White Paper

Models for the New Era of Data Storage

Eventual and Strong Consistency

By Scott Sinclair, Analyst; and Terri McClure, Senior Analyst

June 2015

Contents

Market Challenges

The Need for a Different Storage Model

Data growth used to be the hallmark of massive data environments such as those associated with healthcare, financial services, and cloud storage providers. But now, everyone wants to access everything, everywhere, all the time—and they want to keep all this data indefinitely. Data used in this fashion lends itself to object storage, which sits well with the ultra-flexible world of cloud.

Before going any further, it is important that readers have an understanding of what object storage is and why it is an effective storage method on which to base a cloud infrastructure (public, private, or hybrid). Object storage operates differently from standard file system storage. The differences can be narrowed down to two points that are key to underpinning a cloud environment: scalability and manageability.

With a standard file storage infrastructure (traditional NAS and file servers), content is managed through a hierarchical file system using an index table that points to the physical storage location of each file, and tracks only simple metadata. This approach limits the number of files that can be managed in a single directory. Object storage data is organized into self-described containers of flexible sizes ("objects"). Each object has a unique ID (instead of a file name) with metadata that can include detailed attributes. This metadata can be used to set up automatic storage policies such as the migration of aging data from high-performance to more cost-efficient capacity-based disk or the deletion of data when it expires. Object storage offers a simpler design and greater scalability than a standard storage infrastructure, easily managing billions of individual objects. The most well-known object storage use case is Amazon S3. The scalability and manageability of object-based storage make it a core requirement for these types of hyperscale environments.

Object storage lends itself to two general strategies for providing a balance between performance and resiliency: eventual consistency and strong consistency. Though these strategies may be relatively new constructs to many IT organizations, they are proven architectures that are already helping to meet the market need for dealing with unprecedented data growth. Each of these strategies has its own definition, appropriate use cases, and tradeoffs, all of which need to be understood in order to appropriately identify how organizations should apply technologies that use these strategies to their storage-defined infrastructures. Eventual consistency and strong consistency both have their place in helping companies deal with the rapid growth of unstructured data.

Rapid Growth of Unstructured Data

Rampant growth of data and the resulting effects on storage infrastructure continue as significant problems for IT. In ESG's *2015 General Storage Trends Survey*, IT decisions makers were asked to characterize the primary storage challenge for their organizations, and the top response reported by surveyed organizations was the rapid rate of data growth.

