

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

Introduction: Building mission critical systems using object oriented technology is done in an environment of high pressure and extreme urgency. Combine this with the architectural choices available in the ODBMS arena and the pressures in evaluating and selecting an appropriate object persistence solution and we have a perfect opportunity to either miss a key advantage or succumb to uncertainty and confusion.

In the object oriented design of software applications, architecture is one of the issues at the forefront. That same focus on good architecture has been brought to Objectivity/DB in order to lay a foundation that will enable software developers to build solutions that will stand the test of time. One basic commonality amongst all ODBMS is that they all store and retrieve objects. However, we strongly suggest that how a given ODBMS achieves that basic goal of storage is the key to choosing an ODBMS. Relational Database Systems (RDBMS) have had decades in which to refine and perfect their ways of storing rows and columns of data. B-Trees and indexing methods have become a staple in almost all RDBMS. In the ODBMS world, there are several competing ideas on how storage of objects should be achieved and we will show that Objectivity/DB has an excellent solution to that very crucial question.

This document will give the background to Objectivity/DB storage design decisions and cover key aspects of Objectivity/DB architecture. We will specifically deal with the powerful concept of Objectivity/DB containers and how they may be used to achieve superior designs. We will also show that traditional aspects of software development such as data structures can be used as well as current object oriented development methods.

BACKGROUND: Objectivity introduced containers into the storage hierarchy for four main reasons: to increase system throughput; to ease the migration path from file-based storage to an ODBMS; to make it easier to use standard off-line (hierarchical) mass storage devices; and to make it easier to use standard operating system security features. This White Paper is primarily about increasing system throughput, but the ability to treat a container in a way that is analogous to a conventional file is very useful in some applications, particularly in workgroup environments and in large scale data acquisition or warehousing applications.

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

Many alternative logical and physical storage models were considered and discarded during the requirements analysis and systems design phase. These ranged from the use of the “Black Hole” approach favored in traditional DBMSs to the explicit declaration of both logical and physical storage structures, either in a data dictionary or at runtime. In the traditional database approach there is a single address space and the application programmer has little or no control over the physical placement of data. This task is delegated to a Database Administrator who uses projected and actual system statistics to map the logical structures, such as records or rows, indices and hash tables onto the appropriate physical devices and storage structures (generally files or partitions, but sometimes even down to the cylinder/track level). In a conventional file environment, almost universally favored in the engineering, scientific and PC applications world, the structure of the data is often hidden in the applications and the essential metadata components of a DBMS are missing. However, early ODBMSs often used structured files, sometimes capitalizing on the performance of virtual memory mapped files, because UNIX programmers and users were comfortable with the filesystem paradigm. Some of these ODBMSs have physical object identifiers (OIDs) and this can make space reclamation, cross database references and object migration (as the schemas change) very cumbersome. They also tend to scale very badly as the volume of data and the amount of concurrent access increases.

The evolution of DBMSs is interesting in that successive technologies did not always capture all of the good features of the previous ones. Many users tend to be more comfortable with files that they can see or own than with information locked in a closed DBMS. The CODASYL (network) DBMS standard spawned products that were extremely efficient at navigating through linked lists of related data. The RDBMS world generally ignored these factors and so RDBMSs still offer relatively crude data clustering features and are clumsy and inefficient at performing complex “JOIN” operations. Objectivity tried to meet the demands of object-oriented applications without losing the advantages that each of the previous technologies, including RDBMSs, had introduced. So, although conventional file structures, tuple engines, dataflow techniques and many other alternatives were considered, Objectivity settled on a hierarchical logical storage model that simplifies database administration tasks and meets the four main goals

Dynamic Containers™: The Key To Superior Performance

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

outlined above.

The physical model reflects the logical model. It provides a means to safely and efficiently store and manipulate both very small and very large objects, large collections of objects and extremely complex networks of related objects. There is a single logical address space supported by a hierarchically structured physical space. The application programmer has some control over the physical space. For instance, a programmer may create a new database and specify the hostname and file pathname for the initial database file. Similarly, any object may be initially or dynamically clustered near any other object (or an existing group of objects). However, like its counterparts in earlier DBMSs, Objectivity/DB's storage manager automatically manages critical tasks, such as space reclamation in the cache and filesystem during and across transactions.

