SCTE·ISBE CABLE-TEC EXPO 2019 NEW ORLEANS, LA SEPT. 30-OCT. 3

2019 Fall Technical Forum SCTE·ISBE • NCTA • CABLELABS

Innovations in Software Defined Storage for Cloud DVR and IP-Linear Digital Video

A Technical Paper prepared for SCTE•ISBE by

Stephen Blinick
Sr Director R&D and Principal Architect, Storage Products
Vecima Networks, Inc
4375 River Green Parkway Suite 100
Duluth, GA 30096 USA
520-762-6763
Stephen.blinick@concurrent.com

Table of Contents

Title	Page Number
Table of Contents	2
Introduction and background	3
File-based storage vs object storage	3
Live-linear IP TV & Private Copy cDVR: Two emerging use cases and requirements	5
Challenges and requirements for object storage systems	6
Flash technology for object storage	7
Hybrid object storage	7
Experimental results	9
Conclusion	13
Abbreviations	14
Bibliography & References	14

List of Figures

Title	Page Number
Figure 1 - Data Access Methods	4
Figure 2 - Graphical description of IPTV stream and object storage	5
Figure 3 - Hybrid object storage using flash and disk with egress from both tiers	9
Figure 4 - Average commit/flush latency of an HDD-only object storage system	10
Figure 5 - Count of "long tail" latency events for an HDD-only object storage system	10
Figure 6 - Average commit/flush latency of a flash-only object storage system	11
Figure 7 - Count of "long tail" latency events for a flash-only object storage system	11
Figure 8 - Average commit/flush latency comparison of HDD-only, flash-only, and hybrid object storage systems	12
Figure 9 - Count of "long tail" latency events from HDD-only, flash-only, and hybrid object storage systems	13

Introduction and background

The emergence of object storage as a mechanism to store digital video has modernized the way video content is stored and distributed. Whereas legacy systems that utilize a traditional POSIX filesystem architecture store video assets as files within a directory tree, object storage allows for an asset to be stored as a "data blob" in a potentially large namespace. This method of storing data introduces a number of advantages for digital content storage and distribution.

Filesystems have been around for decades. While many advancements have been made allowing them to scale, their general purpose architecture means that they're designed to work for many different kinds of content, including databases made up of rapidly updated large files, logs that need constant updating and trimming, and various applications that need many of the legacy operations that are part of the POSIX specification.

Object storage, on the other hand, is a relatively modern method of storing and accessing data. While there are many different object storage platforms and protocols, typically all of them utilize a simplified command set that includes creation, reading, and deletion of objects. An object can be of any length, and typically is stored in a "flat" namespace with all other objects, only referenced by its unique name. In some cases, a single-level grouping of objects into namespaces is employed, typically called a "bucket" or "container".

This paper discusses the requirements of modern IP-linear and cloud DVR systems that utilize object storage, the advantages and challenges with object storage, and how those challenges are met with hybrid object storage technology.

File-based storage vs object storage

Object storage has gained wide adoption for digital video content storage and distribution largely due to its many benefits: scalability, performance, modernization of application interfaces, and cost efficiency.

Improved scalability is achieved with object storage realized through the way data is stored and located. Traditional file systems store files in a directory tree. In order to locate some content, the application must recall the set of directories under which a group of files is stored. In addition to this, there are often limitations to the number of files allowed in a single directory, often in the tens of thousands. Once a directory contains too many files, the application software must "split" the directory by moving half of the files into another directory and the other half into two new directories linked underneath the current directory. Additionally, many legacy storage systems based on block storage may present only one block device to the client application. This client application then typically "formats" the block device with a filesystem in order to store and organize data. The same limitations inherent to a filesystem exist in this scenario, with the added complexity and limitations of defining a rigid block device that only one client can typically access at once. In contrast, object storage systems are accessed via a "key-value" interface, where the unique name of the object allows direct access to the content. Modern object systems can scale to many hundreds of millions of objects without adding complexity to the namespace.

Figure 1 graphically describes the different methods through which video data is accessed, and the layers within the data path between the application and the storage.

