Choosing a Higher Performance Database

There is often a great deal of confusion in determining whether an object database management system (ODBMS) is right for you. Most of this confusion can be resolved by focusing on your application’s requirements. In those situations where the application requires the very clear advantages of an ODBMS, choosing which ODBMS becomes a continuation of the exercise of analyzing those application requirements.

We begin by recognizing that not all applications are ideally suited to ODBMS technology. We strongly recommend against using an ODBMS in those applications that are handled perfectly well with relational database management systems (RDBMS). Applications that are well suited to the tabular representation of data, with a low level of relationship complexity are best served through relational technology, since at the very highest level, RDBMS all manipulate rows and columns of data, using multiple-indexing and joins to associate tables. The selection criteria that distinguish between any two RDBMSs are then fairly well-defined; the ACID properties of databases are assumed, and product differentiators are identified in areas such as performance, scalability, dependability, and integration.

Unprecedented data volumes are driving businesses to look at alternatives to the traditional relational database technology that has served us well for over thirty years. Collectively, these alternatives have become known as “NoSQL databases” or better put as “Not only SQL database”. Or another way of looking at these data management requirements is “one size does not fit all”.

There are four main NoSQL data models, Key-Value stores such as Amazon’s Dynamo, and Voldemort; Big Table out of Google, and Cassandra; Document Databases such as CouchDB and MongoDB; and Graph Databases such as Neo4j and InfiniteGraph. These four models are addressing very specific data management problems of scalability and massively parallel processing, but they do not address the issues of complex data and relationships. One could also add Object Databases in to the NoSQL mix to address these issues of complex data and relationships.
Choosing a Higher Performance Database

The same criteria apply in selecting an ODBMS. However, additional criteria also need to be considered when selecting an object database, since data is no longer “rows and columns”.

The very root of ODBMS technology is to store complex objects and the complex relationships between them. People familiar with relational technology sometimes suggest that objects are simply rows in tables, and the data members in an object are like the columns that make up a row. While this model appears to hold up at first glance, further examination shows that objects have many other characteristics that distinguish them from being represented as rows of data members. Specifically, objects have methods and direct relationships to other objects. And, quite importantly, objects can exist independently, in both a logical and physical sense, from any object of the same class. Clearly, this is a departure from the tabular storage of information found in RDBMS, which group all items of the same type in a single location.

So how do you determine whether you need an ODBMS? We strongly recommend that you look at the selection of an ODBMS from the perspective of your application's requirements. Then prioritize those requirements and conduct a detailed investigation of how different ODBMSs meet the high priority ones. To help you in this process, we have developed a list of requirements and organized them into the following six categories:

- Reliability and Integrity
- Data Model Complexity
- Transaction Profile
- Distribution
- Performance
- Environment

We suggest you go through these categories and assign a priority to each requirement, then evaluate different products on how well they meet these needs. As examples, we have performed this exercise for four applications well suited to ODBMS:

- Network management
- Financial trading
- Product information management (PIM)
- Process control
Choosing a Higher Performance Database

The next section of this guide discusses the general requirements of these applications. The final section contains our prioritization of requirements for these applications and a discussion of how well different database features meet their high priority requirements.

Application Areas

Network Management

The term "network management" really describes all mission critical applications that coordinate the delivery of communication services across a computer network. These applications perform such tasks as device configuration, network path management, problem management, and network planning. These tasks require both real time performance and continuous operation, making high integrity, fast access to data, and high availability important requirements. In addition to these environmental requirements, network management applications require efficient handling of complex data. These applications typically involve three kinds of complex data structures:

Managed component configuration. These objects are the hardware, software, and virtual components of the network. They form a composition hierarchy whose underlying structure is fairly standardized. Here is an example of such a generic structure (arrows indicate a composition relationship from the structural parent to the structural child):

Network-> Subnetwork-> Switch-> Board-> Port-> Channel->Bandwidth

When the network encompasses multiple types of hardware, uses complex switching algorithms such as Asynchronous Transfer Mode (ATM), or uses sophisticated industry standards, this hierarchy may be asymmetric and of indeterminate depth.

Storing this configuration hierarchy in a database requires propagating operations from higher levels to lower levels. For instance, removing a switch from the network must also remove all of that switch's boards, all the boards' ports, etc. If an operator wanted to modify the configuration of a switch, the database needs to lock all of its subcomponents to prevent a different operator from modifying the parameters of a board within the switch. Also, nearly every operation performed by network management applications requires configuration data. Therefore, the database storing this information must provide high concurrency, fast navigation, and high availability.

