# A Status Report on the OO7 OODBMS Benchmarking Effort*

Michael J. Carey David J. Dewitt Chander Kant Jeffrey F. Naughton
Computer Sciences Department
University of Wisconsin-Madison

## Abstract

The OO7 Benchmark was first published in 1993, and has since found a home in the marketing literature of various object-oriented database management system (OODBMS) vendors. The OO7 Benchmark (as published) was the initial result of an ongoing OODBMS performance evaluation effort at the University of Wisconsin. This paper provides an update on the status of the effort on two fronts: single-user and multi-user. On the single-user front, we review and critique the design of the initial OO7 Benchmark. We discuss some of its faults, the reasons for those faults, and things that might be done to correct them. On the multi-user front, we describe our current work on the development of a multi-user benchmark for OODBMSs. This effort includes changes and extensions to the OO7 database and the design of a family of interesting multi-user workloads.

## 1 Introduction

The OO7 benchmarking effort is an on-going research project that aims to evaluate the performance of OODBMSs. The first result of this effort, which built upon the foundation laid by earlier OODBMS benchmarking efforts [CS92, RKC87, And90, DD88], was the single-user OO7 benchmark [CDN93]. This benchmark differed from its predecessors in that it attempted to provide a truly comprehensive test of single-user OODBMS performance. Among the performance characteristics tested by the OO7 Benchmark are the speed of a given OODBMS on a wide variety of different pointer traversals (e.g., over cached data and disk-resident data, including both sparse and dense traversals), updates (including updates of both indexed and unindexed data, repeated updates, sparse updates, updates of cached data, and object creation and deletion), and simple object queries (including both exact-match and range queries and both pointer-based and value-based joins).

To date, we have completed "phase one" of the OO7 effort, in which we defined the single-user benchmark and used it to profile the performance of a number of commercial OODBMSs. A full report on this phase, giving a detailed benchmark description and measured performance results for four systems on a common client-server hardware base, is available via anonymous ftp from `ftp.cs.wisc.edu` in the OO7 directory. The ftp directory also contains the source code for the benchmark for five systems.[1] The difference between the number of available implementations and the number of systems for which we have published our measurements is a reflection of the fact that the OO7 Benchmark is not without controversy. The approach used to specify the benchmark, and to limit liberties that might otherwise be taken when implementing it, has been a point of some contention and concern for several vendors who would have liked to participate (but under a different set of ground rules).

Despite the information provided by OO7 and other OODBMS benchmarking efforts, there is virtually no published information available about the "real" performance of commercial OODBMSs. This is because real applications usually involve multiple users, making the multi-user performance of

*This work was funded by Digital Equipment Corporation. This paper initially appeared in the proceedings of OOPSLA '94.

[1] Also available on the World Wide Web at URL ftp://ftp.cs.wisc.edu/OO7.

OODBMSs critical to understand. Unfortunately, the performance studies to date provide little or no insight into what can be expected for multi-user performance. This leaves a critical gap in the available knowledge for consumers of OODBMS technology. The OO7 effort at Wisconsin is currently working to address this gap by developing a comprehensive multi-user OODBMS benchmark. We have changed and extended the design of the OO7 database to better accommodate multi-user workloads, and we have developed a parameterized OODBMS workload that produces an interesting family of multi-user workloads. We are in the process of trying out our benchmark on several OODBMSs, and we are also actively seeking input from OODBMS vendors and users regarding our design.

The remainder of this paper begins with a brief review of the design of the single-user OO7 Benchmark. (Readers familiar with the OO7 Benchmark can skip this section of the paper.) The review is then followed by a retrospective critique of the benchmark and the way in which it was administered; this section of the paper is largely non-technical, presenting some impressions and lessons from our single-user benchmarking experience. We then turn our attention to our current work on multi-user OODBMS benchmarking, discussing proposed changes in the OO7 database and the associated multi-user workload family that we propose to use in evaluating OODBMS performance. A major goal of this portion of the paper is to convey the status of our effort in order to publically solicit feedback from vendors and users of commercial OODBMS technology regarding what we're doing right, what we're doing wrong, and what we might be missing in our current multi-user benchmark design.

# 2 Single-User OO7 Benchmark

As mentioned in the introduction, the original OO7 Benchmark [CDN93] was designed to test OODBMSs on a wide variety of traversal, update, and query tasks in order to thoroughly explore their single-user performance characteristics. As a result, its database structure and operations are non-trivial. The OO7 Benchmark is intended to be suggestive of many different CAD/CAM/CASE applications, although in its details it does not model any specific application. It is important to realize that the goal of the benchmark is to focus on important aspects of system performance, not to model a specific application. Accordingly, in the following when we draw analogies to applications, we do so to provide intuition into the benchmark rather than to justify or motivate the benchmark.

| Parameter | Small | Medium |
|---|---|---|
| NumAtomicPerComp | 20 | 200 |
| NumConnPerAtomic | 3/6/9 | 3/6/9 |
| DocumentSize (bytes) | 2000 | 20000 |
| Manual Size (bytes) | 100K | 1M |
| NumCompPerModule | 500 | 500 |
| NumAssmPerAssm | 3 | 3 |
| NumAssmLevels | 7 | 7 |
| NumCompPerAssm | 3 | 3 |
| NumPrivateModules | 1 (per client) | 1 (per client) |

Table 1: OO7 Benchmark database parameters.

## 2.1 OO7 Database Description

There are two "official" sizes of the OO7 Benchmark database: small and medium. Table 1 summarizes the parameters of the OO7 Benchmark database; their meanings will become clear in a moment when we walk through the database design. In our five implementations of the benchmark, all of these parameters are controlled by a configuration file that is read by the database generation code. As indicated in the table, the benchmark was designed even initially to scale in proportion to the number of clients by having one private "module" per client in both the small and medium databases (modules are described below).