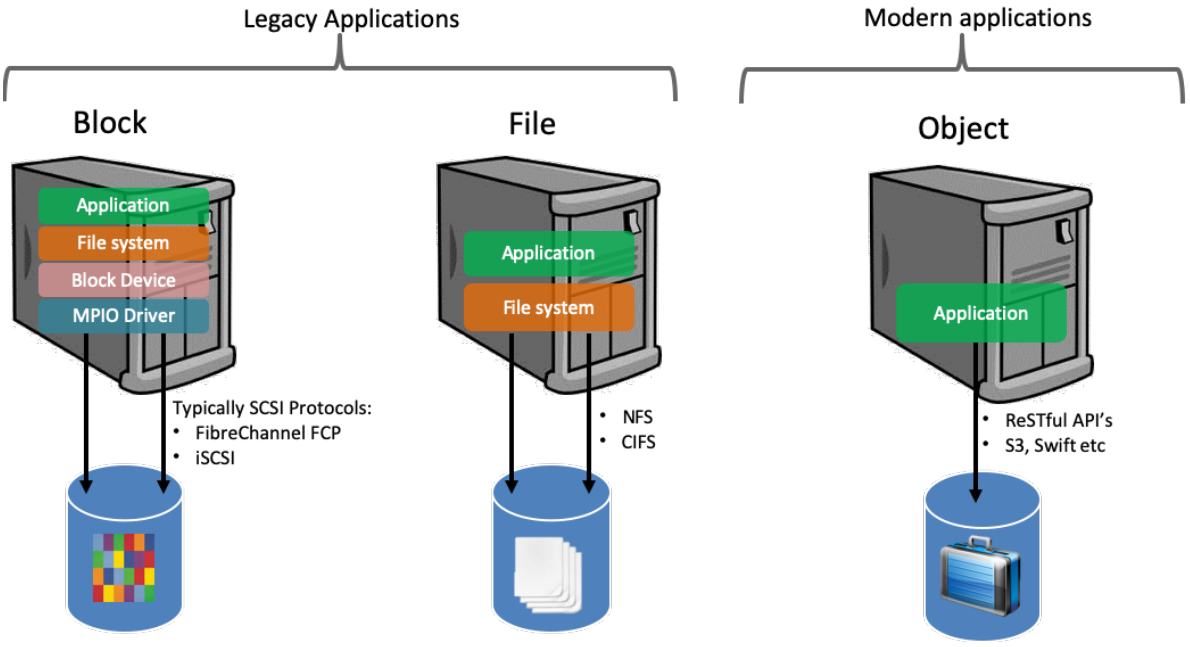

Figure 1 - Data Access Methods

Video content performance is improved when using object storage, when compared with legacy file systems. One reason for this is the aforementioned ability to scale and directly access content without the overhead of the legacy directory structure. Another reason is that legacy filesystems are typically stored on top of traditional block devices (such as a disk drive, or a SAN storage LUN). Each file system traditionally has a finite amount of space based on the size of the block device. Over time, as files are created, deleted, and moved, the finite space is consumed, the filesystem becomes fragmented and performance can degrade. To avoid running out of space and encountering performance problems, users have to add more block devices, each with a finite size and a filesystem, which creates more filesystems underneath the application managing content. This adds complexity as well.

Object storage allows the use of modern interfaces as compared to legacy filesystems. In a traditional filesystem, an application or user must "mount" it to a client, and then explicitly perform operations to open the file, read it, and close it. Also, it is very common for content management applications to keep metadata about a given content file. In a legacy filesystem, this has to be stored as an additional metadata file stored in the same directory as the content file, or possibly in an external database. In either case, there is opportunity for the metadata to become separated or inconsistent with the data itself. Object storage systems, on the other hand, typically utilize REST API commands to access data. This means that a single operation from a client application can locate, open, and retrieve bytes of content. Furthermore, many object storage systems allow the content management application to store additional metadata directly with the object itself. This means that the additional metadata fields are always accessible and in-sync with the content object, which is very beneficial for video storage use cases.