Objectivity/DB Database Architecture

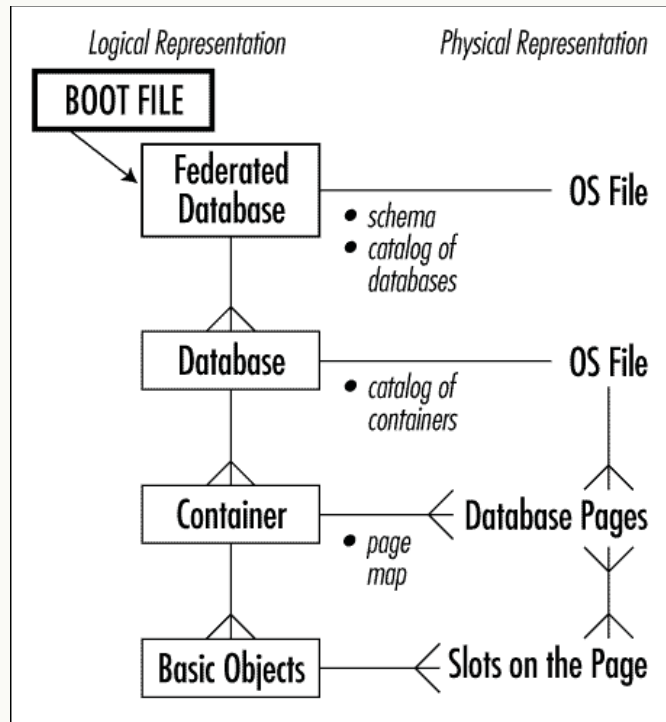
Overview: Objectivity/DB utilizes a 64 bit object identifier (OID) for every object within the federated database (or Federation). The 64 bit OID is split into 4 components as follows:

- First 16 bits - Logical Database ID
- Second 16 bits - Logical Container ID (1 bit internal use, 15 bits container number)
- Third 16 bits - Logical Page within container
- Last 16 bits - Logical Slot within page

Dynamic Containers™: The Key To Superior Performance

www.objectivity.com

Corporate Headquarters:
640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171



The Objectivity/DB storage model thus comprises a Federation (the system database) which logically contains a collection of databases, where each database is represented as a physical file¹. Each database is a collection of containers, where each container is a number of physical database pages. A container contains objects which may be related to many other objects. The container is used to cluster a group of logically related objects together for efficient access and processing.

Based on the OID breakdown above, a federated database can have 65,535 databases (implemented as system files) which can in turn have 32,767 containers, which in turn can have 65,535 pages. Assuming that we use 4Kb pages, this would give us the ability to address a minimum of $5.76 * 10^{17}$ bytes - a very large addressing space that is closing in on the petabyte level. By changing the page size, the address space is affected proportionately. Objects may be larger than a physical page and VArrays may be extremely large, so the physical address space can be many Petabytes with current releases and will stretch to Exabytes with future releases.

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

This is part of the reason that Objectivity/DB is the premier choice for those requiring the ability to store extremely large amounts of data in a database and to manage it.

Objectivity/DB uses the container as the level of locking granularity. Lock management is done by lock server processes which communicates with all the database client processes that request locks. Since individual containers can be tailored to different sizes, a single lock can lock very few objects in a small container or large numbers of objects in a large container.

There are three lock modes supported by Objectivity. The first two are familiar to most database programmers, the read lock and the update lock. As expected, a read lock prevents others from updating, but allows others to read, and an update lock is exclusive. However, since the locking granularity is at the container level, supporting only these two modes would be quite inconvenient to most programmers. It could also degrade system throughput and response times. Therefore, a third locking mode, MROW read, is also supported. MROW (Multiple Readers One Writer) permits an update to exist concurrently with any number of MROW readers. In short, an MROW reader never blocks, and can always read data from a container. Objectivity/DB takes a transactionally consistent snapshot of the container and maintains the ACID properties of an MROW read transaction for each MROW reader.

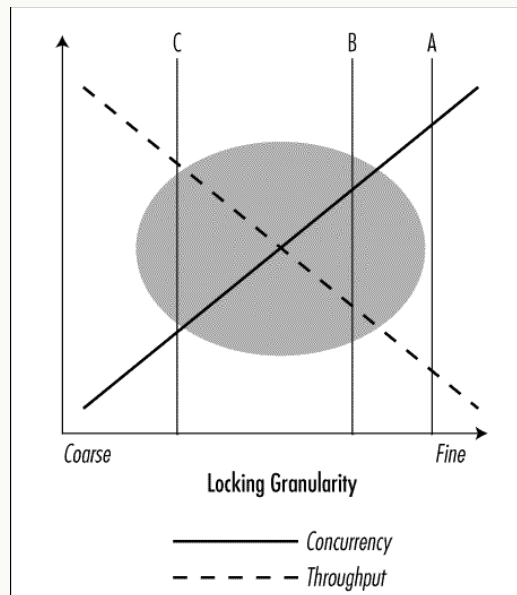
Typical Data Access and Usage: In most OLTP database applications, data is usually used not one datum at a time but usually in bunches or sets that may vary in size. Several associated objects may be created or accessed in a very short time frame. An example would be a customer order with a dozen different items. Most of the items may be in stock and shipped immediately and hence the representative objects modified in a very short time frame as well.

