nexenta™

White Paper

Object Storage: From Niche To Mainstream

By Alex Aizman, January 2014

Table of Contents

Object Storage Overview 3

Object Storage: From Niche to Mainstream 3

- Object Storage Services 4
- Object Storage Versioning 4
- Object Storage Model 4
- CAP Theorem 4
- Directory Hierarchy 4

How Is Object Storage Accessed 5

Will Object Storage Become Mainstream? 5

Will OpenStack Swift Gain Adoption? 5

Intel's Approach to Object Storage 6

When Does Storage Become Interesting? 6

Object Storage Overview

Object storage, also called object-based storage, is a generic term that describes an approach to addressing and manipulating discrete units of storage called objects. Like files, objects contain data -- but unlike files, objects are not organized in a hierarchy. Every object exists at the same level in a flat address space called a storage pool and one object cannot be placed inside another object.

Both files and objects have metadata associated with the data they contain, but objects are characterized by their extended highly-customizable metadata specific for a given user/application. Each object is often assigned an identifier – there's currently a great variety of ways those identifiers are implemented. This enables a server or end user to retrieve the object without needing to know the physical location of the data. This approach is useful for automating and streamlining data storage in cloud computing environments.

Object storage is often compared to valet parking at an upscale restaurant. When a customer uses valet parking, they exchange their car keys for a receipt. The customer does not know where the car will be parked or how many times an attendant might move the car while the customer is dining. In this analogy, a storage object's unique identifier represents the customer's receipt.

OpenStack Swift, also known as OpenStack Object Storage, is an open source object storage system that is licensed under the Apache 2.0 license and runs on standard server hardware. OpenStack Swift and its derivatives can be deployed both as bare-metal installed servers and virtual machines. It is best suited to backup and archive unstructured data, such as documents, images, audio and video files, email and virtual machine images.

Objects and files are written to multiple drives, and the OpenStack Swift software ensures the data is replicated across a server cluster. By default, OpenStack Swift places three copies of every object in as unique-as-possible locations -- first by region, then by zone, server and drive. If a server or hard drive fails, OpenStack Object Storage replicates its content from active nodes to new locations in the cluster.

The system, which is accessed through a REST HTTP application programming interface (API), can scale horizontally to store petabytes of data through the addition of nodes, which typically equate to servers. OpenStack Swift software is based on Cloud Files technology developed by Rackspace Hosting Inc. Rackspace and NASA initiated the project and co-founded the community that develops and maintains OpenStack software, which includes compute, storage and networking components for building cloud computing services.

Object Storage: From Niche to Mainstream

Q: Is object storage a cure for failing filesystems in the face of really massive file numbers in our new world of big data?

A: There are major differences between Object Storage protocols and NAS – these differences are in fact the underlying reasons for the exponential growth of objects. They are driven in part (but only in part) by big data analytics. These file/object differences include:

Object Storage Services

Object storage services have a different model on timing updates. NAS protocols define distinct events for opening, locking, unlocking, reading, writing, synching, closing and deleting files. NAS protocols allow files to be opened for shared write access where these actions can be interleaved on a single shared file from multiple clients.

Object storage protocols define getting and putting (with variations on put for append and delete). Object storage serializes whole object access to objects. Conceptually all actions are instantaneous, they either fetch the entire object or supply a complete atomic update. Each action has “exclusive access” to that object for the instant that action conceptually takes place. Any attempt to reconcile also needs to reconcile the different timing models.

Object Storage Versioning

Related to the above is object versioning. Versioned files are typically associated with SCM systems. Objects are /naturally/ versioned.

Object Storage Model

The Object Storage model is based upon eventual consistency, which allows object storage services to operate in full compliance with the guarantees they issue to their clients even in the fact of network partitions.

CAP Theorem

Related to the above is – a so called CAP theorem, for example:

http://en.wikipedia.org/wiki/CAP_theorem

One of the related generalized conclusions out of the CAP theorem is that it is typically (with all the proper disclaimers on specific apps and use cases) easier, and much cheaper to scale objects. And vice versa, difficult/expensive to scale file based storage.

Directory Hierarchy

NAS files are typically arranged in a directory hierarchy. Objects are typically only stored in a single layer Bucket or Container. It should be noted that they have longer names which happen to include characters traditionally used a directory separators (typically “/” or “\”).

In the future, we expect object systems will add support for hierarchical (virtual) directories. This is actually quite easy: parsing those “/\” separators in object names and providing users the impression that a given bucket/container contains a hierarchy of partially named objects. Today however, objects are typically associated with a flat space.