Our goal in setting the parameters for the "small" and "medium" cases in the way that we did was to arrange it so that for the small database, on high-locality workloads, each client's working set will fit in its cache, while for the medium database, the working set will not fit in the client cache; also, for the small database, the entire database can fit in the server cache, whereas the medium database will not fit in the server cache. We are generalizing here, of course, as exactly what will and won't fit depends upon (1) the characteristics (e.g., the pointer representation) of the specific OODBMS being tested and (2) the sizes of the client and server caches.

Before describing the OO7 Benchmark database further, we note that the full source code for our implementations in all the systems we tested is available by anonymous ftp from `ftp.cs.wisc.edu`.

### 2.1.1 The Design Library

A key component of the OO7 Benchmark database is a set of *composite parts*. Each composite part corresponds to a design primitive such as a register cell in a VLSI CAD application, or perhaps a procedure in a CASE application; the set of all composite parts forms what we refer to as the "design library" within the OO7 database. In the design library, the number of composite parts associated with each module is controlled by the parameter *NumCompPerModule*, which is set to 500. Each composite part has a number of attributes, including the integer attributes `id` and `buildDate`, and a small character array `type`. Associated with each composite part is a *document* object, which models a small amount of documentation associated with the composite part.

In addition to its scalar attributes and its association with a document object, each composite part has an associated graph of *atomic parts*. Intuitively, the atomic parts within a composite part are the units out of which the composite part is constructed. In the small benchmark, each composite part's graph contains 20 atomic parts, while in the medium benchmark, each composite part's graph contains 200 atomic parts. (This number is controlled by the parameter *NumAtomicPerComp*.) For example, if a composite part corresponds to a procedure in a CASE application, each of the atomic parts in its associated graph might correspond to a variable, statement, or expression in the procedure. One atomic part in each composite part's graph is designated as the "root part."

Each atomic part has the integer attributes `id`, `buildDate`, `x`, `y`, and `docId`, and the small character array `type`. The `buildDate` values in atomic parts are randomly chosen in the range `MinAtomicDate` to `MaxAtomicDate`, which is currently 1000 to 1999. In addition to these attributes, each atomic part is connected via a bi-directional association to several other atomic parts, as controlled by the parameter *NumConnPerAtomic*; this parameter is first set to 3, then to 6, and then to 9 in the single-user benchmark. Our initial plan was to connect the atomic parts within each composite part in a random fashion. However, random connections do not ensure complete connectivity (i.e., reachability of all of a composite part's atomic parts from its root part). To ensure complete connectivity, one connection is initially added to each atomic part to connect the parts in a ring; the remaining connections are then added at random.

The connections between atomic parts are implemented by interposing a connection object between each pair of connected atomic parts. Here the intuition is that the connections themselves contain data; the connection object is the repository for that data. A connection object contains the integer field `length` and the short character array `type`.

Figure 1 depicts a composite part, its associated document object, and its associated graph of atomic parts. One way to view this is that the union of all atomic parts corresponds to the object graph in the OO1 benchmark [CS92]; however, in OO7 this object graph is broken up into semantic units of locality by the composite parts. Thus, the composite parts in OO7 provide an opportunity to test how effective various OODBMS products are at supporting complex objects.

#### 2.1.2 Assembling Complex Designs

The design library, which contains the composite parts and their associated atomic parts (including the connection objects) and documents, accounts for the bulk of the OO7 database. However, a set of composite parts by itself is not sufficiently structured to support all of the operations that we wished to include in the benchmark. Accordingly, we added the notion of an "assembly hierarchy" to the database. Intuitively, the assembly objects correspond to higher-level design constructs in the application being modeled in the database. For example, in a VLSI CAD application, an assembly might correspond to the design for a register file or an ALU. Each assembly is either made up of composite parts (in which case it is a *base assembly*) or it is made up of other assembly objects (in which case it is a *complex assembly*).

The first (bottom) level of the assembly hierarchy consists of *base assembly* objects. Base assembly objects have the integer attributes `id` and `buildDate`, and the short character array `type`. Each base assembly has a bi-directional association with three composite parts. (The number of composite parts per base assembly is controlled by the parameter *NumCompPerAssm*.) In the single-user OO7 Benchmark, each base assembly had associations with two kinds of composite parts, private and shared, although only the private associations were used; the shared associations were designed into the database for use in the multi-user benchmark design that we were anticipating for our follow-on work.

Higher levels in the assembly hierarchy are made up of *complex assemblies*. Each complex assembly has the usual integer attributes, `id` and `buildDate`, and the short character array `type`; additionally, it has a bi-directional association with three subassemblies (controlled by the parameter *NumAssmPerAssm*), which can either be base assemblies (if the complex assembly is at level two in the assembly hierarchy) or other complex assemblies (if the complex assembly is higher in the hierarchy). There are seven levels in the assembly hierarchy (controlled by the parameter

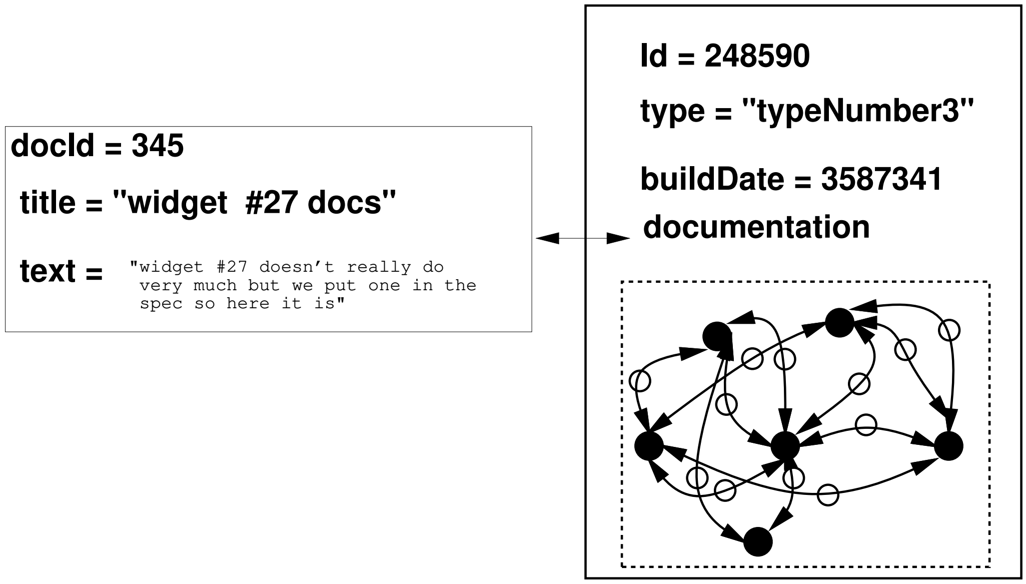


Figure 1: A Composite Part and its associated Document object.