Finally, object storage enables increased efficiency in resource usage and cost. The simplified content storage and access architecture provides increased flexibility in the hardware used for storing content. Object storage systems can be built out of a cluster of multiple commodity storage servers, which are typically cheaper than traditional NAS or SAN hardware devices. In addition to this, more flexible ways

of encoding the data (by erasure-coding or simply storing more than one copy in multiple places) permits for the use of lower-reliability commodity storage devices. As a consequence, the cost to aquire and maintain an object storage system is typically lower than traditional legacy SAN or NAS storage platforms because the data in the system can utilize cheaper hardware, and potentially occupies less space to store it in the system..

Live-linear IP TV & Private Copy cDVR: Two emerging use cases and requirements

Two emerging use cases for digital content storage and distribution are Live-linear IP-based television and the storage of many private copies of IP-based video content assets. With IP-based video delivery being increasingly embraced by service providers, and used for both broadcast and storage of private-copy Cloud DVR, the storage requirements for digital content storage have increased dramatically.

IP-based video assets are typically stored as a collection of objects, each of which represents a segment or fragment of compressed or transcoded digital video. For example, an object may contain data for two seconds of a digitally compressed video stream. Each asset may be stored multiple times, pre-transcoded at different profiles that represent different compression bitrates. This enables clients to select any level of compression bitrate at any point in the video asset by referencing segments of a given compression profile.

In the case of live-linear IP TV, these segments of video must be stored stored within a storage system in real-time as a live broadcast is being captured and transcoded. Each segment of video, stored in an object, is committed to the object storage system immediately. Various clients can then reference these objects, as soon as they are created, in order to stream a live broadcast. Once the objects are created, they can remain in the object storage system as long as there is capacity. Therefore, “lookback” of the live broadcast, which allows various clients to view the live TV stream at any point in the past, can be achieved simply by referencing video segment objects that were recorded earlier and streaming them in the correct order. Multiple clients can therefore access different compression bitrate profiles of broadcast streams either live, or at any point in the past by referencing different objects within the object storage system. The aforementioned capabilities of object storage help enable this by scaling to many millions of objects and making those objects accessible through a simplified protocol that allows direct access in a large namespace.