Network connections. Whereas the managed components configuration corresponded to the physical layout of equipment, network connections correspond to how the equipment provides
Choosing a Higher Performance Database

communication. Each connection represents a chain of network switching devices that routes messages from sender to receiver. Connecting a phone call might involve the following chain of relationships:

Node <-> Link <-> Node <-> Link <-> Node <-> Link <-> Node

Each node corresponds to a port on a network device. Each link represents a slice of bandwidth set aside to service that connection. Since a node may participate in several different connections, the database has to manage a complex web of relationships. In order to route connections, balance network load, and diagnose problems, applications must have the ability to simultaneously move through this web in real time. To meet these needs, the database must provide very fast navigation time along relationships and high read concurrency.

Performance and billing data. Network management applications often track a lot of information over time to analyze performance and perform billing. In order to tune the network or plan for future needs, system administrators may need to execute queries that span petabytes of this data. The database must execute these queries in a reasonable time. Also, since critical automated systems update this data, the database must allow read/write concurrency. Otherwise network service would effectively freeze whenever an administrator made such a query.

Modeling these types of data requires between 20 to 200 classes. We can describe the configuration for any one component with a relatively small amount of data, so the individual object size is small. The configuration and connection data usually consumes 250-1,000 MB of space. Performance and billing data can grow to many petabytes.

Companies typically deploy network management applications on multiple servers and provide for a modest number of GUI clients for the human network administrators. Besides these administrators, hundreds of background daemons act as clients, performing the job of automatically monitoring and managing the network. In general, these daemons have very specific tasks: managing a set of network devices, computing a set of network routings, monitoring specific aspects of performance, etc. This clear delegation means that any particular object is by-and-large "owned" by a single process. So, while many processes may read an object, only the owner has the responsibility of updating it. Most companies deploy these applications on a single type of UNIX platform and, because of their historical association with Bell Laboratories, develop them using C++, though Java's importance is increasing rapidly.
Choosing a Higher Performance Database

Financial Trading

Financial trading refers to applications within the investment banking industry that support trading of financial instruments. People often classify these applications as Back Office or Front Office. Back office applications consummate financial deals, perform routine accounting, and calculate the fiscal impact of an entire day's transactions. They typically run overnight on servers running a RDBMS. Front office applications are the analytic and real-time tracking software that run on traders' workstations. Trading companies often use ODBMSs for front office applications, though they must obviously work with the back office systems. They provide tools for calculating the risk associated with specific instruments, complete portfolios, and the entire company. They also manage the information upon which traders base their investment decisions. This information may include real-time feeds such as stock quotes from Dow Jones online service and multimedia information such as scanned images of corporate reports. Finally, they allow traders to execute trades. Program trading applications can even execute trades without human intervention, based on pre-defined criteria. An overriding concern in all front office applications is response time, both because a few seconds delay could adversely impact the market price of an instrument and because traders' time is very valuable. Data complexity exacerbates these performance requirements. Data complexity stems from four sources:

Specialized instruments. Different trading instruments meet different investor needs. An investment banking firm that can quickly customize instruments to meet specific needs can gain an advantage over the competition. New instruments are usually similar to existing instruments, but have special features or rules. For example, stocks and bonds are basic instruments. Preferred stocks and "junk" bonds are specialized instruments. The object paradigm allows this type of modification through inheritance. Traders may need to "check-out" an instrument for modification and then "check-in" the instrument when it's ready.

Composite instruments. In addition to creating specialized instruments, traders can also meet investor needs by combining existing instruments. For example, a trader might want to combine the purchase of a foreign stock with a currency hedge so profits from an increase in stock price won't be vulnerable to currency fluctuations. These instruments may involve any combination of financial instruments, including other composite instruments. Actually, a portfolio is just like one large composite instrument. As with network management component hierarchies, propagation of operations down the instrument hierarchy is crucial. Tight integration with object-oriented programming languages (OOPs) is also important. A key need for composite instruments is "rolling" up the results for component instruments into the results for the composite. Most OOPs elegantly handle this task. However, if the
Choosing a Higher Performance Database

ODBMS does not integrate well with the particular language, achieving the results for the composite may require significant extra work.