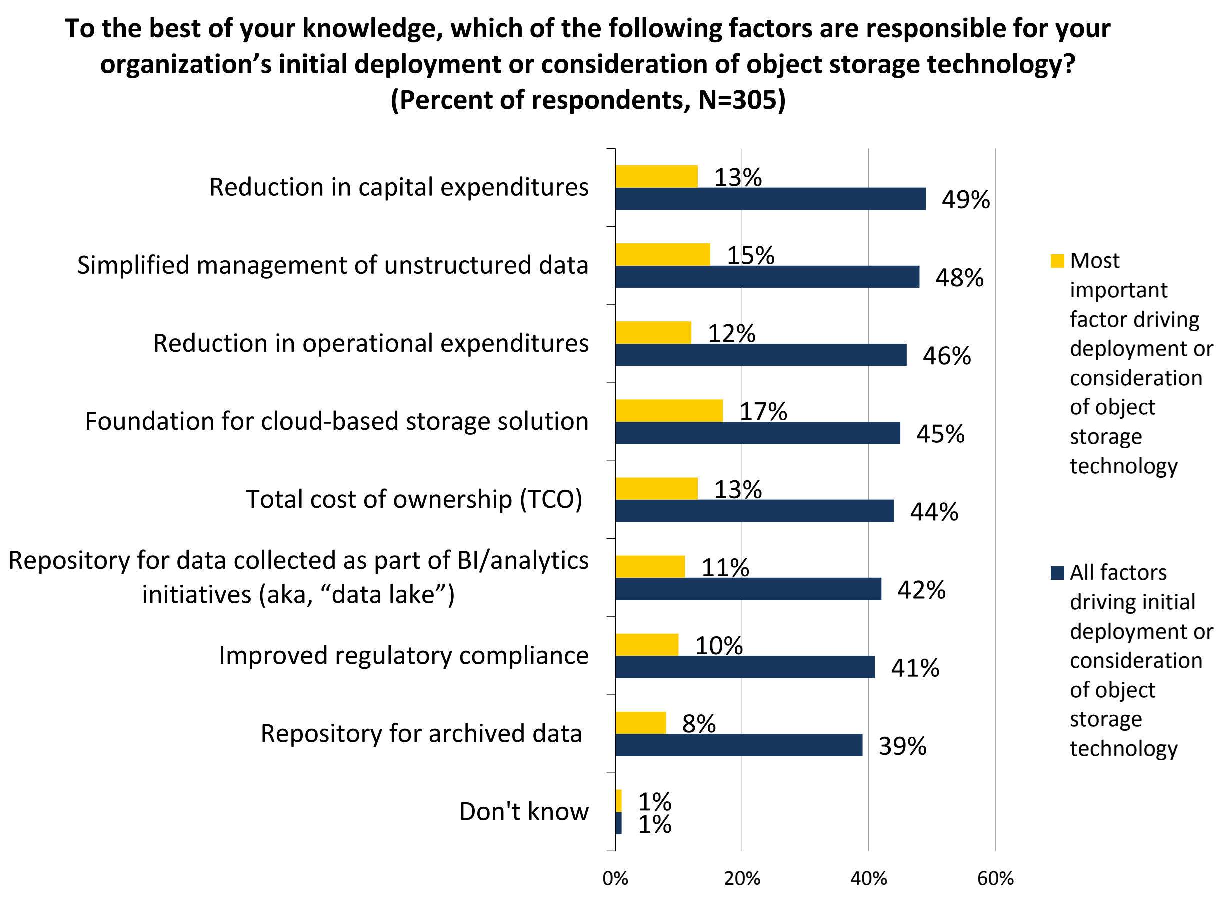
In this new world of IT, fast growing unstructured data environments demand scalability because huge amounts of unstructured data—such as videos, music, images, and files—must be stored *cost-effectively*. This trend is being driven even more forcefully by the price pressures of storing and protecting unstructured data over time on traditional storage systems.

The choice of storage architecture can have a significant impact on helping IT organizations deal with these challenges and, in some cases, they look toward object storage in an effort to stem infrastructure costs. According to ESG research, IT decision makers cited the reduction in capital expenditures (49%) and simplified management of unstructured data (48%) as the top two factors driving the deployment or consideration of object storage.[1] Additionally, organizations recognize how object storage's ability to effectively keep expenditures in check can be expanded to serve a larger infrastructure solution. Respondents to ESG's research listed object storage's ability to

[1] Source: ESG Research, *General Storage Trends Survey*, May 2015.

serve as a foundation for a cloud-based solution as the top response when asked for the most important factor driving the consideration of the technology.

Figure 1. Factors Responsible for Initial Deployment of Object Storage Technology



Source: Enterprise Strategy Group, 2015.

The Move to the Cloud Delivery Models: A New Era in Data Storage

Given the fact that just about everything is now online and expected to be available 24x7, high availability is by far the most important requirement companies have for their storage systems. This, along with high performance, scalability, and price pressure are all driving IT to look at cloud delivery models in order to satisfy data storage volume expectations.

This brings us to consistency models and why they are important to enabling web scale. Amazon Web Services (AWS) was one of the first companies to appreciate the value of eventual consistency in enabling highly distributed, unstructured data storage environments via its S3 online cloud storage service. Using eventual consistency, AWS focused on the tradeoffs between high availability and data consistency. As a result, S3 took six years to accumulate its first one trillion objects, and less than one to accumulate its second trillion—which demonstrates the staggering growth of unstructured data outside traditional storage.

Unfortunately, due to the shift in delivery model and the sheer volume of data stored in these clouds, when there is downtime, the ripple effect is felt across many departments and workgroups within the company. The cost of downtime increases in line with the users and data consolidated onto private or public cloud systems. This can lead to the point where hundreds of applications and thousands of users are negatively impacted.

