Object Servers
A new look at external computer storage

By Samek Mokryn

Introduction
For the last fifty years a majority of computer storage systems were still based on record servers. However, the role of such systems and the challenges they faced, changed rapidly over time – from being an extension of internal computer storage to becoming an on-line depository of vast amounts of data. As no information was provided to storage servers about data content and its organization, it became very cumbersome to manage, backup and conduct data searches on these systems. As a result, storage systems are very dependent on applications and application servers utilizing their data.

In order to solve these problems we propose Object Servers, who can function as data object depositories that organize, search and backup the objects autonomously. Object Servers can be described as huge automated data warehouses, where each package contains some data, all referred to by its object ID, equivalent to a package tracking number. These packages can be of various sizes, from small records to large video files.

The main difference between the Object Servers and existing storage systems is the level of abstraction in reference to data – data objects versus atomic bit records. This extended view provides an Object Server with new capabilities and functionality, raising the role and importance of data storage to a completely new level.

Target Applications
The Object Server is intended to store final, rarely changing data objects like web pages, emails, pictures, documents and video files. This is a *heterogeneous*, shared storage system, where the same data is accessed by many communication streams or application servers. Its definition assures a coherent view of the data, allows storage systems to efficiently perform data access control function, object searches with various search criteria, and provides data administrators with a very efficient backup facility. Other data mining applications can be added on top of the Object Server, allowing for views of the data not possible by other means.

Semantics of Data Object
There are three basic elements of the Data Object:

1. Object ID
2. Object Descriptor
3. Data content

<u>Object ID</u>
Object ID is a system-unique number assigned by the Object Server at the moment the Data Object is created. All subsequent operations use this number as an object identifier.

Object Descriptor

Object Descriptor contains all relevant information about an object. It contains the Data Object name, originator ID, destination address (individual, group or list ID, which is an object in its own rights), search keys, replacement policy and other relevant information. If needed, the Descriptor can point to another object which contains a supplementary descriptor.

Data Content

Data Content is provided by authoring applications and cannot by modified at any time. In its basic application, the Object Server doesn't look into this Content, except for its own Control Data Objects such as lists of destination addresses.
It is possible for the Object Server to encrypt the data and handle all the security aspects of the object.

Typical data flow

1. Authorized Application Server delivers the Data Object to the Object Server.
2. If the operation is successful, the Object Server returns the ObjectID it assigned to the Application Server, otherwise the operation is rejected.
3. Any Remote Application requests a search of objects based on the object's name, search keys, owners' IDs or any other relevant information.
4. The Object Server returns list of Object IDs that satisfy the search criteria.
5. The Remote Application requests a delivery of the Data Object based on its ObjectID.
6. If allowed, the Object Server delivers the Data Content of the Data Object to the Remote Application.

Please note that steps 3 and 4 are optional. If any application acquired the ObjectID by other means, it can attempt to access the Data Object at any time.

Data consistency and integrity

Data Objects are always consistent (never partially old and partially new as a result of a write in progress). Since the objects never change, there is no need to synchronize them to the applications which create the data, otherwise needed to assure the storage server that all data change ended. This synchronization (Sync Points) is application or system specific, and creates time dependency between the storage and applications. This dependency is very difficult to manage especially in large heterogeneous storage systems, which serve multiple applications and users, due to the lack of standardization and proprietary control interfaces.

In the case that an object is being replaced (if a replacement policy of such object allows it), the original version of the object is deemed current until all accesses to this object stop. When the new version replaces the old version of an object, all relevant pointers are updated, the replacement is acknowledged by the Object Server to the authoring Application Server and access to this object resumes. This operation is completely internal to the Object Server and as such doesn't affect any external applications.

Data integrity is another important characteristic of the Object Server's concept. Even if it is extremely rare, it does happen that sometimes storage servers provide users with wrong data (data miscompare) as a result of an internal error or write error. Object Servers can eliminate this problem by providing an internal data object verification policy by creating an object's checksum on receive, and verifying it each time the object is being delivered to users, assuring them that the right data was delivered. Again, due to the fact that objects are never partially modified, such error checking can be easily implemented. Please note, that the worst scenario in all storage applications is to deliver wrong data without an error notification.

Backup and Restore

The simplicity of the Backup process is one of the greatest advantages of the Object Server concept. Since the data always remains consistent, there is no need to delay the backup until the data is synchronized. Furthermore, there is no need to keep separate (and expensive) data snapshots of the frozen, synchronized data, in conjunction with the current data.

All backups are incremental and only the relevant objects are stored. These backups can be performed concurrently with the Object Server delivering and accepting new data, making the backup process completely autonomous from the data traffic and never interrupting the data delivery service. Finally, the data backup can be performed continuously. As a result, the "backup window", i.e. the difference between the current data and the last backup performed, can be reduced to nearly zero, drastically reducing the risk of data loss as a result of a system failure.