*NumAssmLevels*).

Each assembly hierarchy is called a *module*. Modules are intended to model the largest subunits of the database application, and are used extensively in the multi-user workloads; they are not used explicitly in the small or medium single-user OO7 databases, each of which consists of just a single module, but were included for their usefulness in our planned multi-user follow-on work. Modules have several scalar attributes — the integers `id` and `buildDate`, and the short character array `type`. Each module also has an associated *Manual* object, which is a larger version of a document. Manuals are included for use in testing the handling of very large (but simple) objects.

Figure 2 depicts the full structure of the single-user OO7 Benchmark database. Note that the picture is somewhat misleading in terms of both shape and scale; the actual assembly fanout used is 3, and there are only $(3^7 - 1)/2 = 1093$ assemblies per module in the small and medium databases, compared to 10,000 atomic parts per module in the small database and 100,000 atomic parts per module in the medium database. Also, as mentioned earlier, the composite parts associated with a given module are private to the module in the sense that they are only referenced by base assemblies of that particular module.

## 2.2 Single-User OO7 Operations

The operations of the single-user OO7 Benchmark can be roughly grouped into three categories: read-only traversals, updates, and queries. Each category, in turn, contains a number of different operations in order to achieve broad OODBMS performance coverage.

### 2.2.1 Read-Only Traversal Operations

The first category, read-only traversals, consists of four distinct traversal operations. The first two are based on traversing the assembly hierarchy. The traversals start from the top (i.e., at the module level) and proceed in a depth-first manner; as base assemblies are reached, each referenced composite part is accessed. The first traversal is a dense traversal that accesses the entire atomic parts graph of each encountered composite part (via a depth-first search). The second traversal is a sparse traversal, which accesses only the root part of each composite part, leaving the remainder of the atomic parts graph untouched. The third and fourth traversals focus on the manual object associated with a module. The third traversal scans the entire text of the manual, while the fourth traversal touches only the first and last characters of the manual's text (to test random accesses to very large objects).

Each of the traversal operations is implemented using methods of the object classes involved, as each one is supposed to represent an operation that was anticipated (unlike the operations in the query category) at application design time. For each traversal, both a cold time (based on starting with an empty cache) and an average hot time (based on reaccessing exactly the same data repeatedly) are reported. In addition, for each hot time, two cases are reported

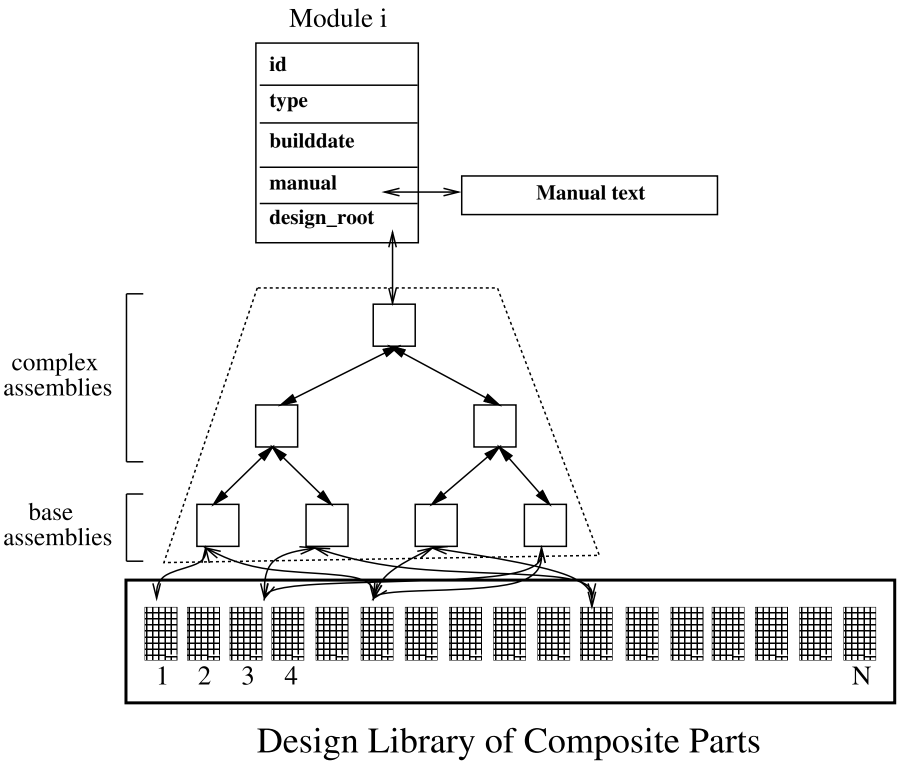


Figure 2: Structure of a module.

— with the traversals being performed either within one transaction or as many transactions — to expose caching performance both within and across transaction boundaries.