The Quest to Go 'Cloudscale'

Achieving availability at scale is a complex challenge that traditional, scale-up storage systems simply cannot meet. In order to achieve "cloudscale," large multi-node or clustered scale-out systems are a core architectural requirement, but balancing performance and availability across these clusters is a difficult challenge. For example, the same piece of unstructured content may be stored in and accessed from multiple nodes, so data in the nodes must maintain some level of consistency to return the most updated data. However, keeping a large-scale cluster consistent could cause considerable latency and grind performance to a halt.

In an effort to alleviate the problems associated with achieving availability at scale, vendors have developed techniques to achieve distribution transparency, in which individual user systems appear as a single system rather than multiple collaborating systems. Back in the mid-90s, there was a growing belief that availability might be the most important characteristic of large Internet systems, but there was no consensus on the characteristic that it should be traded off against. Toward that end, in 2000, Eric Brewer, a systems professor at the University of California at Berkeley, presented the CAP Theorem, which is based on the concept that of the three properties of shared data systems—data consistency, system availability, and tolerance to network partition—only two can be achieved at any given time. According to AWS CTO, Werner Vogel, "…in larger distributed-scale systems, network partitions are a given; therefore, consistency and availability cannot be achieved at the same time. This means there are two choices on what to drop."[2] Prioritizing *availability* means that under certain conditions, the system will not be consistent (i.e., the eventual consistency model), and prioritizing *consistency* means that the system will not always be available (i.e., the strong consistency model).[3]

In order to deliver both consistency and availability, vendors like AWS, Rackspace (which has >150 petabytes of unstructured data stored on OpenStack Object Storage [code named Swift]), and SwiftStack are using an eventually consistent model to meet their data growth requirements at scale.[4] Clearly, these companies have committed to eventual consistency because, as Amazon's experience proves, it works for large-scale geographically distributed environments.

A Deeper Dive into Eventual Consistency Compared with Strong Consistency

Understanding the differences in consistency approaches can help IT administrators, architects, cloud builders, and executives decide on which model may work best for *their* workloads, and whether data inconsistencies are acceptable for their particular applications. At a high level, eventual consistency models are widely deployed in cloud infrastructures in order to manage the exponential growth of unstructured data; accommodate geographic data dispersion requirements and mobility needs; comply with performance demands; and support a large number of concurrent users. Strong consistency models, which are optimized for file systems and block storage to support operating system and database uses, are employed when data consistency is highly important for applications, such as for financial transactions or statistics. But there are a lot of elements to take into consideration when choosing which model to deploy.

Consistency Considerations

Data is in a consistent state within a storage system when any changes to that data are written to all nodes where data is housed, or, more specifically, all nodes where replicas (or erasure-coded data) exist. This is a nuance associated with scale-out systems (as opposed to more traditional scale-up systems). Any system with finite controller resources is a scale-up system and cannot scale out, whereas distributed systems in which controller resources get added with each node (along with memory and storage capacity) can scale out. Object stores are scale-out systems, which adds processing and storage capacity as the system grows—that is part of the secret sauce

[2] Source: Werner Vogel's Weblog: All Things Distributed, *Eventually Consistent – Revisited*, December 2008.
[3] For more information on the CAP theorem and its associated technology, please see the report, *Brewer's CAP Theorem*, 2012.
[4] Source: *Openstack Swift at Scale*

that allows object stores to scale. The other part of the secret sauce is how the system manages storing and protecting data.

In scale-out object stores, data is often protected by storing copies of data or segments of erasure code on multiple nodes so that if one node fails, the data can be retrieved from another node (in the case of storing exact copies), or can be rebuilt from the remaining erasure code data. Note that this does not mean every object is stored on every node. Many object stores that store multiple copies of data allow users to specify how many copies are required to meet availability requirements—typically three is the recommended amount. This means that only three of the nodes in the cluster will have a copy of a particular object. When erasure code is used, the code can be striped across many nodes—often eight or more—but not the entire cluster. For both of these models, multiple nodes can fail while data remains available, which is why multi-node scale-out systems are ideal for meeting the scale and availability requirements for public and private cloud environments.