Object Servers eliminate the need for expensive and wasteful volume snapshots and backups.

Deletions are handled by maintaining and backing up the delete lists (which are also Objects), without the need for vast volume backups as are required today on record servers. Restores are handled by starting with a backup from a particular date, while referencing the delete lists created after the initial backup.

Object Servers even maintain the ability to perform the restore process out of sequence, restoring the most important (high priority) data objects first, and as such dramatically reducing the restore window.

There is really no need for periodic system backups, which due to storage system size can be quite time consuming and expensive.

Performance

Generally, the performance of any storage system depends on its ability to adopt existing resources to traffic patterns and minimizing operations required to provide and maintain data. This ability strongly depends on a system's information about the data organization and its global view on traffic patterns from all users.

The Object Server concept gives the storage system much more information about the data than any other storage system architecture. This information allows for very efficient resource assignments (like caches, descriptor and control lists, etc.). There is no need to copy, prefetch or otherwise manipulate temporary files, empty space or other information that is not necessary for current operations. All these unnecessary operations burden system resources and can drastically slow the system under very heavy load conditions, at the exact point where performance is really important. While all modern systems can perform relatively well under light load conditions, these are not the conditions large external storage systems are intended for.

Additional performance gain can be achieved for traffic patterns that require search operations, since all searches are performed internally, without the need to send the descriptors, pointer lists or even the data to external search engines. Since the Object Server serves all user communication, the integration of the communication, storage and search engine in one place will always perform better than a system where all of these functions are dispersed between various system elements.

In addition, the Object Server stores Data Objects in the media sequentially. There is no file fragmentation; neither is there a need to perform file defragmentation. Please note, there is an order of magnitude performance difference between the disk accesses to sequential data versus random data. Currently, data fragmentation is also common in file servers, which store modified blocks in different places on the disk and perform additional mapping, resulting in a very poor read performance of modified files.

Cost of Ownership

The Cost of Ownership is greatly reduced due to better system utilization than other solutions. First, there is no need for expensive volume snapshots, which can double the storage (and cost) requirements for the size of the system in relation to the desired data storage capacity.

System utilization is increased further due to the knowledge that the system possesses about the data. No storage is wasted for temporary or deleted files. Due to the capability of continuous backup and fast restore, it becomes possible to save on the substantial cost of the redundant storage systems. In addition, seldom used objects can be removed to a cheaper secondary storage and brought back only when specifically requested. In other words, the Object Server can perform the hierarchical storage function.

Cost reduction also stems from the integration of search, storage and communication facilities being in one place, where the savings come from integral resource utilization, shorter communication paths and the advantages of scale.

Further, additional savings come from lower maintenance costs. Today, data management, especially in heterogeneous, often incompatible systems, is tremendously difficult and expensive. A higher level of system intelligence, function integration and the uniform view of data objects, allows for much easier data management, subsequently reducing maintenance costs.

Relationship between Data Objects
In its initial form, the Object Server doesn't maintain any relationship between Data Objects, except its own. However, it will be possible in future versions to maintain relationship lists that can be very useful during data searches.

Comparison to file servers
The Object Server is not a file server. It doesn't allow for the modification of data already deposited, and as a consequence it doesn't maintain a file pointer – the basic construct of file servers. The descriptors also contain different information than the file control blocks.

Final Thoughts
The Object Server is a brand new storage application designed to share data in a heterogeneous environment with a simple application interface. It trades the data updates for data consistency and maintains independence from applications.

Due to the fact that the storage system now has information about the data, it allows the system to manipulate the data effectively, which provides a basis for future, more sophisticated applications.

About the author:

Samek Mokryn is the founder and president of HalStor, Inc. (www.halstor.com), a design and consulting company specializing in storage and communication interfaces and architectures. He founded HalStor, Inc., after more than 30 years of experience in product development, design and project management.

In 1996, Samek founded C-Star Corp., a company that focused on the development of new data technologies. C-Star later became SANgate Systems - a storage appliance company - where he invented SANgate Systems' breakthrough technologies for managing stored data. SANgate Systems later became Sepaton, Inc., (http://www.sepaton.com) currently located in Marlborough, MA.

Samek previously held senior-level technical positions with EMC Corp., including leading the development team for ESCON connectivity within EMC's Symmetrix Information Storage Systems. He also influenced EMC's product and marketing policies. Before joining EMC, he served in various development roles related to I/O controllers and computer systems.

Samek holds an MSEE degree from Columbia University and earned a BSEE degree from Technion of Haifa, Israel.
He holds two patents in the area of a data replication.