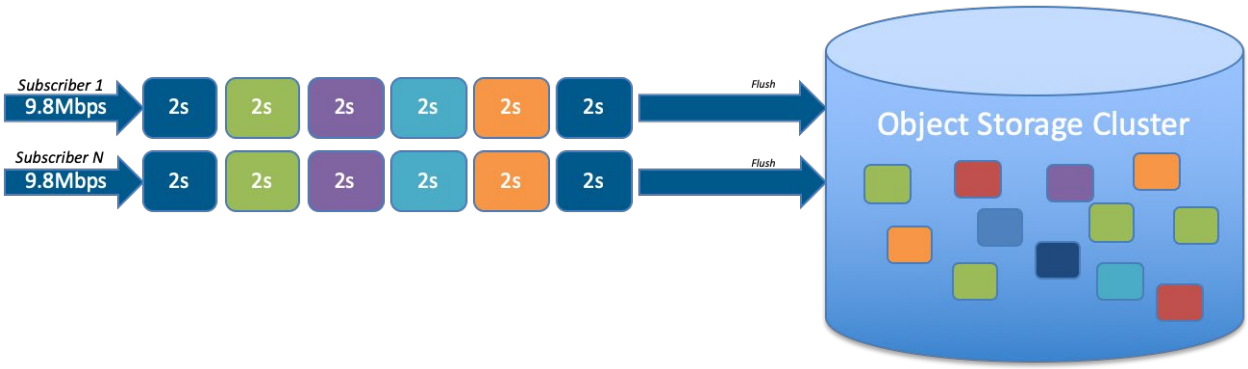

Figure 2 - Graphical description of IPTV stream and object storage

Another emerging use case for digital video content in object based storage is "cloud DVR". In this scenario, video content recorded by a customer which traditionally would have been stored in a set-top box in their home can now be stored at a centralized off-site location. If a customer decides to record a program, the cloud-DVR storage system can create a "recording" simply by saving the IP-based video segment objects of a broadcast stream that are associated with the program. The customer can then access this recording by instructing their set-top box to stream these video segments at any time. Because of fair-use copywrite law, the cloud-DVR operator cannot simply save these segments once for each program and serve them to multiple customers. Instead, the operator is compelled to individually record each segment in a unique location for every customer that wishes to record the program. For example, if 10,000 customers wish to record a program at 8PM on a given channel, the Cloud-DVR storage system has to write each video segment, in real-time, to 10,000 different sets of video streams, each with its own segments. Each customer would then 'own' a copy of the recorded program, and can watch or delete the program at a later time.

Challenges and requirements for object storage systems

Both of these use cases present similar requirements and challenges for the object storage system: scalability and latency. When there are large numbers of individual channels with many profiles (each representing a video stream), or in the case of private-copy Cloud DVR with a large number of subscribers, the number of individual object segments being written at once will increase drastically. The result is that many millions or billions of individual objects must be stored and tracked within a single namespace in the object storage system. Although object storage is well-suited to handling a large number of objects, there is inherent overhead involved at such a scale. This overhead includes internal metadata, fragmentation, and background bookkeeping tasks. When an object storage system is required to scale at these levels, the amount of internal IO operations to maintain the objects increases, adding to the workload of the system.

With regards to latency, because of the nature of real-time live video, any segment (in the form of an object) must be committed to storage immediately. That means that each object write, also known as a "put", must be acknowledged quickly every time. Each stream is an endless sequence of new object fragments, arriving at the target storage system in parallel. If even a small percentage of transactions encounter a "long tail" latency event, meaning that an individual IO in a stream of many takes several times longer to complete, queues and buffers in the application will back up and the entire recording can fail. While it's true that the absolute latency required for a video object storage system typically isn't as stringent as for a transaction processing system such as a database, at very high scale and throughput, video storage systems demand very little jitter or "long tail" latency. In fact, video storage systems as described here may require higher latency consistency than an enterprise database. For example, whereas a database may require a lower average latency of perhaps 1 millisecond, it can tolerate 95-97% of all IO operations making up this average. This means the remainder of IO's can be longer (even several times the average). In contrast, very high throughput video systems working to commit real-time information have small buffers, and while the average latency may be in the 100's of milliseconds, they require 99.9% or more of the IO's to make up this average.

In a traditional object storage system, even very large systems with 1000's of servers, the data is stored on spinning disk drives. A spinning disk drive is well suited for video object storage, because disk drives are generally very good at storing large amounts of sequential stream data. But when asked to "seek" – move the disk head between locations on the disk – the latency introduced to complete a transaction can go up by several times. The operations previously described to handle a very large-scale object storage

system receiving thousands or millions of digital video fragments such as internal bookkeeping, background scrubs, and additional metadata management all contribute to "seeks" for spinning disk drives. Traditional "single-tier" object storage sytems attempt to mitigate this by adding more disk drives to hopefully reduce the chance that any drives required for a transaction are delayed by these additional operations. Unfortunately, there still are cases where enough drives involved in an IO transaction are "busy" or performing other recovery tasks such that a few operations get delayed to several times beyond the acceptable average latency. As stated above, modern video applications such as IPTV and private-copy Cloud DVR do not tolerate even a small amount of these "long tail" transactions, and this results in failures to record a stream.

Flash technology for object storage

In contrast to spinning disk drives, flash technology (solid state media) has offered an alternative that is far faster in terms of throughput and especially latency. The general term "flash storage" refers to solid-state storage media typically in the packaging and formfactor of a disk drive. Solid state media means high-density chips that are able to be used to hold data persistently even without power. They can be programmed and erased very quickly. Flash devices come in various types and formfactors. The most common is "SSD", or solid-state disk, which is a packaged device that plugs into the same slot as and behaves like a disk drive. For example, a flash device could be a "SATA-attached SSD". The other most common type of flash device is an NVMe device. This is a set of flash media packaged into a device that connects via a PCI-express bus for maximum performance and minimal protocol complexity between the application and the device. The products that use solid-state media include an internal processor which communicates to the host system / server and facilitates read & write operations, allowing the system that uses the device to treat it as a storage device and issue read/write commands to store data to the solid-state chips. From the application's perspective, a flash device is simply a very fast storage device or disk drive. An object storage system is built out of multiple storage devices spanning multiple host systems or servers. Therfore, an object storage system built out of a set of flash devices achieves performance gains over a spinning disk-drive based system.