#### 2.2.2 Update Operations

The second category, updates, includes two update traversals. Both of these execute the first (dense) assembly hierarchy traversal operation but perform updates along the way. The first update traversal changes the value of two unindexed attributes of either one, all (once), or all (repeated a total of four times) of the atomic parts that it encounters. The second update traversal is similar, but instead changes the value of one indexed attribute of the atomic parts. The update category also includes an insert operation, which inserts five new composite parts into a module, and a corresponding delete operation, which deletes the same five composite parts in order to restore the module to its original logical state. Together, the collection of update operations in the benchmark tests the efficiency of a number of update-related implementation issues, especially the recovery mechanism (usually based on writing log records or shadows) and the index manager.

#### 2.2.3 Object Queries

The final category, queries, is a set of seven object queries that range from simple to moderately complex (at least given the state of OODBMS query languages and implementations as of the initial OO7 benchmarking exercise). Included are an exact-match query that looks up one atomic part, two range queries that look up 1% and 10% of the atomic parts, respectively, and a sequential scan query that accesses every atomic part in the type extent for atomic parts. The three more complex queries in the query category are essentially "join" queries. The first two are pointer joins, i.e., they are navigational in nature. One is a three-level path query that goes from a document title to the base assemblies that use the document's corresponding composite part; the other is a two-way pointer join of two connected type extents (composite parts and base assemblies). The last join query is an ad hoc (value-based) join between two other type extents (between documents and atomic parts, based on matching the documents' ids and the document id field that atomic parts have). Each of the OO7 object queries are written as declaratively as possible using whatever facilities the tested OODBMS has for expressing, or otherwise implementing, queries.

# 3 Single-User Lessons

As mentioned in the introduction, we implemented and ran the single-user OO7 Benchmark on five different OODBMSs (four commercial systems plus our own research prototype). We were eventually permitted to report results for four of the five systems. This experience taught us quite a bit about the systems tested as well as the overall state of the commercial OODBMS industry. The various technical lessons that we learned are reported in [CDN93]. Here we look back on some of the other, less technical issues and lessons (including a few design flaws) that resulted from the initial OO7 work.

## 3.1 Specification Woes

One issue that concerned us from the very beginning of our work was how to ensure that the OO7 benchmark implementations for two different OODBMSs are really providing a fair, "apples to apples" comparison of the systems. Essentially, this question boils down to "what really is *the benchmark*?" (in terms of the degrees of freedom in its implementation). Perhaps less intuitively, we found that in OODBMS benchmarking this also boils down to the question "what really is *the system*?" Every system tested had a rather different set of features as well as a different set of potentially useful performance knobs (each requiring a different level of wizardry and a different degree of knowledge regarding specific details of the OO7 database and operations).

An easy out would have been to simply specify the benchmark operations at a high level (e.g., in English) and let each vendor implement them in the best way that they could using algorithms of their choosing and every performance feature available in their system. However, we specifically wanted to avoid this, as it would have defeated our purpose. That approach would have turned the OO7 effort into a programming contest for wizards rather than a comparison of OODBMS engines as they would likely be used by intelligent, but not exceptional, application programmers. Instead, we chose a different approach — all official OO7 numbers to date have been obtained by us, running code that we either implemented personally or else audited very carefully, and each system was tested on identical hardware at the University of Wisconsin. Was this really necessary? Unfortunately, yes.

As just one example of why we conducted the benchmark this way, most OODBMSs (all commercial systems that we tested, in fact) provide a set facility. This provides a convenient way to represent the fact that a given object, A, has a one-to-many association with a set of other objects, B, C, and D. Using the set facility, you can store in object A a set of references to the objects B, C, and D. The manuals of the various systems, as well as the provided example programs, do indeed implement one-to-many associations in this way. However, each OODBMS essentially provides the full power of C++, making it possible to bypass the system-provided set facility. Instead, one could use C++ to code a special association class tailored for the OO7 benchmark. This hand-coded association class might be much faster than the standard system-provided class. Should doing so be an acceptable approach?

Our answer was that system-provided classes for things like sets must be used wherever they are appropriate in the benchmark — otherwise users who employ system-provided facilities would be unable to duplicate the benchmark results. Moreover, one would hope that these system-provided facilities, described in the manuals as the "recommended" approach for modeling relationships, would be reasonably well implemented. However, in some cases we were criticized with the comment that "sets are not part of the engine, so why are you benchmarking them?" Similarly, we were criticized in some cases for not using the fine-tuning features offered by a system when doing so meant hard-wiring details of the OO7 database parameters (e.g., exact sizes of all objects, exact cardinalities of all sets, and so on) into the benchmark code. Again, we resisted vendors' urgings to do this, as we felt that typical OODBMS application programmers would not be able to use such features nearly as well as an expert programmer who was implementing the OO7 Benchmark with full, a priori knowledge of all of the database parameters. Our focus was on obtaining OODBMS performance information that would be useful to typical programmers.

When we ran the original tests there was no clear way to resolve this issue. We started by specifying the benchmark in English, and we simply made public our own implementations for five systems to clarify the intent of the English. Obviously, this needs to be tightened up. Luckily, now there is hope. The ODMG committee, whose membership includes all major OODBMS vendors, recently proposed a standard data model for OODBMS applications [Cat94] and embeddings for this model in C++ and Smalltalk. A clean way to specify the benchmark in the future will be to provide an implementation of the benchmark using this standard, thus ensuring that all systems running the benchmark will run exactly the same code. A particularly interesting (but time consuming) exercise would be to run the benchmark both this way and using all of each vendor's bells and whistles (or at least those made available through optional extensions of the ODMG standard) to see what each system's "dynamic range" might be.

## 3.2 More Specification Woes

A related problem with the specification and implementation of the initial OO7 Benchmark — or perhaps just an extreme example of the issues discussed above — arose with respect to the queries in the benchmark. When we began the OO7 project, the target OODBMSs varied wildly in their support for declarative query processing. Some systems had no declarative query language. Among those systems that did support a declarative query language, the query languages had widely differing expressive powers. For this reason, we implemented each of the OO7 queries in the system-provided query language if possible; otherwise we hand-coded a "reasonable" evaluation algorithm for the query in C++, using the same algorithm across all "query-impaired" systems to avoid the programming contest problem. We have been justifiably criticized that this approach could penalize a system for providing a query processor.