The database designer is able to tailor the page size in Objectivity/DB to the size of the bunch of data that is usually created or modified. This means that a single page could be about the right size for all the items that are on a typical order. However, there is no reason to insist that the page can contain only one order and its items. It could be larger to accommodate several orders and their items together. This brings up the issues of concurrency

Dynamic Containers™: The Key To Superior Performance

and performance when it comes to creating and updating objects by multiple users since Objectivity/DB locks at the container level.

By varying the size of containers you can trade off concurrency versus performance as in the graph below.



An alternative architecture could provide locking on individual objects or pages. With object-level locking architectures, locks have to be requested and granted for each object regardless of the fact that they may be associated and therefore will be updated as a group. The associated objects may even be on the same page due to clustering but cannot benefit from page level locking. This puts a theoretical limit on throughput of object-level locking systems. In the graph above, an object-level locking architecture will be on the right side of the graph at line A where concurrency is high but throughput is low. Page-level locking architectures are much better in that a single lock will allow you to update many objects within one page. However, when you wish to update objects across many pages you get back to having to manage quite a few locks and therefore throughput will still plateau at some level. Line B in the graph above illustrates that page-level locking also dictates the concurrency and throughput that may be obtained. Container-level locking allows the designer to vary the size of individual containers and

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

the number of objects within that container according to the application's needs. By placing one object per container, object-level locking is available. By having minimally sized containers, a form of page-level locking is thus available. Finally, by having containers with multiple pages, you can maximize throughput by requesting only one lock. In reality, most objects are not standalone. They are usually associated with other objects which can benefit from being on the same page or container and thus can be locked at the same time. This is represented by the shaded elliptical area in the graph above. The designer can thus move within that area on the graph and achieve varying levels of concurrency and throughput within a single application.

The fact that containers can be created at a specific size by the applications to store specific objects means that you can lock varying amounts of data by varying the size of the container. With this flexibility, the designer can make a very small container to store few objects as well as large containers to store large numbers of objects. Furthermore, containers can both grow and shrink in size as required. This gives Objectivity/DB variable locking granularity within the same database.

Basic Approaches to Utilizing Containers: One of the key elements of designing a system using Objectivity/DB is the mapping of the Application Object Model onto the Objectivity logical storage hierarchy and thus taking advantage of the underlying physical implementation. We will cover two of these, the Container Pool Pattern and the Data Acquisition Pattern in some detail.

Container Pool Pattern: The concept of container pools is a very simple and useful one. It is basically the generic version of the approach taken by the RentalFleet example, explained below.

In designing and implementing a database in Objectivity/DB, you create a pool of containers at database creation time that will be large enough to contain all objects that the database may be required to hold for the foreseeable future. The trick is to use basic probability to calculate how large this pool of containers will need to be for your application.

Dynamic Containers™: The Key To Superior Performance

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
 Suite 210 Sunnyvale,
 CA 94086-2486 US
 Tel: (408) 992-7100
 Fax: (408) 992-7171

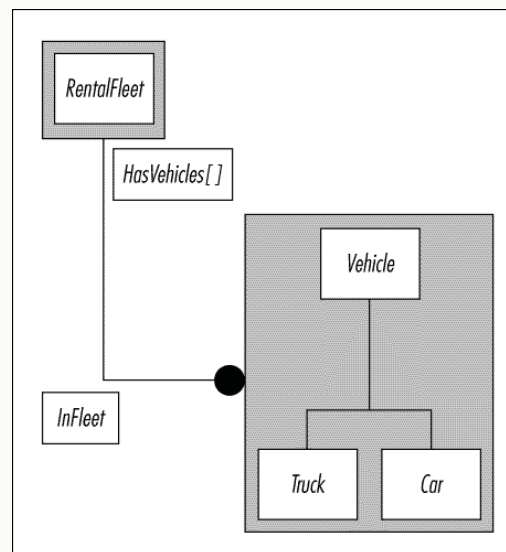
If your application has a worst case scenario of 12 clients needing to perform updates at any given moment and you wish to calculate the probability that there will not exist any update locking contentions if you use 1000 containers, you can do so as follows:

$$\text{Probability} = (1000! / (1000 - 12)!) / (1000^{12})$$

which works out to approximately 93.58%. Increasing the number of containers to 10,000 will increase the probability of no contention to approximately 99.34%. Even if there is contention, ODBMS updates are usually very fast. Recall that readers do not block writers in Objectivity/DB.

The use of container pools can be quite straightforward. Picking a container from the pool at random or by round robin assignment are such methods. You may find the random method in the `d_session` class (available from Objectivity infocenter) useful. Or you can use more sophisticated schemes including hashing or named containers or containers with specific purposes. We will propose a pattern utilizing some of these ideas in the next section.

Using the object model below, we will step through the example of an automobile rental fleet.