**Derivatives.** Derivatives are instruments whose value is not determined by the underlying value of a set of securities, but rather by a link to an external financial value. For example, a mortgage derivative's value fluctuates with the average interest rate of home mortgages. To support derivatives, an ODBMS must allow relationships between arbitrary objects. Moreover, the database should allow traversing the relationship in either direction. To calculate the value of the mortgage derivative, we have to traverse the relationship from the derivative to the relevant interest rate. On the other hand, to calculate the company's exposure to fluctuations in the interest rate, we must traverse the relationship from the rate to all associated derivatives in all portfolios.

**Staged payment schedules.** Many deals involve staged payouts that change over time. A bond may pay off 5% on 1/95, 6% on 1/98, 7% on 1/01, etc. A principal problem with these types of data is that they can vary in length. Price histories that go back for an indeterminate number of years present a similar problem. Gracefully handling variable length data makes ODBMSs extremely valuable.

Describing a particular instrument usually requires a small amount of data. In most cases, the combined size of all portfolios and instruments ranges from 50-500 MB. Applications that provide multimedia decision support data could store petabytes of data.

Most deployed financial applications involve a small number of trader workstations on a local area network. Concurrency control among traders is fairly easy, since each tends to have specific ownership of a set of portfolios. However, a new trend known as team trading, whereby a group of traders may have joint responsibility for a single portfolio, requires sophisticated concurrency control. Most companies deploy these applications on UNIX workstations and develop these applications in C++, with a significant minority using Smalltalk. With the rise in popularity of Java, we expect Java to soon challenge C++'s lead.

**Product Information Management**

PIM systems track documents and other electronic data generated during the lifecycles of complex manufactured products, like automobiles and airplanes. During the development of such a product, each subsystem passes through requirements definition, design, manufacture, and testing phases. An organization needs to track the information generated during these phases so that it can maintain a record of how requirements get translated into features. Also, to assemble the final product, all the
Choosing a Higher Performance Database

Subsystems need to work together properly. Since organizationally and geographically separated groups may build different subsystems, coordinating their efforts can become difficult. PIM systems store all pieces of information created during a product's lifecycle and the relationships among them, making information available to every team that needs it and providing a trail for diagnosing the source of product defects. Furthermore, advanced PIM systems also manage workflow, ensuring that appropriate personnel know about and authorize design changes. The additional need to store two types of complex data make an ODBMS an attractive solution to these problems:

**Component assemblies.** More than any of the other application areas covered here, PIM systems require support for composite objects. A given assembly may contain hundreds of subassemblies, each of which contains hundreds of parts. The underlying database must allow the manipulation of these assemblies as logical wholes.

Each assembly, subassembly, and part has associated requirements, design, manufacturing, and test documents. Different engineers need to modify different documents and these modifications may take several days. The system needs to support check-in and check-out of documents and document versioning. The system must also support complex queries so that engineers can find specific documents to diagnose problems, assess the impact of changes, etc.

**Workflow rules.** To make sure that all teams receive the information they need, advanced PIM systems maintain a workflow model for a given product. For instance, all teams need to know about changes to other components that interface with their components. Certain types of changes may require sign-off from a number of different people. Managing this process requires complex relationships among workflow elements and from workflow elements to documents. Systems that manage workflow must support queries so that engineers can check the status of documents managed by the system. As the product matures, workflow rules may change. The system must flexibly accommodate these changes.

Information elements vary widely in size and format, from complex CAD drawings to simple test results, requiring varying sized objects. As a product moves through its lifecycle, the number of these objects grows quite large. The database for a complex product with many subsystems, such as a passenger aircraft, can grow to hundreds of petabytes.

All engineers involved in the development effort usually have clients running on their workstations and each department usually has its own database. Workstations often vary in type from team to team. The total number of clients involved in a complex project can number in the thousands. PIM systems are
Choosing a Higher Performance Database

mission critical. They require high integrity because corrupted documents could delay product development or introduce flaws into the product. However, since PIM systems aren't directly involved in the manufacturing process, they do not require the continuous operation of the other application areas. Most companies develop PIM systems in C++, though some companies have begun using Java and Smalltalk.