When compared to a traditional spinning disk drive, flash devices achieve much lower latency, faster throughput, and can be considerably more expensive depending on how costs are compared. If using cost-per-gigabyte, flash is much more expensive than spinning disk drives, often about 10x. However, when comparing cost-per-IOPS (IO's per second), the flash device becomes cheaper. This is because a flash storage device can typically achieve 700 times the amount of IOPS of a spinning disk device, at 1/2000th the latency. Clearly if there is a need for performance, it is much more economical to employ a flash device. When architecting an object storage system out of a single type of storage device however, the cost required to build it out of flash devices is prohibitive.

When considering the emerging digital content storage workloads: IPTV and private-copy Cloud DVR, it becomes clear that these workloads needs both flash device performance to handle the massive streams of small objects being written, but also the economical scale and capacity of spinning disks. A novel approach is required.

Hybrid object storage

Legacy block and file storage systems have employed "Hybrid storage" technology for many years. Hybrid storage refers to the ability for a single storage system to present a file or block device to an application, but internally use different types of storage devices to house the data. From the application's point of view, there is a single or set of files or block devices that it can use. Each one can be read or

written at any time. However, internally the storage system is moving all or parts of the data of each block device or file between different types of storage. One common example for this is the ability to move a block device or file from costlier "online" storage to cheaper "archive" storage when it is no longer in use. Another similar use of hybrid storage is to promote a block device or file from disk storage to flash storage to boost the performance of IO accesses to it. In traditional systems, the hybrid storage capability is accomplished because every access to a sector of the virtual block device, or file from the client is "funneled" through the interface of the storage system. The storage system therefore keeps an internal map that it refers to about where data resides at any given point in time and is able to internally acess the data at its current location in order to return it to the requesting client application. These traditional storage systems continue to exhibit the same aforementioned limitations even when employing hybrid storage.

Hybrid object storage refers to a similar concept: the utilization of multiple types of storage devices to house a single type or instance of data. However, in the case of object storage, there is no direct client connection such as a mounted filesystem or a connected block device to the object storage system. Therefore, there is no single directory or mapping internal to the storage system that can intercept IO operations and direct them to the different storage devices. As previously mentioned, objects are referenced by a name or key, which allows the associated data (the object's data) to be referenced directly. Therefore, the approach and challenges of hybrid object storage are unique as compared to legacy storage systems.

Because object storage systems utilize a flat namespace, they are typically comprised of a single type of storage device, spread across many independent servers clustered together. The most trivial implementation then would be to use different types of storage devices in multiple namespaces, or even multiple object storage systems. In this case, the application must "put" and "get" each object directly to the namespace or object storage system it wishes to store the object on. This means that the application must be aware of the different types of storage available, and manually control which objects are stored where. Furthermore, the client application then must "get" objects from one namespace or system, and "put" them into the other system if it wishes to move objects between the two types of storage. This is antithetical to the concept of "hybrid storage" where the application should be unaware how much and what type of devices are present in the storage system it is using. If multiple storage devices are used in this way, new bottlenecks are created because the application itself must manage the storage system (possibly requiring new metadata). This becomes complex if the application is running on a distributed set of systems and expending resources to move data between the storage namespaces.

Therefore, a true hybrid object storage architecture must meet the following requirements:

- Data location and relocation of any data must be handled without knowledge to the application
- Application interfaces must remain as they would with a single storage device type object storage system: single namespace, put, get, delete, etc
- The application must be able to create, delete, and read the objects within the hybrid system at any time, even if a relocation of the object data is in progress at time time of the operation
- No coordination among multiple clients or applications is required: One application can create objects while another consumes them so that scaling of both types of applications can happen independently.