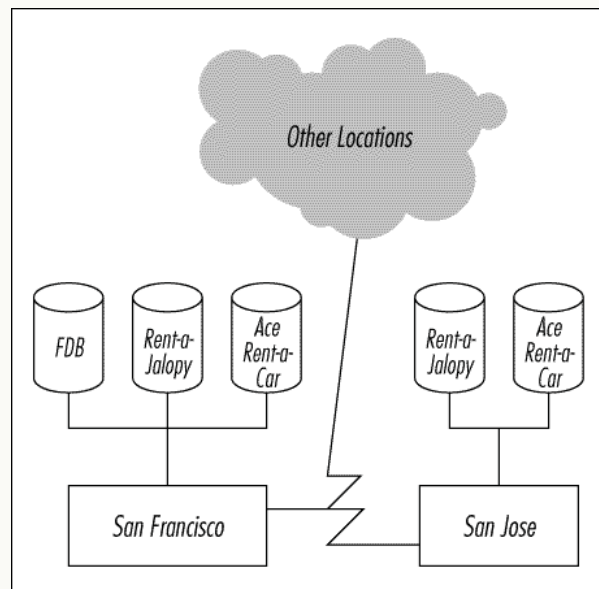
Dynamic Containers™: The Key To Superior Performance

www.objectivity.com

Corporate Headquarters:
 640 West California Ave.
 Suite 210 Sunnyvale,
 CA 94086-2486 US
 Tel: (408) 992-7100
 Fax: (408) 992-7171

Consider the RentalFleet example. Let us assume that we are dealing with a worldwide system of RentalFleets. A RentalFleet is an organization that rents vehicles of different classes. The idea is that each RentalFleet organization keeps data for vehicles at each location at that location itself, i.e. in a database that resides at that location. Therefore the Rent-a-Jalopy company would have databases in each location it served, say San Francisco, San Jose and London's Heathrow airport. It is also possible to return a vehicle to a different location than the one from which it was rented. Furthermore, Rent-a-Jalopy is a tight organization which obtains its vehicles from Ace Rent-a-Car.

Assuming that both Rent-a-Jalopy and Ace Rent-a-Car are in the same federated database, we would have a situation like the one below.



Now we can think about the containerization strategy. On a per location basis, if we put all vehicles in one container then this would only work if there is only one updating process at any time. This is probably not the norm. Next is the strategy to spread the vehicles across a number of containers to increase concurrency and therefore throughput. You may think of splitting all the vehicles into containers which represent the class of car.

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

There would be economy, compacts, mid-size, full-size, luxury, SUVs, minivans and trucks as the different classes. This will certainly spread the vehicles across a number of containers. However if San Francisco is hosting the Fortune 100 top CEOs, they may all arrive together wanting the luxury cars! There will be conflicts accessing the luxury cars container. We therefore need a different clustering strategy.

We can use simple probability to determine the number of containers that would be needed. Let us say that the maximum number of operators at any counter of any location is 10. The minimum number of containers that would be needed in such a scenario would be 10. However, allowing for collisions in our imperfect world, we could set the number of containers in this application to some factor larger than unity (such as two) times the number of operators. Now we can be reasonably certain that we are going to get a good degree of concurrency without going to the extremes of one object per container.

Data Model Abstraction: When writing ODBMS applications in C++, issues of encapsulation become more important than normal because the physical storage of objects is something which the application developer should not have to worry about when he is re-using libraries of persistent classes written by the schema developer. It is not possible to make using persistent objects as easy as using transient objects but our goal should be to get as close as possible to this ideal. A detailed implementation example is shown in the Appendix.

By imposing an abstraction layer between the business objects and the underlying data model, it becomes much easier to tune the performance or reorganize the data layout of your persistent objects at a later time. By following the pattern described here, the lifetime of persistent objects is all managed in a small number of methods which can be changed without affecting the rest of your application. This may be more work in the beginning but like other software engineering practices, it will pay off later.

Data Acquisition Pattern: Large scale data acquisition systems may be characterized as having three distinct phases: an acquisition phase; a processing phase; and a presentation phase. In the acquisition phase data

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

arrives in continuous streams or in batches. The data may arrive much too fast for a single processor or disk to handle, e.g. one Objectivity customer is projecting 20 Terabytes per day. Spreading a container pool across multiple machines and files makes it possible to store the arriving data safely. The application can use a near-infinite random access “virtual tape”. The containers are in a federation so other applications see a single address space and need not concern themselves with the physical placement of the data. The second phase is a number crunching phase and it may build indices, hash tables (ooMaps) or associations across the objects. It may even recluster or replicate some of the objects into existing or new containers to make subsequent access more efficient. The third phase benefits from the computing done during the second phase. If programmed correctly, the containerization is generally transparent at this stage.