**Process Control**

Process control refers to applications that monitor and control physical processes such as chemical manufacturing and environmental control for buildings. These applications perform two kinds of tasks. They monitor information about the state of the system, i.e., reactor vessel temperatures, flow rates, and yields. They also exercise control over the system, i.e., open valves, apply more heat to reactor vessels, and increase the flow of cooling systems. Process control systems are often organized in a hierarchy. A top-level system monitors the entire factory. Lower-level systems monitor individual manufacturing processes, known as cells. These systems constantly adjust processes to maintain performance within tight tolerances. They respond to different conditions by applying a combination of standard algorithms and custom rules. Human operators adjust these rules over time to optimize performance. The basis for adjusting rules often comes from querying process histories to discover problems and potential improvements. Clearly, these systems require the highest level of reliability because of the potentially dangerous nature of the processes they control. Two types of data add complexity to this reliability requirement:

**Control and sensor hierarchy.** The system maintains a complete model of all controls and sensors in a manufacturing cell. This model consists of a component hierarchy, imposing the same requirement for composite objects as the other application areas, and the physical relationships among components, imposing the requirement for complex relationships. In large scale manufacturing processes such as an oil refinery, this model can become quite large.

**Sensor data.** In a complex manufacturing process there is a great deal of sensor data. Effectively reviewing and responding to this data requires rapid navigation. Some systems maintain detailed time series data for each sensor, imposing a requirement for the ability to handle large amounts of variable length data.

Modeling individual components does not require large objects. Likewise, the data from each sensor at any one time is not large. However, the number of components and sensors, plus historical sensor data can lead to databases of several petabytes. In these applications, each control and sensor generates a
Choosing a Higher Performance Database

A continuous stream of data. These components send the data to a database where the process control system itself reads it. While this division of labor means that each object usually only has one updater and one reader, they must have simultaneous access to the same data.

Most companies develop process control applications in C++ and deploy them on UNIX or Windows NT workstations.

Requirements and Features

The previous section discussed the general problems faced by four types of applications. This section maps these problem statements onto our ODBMS requirements framework. As discussed in the Introduction, this framework organizes requirements into six categories:

- Reliability and Integrity
- Data Model Complexity
- Transaction Profile
- Distribution
- Performance
- Environment

For each of these categories, we first define the individual requirements and provide some guidance in how to weight them for different applications. We then weight these requirements as High, Medium, or Low for the four applications introduced in the previous section. The rankings come from the extensive experience of Objectivity engineers who have assisted customers in building these types of applications. We also discuss how Objectivity/DB meets these application requirements.

Reliability and Integrity

This category concerns requirements relating to data safety and database availability. Specific requirements are:

- Data Integrity. A requirement for high data integrity means the cost of data corruption is high.
- Referential Integrity. A requirement for high referential integrity means the cost of dangling references is high. The database should delete the relationships between two objects when one of the objects is deleted.
Choosing a Higher Performance Database

- 24x7 Operation. The application needs to operate continuously or near-continuously.
- Failure Recovery. The application needs the database to recover from a failure as quickly as possible.

<table>
<thead>
<tr>
<th></th>
<th>Network Management</th>
<th>Financial Trading</th>
<th>Product Information Management</th>
<th>Process Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Integrity</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Referential</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Integrity</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>24x7 Operation</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Failure Recovery</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 1 - Reliability and Integrity Requirements

Table 1 shows the ranking of these requirements for our example application areas. Not surprisingly, all four applications require high data and referential integrity, principal reasons for using a DBMS in the first place. Objectivity/DB provides high data integrity through its use of object handles and object references. Developers use references or handles as the type for variables that refer to persistent objects. These types make variables independent of the in-memory location of persistent objects. An alternative implementation is to simply convert object identifiers (OIDS) to in-memory pointers. Unfortunately, this technique, sometimes known as "swizzling", can lead to data corruption. Pointers become invalid after a transaction commits because object lengths can change, causing their memory location to change as well. Reusing invalid pointers corrupts whatever objects occupy the memory location to which those pointers refer. With Objectivity/DB, references and handles re-acquire the memory locations of objects if they change. The only performance penalty of using a reference or handle is an extra pointer dereference. The time required for the extra dereference has proven to be completely inconsequential within the execution of Objectivity/DB applications.

In addition to data integrity, Objectivity/DB provides referential integrity through bi-directional associations. The database automatically deletes an association when either of the objects in the association is deleted.
Choosing a Higher Performance Database

Except for PIM, our example applications also need 24x7 operation and failure recovery. Important features for achieving 24x7 operation are online backup and restore and low mean time between failures (MTBF). Objectivity/DB offers both. In fact, its MTBF is so low that when a potential customer asked an existing customer how long it took Objectivity/DB to recover from a failure, the customer replied, "I don't know, it's never failed." As for failure recovery features, Objectivity/DB offers automated recovery of server processes, but its low MTBF means customers very rarely need to use this feature. Objectivity/DB also allows administrators to partition databases and clients into autonomous workgroups, duplicating all the necessary database services for each workgroup. Such partitioning insulates each workgroup from failures elsewhere. In addition, Objectivity/DB replicates data across those autonomous partitions. This replication is performed in such a way that combines the data integrity of synchronous data replication with the availability of asynchronous data replication.