In any scale-out system, consistency of data across the nodes is a core consideration. Consistency is the guarantee that the state of the data can be brought back to a valid state with the *most recently* completed transaction. Within enterprises, this is normally done with hardware and software designed to work together to provide the highest levels of consistency (strong consistency). However, it should be remembered that the highest levels of consistency are not always necessary; in fact, realizing a consistent state for each transaction—as with strong consistency—can be difficult at scale, especially when it comes to multi-site configurations. For example, operating systems need consistency and databases with transactions need consistency, but backup files, log files, and unstructured data do not need that same consistency. This situation has led to the emergence of eventual consistency models that can allow storage systems to scale even as they meet performance and availability requirements.

Strong Consistency: What Is It and When Do You Use It?

The state of strong consistency occurs when data is written to all nodes for purposes of protection, and reads are guaranteed to return the most recent data. In this environment, all nodes must be queried to ensure that all updates have been written to all nodes and the read is returning the most recent copy. In other words, strong consistency guarantees that the database is in a consistent state when a transaction is finished and before the next transaction can be handled.

This is how most data center storage arrays and POSIX-compliant file systems support business and back-office applications, as well as virtual machines that absolutely must return the most recent data and maintain transactional consistency across all copies. For example, when removing money from one bank account and adding it to another, the total balance for both accounts should be the same as it was before the transactions. The strong consistency model is required for real-time systems. Strong consistency offers a positive user experience when consistency of object stores is paramount, as in financial transactions or other relational database types of applications. However, strong consistency implies a compromise on the scalability and performance of the system.

Eventual Consistency: What Is It and When Do You Use It?

Eventual consistency changes the rules slightly. In this model, when data is written across multiple nodes for data protection purposes, the read returns the version it finds first, whether or not it is the most recent version. In the eventual consistency model, there is a guarantee that the system will *eventually* become consistent and have the most up-to-date version of data for all copies of the data. While that is determined, however, reads are not "held hostage" until the cluster reaches a consistent state. Eventual consistency only matters when you need to change something (e.g., overwrite all or part of a piece of data), so it can be very effective for most every data type.

Like the strong consistency model, certain use cases for the eventual consistency technology make it a good design choice that optimizes other factors, such as performance and scalability, which are particularly important for massive, highly distributed infrastructures with lots of unstructured data serving global sites. Examples include "follower" lists on Twitter, "friend" lists on Facebook, log data for scientific applications, and active archiving of 4K and HD video. In general, any use cases in which providing an answer is more important than providing the most up-to-date answer will be served well by eventual consistency. If the users of an application will not notice (user-perceived consistency) or care if updates are reflected consistently at all times, then eventual consistency makes sense for its read performance benefits and scalability.

Eventual consistency is a failure handling mechanism. The assumption is that hardware will fail, so the question is: What happens to the system in the event of a failure? Systems can either choose to be eventually consistent or strongly consistent. That being the case, if a drive has gone down, do you reject storage operations until the drive comes back up—which is strong consistency—or do you accept the operation and reconcile it later—which is eventual consistency?

How OpenStack Swift's Consistency Strategy Works

Object storage is designed to maintain consistency while managing the data within a distributed storage system. In order to do this, the metadata and the controllers communicate to ensure that the metadata is all up to date and that all objects are written to as many nodes as required. For example, if someone updates her profile for a file sync and share application, and the company policy is to have three copies of the data, OpenStack Swift writes it to Node A, Node B, and Node C all at the same time. That way, the metadata and the system are aware that a state of consistency has been reached. However, if the drive where the data resides on Node A subsequently goes down, then the auditors within the system comply with the company policy of maintaining three copies (eventually) and instruct the system to replicate the data to either another location on Node A or another available node that meets the protection policy.

In the case of an eventually consistent model, if your LinkedIn profile gets updated and that information goes to Nodes A and B, it can be read without having to wait for Node C to be updated. By way of comparison, with a strong consistency model, the user has to wait until Node C is updated before being able to read the information.