Leveraging Containers: Since containers are basic objects in Objectivity/DB, you can subclass from them. This means that you can add attributes to containers that will give you tremendous flexibility. Examples include:

- Linked lists of containers
- Trees or networks of containers
- Multidimensional pointers in containers
- Containers with attributes relating to the objects in them

Since we can combine data structures with containers, we can design solutions that are extremely elegant and powerful as we will demonstrate below.

DEPLOYMENT EXAMPLES: The COM21 Solution

Com21 developed their NMAPS (Network Management and Provisioning System) using Objectivity/DB. The NMAPS system maintains information about a network of cable modems (STUs) and their hub controllers (HCXs). The object model is an abstraction/mapping of the actual physical hardware involved in the cable network. Classes include such real-world abstractions as HCX (the cable hub), HCX-slot (slot for card within the hub), HCX-card (the card that fits within slot), STU (the modem attached to the hub), Subscriber (individual subscriber to the service) and QosGroup (quality of service group for set of subscribers - different levels of quality of service are availability). By modeling the components of their cable network as objects,

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

application performance is improved and actions such as polling the HCX hardware to determine the STUs to which it is attached is unnecessary.

Com21 uses a variety of containers to store the objects and uses dedicated containers for each of the following classes of objects: Quality of Service group objects, Subscriber objects, HCX component objects and attached STUs (in a single HCX container), Alarm objects for each HCX, Polling Data and each Map (which maps Account number to Subscriber Map and STU to address). The container architecture provides for a division of objects based on function although alternative approaches of mixing and matching could easily have been performed. Customized Alarm containers (deriving from the container class ooContObj) themselves have references to Alarm summary information for that container. This is a proper logical division as the Alarm summary is for that container. The Objectivity/DB customizable container architecture easily enables good design.

Each level in the Objectivity/DB architecture provides additional levels of resolution. For example, all containers could be queried for a summary of all Alarm summaries. This query should not be concerned with the details of individual Alarm objects, provided the container is keeping an accurate summary. A customized HCX container allows for both physical and logical containment of HCX components and attached STUs. This breaking down of the problem into these components simplifies the design and the application designer knows implicitly where to find the objects. It would be difficult to imagine a solution to this problem without the container architecture due to both the complexity of a cable modem network and the difficulty of mapping physical containment using only associations.

When application polling is enabled, billing and performance data is periodically gathered in multiple containers within the polling database. Multiple containers allow multiple daemons to concurrently update information in the database. Using the MROW (multiple readers, one writer) feature on read transactions provides the highest level of concurrency allowing this data to be summarized without any interference from the updating processes.

This COM21 solution is a good example of utilizing containers of different sizes and contents to provide variable concurrency and throughput appropri-

Dynamic Containers™: The Key To Superior Performance

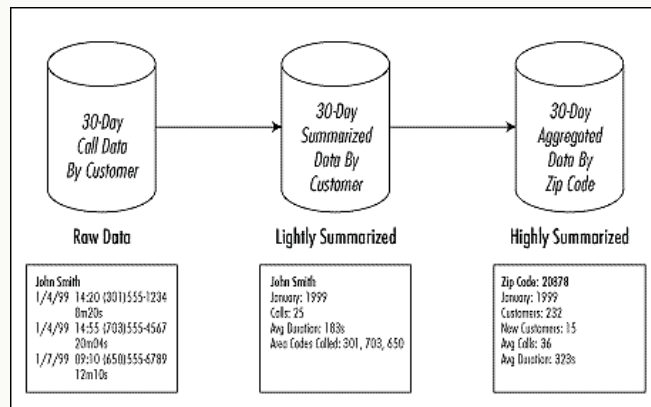
ate for the application.

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Data Warehousing: Imagine a cellular phone company wanting to implement a data warehouse to analyze user behavior. In designing such a data warehouse, data is stored and generated to several levels of granularity. We can break down the levels as in the diagram below.