Data Model Complexity

This category concerns requirements relating to the complexity and size of data. Specific requirements are:

- Complex Relationships. The application needs to store networks of objects with many-to-many relationships, such as which customers purchased which products.
- Composite Objects. The application needs to store structural hierarchies of components.
- Complex Objects. The application needs to store data that varies in length.
- Large Objects. The application needs to store objects greater than 10KB.
- Small Objects. The application needs to store objects of less than 100 bytes.
- Modifiable Structure. The application needs to store data whose underlying structure, or schema, changes often.
Choosing a Higher Performance Database

<table>
<thead>
<tr>
<th>Complex Relationships</th>
<th>Financial Management</th>
<th>Product Information Management</th>
<th>Process Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Composite Objects</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Complex Objects</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Large Objects</td>
<td>Low</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Small Objects</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Modifiable Structure</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 2 - Data Model Requirements

Table 2 shows the ranking of these requirements for our example application areas. The table shows that complex relationships and composite objects are generally a high priority. This need stems from the nature of object-oriented data, which generally have many relationships among objects and many composite objects. If the weighting for these requirements were low, you might want to consider a relational database. As mentioned in the previous section, Objectivity/DB supports bi-directional associations as one of its primary features, as do most other ODBMSs. Bi-directional associations provide an arbitrary number of two-way connections between objects. To handle composite objects, Objectivity/DB allows propagation of delete and locking operations down the composition hierarchy, ensuring these operations treat composite objects as logical wholes.

To further support both complex relationships and composite objects, Objectivity DB provides advanced clustering features. Developers can put related objects into containers so that the database will lock them together. Containers increase locking efficiency by reducing the number of locks. They also reduce deadlocking by locking groups of objects accessed at the same time. This allows Objectivity/DB developers to cluster closely related objects on the same data page, greatly increasing the efficiency of disk access, network transfer, and cross-transaction caching. An alternative is to have the database use a simple clustering heuristic, such as clustering objects together according to the time when they are created. Such a heuristic robs the developer of an important performance tuning mechanism and does not take advantage of the inherent inter-relations present in object-oriented data.
Choosing a Higher Performance Database

Complex objects are somewhat lower priority in this type of application. However, Objectivity/DB provides some unique features. For instance, it allows objects to have multiple variable length arrays of data. A simpler implementation would be to limit objects to a single variable length array. However, this limitation could force developers to either split an object into two pieces or write extra routines to combine two logical data structures into one variable length array.

For the applications considered here, small objects are much more important than large ones. Objectivity/DB handles small objects extremely well because it sends a page of objects from the server to the client, making transfers of small objects efficient. Objectivity chose this approach because sending only one object at a time would put an unacceptable load on the network when accessing many small objects. If an application does require large objects, such as multimedia documents, Objectivity/DB provides an important feature that makes them easier to handle. The client automatically manages memory when the amount of data sent from the server exceeds available memory, freeing developers from the need to write manual memory management routines to handle this situation.

As for modifiable data structure, or schema, the applications considered here fall into two groups: those where the structure changes very little and those where it changes often. Objectivity/DB allows developers to modify their schema without bringing the database down. No longer must the database be shut down, unloaded, filtered, and re-loaded in order to alter the database schema. Additionally, developers are able to create a hierarchy of schema definitions, version objects and migrate existing objects from the old schema to the new schema.

Transaction Profile

This category concerns requirements relating to how users access data. Specific requirements are:

- **Many Readers per Object.** Many different clients need to read the same data at the same time. 1-5 readers is low, 6-100 readers is medium, and more than 100 readers is high.
- **Many Updaters per Object.** Many different clients need to update the same data at the same time. 1-3 updaters is low, 4-20 updaters is medium, and more than 20 updaters is high.
- **Long Transactions.** A high need indicates many transactions need to survive the termination of a process. Long transactions occur when a person wants to establish ownership over an object for an extended time.
- **Short Transactions.** A high need indicates many transactions occur over the life of a process and none survive its termination.
Choosing a Higher Performance Database

- Large Transactions. A high need indicates most transactions involve amounts of data greater than 1MB.
- Small Transactions. A high need indicates most transactions involve amounts of data less than 100KB. Note that the priority for small transactions should be inversely proportional to the priority for large transactions.