In short, object storage supports sequence: write-entire-object-then-read—entire-object, as opposed to POSIX (and NAS on top of POSIX) open-read-write-…-read-write-close. Objects are “eventually consistent” – the property that definitely benefits applications that require massive scale and, simultaneously, do not require instant multi-user consistency or POSIX-like semantics of concurrency working on portions of the file. One such application happens to be big data analytics, there are others of course.

However, financial applications, for instance, will likely require OLTP and non-object back-ends for the foreseeable future.

How Is Object Storage Accessed

First – internal inter-server protocols must not be any of the above. As a side, we've done some work to that end under codename Replicast - storage and transport stack that enables efficient and reliable replication of storage content in a distributed and scalable system of many servers.

Refer to: http://nexenta.com/corp/products

As far as external, user and application interface – it will be a flavor of the REST API, although unfortunately Nexenta does not see the industry coming together to develop an object access REST-based standard that the majority would agree upon. Amazon S3 and OpenStack Swift are two examples of popular non-standard APIs. This (lack of standardization and lack of designing the future API for advanced features) is frustrating. However, it reflects the current state-of-the-art where the bare minimum is (for now) sufficient while advanced features like source deduplication, compression, and encryption are simply not present in the existing solutions.

Will Object Storage Become Mainstream?

Q: Should object storage be regarded as some kind of niche storage technology? Only of interest to massive data environments such as: health customers, financial services players and cloud storage providers. Or will it cross the chasm into mainstream use?

A: Object storage won't stay a niche technology. In fact, we believe that NAS will gradually relegate itself to specific applications requiring instant consistency and limited scale (see above). Bring your own device (BYOD) will remain a tremendous driver and the force behind objects, along with big data.

Backup, archive and disaster recovery will get gradually re-implemented via "objects", thus boosting the already exponential growth of the "cloudified" petabytes. Working with documents from your virtual PC or smartphone will be done via direct object backend – as opposed to, for instance, dropbox relay from your local folder where you store POSIX files. This is as mainstream as it gets.

Will OpenStack Swift Gain Adoption?

Q: What impact will OpenStack Swift have in the next year or five years?

A: The jury is still out! There are limitations and flaws in the OpenStack Swift design. The big advantage of OpenStack and OpenStack Swift, is the question "is it the right time and place for this combination". Both in terms of community and its mission to allow people build value-add Clouds. If so, OpenStack Swift has a very good chance of becoming mainstream.

OpenStack Swift fills the niche that is not covered by dropbox et al. BYOD users will eventually use direct get/put, and OpenStack Swift can be the preferred backend. Amazon, Google and Microsoft will compete for that space.

On the open source side, OpenStack Swift carries a powerful promise of a common codebase to build private/public clouds. That's definitely an asset! However, the time to consolidate has not yet come - many startups fork and change the OpenStack Swift internals. Additionally, Ceph would be a salient example of one competing technology that happens to provide external RESTful API (also called Swift) but has a totally different codebase and a different set of features, including distributed block.

The bottom line is that intensive development is happening on multiple fronts and OpenStack Swift has good chances of becoming mainstream.

Intel's Approach to Object Storage

Q: What makes Intel's approach to object storage interesting and which other main players should we focus on?

A: Intel's continuing work on the x86 architecture and instruction sets to accelerate SHA, RAID, CRC and erasure coding – is very timely and promising. Those are the functions that a storage appliance executes, generating sometimes multiple processor cycles per each stored byte. Local deduplication, for instance, uses cryptographic strong hashing - this may be SHA-256, SHA-512 or SHA-3. Therefore, deduplication definitely requires specific capabilities from the CPUs (or GPUs if available).

Other than specialized hardware acceleration, a storage server does impose requirements that are otherwise present for general-purpose computing applications. Other major CPU alternatives include AMD and ARM – in the latter, developments like Calxeda are early but quite intriguing.

Speaking of ARM and specialized board designs in general and to compete for the storage space, Intel will have to address its Atom's issues which include memory limitations, lack of GPU, power usage.

When Does Storage Become Interesting?

Q: Is this the point at which DBAs and DevOps professionals (or even pure-play programmers) start to find storage interesting at last?

A: Here are a few non-exact but plausible assessments. Almost every web application nowadays uses sqlite or mysql database for its (special application) persistence. In fact, OpenStack itself uses sqlite for its own specific persistence on both OpenStack Swift and compute sides.

Almost always the corresponding logic that reads and writes mysql and sqlite records does not require compound transactional semantics. All that is required is: bulletproof data integrity and atomic read, and atomic write behavior. Instant availability of the written data is also often required.

Which leaves us in the middle. Instead of maintaining 73 specific databases for various applications (think TCO) – why don't we use object storage for all our needs (including but not limited to – web application needs)? Let us stop paying such close attention as to where the physical storage is actually located? There's definitely a strong motivation to unify and simplify that part.