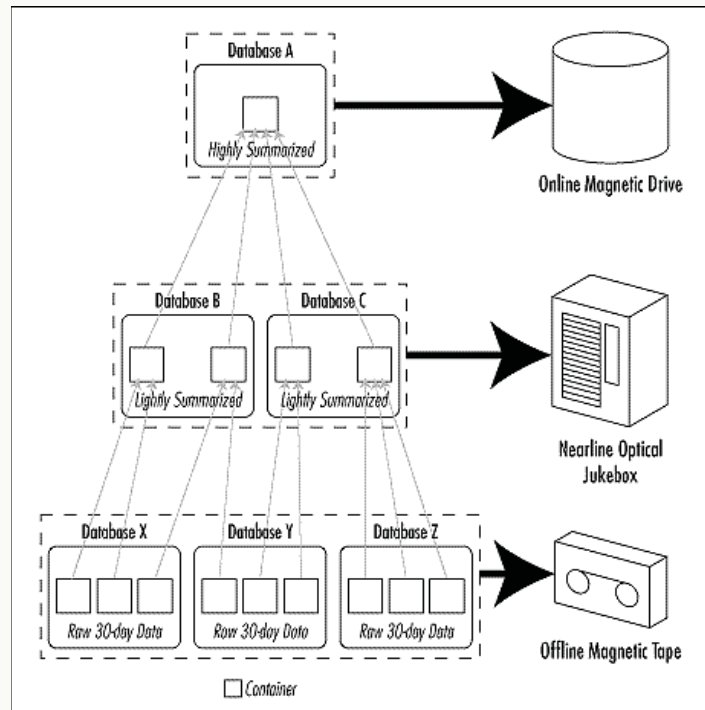
At the finest granularity, every call description record (CDR) in a 30-day period for each customer is kept. At the next higher level, a lightly summarized history of the customer activity since inception is kept. This lightly summarized history may contain statistical information by month for that customer such as calls by hour of day, day of week, area codes of numbers called, average duration of calls, etc. At the next level of granularity, customer data in each zip code served by the wireless phone company are aggregated in order to provide gross numbers that are useful for statistical activities. These may include number of calls made from a zip code by all customers, roaming call activity, customer churn rate, etc.

In implementing such a data warehouse in Objectivity/DB, we will use different databases and containers for the raw, lightly summarized and highly summarized data in a federated database. We can use a design such as that below.

Dynamic Containers™: The Key To Superior Performance

www.objectivity.com

Corporate Headquarters:
640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171



You can then use a hierarchical storage system (HSM) to move the databases containing the CDR records into nearline or offline storage such as optical disk and magnetic tape when analysis of the high granularity data is complete. The summarized data can be kept online on regular hard drives while the lightly summarized data on an optical disk jukebox (nearline) and finally the raw data can be offline on magnetic tape. Data can also be moved back from near line or offline storage on demand albeit with some delay. Objectivity/HPSS (High Performance Storage System) and the Objectivity Open Object File System make it easy to interface with industry standard (HPSS) or existing HSMs.

Such a system with three different levels of data granularity would cause containers to be of different sizes. In terms of the concurrency-throughput graph, such an application would probably use a subset of the area on the graph that would consist of the right half of the shaded ellipse.

Performance Monitoring: In many applications, it is necessary to capture

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

large amounts of data for analysis and to retain some interval of data obtained. Utilizing our example on the wireless phone service above, we may have a requirement that we wish to record the following types of attributes of each call for the purposes of establishing quality of service:

- Call start time
- Call start cell location
- Call end time
- Call end cell location
- Signal grade (A, B or C)
- Signal lost flag (Y, N)

Note that there is no requirement to store customer information - just raw call information. This data is then analyzed to assess whether there is a need to improve signal quality in particular locations. We may wish to keep 90 days worth of call data but no longer than that to establish quality of service trends.

It is easy to store up to millions of calls per week for a regional wireless phone service provider. Since there is a requirement to delete data more than 90 days old, it could be necessary to delete hundreds of thousands of records each day. In a RDBMS, deleting such a number of records also means modifying huge index trees. This will typically take hours in an RDBMS. However, if we created containers to store the call record objects for a single day, we can just drop the containers in Objectivity/DB in seconds.

In terms of visualizing this application on the concurrency-throughput graph, this would be represented as line C on the graph which is near the leftmost edge of the elliptical area.

Data Warehousing usually requires that multiple servers be used to contain, manage and query the data. With Objectivity/DB, data distribution and management are very transparent and scalability is unmatched.

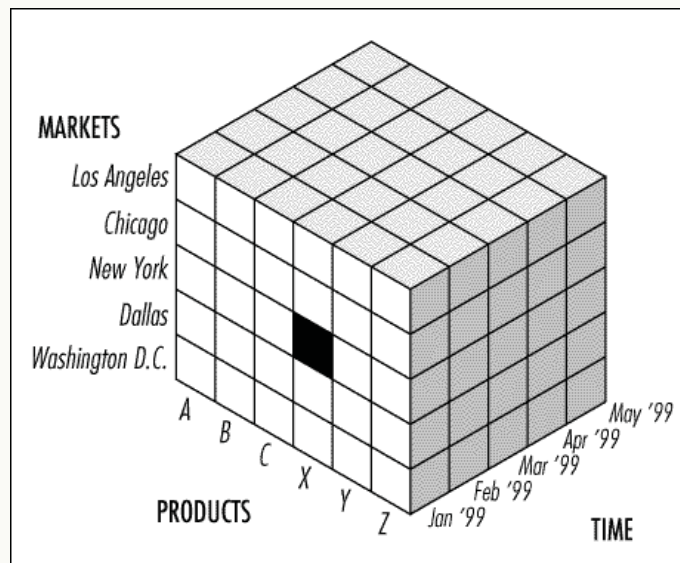
OLAP Cubes: Online Application Processing uses a multidimensional database and is often viewed as a cube. The diagram below shows the classic OLAP cube with the three axes to represent time, product and