Due to this problem, the query processing aspect of the OO7 Benchmark is an aspect of the benchmark that needs significant improvement. Once again, as the OODBMS industry matures, however, there is hope for avoiding this situation. Specifically, the ODMG committee has also proposed a standard query language, called OQL, that ODMG-compliant OODBMSs will be required to support. In light of this proposal, we will revisit the query portion of the benchmark. For the next major iteration of the single-user benchmark, we intend to express the queries using the OQL query language standard. If a given system does not support OQL or an equivalently powerful declarative language, it will not be permitted to participate in the query portion of the benchmark.

## 3.3 Auditing the Auditors

One legitimate concern with benchmarking is: who will audit the benchmark implementation and results? As mentioned above, our initial solution was to do most of the implementation work and all of the benchmark-running work ourselves, very carefully auditing (and toning down when necessary) those implementations that were vendor-provided. Moreover, in the past, we (the UW OO7 team) have set the rules for the benchmark and have interpreted them and resolved all disputes. We do not enjoy playing all of these roles; ideally, some standards group like the TPC should audit benchmarks. Perhaps in the future OO7 (or some derivative benchmark) will be adopted by ODMG or a related standards group who will put a mechanism in place to clarify the rules and audit the implementations and results. We think this is a natural progression for a benchmark — being proposed in its initial form by a small team of people, and then being revised, formalized, and administered by a standards body that both vendors and customers can trust. This progression (which produced TPC-A through TPC-C [Gra93]) seems more efficient than the alternative of letting a large committee of vendors define a benchmark from scratch (*a la* the approach taken in TPC-D [TPC94]), as it is difficult for a large committee with diverse goals and mutual fears to define something as complex as a database benchmark.

## 3.4 Freedom of Information

Specification and auditing issues aside, one overriding problem that a would-be benchmarker must face is that many OODBMSs (like almost all relational DBMSs) have clauses in their software licenses that prohibit the release of performance results without explicit permission from the OODBMS vendor. That is, for some systems, it is actually a violation of the system's license agreement, and therefore a very real potential source of a law suit, to purchase an OODBMS, run a benchmark, and publish the results. Moreover, as we discovered, this is not just idle jargon in the license agreement. If you do go ahead and benchmark such a system, you will almost surely be hearing from the vendor's lawyers.

While it is apparently perfectly legal for software licenses to be written in this manner, one has to wonder why some vendors feel the need for such clauses — and what the impact is on the information systems industry. If other industries followed the DBMS industry's lead, Road & Track could not publish its new car road tests, and Consumer Reports could not evaluate dishwashers and stereos. Closer to the computer industry, it would be prohibited to publish benchmark ratings for new microprocessors. Clearly, this clause has a very strong stifling effect on would-be benchmarkers. It certainly made our job much harder; at each step of the way we had to negotiate with some of the companies involved to keep them from firing up their lawyers and dropping out of the benchmark. (At times it also kept our FAX machine humming with legal correspondence, leading us to duck and shout "incoming!" each time the FAX phone rang in our secretary's office.)

Dealing with this issue was very painful, both for us and for our technical colleagues who work for the various OODBMS vendors. Having said that, we should also note that in general our contacts in the companies involved were helpful and supportive in what was an extremely high-stress situation for everyone involved.

### 3.5 Interpreting the Results

Given that one has managed to overcome (or ignore for the moment) the sticky issues above, producing a set of numbers, one is then faced with yet another issue: What do the benchmark results mean? This is viewed by some as being a major problem with the OO7 Benchmark, and many have urged us to make the OO7 Benchmark results easier to use by providing a mechanism to condense the test results into a single number for the purpose of ranking systems.

We agree that the benchmark report currently includes far too many numbers; we will talk in a moment about our plans to condense the OO7 Benchmark to a more manageable size. However, we do not anticipate *ever* reducing the benchmark to a single number. This is not due to bashfulness or fear of hurting people's feelings; rather, it is a result of (1) our decision to favor a comprehensive benchmark rather than a concise one but narrow one, and (2) our belief that for a knowledgeable customer, a multiple number benchmark is far more useful than any single number benchmark could be. That is, to use the OO7 results, we recommend that you determine the main performance demands of your application and then look at the OODBMSs' performance on the portion of OO7 that most closely matches those demands.

To illustrate some of the difficulties with a single number condensation of the OO7 results, here are three ways that have been suggested (e.g., in trade journal articles covering OO7 or marketing literature from OODBMS companies) for producing a single OO7 number.

1. **Number of first places on benchmark tests.**

   In the benchmark report we present 105 timing results (tests) from each system.[2] To produce this one-number performance metric, one simply counts the number of times that each system had the fastest time on one of the 105 tests.

   By this measure, the results as of this writing are:

   | Rank | System | Score |
   |---|---|---|
   | 1 | E/Exodus | 61 |
   | 2 | Versant | 22 |
   | 3 | Ontos | 19 |
   | 4 | Objectivity | 3 |

2. **Weighted ranking of places.**

   To produce this alternative one-number OO7 performance metric, one gives each system one point for a first place finish on a test, two points for a second place finish, three points for third place, and four points for fourth place. These numbers are then added up for each system, so a lower overall number is better.

   By this measure, the results are:

   | Rank | System | Score |
   |---|---|---|
   | 1 | E/Exodus | 173 |
   | 2 | Ontos | 264 |
   | 3 | Versant | 283 |
   | 4 | Objectivity | 330 |

3. **Geometric mean.**

   Yet another possible approach, which weights each number as being equally important (and therefore presumes that all key performance areas have been equally well covered), is to use the geometric mean of the results as the one-number metric — taking the 105th root of the product of the 105 individual test results. This is the approach taken by the TPC in their new TPC-D benchmark.

