

Writing Sample For Jeffrey

Hal Stern on three-tiered client/server, see p.50

# Advanced Systems

Client/Server Products for Unix Professionals

\$3.95 Canada \$4.95  
April 1990

IDG

The heavyweight

champ of the

ODBMS market

Advanced Systems

**BEST**

PRODUCT  
1990

## ObjectStore Delivers

### Reviews

OpenVision manages  
the multitudes

*First Impressions:*

Ami Pro vs. WordPerfect

HP's Aptrex X terminal

### Columns

Persistently C++

Passion for Perl

Interviewing goofs

PC TCP/IP

The network is the  
data center

REPRINT

# Advanced Systems

R e p r i n t

## ObjectStore delivers

By Jeffrey Blake

*The reigning champ  
of the burgeoning  
ODBMS market  
boasts performance  
and scalability.*

Users whose relational databases don't adequately handle their complex datatypes, such as audio and video images, often turn to object-oriented database-management systems (ODBMS) for relief. Once a user decides to invest in an ODBMS, the challenge is to choose the best.

There are three kinds of ODBMS: object-relational databases; "pure" or semantic databases; and C++ persistent databases. Each has its advantages. The object-relational model (exemplified by Hewlett-Packard's OpenODB and UniSQL's UniSQL/M) combines both object and relational technologies. Semantic databases (including ADB Inc.'s Matisse, Servio's Gemstone, and Itasca Systems' Distributed ODBMS) boast a "pure" object approach and the ability to store and execute methods inside the database. C++ persistent ODBMS (such as Object Design's ObjectStore, Ontos Inc.'s Ontos DB, Objectivity Inc.'s Ob-

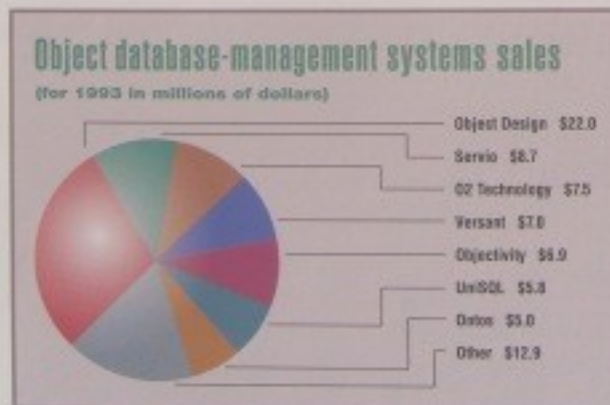
jectivity/DB, and Versant Object Technology's Versant ODBMS) outpace alternatives in terms of speed, and have gained a strong foothold as favorites in the market, in part due to their close integration with C and C++.

Object Design's ObjectStore is the reigning heavyweight champion of the burgeoning ODBMS market, commanding about 30 percent of all ODBMS sales (see the pie chart **Object database-management system sales**) due to savvy technological partnering, a strong technical implementation, and top-notch support.

### Who uses ODBMS?

ODBMS are ideal for complex applications that rely on nonrecord-oriented data, such as diagrams and photos or time-series data. These applications might need to handle 2-D and 3-D graphics, geometry, topology, text, bit-mapped images, voice, and video data. The applications capitalize on using object-oriented methodologies that model the real world. Applications especially suited for ODBMS include simulation and modeling, because the object-oriented paradigm and ODBMS provide natural capabilities for data modeling of real-world objects, and simulation objects can be stored directly into the database without translation. Mechanical and electrical CAD fit too because drawings can be stored and retrieved easily. These disciplines use complex datatypes as images, mechanical drawings, and documents not easily stored in a relational database. Also, when modeling the real world the object-oriented paradigm is well suited. ODBMS can store objects without translation, making the design and development simpler.

The number one concern of most ODBMS users is performance. The second concern is scalability. We evaluated ObjectStore by ex-



Source: International Data Corp., 1994.

**About the author:** Jeffrey Blake (jeffrey.blake@advacod.com) is a software engineer specializing in Unix and object-oriented supercomputer application development. He has more than 15 years of software development experience. In the last five years, he has focused on object-oriented Unix applications and is currently working as a member of the Advanced Simulations Concept Team at a Navy Research and Development laboratory in San Diego.



analyzing its features and then wrote some small programs to test scalability. Since we were not comparing ObjectStore with other ODBMS, we focused on discovering the performance features and how it would scale up to handle large applications.