Table 3 shows the ranking of these requirements for our example application areas. The four application areas covered have very similar data access characteristics. They have a medium number of readers per object and few updaters. Transactions are generally short and small. Objectivity/DB provides excellent support for this type of transaction through a flexible locking mode called MROW (Multiple Readers, One Writer). This mode allows clients to read objects even when another client is updating them. Since most applications have many readers per object, but only one or two updaters, conflicts between readers and updaters are by far the most common. MROW eliminates this conflict with no performance penalty. Note that this feature is not a “dirty read”. MROW allows readers to only see committed data.

Objectivity/DB provides support for both long and short transactions. Check-in/check-out features address long transactions. In the particular applications, short, small transactions are a higher priority.
Choosing a Higher Performance Database

For this type of transactions, the efficiency of network transfer, cross-transaction caching, and concurrency determine the overall performance.

Performance for many small transactions can suffer due to the overhead incurred for every network transfer. To ameliorate this problem, Objectivity/DB sends an entire page of objects to the client at one time, amortizing network overhead over all the objects on the page. The client may or may not use all the objects on the page in a given transaction. However, since Objectivity/DB allows developers to define the clusters of related objects on a given page, the client is likely to use these objects in future transactions, saving the overhead associated with individual requests for objects. Accessing each object individually would necessitate a call to the server over the network. Simply using a persistent linked list would require a remote procedure call for every node traversal.

In addition, Objectivity/DB provides a sophisticated client mechanism that automatically caches all the objects on the page. When it needs to access some of them in the future, it receives a time stamp with the lock to make sure no one else has updated them. The client gets in-memory performance for these cached objects. Checking the time stamp is very important. Without this mechanism or other cache consistency protocol, cross transaction caching could cause severe integrity problems because the cached version of the object could get out of synchronization with the committed version. Objectivity/DB automatically ensures the integrity of the objects in the cache, allowing developers to concentrate on the application rather than the memory management and integrity of cached data.

Data access concurrency is controllable in a completely flexible manner via the container clustering mechanism described earlier. In systems with high performance requirements, logically related objects are typically accessed together. Objectivity/DB allows locking to be precisely matched to this logical clustering to guarantee peak performance. Locking individual objects rather than object clusters would severely impact performance. Not only would it require additional processing during the transaction, but the proliferation of locking requests would also contribute to a server bottleneck.

**Distribution**

This category concerns requirements relating to the application's network configuration and the database's distribution. Specific requirements are:

- Many Clients. Many client processes need to connect to the database at any one time. 1-100 clients is low, 101-1000 clients is medium, and more than 1000.
Choosing a Higher Performance Database

- Many Databases. Clients need to access data in many databases. 1-3 databases is low, 4-50 databases is medium, and more than 50 databases is high.
- Many Sites. Clients and databases at many sites interact with each other. 1-3 sites is low, 4-10 sites is medium, and more than 10 sites is high.
- Heterogeneous Platforms. Clients need to access data stored in different data formats. These differences include floating point format, integer format, padding, and alignment, and differences between 32-bit and 64-bit representations.
- Flexible Configuration. The application requires flexible changes to the distribution architecture, including the easy addition of clients and servers.

<table>
<thead>
<tr>
<th></th>
<th>Network Management</th>
<th>Financial Trading</th>
<th>Product Information Management</th>
<th>Process Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many Clients</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Many Databases</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Many Sites</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Platforms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flexible Configuration</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 4 - Distribution Requirements

Table 4 shows the ranking of these requirements for our example application areas. In general, the requirement for many clients and many databases is medium. Objectivity/DB's fully distributed architecture meets this need. Objectivity/DB organizes databases into federations. A federation can contain up to sixty-four thousand databases. Clients have a single logical view of all databases within a federation and all services are coordinated across databases within a federation. The single logical view frees developers from having to program clients to explicitly open individual servers or databases. Such a limitation could mean extensive reprogramming as the number of databases increases and administrators move objects to the new databases for load balancing. Coordinating services across a federation means that developers do not have to worry about such potentially crippling conditions as deadlocks across databases.
Choosing a Higher Performance Database

Perhaps Objectivity/DB's greatest strength lies in meeting the requirement for completely flexible database configuration. Typically, Objectivity/DB applications start out with a traditional centralized client-server model, in many cases with the intent of evolving into a more distributed architecture later.