With this capability, multiple clients and applications can achieve both high performance and large capacity to achieve a large number of streams or subscribers with a scalable retention period of the data. Large amounts of small fragments from multiple streams of object fragments are all written with low latency to the flash tier. In the case of IP Linear live TV, these objects are immediately referenced for

broadcast by live TV systems. Egress of these video streams that occurs close in time to live also comes out of the flash tier. Independent from the multiple applications, a data migration agent can seamlessly move the data after some time to a more efficient disk storage tier. Clients can stream the same data, referenced via the same object names, and the data will be pulled out of the disk tier automatically once it resides there. If a set of objects are being relocated while they are being streamed, this succeeds and the client application is unaware that the objects are moving.

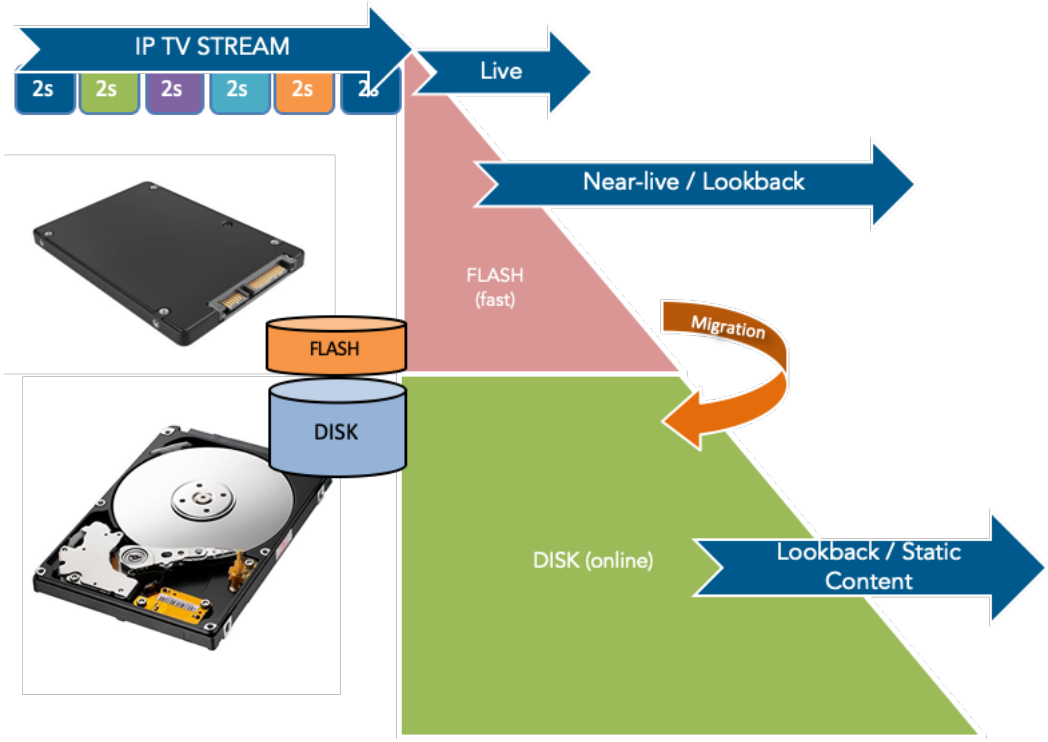

Figure 3 - Hybrid object storage using flash and disk with egress from both tiers

Experimental results

In order to illustrate the real-world performance benefits of hybrid object storage, performance measurements were obtained from a system running the "private copy cloud DVR" workload described in this paper. In this case, the system was configured in three ways: first, as a HDD-based object storage system, second, as an all-flash object storage system, and lastly, a system comprised of a set of HDD storage devices and flash storage devices, with hybrid object storage software managing both sets as a single object storage platform.