We installed ObjectStore in 15 minutes on a Sun SPARCstation 330 (a SPARCstation 2-class machine) running SunOS 4.1.3, X11R5, and OSF/Motif with 24 megabytes of RAM and 400 megabytes of free disk space. ObjectStore requires at least 32 megabytes of free disk space and 8 megabytes of swap space on each client, and it uses 16 megabytes in /tmp to store client cache, which can be redirected elsewhere. Of course, you'll also need sufficient disk space to store ObjectStore databases.

#### How does it work?

We stepped through numerous tutorials, which educated us about ObjectStore's robust, fast, and scalable characteristics. After reading ObjectStore's documentation, we still didn't understand how ObjectStore makes objects persistent—that is, how it creates objects that survive beyond the lifetime of the process that creates them. This became clear only after we crafted some persistent objects.

ODBMS have different methodologies for implementing persistence of objects. ObjectStore stores persistent data on disk in databases. Each database is made up of segments, which are variable-sized regions of memory that can be used as the unit of transfer from persistent storage to program memory. Each segment, in turn, is made up of pages that also can be used in a transfer.

ObjectStore employs a powerful strategy that relies on its patented Virtual Memory Mapping Architecture (VMMA). VMMA pro-

vides a single view of memory, instead of dividing it into program memory and database memory. With ObjectStore, C and C++ programs have direct, transparent access to the database; ObjectStore data is handled by programs in the same way as regular transient data. This feature is called "persistence independent of type." Other ODBMS use persistent classes from which each persistent object must inherit—an indirect solution that creates overhead for each object, making the application larger and less flexible.

Moreover, references to persistent data already encached (in memory and ready to be loaded into persistent address space) on the client host are handled at the same speed as virtual-memory references. ObjectStore won't run without a virtual-memory operating system. It capitalizes on the virtual-memory management of the operating system to enhance performance. ObjectStore detects any reference in a running program to persistent data and automatically transfers the page containing the referenced data to the application's cache. Then the page containing the referenced data is mapped into virtual memory. At times the referenced data is already in the cache and simply requires the virtual-

*continues*

## Reviews

continued

memory mapping; however, at other times the page is already mapped into virtual memory and ready for access. Once persistent data has been mapped into virtual memory, users can access it as fast as they access transient data. Other ODBMS use "handles" to objects that first must be translated before retrieving the object. This indirect route can consist of many extra instructions. It simply takes fewer instructions to access an object under ObjectStore. It doesn't depend on the amount of RAM. Obviously, more RAM is better for almost any large application. More RAM means less fetching of data.

ObjectStore keeps track of objects in the database by using database roots, which provide a way to give an object a persistent name and allow objects to serve as initial entry points. When an object has a persistent name, any process can look up the object to retrieve it. Once you have retrieved an object you can retrieve any object related to it via navigation (using data member pointers) or a query. Typically, each database has a small number of entry-point objects (named objects), each of which allows access to a collection of related objects. Persistent objects are created in a particular database. The database name is used in the overloaded new constructor

for the persistent object. Of course, before you perform any of these actions, a database must be created, initialized, and opened.

If you are not familiar with database operations, some of this may seem foreign. After some time we found the setup code simple, and often duplicated this code when developing new applications. To obtain example code illustrating how persistent objects are created and stored in the database, send a blank e-mail message to [objectstore@adix.com](mailto:objectstore@adix.com).

### Features

ObjectStore boasts many features: a

## Object, relational, and object-relational choices

**T**he competitive enterprise faces some fundamentally different choices when deciding whether to stay with relational technology or move to an object data model.

RDBMS provide mature capabilities to reliably conduct high-volume transactions on a limited set of datatypes—essentially numbers and strings. Data are populated into rows and columns in a table, and tables are associated with one another by joining fields that match in the two tables.

Due to the evolution of a standard structured query language (SQL) and the pervasive use of RDBMS, this is a reliable and well-understood process. RDBMS vendors have been able to produce significant breakthroughs in transaction management, database administration tools, application development tools, data replication, and data distribution, because fundamental storage and access methods were generally agreed upon.

ODBMS were created initially to give object-oriented programmers a place to permanently store the objects that their OO programs manipulated. They emerged to handle an increasingly important task where RDBMS products generally failed—the storage and manipulation of large unstructured datatypes, such as graphics, sound, and compound documents.

Proponents of the object data model point out that the world does not fit neatly into table structures, and a natural model of today's business processes does not either. A lot of relationships are hierarchical (like the parts to an automobile) or highly variable and complex (like employee/manager/work-team relationships in today's flatter organizations).