Within the context of this information, time is relative, and "eventual" can be measured in anywhere from milliseconds to minutes or longer, depending on many factors (e.g., size or geographic distribution of the cluster, object sizes, available bandwidth over a WAN, or how busy the system is). So while a time lag may exist as the system updates to ensure all three copies are consistent when an update is made, the lag likely will be short enough that it won't matter in some applications such as unstructured data, Amazon S3, or Dropbox files. Even in the rare and unlikely case where a read for a specific piece of content is requested prior to an update of that specific content achieving full consistency across all three copies, it is highly unlikely that the read will access stale data. Object systems are designed to service reads from the fastest responding node housing a copy of the data. It is extremely rare that the fastest responding node would also be the slowest to complete an update. For example, latency's largest impact is seen with remote data centers. Object storage systems are typically designed to have live users read data from the local nodes that are already in sync and not from the remote nodes residing in a remote data center where latency may be a factor.

The Pros and Cons of Eventual and Strong Consistency

It's important to know when to use each of these consistency models and what some of the tradeoffs are for each, as listed in Table 1. Eventual consistency gets the nod when it comes to robust scalability because it is not subject to the vagaries of latency associated with multi-node and geographically distributed data sets. That makes it attractive for fast-growing unstructured data stores and next-generation applications, which can operate well even in rare instances of receiving "stale" or outdated data that might be only milliseconds old. Strong consistency is preferable in highly structured environments such as financial institutions and for supporting virtual machines where the risk of receiving outdated data creates unacceptable business conditions.

Table 1. The Pros and Cons of Eventual and Strong Consistency Technologies

Features	Eventual Consistency	Strong Consistency
Supports robust scalability	Yes	No
Always reads most recent data	No	Yes
Good for geographically distributed data sets	Yes	No
Primary storage for relational databases and virtual machine environments	No	Yes
Snapshots, backups, and images of relational databases and virtual machine environments	Yes	Yes
Supports unstructured data sets and next-generation applications	Yes	No

Source: Enterprise Strategy Group, 2015.

The Bigger Truth

Never in its history has the IT industry been engulfed in such a tsunami of change with so many challenges and opportunities as it is now. Legacy infrastructures are buckling at their seams as more, and seemingly ubiquitous, bandwidth enables new delivery models. Spurred on by market-changing forces such as mobility, virtualization, and cloud computing, enterprises are updating their IT infrastructures to provide greater efficiencies and accommodate changing consumption models. This change can't happen soon enough as IT is now essentially competing with cloud service providers to provide technology services to their organizations. Business stakeholders are asking for fast, flexible services with the same properties and (perceived) price points as cloud service providers such as Amazon. Through it all, enterprises have no choice but to maintain a competitive pace in the global marketplace.

Storage strategies have become critically important as users struggle to keep up with the exponentially expanding speed of unstructured data growth. Deploying the right storage model with the right consistency model is key to transforming the storage infrastructure to meet today's needs: It is how IT can help its company achieve new levels of scale and flexibility at a much better cost point than using traditional storage.

Eventual consistency—which guarantees that the state of data can *eventually* be brought back to a valid state with the most recently completed transaction—is not a high-profile player on this new IT stage, but it is a proven technology that, for many years, has been playing an essential role in transcending scalability limits and enabling high-availability and high-performance environments. Given the runaway numbers of objects organizations are facing and the high growth of users accessing data, its value has never been higher. For example, Dropbox, which supports millions of concurrent users and billions of objects, is a model that requires eventual consistency in order to function.

Operating as it does against a backdrop of clustered, scale-out architectures in cloud-based environments, eventual consistency is not perfect because like other forms of consistency—most notably strong consistency—it requires tradeoffs that impact the speeds at which data is returned at the end of transactions. As such, it introduces the possibility of "stale data"—a real but minimal risk considering that the time to consistency is typically measured in milliseconds (longer for geographically dispersed systems)—which may not be acceptable in various operating environments, such as financial services. Strong consistency is not going away—it will continue to be used for virtual machine environments, financial services, and many database applications. However, choosing right and optimizing technologies for the correct use cases is what helps meet the challenges of today's applications. For example, you wouldn't want to use an all-in-one tool like a Swiss Army knife to perform surgery, when tools specific to the type of surgery are what you need, i.e., tools that are optimized for that use. Similarly, it's important to know when it is appropriate to use eventual consistency or strong consistency.

Like strong consistency, eventual consistency comes with tradeoffs and isn't the solution to every storage problem. However, given the scalability, performance, and high-availability demands users face in this new and challenging world of IT, eventual consistency is a proven and required technology that gets the job done.