested and equal weightage is given to all the three structural modifications in section 2.1.

4. CONCLUSION

In this paper, the need for exploiting the object oriented features of object oriented database product while testing it, is emphasized. The lacuna in the existing benchmarks is identified. The existing benchmarks are either too generic or too narrowed down for usability. In this paper, the genericness of the structure is supported. At the same time the design expectations from a good object oriented bench mark quantitatively as well as qualitatively are listed. Parameters and test cases are defined for both single and multi-user environments.

The 007 benchmark provides the basic structure of an OODBMS benchmark. If it is extended to support the features listed in this paper, it can be used more effectively.

5. REFERENCES

[1] Anon et al. 1985. A measure of Transaction Processing Power, Datamation, 31(7):112-118.

[2] Anderson et al. 1990. The HyperModel Benchmark, Proc. of the Int. Conference on Extending Database Technology.

[3] Ashutosh Tiwary, Vivek R. Narasayya, Henry M. Levy: 1995. Evaluation of OO7 as a system and an application benchmark, In OOPSLA Workshop on Object Database Behavior, Benchmarks and Performance, Austin, Texas.

[4] Banerjee et al. 1987. Semantics and Implementation of Schema evolution in Object–Oriented Databases, In Proceedings of ACM SIGMOD conference.

[5] Carey et al. 1993. The OO7 Benchmark, In Proceedings of the ACM SIGMOD Conference.

[6] Carey et al. 1994. A Status Report on the OO7 OODBMS Benchmarking Effort, In proceedings of the ACM OOPSLA Conference.

[7] Cattell R. G.G and Skeen. J. 1992. Object Operations Benchmark, ACM Transactions on Database Systems. 17 (1):1-31.

[8] Cattell R.G.G. 1993. An Engineering Database Benchmark, In: The Benchmark Handbook for Database and Transaction Processing Systems 2nd ed., J. Gray ed., Morgan Kaufmann.

[9] Darmont. J, Petit.B and Schneider. M. 1998. OCB: A generic benchmark to evaluate the performances of object-oriented database systems, Proceedings of 6th International Conference on Extending Database Technology Valencia, Spain, LNCS.

[10] Darmont Jerome, and Michel Schneider. 2002. Object-Oriented Database Benchmarks, Advanced Topics in Database Research, Volume 1. IGI Global, 34-57.

[11] DeWitt et al. 1990. A study of three alternative workstation-server architectures for object oriented database systems. Proceedings of the Sixteenth Very Large Data Bases Conference, Brisbane, Australia, pp. 107-121.

[12] Gray.J. 1993. The Benchmark Handbook for Database and Transaction Systems (2nd Edition), Morgan Kaufmann.

[13] Geetha. V and Sreenath. N. 2011. A Multi-Granularity Lock Model for Object Oriented Databases using Semantics, International Conference on Distributed Computing and Internet Technologies, Bhubaneshwar, India, Proceedings in LNCS.

[14] Geetha. V and Sreenath. N. 2011. Semantic Based Concurrency Control in OODBMS, International Conference on Recent Trends in Information Technology, Chennai, India, in Proceedings in IEEE Computer Society.

[15] Jun, W. And Gruenwald. L 1997. Experiences with the 007 Benchmark for Concurrency Control Technique Performance Evaluations, ACM Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '97) Workshop on Experiences Using Object Data Management, pp. 1-5.

[16] Karey M.J. et al. 1997. The BUCKY Object-Relational Benchmark, In proceedings of the 1997 ACM – SIGMOD International Conference on Management of Data, Tuscon, Arizona.

[17] Kim.W, Bertino,E and Garza.J.F.1990. Composite Objects revisited, Object oriented Programming, systems, Languages and Applications, pp 327-340.

[18] Lakey. B. 1989. Developing benchmarks for comparing relational and object-oriented database systems. Master's Thesis, Oregon Graduate Center, Beaverton, Oregon.

[19] Lee. S.H et al. 2000. The BORD Benchmark for Object-Relational Databases, Springer-Verilag Lecture Notes in Computer Science.

[20] Riehle.D, Stephen P. Berczuk. 2000. Properties of Member Functions in C++, Report.

[21] H.Schreiber, Justitia:. A Generic Benchmark for the OODBMS Selection, Proc. of the Fourth International Conference of Data and Knowledge Systems for Manufacturing and Engineering, (1994)

[22] Schreiber and Justitia. 1994. A Generic Benchmark for the OODBMS Selection, Proc. of the Fourth International Conference of Data and Knowledge Systems for Manufacturing and Engineering.

[23] Shanely.K 1998. History and overview of the TPC, Transaction Processing Performance Council, http:// tpc.org.

[24] Stonebraker. M. 1996. Object – Relational Database Systems: the next wave, Morgan Kaufmann.