ODBMS give the enterprise the ability to define arbitrary datatypes and manage those datatypes within the database. Classes provide a more natural model than tables. Methods, with polymorphism, deal with real-world variability better than stored procedures.

Perhaps most importantly, the object data model enables applications to navigate through complex relationships that are many levels deep, without having to create joins

between relational tables. (Table joins are very resource-intensive computing operations.)

With the burgeoning importance of multimedia, ODBMS solutions have matured, but few offer the sophisticated management and administrative tools traditional businesses demand. There is a growing trend to reverse engineer many relational capabilities (such as SQL support) back into ODBMS, but these efforts will take time.

Meanwhile, RDBMS vendors look to extend their products' object capabilities, but most now realize that major advances in object support will only come through a substantial rewrite of RDBMS products.

A third data model—the object-relational model—is beginning to generate a great deal of interest. UniSQL (Austin, TX) and Ilustra Information Technologies (Oakland, CA) offer slightly divergent views of the object-relational data model, but both are shipping products that fully support most of the capabilities of pure ODBMS and RDBMS. They appear to do a good job providing performance and flexibility in dealing with unstructured data, while continuing to provide SQL support, fast associative retrieval, and other crucial RDBMS administrative capabilities. Both vendors state that fully supporting an object-relational data model requires the database kernel to be written from scratch with this goal clearly in focus.

Yet another group of products approximate the object-relational data model by layering third-party packages with object capabilities upon traditional RDBMS. Such products include OpenODB from Hewlett-Packard (Palo Alto, CA), Enterprise Objects Framework from NeXT Computer (Redwood City, CA), and Persistence from Persistence Software (San Mateo, CA). These products are designed to create a layered object environment where data appears in class structures, to sit upon existing RDBMS datasets. They enhance functional abilities, but must cope with the reality that data remains mapped to tables in the underlying relational data store.—Barry D. Bowen



## Reviews

large class library, indexes, cursors, exception handling, relationship support, variable locking, deadlock detection, and an array of system administration tools—all the things necessary for the development of mature object-database applications.

ObjectStore's class library contains 20 different categories. One highlight is an object class called collections, which group together other objects. Collections include subclasses of sets, bags, lists, and arrays. We used collections to contain sets of related objects that were accessed by setting a root head (an object that stores information about a collection in the database) to the top of the collection. Collections are essential to efficient object traversal and the use of queries. They add built-in capability to an application. They can change their behavior and representation at runtime, be queried and indexed, be used to keep track of extents (instantiations of a particular class), and best of all, be used transiently or persistently.

(Make note that ObjectStore recently has been integrated with Iona Technologies' CORBA-conformant object request broker [ORB] and Expertsoft Corp.'s XShell distributed object management environment.)

### Queries

Contrary to popular belief, ODBMS do support queries. ObjectStore has a healthy query capability built into the collections class that allows the programmer to select target objects or values found in objects. ObjectStore doesn't have a traditional SQL interface with the capability to do ad hoc queries. Instead, queries are treated as ordinary expressions within the language. And, like RDBMS, ObjectStore provides indexes to permit more efficient implementations. Also, a built-in query optimizer selects the most efficient way to execute a query. Learning to specify queries over your object collections is a bit tricky at first. We usually specified that the results of a query be placed into a collection and then iterated through the resulting collection to discover the query results.

ObjectStore's relationships feature helps maintain referential integrity between two objects. Relationships are useful in modeling complex objects that have many interconnected objects. The relationships mechanism enforces in-

tegrity between two or more objects. By declaring two objects to be inverse of each other, integrity can be automatically enforced as an update dependency. This means when one object is deleted, the other is automatically deleted.

For now, ObjectStore's implementation looks good. However, when C++ exception-handling catches up and has its own exception facility, ObjectStore's implementation may be incompatible.

ObjectStore uses a flexible and smart locking policy that allows locking of either pages or segments. A segment can be as large as 4 gigabytes or as small as 512 bytes. Pages are usually 4 kilobytes. The developer can pick and choose what gets locked. According to Object Design, no object-level locking is provided because this has proven to be inefficient and overkill. Applications typically manipulate data in groups; to lock on smaller pieces causes too much paging.

ObjectStore provides automatic deadlock detection. To resolve the deadlock, a victim has to be selected. ObjectStore stands out in its ability to let the developer select the deadlock policy. (Other ODBMS hardcode the policy into the system.) In addition to the typical database rollback capability, ObjectStore provides archive logging that allows for roll-forward recovery. This important feature allows old databases to be updated to a new configuration. Archive logging is a feature we haven't seen in other database systems.