   By this measure the results are:

   | Rank | System | Score |
   |---|---|---|
   | 1 | E/Exodus | 14.19 |
   | 2 | Objectivity | 33.88 |
   | 3 | Versant | 33.94 |

   (We couldn't place Ontos in this ranking because as of this writing we do not have numbers for the medium-9 OO7 database results for Ontos.)

While we don't ever want a single-number benchmark, OO7 certainly errs on the side of generating too many numbers. This is because benchmarking is a learning process, and only through iterative refinement can one end up with a minimal yet somehow sufficient set of benchmark operations. We plan to fix this problem in OO7 by further reducing the set of tests involved in running the benchmark. This will be done by requiring one or a few numbers for each of a few interesting categories of workloads, e.g., hot traversals (how fast can an application program traverse in-memory data?), cold

[2] This is not quite as outrageous as it might seem, as the OO7 technical report includes a number of graphs. The total of 105 numbers comes from counting every point on every graph.

sparse and dense traversals (how fast can an application program traverse data not yet in memory?), one or two update traversals, a smaller set of queries, etc. Some of the current OO7 parameter variations (e.g., the 3/6/9 variation for the atomic parts graph, and the some/all/repeated update variation) have come to show little or no new information as systems have matured and their early performance bugs have disappeared. These variations can now be eliminated.

### 3.6 Application Coverage

One final issue that is becoming more and more worthy of discussion is the question of what sort of application(s) an OODBMS benchmark like OO7 should be loosely based on. When we began this work, the answer was quite clear, and as a result the current single-user OO7 Benchmark has a strong CAD/CAM flavor. This reflects the fact that CAD-type applications were the initial target application for most OODBMSs. However, this is now changing: we are seeing OODB systems applied in diverse application areas ranging from CAD to telecommunications to finance, with some estimates putting the CAD portion of the OODBMS market in the 30-40% range and dropping. Clearly, emerging OODBMS application domains may have workload characteristics that are not well covered by OO7. We are actively seeking feedback about such applications to help us in future revisions of the benchmark, especially with respect to their multi-user characteristics. Such information would be very helpful for our current multi-user OO7 Benchmark effort, to which we now turn our attention.

## 4 Towards a Multi-User OO7

As an initial step towards filling the information gap that the introduction mentioned in the area of multi-user OODBMS performance characteristics, we are currently developing a multi-user OO7 OODBMS benchmark. There are five systems currently involved in this effort: E/Exodus [CDF+86, RCS93], O2 [Deu91], Objectivity [Obj92], Ontos [Ont92], and Versant [Ver92].

Our experience so far is that designing a multi-user benchmark is a much more difficult problem than designing a single-user benchmark. This is partially due to the fact that the number of dimensions along which the workload can vary is greater in the multi-user case, and partially because multi-user workloads inherently involve complex interactions of multiple concurrent activities. Also, if there is no agreement about what constitutes the canonical single-user OODBMS workload, there is even less agreement in the multi-user arena. For this reason we regard coming up with a monolithic workload that generates a single-number system evaluation (*a la* TPC-A or TPC-B) as nothing short of hopeless. Our response to date has been to develop a fully parameterized workload that is made up of primitives that can be combined to generate a range of workloads with a wide variety of different characteristics. In this sense the multi-user OO7 Benchmark, in its current form, is really a customizable benchmark generator that we hope will be useful to sophisticated consumers of OODBMSs.

In the remainder of this paper we give a brief description of our ongoing work on the multi-user database and its workloads. Our hope is that reading and hearing about this work will lead some of the more informed members of the OODBMS community, like vendors and serious application developers, to provide us with feedback based on this description. Such feedback would enable us to improve our design, helping us to produce a more relevant and therefore more useful benchmarking tool for the OODBMS community to share.

### 4.1 The Multi-User OO7 Database

The multi-user OO7 database is a scalable extension of the single-user OO7 database described in Section 2.1. As mentioned there, scalability for multi-user benchmarking was a goal even in the initial OO7 database design. However, the process of designing the current multi-user OO7 database led us to make some minor changes to the initial single-user OO7 Benchmark database, and to extend it slightly, as we did not correctly anticipate certain sharing and data contention issues that quickly became clear when we began running some initial multi-user experiments.

In the initial OO7 Benchmark database, as discussed in Section 2.1, each base assembly in a given module had associations with two kinds of composite parts — private and shared. Both were references into the design library, with private composite part references referring only to the composite parts that were privately associated with the given module; the shared composite part references were randomly assigned to point anywhere within the entire design library (i.e., to private parts of any module). The unfortunate aspect of this design was that there were no truly private composite parts; e.g., it provided no way to have a client transaction read from its private composite parts and update shared composite parts without interfering with the private reads of other clients.

We have therefore abandoned this design, removing the shared composite part associations from the definition of the benchmark's base assembly object type. Instead, for the multi-user benchmark we have added one additional module, the "shared" module, to the database. One last change, alluded to earlier, is that we also eliminated the 3/6/9 connection count variation scheme from the single-user benchmark; for the multi-user OO7 database, the number of outgoing connections per atomic part is simply fixed at three.

For the multi-user OO7 Benchmark, an important issue that we faced was how to scale the database with the workload. As clients are added, should the database remain the same, potentially leading to more and more contention? Or should the database grow with each additional client, the idea being that each additional client represents a designer who will be working on an additional piece of a CAD design? We chose the latter approach, which implies trying to keep contention more or less constant as the number of clients scales up. This, in turn, requires carefully scaling the database in terms of both its private and shared modules.

To scale the benchmark database in proportion to the number of clients, we have one private module (as described in Section 2.1) in the database for each client in both the small and medium databases. In addition, we grow the shared module in proportion to the number of clients. Roughly speaking, the shared module, which has the same depth as the private modules in the benchmark database, can be thought of as a "mega-module" consisting of as many submodules as there are clients in the database. Each submodule of the shared module therefore has one less level than than a private module, with the submodules being hooked together via the root complex assembly of the shared module. The shared module brings to the design library its own additional set of composite objects; each submodule of the shared module adds *NumSharCompPerClient* (currently set to 200) composite objects to the design library, and these are the composite objects that are referenced by the base assemblies of that submodule.