Any individual Objectivity/DB "client" machine has a Local Server that contains the ability to provide full database functionality to the local application. Accessing data on a "server" machine requires a Remote Server process. There are two choices for the Remote Server process. First, there is Objectivity’s Advanced Multithreaded Server (AMS) which centralizes database services and offers advanced performance features such as asynchronous writes. Alternatively, the Remote Server may be the NFS server daemon native to the platform.

An application may require any number of "client" and "server" machines to be connected. In fact, both the Local Server and Remote Server may be present on the same machine. Meeting the needs of multiple application architectures lets Objectivity allow customers to evolve their application architecture to meet their changing requirements.

Another Objectivity/DB strong point is its full support of heterogeneous platforms. The interoperation of heterogeneous platforms is completely transparent to the application. Data residing on any system may be accessed by any application on any combination of platforms simultaneously.

Whenever a client accesses an object, the database automatically checks to see if the data formats differ and performs any necessary conversion. Since Objectivity/DB does not store objects in their native virtual memory page format, it has no trouble with padding and alignment differences. Any such system would require manually altering source code files to account for these differences. This time-consuming and error-prone procedure would not even be possible when using third party libraries. In contrast, Objectivity/DB’s approach makes padding and alignment completely transparent to the developer.

Performance

This category concerns requirements relating to the speed with which the database completes different operations. Specific requirements are:

- Scalability. Performance measured on a per object basis needs to remain as constant as possible. Application also requires easy addition of resources to increase performance.
- Reading Objects. Application needs excellent response time for read transactions.
Choosing a Higher Performance Database

- Traversing Relationships. Application needs excellent response time when navigating from one object to another via a relationship.
- Updating Objects. Application needs excellent response time when updating objects.
- Executing Methods. Application has complex methods that require fast execution time.
- Making Queries. Application requires excellent response time when making predicate queries.

Table 5 shows the ranking of these requirements for our example application areas. To a large extent, we've already covered scalability and performance for reading objects. Sending objects to clients a page at a time, clustering related objects together, and cross-transaction caching, make read access fast. These features are also efficient, allowing servers to gracefully handle additional clients. Furthermore, Objectivity/DB's flexible architecture allows the easy addition of resources to improve scalability.

One of Objectivity's big performance advantages comes from processing queries and methods on the clients rather than the server. The advantage may seem counter-intuitive at first. How can a client process queries and methods faster than a server? First, a server actually serves many clients. Second, the relative performance advantage of servers over clients has decreased. A fast server today has a performance of 250 MIPS. With the rise of Pentium PCs, clients average 50 MIPS. In a network of 10 clients and 1 server, the clients have twice as much processing power. If clients are UNIX workstations
Choosing a Higher Performance Database

or if the client to server ratio is greater, this advantage increases. A typical objection to performing queries on the client is that the client would have to receive the entire database over the network to search all objects. In reality, nearly all queries will involve indexes. The client only needs to receive the relevant part of the index. Cross-transaction caching means that the index will remain in client memory and the next query won’t require any index pages.

Traversing relationships is a high priority item for most ODBMS applications. This is an area in which ODBMS technology provides a quantum leap in performance over RDBMS technology. ODBMS technology removes the need for performing expensive JOIN operations in navigating complex data relationships. Overall system performance can also be improved by replicating data, and moving it closer to the client application. Objectivity/DB implements replication in an extremely flexible manner that allows application requirements to dictate the configuration and the granularity of replicated data. By default, data is read from the closest replica, thus tremendously improving read performance.

Environment

This category concerns requirements relating to the application's development and deployment environments. Specific requirements are:

- **C++ Support.** High performance applications may be written in C++.
- **Java Support.** Many web-based and service oriented architecture (SOA) systems are written in Java.
- **Microsoft C#** is also being used for Windows .Net applications
- **Smalltalk Support.** High need for Smalltalk support means the applications are developed almost solely in Smalltalk.
- **Standards Support.** The application needs compatibility with database standards such as SQL, ODBC, and ODMG-93.
- **Legacy Integration.** The application requires direct access to data stored in other types of databases.
Table 6 shows the ranking of these requirements for our example application areas. C++ is the most important OOPL for the example application areas, though Java's importance is increasing rapidly. Objectivity/DB supports all three languages. Note that, if the application does not require support for an OOPL, a relational database may be an acceptable choice.