Dynamic Containers™: The Key To Superior Performance

www.objectivity.com

Corporate Headquarters:
 640 West California Ave.
 Suite 210 Sunnyvale,
 CA 94086-2486 US
 Tel: (408) 992-7100
 Fax: (408) 992-7171

market. By making graduations across each axis, we could break up the data into smaller pieces, called “cubelets”. We can picture queries as operations on rectangular slices cut in that cube. If we have a sale of Product X which was sold in the New York market in January 1999, that piece of data is then in the cubelet which is at the intersection of Product X, January 1999 and New York (see blackened area in diagram).



If we wanted to analyze data in just these three dimensions, you can effectively store all data in cubelets and make each cubelet a container in actual implementation. This natural partitioning of data into a cubelets/containers simplifies searches as well as analysis since we do not have to scan through large tables as in a RDBMS. A new product means we have to create a new array of containers in which to put the data. A scan over the time history of a given product in a given market would mean that we would retrieve only the relevant containers for that product for a specific market. We would not be scanning through and filtering non-qualifying data at all much less bring them from disk to main memory. Of course, in reality we would use more dimensions and therefore the cubelet could be viewed as a cube in itself with its own dimensions and implement containers at that level or we could implement additional dimensions in terms of data structures within the container. Combine this with the fact that indexes can be created within a

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

container for data only in that container and you have a fairly good query capability even if you did not design in a dimension initially. Objectivity/DB also excels at scanning data even if it is not indexed and this makes it a strong contender when you wish to perform OLAP or multidimensional database type work. In fact, the time it takes to scan a flat file and an Objectivity/DB container with the same data is not significantly different.

Whatever the actual design or implementation, we have within Objectivity/DB a powerful architecture utilizing containers.

The Coherent Networks Success Story: Coherent Network International (CNI) is a company that provides products and services for data integration and knowledge management to the telecommunications and utility industries. Their SmartMaps Solutions Framework enables customers to get a single view of the many pieces of disparate and often incompatible data that most organizations in those industries possess. These separate sources of data may be CAD drawings, inventories and schedules on paper, legacy databases on mainframes and ODBC accessible databases.

CNI's object model groups metadata, state information, geospatial data and results of data validation in logical clusters. This hierarchy of related objects maps directly to Objectivity/DB's container based storage hierarchy. For example, a query on a telephone pole's location does not return just data about that pole but about all the objects that are spatially proximal to that pole including cables, wires and nearby poles and exchanges. This natural conceptual grouping of objects is put into a single container which makes the query extremely efficient and fast, since it is mostly restricted to searching through a single container. Adjacent map sections are mapped into containers of their own so that when you traverse across a section boundary, you also go to a nearby container.

A single customer may serve several different geographical regions and each region is mapped into an Objectivity/DB database. This distribution into the Objectivity/DB databases is a good physical and logical separation of data. This is a case where the database software component of the project was not treated as a black box at the design stage and the result has been a superior product.

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

Space Telescope Institute: The Hubble Space Telescope supplies images and spectral data to scientists around the world. The second Guide Star Catalog or GSC II, is an all sky survey which is primarily used to verify the astrometric positioning of other observational data, including data obtained from the HST instruments. This survey uses Objectivity/DB as the storage engine for the enormous amounts of catalog data produced by the survey. The current federation is 500 Gigabytes, and, when completed, the final catalog will comprise 4 Terabytes in 32 thousand distinct databases.

The survey data is scanned from photographic plates (which may contain millions of astronomical sources) and divided into regions of sky at the Space Telescope Science Institute. Each, individual region is stored within Objectivity/DB as a single database. The GSC II catalog uses a sophisticated tessellation scheme called Hierarchical Triangular Mesh (HTM) to partition the sky into more manageable units, which in practice are spherical triangles. The astronomical sources contained within the triangles are all stored within a single Objectivity/DB container. As a result, each region/database contains multiple triangles/containers which in turn contain all of the catalog data for the measured sources, such as name, position, brightness, and an image identifier. The actual implementation of the HTM uses the principles of Computational Geometry and involves specialized quadtrees to index the large and complex databases.

This natural use of Objectivity/DB's containers and databases enable proximity, and other complicated spatial queries to be quickly executed. These same queries would be impossible or prohibitively expensive using the architectures of other ODBMS or RDBMS vendors. As an example, all sources which satisfy a specific proximity query might actually be contained within a single container. This clustering at the container (or at a larger scale, the database) level is what enables queries to be extremely fast. This container architecture, therefore, is the dominant reason why Objectivity/DB was selected for the persistent implementation of the HTM within the GSC II project.