## 4.2 The Multiuser OO7 Workload

As mentioned at the beginning of this section, our approach to multi-user workload generation has been to define a parameterized workload that can be used to create workloads with a variety of different characteristics. By doing so, we are able to thoroughly explore the space of multi-user OODBMS performance. The multi-user OO7 workload consists of a set of clients, each running a series of (parameterized) traversal transactions that are themselves made up of primitive operations. Different clients can be told to run traversal transactions with different parameters, thus allowing a wide range of different workloads to be generated (e.g., with varying degrees of inter-client data sharing).

```
beginTransaction;
  for RepeatCount do

     if this is a shared transaction
         start at the root of the assembly
         hierarchy of the shared module;
     else
         start at the root of the assembly
         hierarchy of module k;

     Follow a single random path down the
     hierarchy to a base assembly;

     From the base assembly, perform some
     operation on a composite part;

     Sleep(SleepTime);

  end;
endTransaction;
```

Figure 3: Generic multi-user OO7 transaction.

### 4.2.1 Traversal Transactions

Each client in the multi-user OO7 Benchmark has a distinct client number. Pseudo-code for a generic multi-user OO7 traversal transaction for client $k$ is shown in Figure 3. The transaction repeatedly chooses a single path through the assembly hierarchy, performing some operation on a single composite part that it reaches via the chosen path. In particular, if the parameter `RepeatCount` is set to one, the transaction will visit only one composite part.

Each time through the loop, there are two possibilities for what each assembly hierarchy traversal can do when it visits a composite part:

1. Do a *read-only* depth-first search traversal of the atomic part subgraph associated with that composite part.

2. Do a *read-write* traversal. Specifically, do a depth-first search of the associated atomic part subgraph that swaps the `X` and `Y` coordinates of each atomic part as it is visited.

We call these operations on composite parts *basic operations*. Since each of these (read-only or read-write) operations can be done beginning with a traversal of either the client's private module or the (globally) shared module, there are a total of four possible basic operations. Given this description of the basic operations, we can now describe the parameters of the traversal transactions used to generate the OO7 multi-user workload. We consider each parameter below.

- Percentage of Each Basic Operation.

  The percentages of each of the four basic operations is best described by a vector. For example, (100, 0, 0, 0) specifies a workload in which each transaction contains 100% read-only operations on its private module. Similarly, (80, 10, 10, 0) specifies a workload in which each transaction contains 80% read-only operations on its private module, 10% read-only operations on the shared module, and 10% update operations on its private module. In more detail, these percentages are interpreted to be probabilities for each operation: each time through the loop, when the transaction reaches a composite part, it "flips" a biased coin (i.e., generates a random number) to decide which kind of basic operation to perform on it. On the average, then, if the RepeatCount parameter of Figure 3 is set to 100, a transaction drawn from the (80, 10, 10, 0) vector will contain about 80 private read-only operations, 10 shared read-only operations, and 10 private update operations.

- Repeat Count.

  By varying the RepeatCount parameter, which determines how many basic operations a transaction contains, it is possible to generate transactions of arbitrary length — ranging from short, traditional TP-ish transactions to longer, CAD-like transactions.

- Sleep Time.

  The SleepTime parameter controls the "intensity" of the transactions. If this parameter is set to zero, the transaction is never idle (unless it is waiting for the OODBMS running the transaction); this is perhaps suggestive of a CAD program such as a design rule checker. By specifying a longer sleep time, one can model a transaction of an interactive session that involves think times between the database operations.

The runtime arguments to a given benchmarking run specify all three of these parameters — the percentage vector for basic operations, RepeatCount, and SleepTime — in the form of command line arguments.

### 4.2.2 Multiuser Workload Generation

The parameters just described (operation percentage vector, operation repeat count, and sleep time) make it possible to generate a wide variety of multi-user transaction workloads. In addition, a final very important parameter of the multi-user OO7 Benchmark is the number of clients, as mentioned in the preceding multi-user benchmark database description. By varying the number of client workstations, and by varying the parameter set given to each client, many different workloads can be experimented with.

Of particular interest are the various *data sharing patterns* that can be generated from the framework that the OO7 multi-user benchmark parameters provide. One class of workload that can be generated are symmetric workloads, where all clients behave similarly with respect to accessing their "private" data and the system-wide shared data. Such workloads can be generated by running every client with the same set of input parameters. Potentially interesting examples include the private read-only workload mentioned above, or the largely private (80, 10, 10, 0) read-mainly workload. Similarly, if an operation vector of (0, 0, 100, 0) is given to each client, a private read-write workload can be generated. In addition, the transaction length parameter allows the different sharing patterns to be applied to either short transactions or long transactions, and the sleep time parameter allows both compute-intensive and pause-intensive transactions to be explored.

In addition to symmetric workloads, it is also possible to use the OO7 multi-user benchmark parameters to generate interesting asymmetric workloads. One such workload might be a producer/consumer workload, where one client generates information that others read (such as stock price quotations). By running a "producer" client with an operation vector of (0,0,0,100), one could have it be a provider of shared information. The rest of the clients, the "consumers", could be run with operation vectors of (0, 100, 0, 0) so that each reads the shared information provided by the producer's updates. A somewhat less contentious version of the consumers could be generated by running them instead with an operation vector like (50, 50, 0, 0), causing them to spend only half of their time reading from the shared module (with the other half being spent on private reads).

#### 4.2.3 Other Operations Under Consideration

In addition to the family of multi-user workloads that can be generated using the parameterized transactions described up to this point, we are currently considering (and experimenting with) several additional workload variations. These additional variations are essentially a first attempt to make the OO7 Benchmark somewhat more broad than just being a CAD-like benchmark. We briefly mention some of our additional (in progress!) ideas along those lines here.