### Server

ObjectStore's intelligent and efficient database server stands out. Its implementation makes use of the host operating system's memory-management system. One of the biggest costs in a database is the time it takes for a client to communicate with the server. ObjectStore's state-of-the-art server is multi-threaded, with a thread provided for each client. Starting a thread is cheaper than starting a process and faster than context-switching between processes.

In the next release, ObjectStore promises asynchronous I/O in its server, and thus will be able to service more clients simultaneously. Since the server is page-based, it doesn't fight the memory management of the operating system. Additionally, the server employs resourceful caching and locking policies

continues

## Advanced Systems Test Strip

### ObjectStore 3.1

Object Design Inc.  
25 Mill Rd.  
Syracuse, MA 01802  
800-862-0620  
617-674-6000  
617-674-6019 fax  
tpodi.com, info@odi.com



**Pricing** Floating license starts at \$3,500.  
**Platforms** Unix (including Solaris 1 and 2, HP-UX, AIX, IRIX, OS/390), OpenVMS, Windows 3.1, Windows NT, OS/2.

**Summary** This leading ODBMS scores big with strong, scalable performance, robust features, and impressive documentation and support.



**How Test Strip works:** Products are compared with others in their class, and we judge different products on different categories as needed. **Features** evaluate capacity, expandability, reliability, and availability. **Documentation** summarizes the quality of each product's printed and on-line documents. We give high marks for excellent, accurate, and indexed paper manuals accompanied with on-line pages and high-caliber hyperlinked on-line help. **Administration** (not applicable in all reviews) gives credit for tools that aid in system administration. **Support** (where applicable) summarizes installation, warranty, and technical support policies including hours of availability. **Performance** (for hardware) summarizes tests of various comparative metrics. **Performance** (for software) summarizes how well the product accomplishes its intended tasks. **Interface** (where applicable) rates whether the product's user interface is intuitive and straightforward. **Compatibility** (where applicable) includes compliance with de facto standards. Weighting is based on reader surveys and expert knowledge. Total of dimensions is divided by 50 and rounded to one decimal place to yield an overall rating on a scale from 0 to 10. Adjust the weighting to customize the Test Strip to your own needs.

## Reviews

continued

that result in minimal data caching overhead; data is cached across transaction boundaries. If data is still valid, the server doesn't have to refetch the data and thus minimizes communication between client and server. Also, the server's minimal locking overhead and adjustable locking granularity allow the client to give up a lock only if needed. Other ODEMS systems are not as efficient in this area.

ObjectStore has a collection of about 30 different utilities for copying, exporting, listing, and otherwise manipulating databases. We found some gems, including *essize*, which report database size and a variety of other useful pieces of information, including the contents of all segments in a database.

Another powerful utility is *asterifydb*, which verifies that all pointers in the database are valid. This tool particularly helps track down "loose" (not tied down to a particular place so they can be retrieved) pointers. A powerful, easy-to-use graphical tool called *astbrowser* lets users peruse a database by examining schema and objects in the database. ObjectStore has about 20 powerful system administration utilities that perform on-line backup, cleanup, shut down and restoration of a database, and display statistics on all clients currently connected, among other things.

### Documentation & support

ObjectStore comes with a set of six software manuals. The 750-page reference guide is a comprehensive encyclopedia that describes the API for all functionality provided by the ObjectStore ODIIMS. It includes sections covering classes, global functions, macros, the Data Definition and Manipulation Language (DDL—a convenient notation alternative to C++), the C Library Interface, and exceptions. The tutorial describes a series of sample programs designed to present key concepts used to write and build ObjectStore applications.

The two platform guides discuss installation requirements, system administration, release notes, and how to contact technical support.

The DML and Library Interface user guides serve as "how-to" books that present a product overview covering fundamental concepts and numerous chapters describing the APIs for persistence, transactions, collections, queries and indexes, data integrity, version management, schema evolution, dynamic schema access, and data compaction.

Although the sheer volume of information and the complexities of getting started may be daunting, ObjectStore comes with one of the most impressive sets of documentation we have seen. The introductions are wonderful and informative and the text overflows with descriptions, pictures, and examples.

Additionally, ObjectStore's technical

support impressed us with top-notch technical knowledge, and they offered to help us get started by spending time discussing our design concerns. Phone and e-mail support ([support@odi.com](mailto:support@odi.com)) is offered from 8 a.m. to 6 p.m., EST. When using e-mail support, Object Design will promptly send back a receipt including a reference number. It makes you feel that they won't forget you.