Objectivity also leads the industry in standards compliance. Objectivity is a founding member of the Object Database Management Group (ODMG). Objectivity also offers SQL and ODBC access to its database, allowing developers to use hundred of development tools that allow data access through these standards. While legacy integration in the form of gateways does not have the importance of SQL and ODBC access, Objectivity is exploring the possibility of building gateways to RDBMS products.
Choosing a Higher Performance Database

Additional database selection criteria:

When selecting an ODBMS, concentrate on your particular application's needs. After defining your priorities, you should open a dialog with ODBMS vendors and investigate how they can satisfy your needs. Objectivity is more than willing to participate and assist in these evaluations. To learn how well Objectivity/DB matches up against your particular application requirements, please contact Objectivity as shown on the back cover of this document.

While this guide has focused primarily on technical issues, you should also investigate the vendor itself. You should ask questions such as:

- Is the vendor financially stable?
- Does the vendor have an experienced management team?
- Will the vendor incorporate customer requests into future releases?
- Does the vendor offer 24x7 support?
- Does the vendor offer consulting services to facilitate technology transfer?

Spend some time investigating the answers to each of these questions for all your potential vendors. For Objectivity, you'll find the answers to all of these questions are a resounding, "Yes!" This is evident in the fact that Objectivity has many customers who have successfully deployed applications. After all, Objectivity's success can only be measured by the success of our customers' deployment.
Choosing a Higher Performance Database

Objectivity, Inc. and Objectivity/DB: Innovate with Confidence.

Objectivity, Inc. provides distributed data management solutions for business, science and government organizations, and is the enabling technology for some of the most complex and mission critical systems in operation around the world today.

Objectivity/DB benefits include

- **Scalability**: Uniquely distributed, federated architecture provides a single logical view to virtually unlimited data sources. A single Objectivity/DB federation may contain more than 65,000 individual databases, and provides more than a yottabyte (one quadrillion gigabytes) of addressing space. That's a thousand times the space on all the hard-drives in the world today, by the way.

- **Performance**: Near real-time performance on vast volumes of complex interrelated data.

- **Flexibility**: Design and deploy the system you want, with complete interoperability among heterogeneous operating systems and language bindings. Objectivity/DB supports C++, C#, Java, Python, Smalltalk and other programming languages, and can run on virtually any hardware.

- **Reliability, Low Maintenance**: Designed to support stand-alone embedded systems or data services across vast networks, with little or no administrative overhead. Objectivity/DB is used in systems and devices that simply must work, all the time.

With Objectivity/DB, you can manage your data, rather than your data management system. Objectivity, Inc. provides a solution that addresses today's data and systems requirements, and enables developers and integrators to design and build the systems they want and need.

Objectivity, Inc. has offices and representatives worldwide, and works directly with organizations, integrators and technical teams to recommend solutions and support options specifically tailored to your project and technical requirements. Contact us to schedule a consultation.
## Choosing a Higher Performance Database

### About the company and technology

- **U.S. owned and operated, with GSA schedule contract (#GS-35F-0119R) and clearances**

- **20 years of experience working with agencies and organizations with the most complex security, intelligence, industrial and business requirements.**

- **A patented data management solution that has been built and optimized for virtually unlimited performance, flexibility, scalability and reliability (zero administration and no unplanned downtime).**

- **Our team of expert systems engineers and support staff are ready to assist you with any or all phases of your project.**

- **Our solution has been recognized by the U.S. Government for enabling and supporting mission-critical national security and intelligence systems.**

- **Objectivity/DB is unique in its support for distributed architectures and complex data management requirements including high-speed data ingest, data fusion and correlation, massively parallel processing, relationship analytics and graph traversal, and, of course, the ability to scale to meet the most demanding data storage requirements.**

### Contact our technical representatives to schedule a consultation today.

**Objectivity, Inc.**  
640 West California Ave., Suite 210  
Sunnyvale, CA 94086-3624 US  
Phone: (408) 992-7100  
FAX: (408) 992-7171  
www.objectivity.com
Choosing a Higher Performance Database

Bibliography

Introduction to Object Technology


Introduction to ODBMSs

- Selecting ODBMSs

ODBMS Architectures


ODBMS Performance Benchmarks