**Finer Granularity Traversals**

Each time one of the aforementioned traversal transactions visits a composite part, it traverses the entire atomic part subgraph that is associated with the composite part. Each such subgraph traversal touches 80 objects in the small database (20 atomic parts plus 60 connection objects) and 800 objects in the medium database (200 atomic parts plus 600 connection objects.) While this seems fine for CAD-like transactions, having transactions access such a large number of objects is too heavy-weight for modeling some varieties of transactions. To address this problem, in addition to the traversals described previously, we intend to add another type of transaction — very similar in structure to the original multi-user traversal transaction — that stops its traversal and performs its basic operations at the base assembly level. Our intent is to use this workload to characterize applications that have finer-grained interactions between client transactions.

**Set Update Operations**

One of the few items of feedback that we have received regarding multi-user workloads is that concurrent set update (e.g., insert and delete) operations are important for some types of OODBMS applications. To explore performance issues raised by multi-user set updates, we have added an associated set object to every module (both shared and private). This new set will contain primitive objects that are distinct from the other objects in the benchmark database. We are currently exploring these issues by testing the following operations:

1. Generate.

   Every client generates some number of new objects and inserts them into the set of its private module. While there is no explicit contention among the clients, depending upon how the system is implemented, clients are likely to contend for both logical and physical resources (e.g., OIDs and database disk bandwidth) at the server.

2. Migrate.

   Every client migrates the objects in its private set into the set of the shared module. Here, clients are contending for this shared set due to concurrent insert operations.

3. Migrate back.

   Every client iterates through the shared set, moving the objects that it put there back to its own private set. (This operation has caused terrible data contention in our initial experiments, so it may need to be modified.)

4. Destroy.

   Every client destroys the objects in its private set and deletes them from the set.

**Your Favorite Operation Goes Here!**

As we have mentioned already, we are very interested in hearing from OODBMS vendors and customers regarding the nature of their multi-user workloads. If we are successful at soliciting such information, perhaps in the form of critical comments on the preliminary multi-user OO7 Benchmark design that we have described, we will likely extend (and/or modify) our design further in the future.

## 5 Conclusion

This paper has reported on the current status of the OO7 OODBMS benchmarking effort at the University of Wisconsin. The first half of the paper was devoted to a review and retrospective evaluation of the single-user OO7 Benchmark. We discussed a number of issues that arose concerning the design and administration of the benchmark, explaining what we did, why we did it, and what we might do differently the next time around. We then described our in-progress work on multi-user OODBMS benchmarking, detailing a set of proposed changes to the OO7 database and an associated multi-user workload family for evaluating OODBMS performance. A major goal of the latter part of the paper is to publically solicit feedback from vendors and users of commercial OODBMS technology regarding, in their opinions, what we're doing right and what we might be missing in our proposed multi-user benchmark. We are now in the process of running a variety of multi-user workloads against the multi-user OO7 database; our parameterized workload approach has already paid off, allowing us to explore the OODBMS performance space on five systems without a continual modify/compile/debug cycle between every data point.

We are hoping that, through extensive experimentation and feedback from the OODBMS community, we will be able to develop a set of workloads that strikes a reasonable balance between simplicity and completeness. However, all of the benchmark implementations will be configurable by varying the run-time parameters of the benchmark. Moreover, as our work progresses, we intend to make our benchmark implementations freely available so that others can independently and efficiently conduct their own tests on workloads that interest them. We expect this to be quite useful, as the results from the different data points we have explored so far in the multi-user workload space indicate that the benchmark will indeed find significant variations in the multi-user performance characteristics of the systems being tested. (We intend to publish a full set of results once we are satisfied that we have a solid multi-user benchmark design.)

# References


[And90] T. Anderson et al. The HyperModel Benchmark. In *Proceedings of the EDBT Conference*, Venice, Italy, March 1990.

[Cat94] R. Cattell. *The Object Database Standard: ODMG-93 (Release 1.1)*. Morgan Kaufmann, San Mateo, CA, 1994.

[CDF+86] Michael J. Carey, David J. Dewitt, Daniel Frank, Goetz Graefe, M. Muralikrishna, Joel E. Richardson, and Eugene J. Shekita. The architecture of the EXODUS Extensible DBMS. In *Proceedings of the Twelfth International Conference on Very Large Data Bases*, pages 52–65, 1986.

[CDN93] Michael J. Carey, David J. DeWitt, and Jeffrey F. Naughton. The OO7 benchmark. In *Proceedings of the 1993 ACM-SIGMOD Conference on the Management of Data*, Washington D.C., May 1993.

[CS92] R. Cattell and J. Skeen. Object operations benchmark. *ACM Transactions on Database Systems*, 17(1), March 1992.

[DD88] J. Duhl and C. Damon. A performance comparison of object and relational databases using the sun benchmark. In *Proceedings of the ACM OOPSLA Conference*, San Diego, California, September 1988.

[Deu91] O. Deux et al. The $O_2$ system. *Communications of the ACM*, 34(10), October 1991.

[Gra93] Jim Gray. *The Benchmark Handbook*. Morgan Kaufmann, San Mateo, CA, 1993.

[Obj92] Objectivity, Inc. Objectivity reference manual. 1992.

[Ont92] Ontos, Inc. Ontos reference manual. 1992.

[RCS93] Joel E. Richardson, Michael J. Carey, and Daniel T. Schuh. The design of the E programming language. *ACM Transactions on Programming Languages and Systems*, 15(3), July 1993.

[RKC87] W. Rubenstein, M. Kubicar, and R. Cattell. Benchmarking simple database operations. In *Proceedings of the ACM SIGMOD Conference*, San Francisco, California, May 1987.

[TPC94] TPC. TPC Benchmark$^{TM}$ D (Decision Support). Working draft 6.5, Transaction Processing Performance Council, February 1994.

[Ver92] Versant, Inc. Versant reference manual. 1992.