Extended support contracts and classes also are available. We attended the one on C++ programming with ObjectStore, in which competent instructors presented the basics. These classes may be just what the doctor ordered to get over the ObjectStore hump.

### Ракет и punch

ObjectStore delivers performance with a punch and can successfully scale up to large applications without performance degradation. ObjectStore owes its speed in large part to its unique Virtual Memory Management Architecture (VMM), which allows objects to be loaded into memory in the same way they are represented in the application. There is no impedance mismatch, and its single-level storage allows objects to be accessed as fast as transient data. Other ODBMS use handles to objects that require an additional level of indirection. ObjectStore's persistent objects are accessed by a single-pointer dereference (accessing the contents of the variable to which a pointer points). ObjectStore's efficient server implementation provides smart caching, minimal per-object network overhead, and tunable fetching and locking policies.

After completing our scalability test (see the sidebar **Scalability benchmark test shows effect of clustering data**), we had no doubts about ObjectStore's ability to stand up to large applications. All databases degrade in performance when the server must constantly fetch data. ObjectStore's server controls and minimizes swapping. What convinced us that ObjectStore can "scale up" is its tunable fetching policy and effective clustering facility that enhances "locality of reference." Our scalability test demonstrates that careful clustering of the data produces dramatic speedups.

Beyond handling large applications, ObjectStore also effectively handles additional users because its server is page-based and multithreaded. This means

confirms

The ObjectStore browser lets users view the actual values of the code by simply selecting the value of the appropriate Document. In the example, the code corresponding to "[0] [Document \"0x86ca000\" is displayed.





## Reviews

### Scalability benchmark test shows effect of clustering data

**W**ith large databases, the key to good performance is effective clustering of the data. We set up a benchmark application to test ObjectStore's ability to effectively cluster objects in a database. We already know that ObjectStore was fast when the objects were already encashed due to "single-level storage," so our test focused on ObjectStore's ability to reduce the number of page faults.

Our test application used a simple document data model made up of a hierarchy of documents, chapters, and sections. A document can have many chapters, and each chapter can have many sections. The test measured the time it took to search for a text string inside a document database.

We built three different document databases to test three levels of clustering. The first level is default clustering, in which all documents created are placed into a single default segment in the database. This default high watermark strategy causes the segment to grow as each new document object is placed adjacent to the last one. In the end, this will lead to poor query performance since the search will have to examine large portions of this one big segment.

Using the second level of clustering, our program created a new segment in the database for each new document, thus allowing the query to search through less data than the first case. The third level refines this strategy to search through the smallest possible dataset. In this case we split off portions of the model not searched into separate segments.

The test consisted of entering a document name and a keyword to be found in the text of the document. The query works by first obtaining a pointer to the top of the documents collection, where all documents are stored in a list.

After the correct document is found, a keyword search begins. The keyword search iterates through each section of each chapter in the document. In the case of the default clustering, when the database is large enough, some amount of data will be transferred to the client program during the search. In the second and third cases, data transfer of other documents is avoided by ensuring that the documents and all associated data are on separate segments.

The same query was run on three test databases, each containing 60 documents made up of 25 chapters. Each chapter contains 10 1-kilobyte sections of text, with 4 keywords per section. Each keyword was selected at random from a predefined list of 25 keywords.

#### Query results

Database	Size in bytes	Search time (sec.)	Pages fetched
cluster1.db	16474112	5.65	63
cluster2.db	16839664	3.75	73
cluster3.db	17332864	2.88	16

Database size differences are due to some per-segment overhead.

The results show that cluster1's 5-second query takes less than a second in cluster3. This optimization does not require a fistful of extra code; the different allocations take place in a single line. The tuned application is prevented from paging by limiting the working set necessary to perform each search. And we know that when ObjectStore doesn't have to fetch new pages, it can quickly access objects with a single pointer dereference.

Effective clustering of data makes ObjectStore apps fast.

*continued*

the server uses an efficient data-fetching policy and controls new clients by adding threads instead of starting new processes. Multithreading means the cost of adding a client is low since it minimizes processing overhead.

Last but not least, since ObjectStore implements persistent objects independent of type (no persistent class to inherit from), there is very little persistence overhead in terms of RAM and disk space. Applications don't drag

around the extra space per object required by the persistence class. This makes applications smaller. When each object is smaller, less page faulting occurs, resulting in good performance.

ObjectStore's unique VMMA strategy, smart fetching policies, robust API, and performance tuning tools deliver good performance even with large applications. The ODBMS' strengths, combined with quality technical support and documentation and competitive pricing, indicate ObjectStore has earned its prominent market position. ■