CONCLUSION: We hope that this paper has enabled you to understand a feature of Objectivity/DB that is one of its greatest strengths. What may appear initially as bump in the smooth road is really a stepping stone that

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

enables the software engineer to solve problems more elegantly. We have shown through examples that combining traditional thinking and design methodology with Objectivity's container architecture results in powerful and unique solutions that you may not have realized were possible. Furthermore, with the flexibility offered only by Objectivity/DB in the aspect of locking granularity versus throughput, you can customize your application in ways impossible with other ODBMs. By taking advantage of the container architecture, high throughput, storage flexibility, security and scalability are much more easily obtained when compared with other architectures.

APPENDIX: Container Pools Pattern Implementation

This section gives a sample implementation of the Container Pools Pattern discussed earlier in this white paper.

Object Creation/Deletion Abstraction: The main area where encapsulation is key is in the area of object creation and deletion. The application developer is used to having the new operator to create objects. We will prevent this by making constructors protected.

The first step is to make the constructors for your persistent-capable classes protected to prevent the application developer from using them directly. Next, you write static create methods for each of these classes which create objects and return handles to them.

Example:

```
class Data_node : public Graph_node {
public:
    static ooHandle(Data_node)& create(ooHandle(Data_node)& inout,
                                     Const ooVString& value);
    void send_results();
protected:
    Data_node(ooVstring s):Graph_node(s) {}
private:
    static ContainerPool pool;
};
```

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

```
ooHandle(Data_node)& Data_node:  
:create(ooHandle(Data_node)& inout, Const ooVString& value)  
{  
    inout = new (pool.pickOne(false)) Data_node(value);  
    return inout;  
}
```

In the example above, the application developer would call the create method and get back an ooHandle to the new persistent object. Note that the create method as implemented above delegates the responsibility of the physical placement of the object to the pool object which is of type ContainerPool. This object's sole function is to return a useful container whenever its method pickOne() is called.

The advantage of doing abstraction of the creation and deletion of these persistent objects give you three advantages:

1. No need to use containers by the application developer
2. To change the way an object is created/destroyed, you only need to modify the create or destroy method
3. It is easy to search for code where you are creating or destroying certain objects by searching for classname::create or classname::destroy

A ContainerPool Class

The pool object above is of the ContainerPool class which we will now discuss. This class is available for download from the Objectivity InfoCenter, and can be used as-is, or adapted to suit your own needs. This class is general-purpose enough to permit distribution of objects across multiple hosts, directories and databases. There is a simpler facility for managing pools of containers in a single database, which is part of the d_session class, which is also available on the InfoCenter.

This particular ContainerPool class can be persistent (embedded in another persistent-capable class) or transient. Note that all containers themselves are persistent but the pool object itself can be persistent or transient. A

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

Dynamic Containers™: The Key To Superior Performance

persistent container pool does not have to be reinitialized each time a new process needs it and can be shared among multiple clients. However, if it maintains state information such as how many times a container has been requested from it, or which is the next container to be used, there could be lock contention because you need an update lock on the container pool each time you call member functions which change the state information. If multiple container pools exist which effectively “share” the same real pool of containers, this is perhaps preferable to having to share the single ContainerPool object.

Each ContainerPool instance holds onto an ooVArray(ooRef(ooContObj)) and each call to pickOne() returns an ooRef to one of the container in this array. The actual rules to determine which container is returned can depend on your own pickOne() implementation. For a more intelligent distribution mechanism, you can subclass ContainerPool and add more pickOne() methods which take different kinds of arguments including objects for clustering hints to the ContainerPool.

The ContainerPool class must be initialized once during its lifetime with the init() member function. This member function takes three forms.

1. A reference to an existing container which causes pickOne() to return the same container for all subsequent calls. This is useful for simple single-user benchmarks.
2. A name, number of databases and number of containers per database so that init() can create them.
3. A dbspec object, where you must pass an object that implements the following interface:

```
class Dbspec {  
  
public:  
  
virtual int numDBs() const = 0;  
  
virtual const char* host(int I) const = 0;  
  
virtual const char* path(int I) const = 0;
```

Dynamic Containers™: The Key To Superior Performance

www.objectivity.com

Corporate Headquarters:

640 West California Ave.
Suite 210 Sunnyvale,
CA 94086-2486 US
Tel: (408) 992-7100
Fax: (408) 992-7171

```
virtual const char* sysname(int I) const = 0;
```

```
};
```

As long as your derived class supports this interface and returns valid hostnames, pathnames and so on, you can implement the class in any number of ways. Using this form of init allows objects to be distributed across multiple hosts, as well as multiple databases. To change the distribution scheme, you only need to modify your own Dbspec-derived class.