The following charts and graphs show the performance of the various configurations discussed in this paper. In each of these graphs, the focus is on latency – both the average latency over time as well as the count of IO transactions that exceed ½ a second or more. While the average latency can be a good indication of system performance, in a video stream, late delivery of even a single fragment can dedgrade or interrupt the viewer experience. We use the term "long tail latency" to describe any transactions that take over a specified amount of time. In these results, we consider the bucket that counts operations taking 2.5s or more as "long tail" latency events.

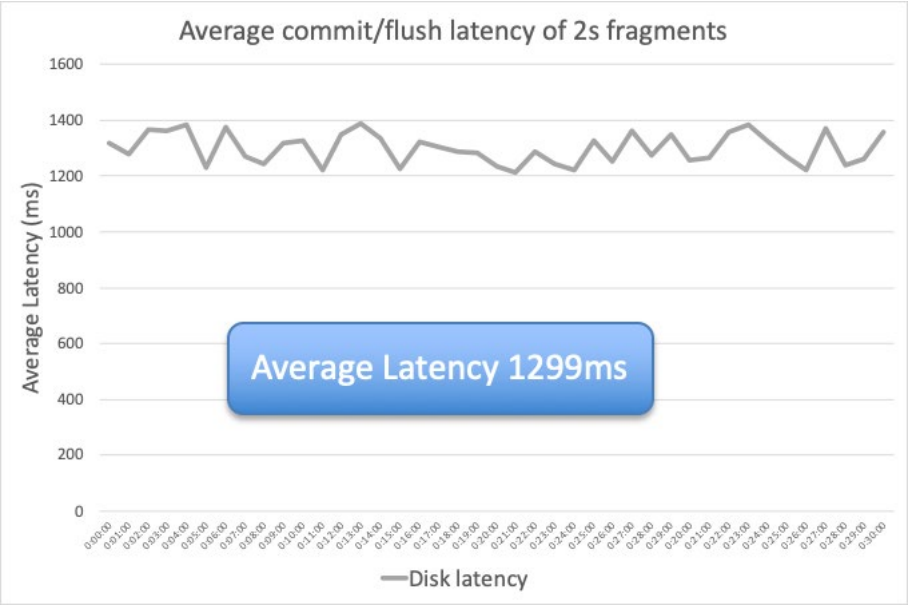

Figure 4 - Average commit/flush latency of an HDD-only object storage system

In the chart above, a continuous stream of IP video fragments are being written to a distributed object-storage array. Multiple (tens of thousands) of streams are being written simultaneously. Each stream is flushing a fragment to the object storage array every 2 seconds. The latency displayed in this chart shows the span of time during the test at steady-state and the average latency of each fragment flushes over this time.

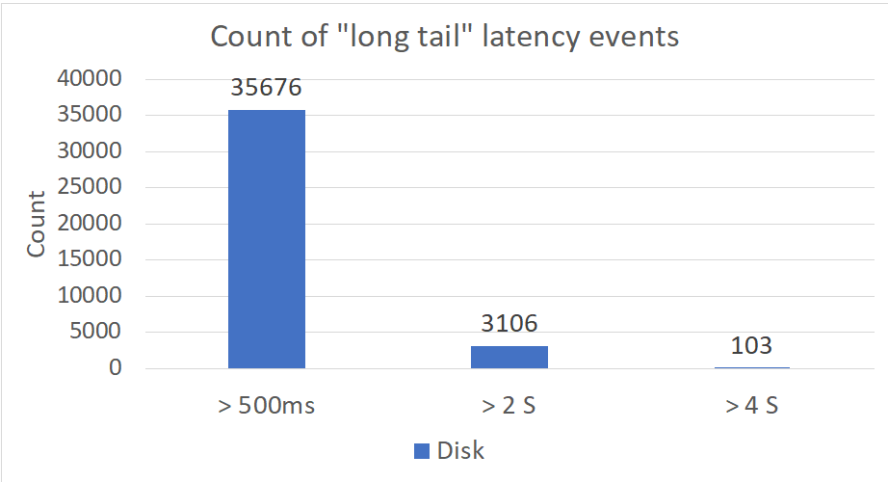

Figure 5 - Count of "long tail" latency events for an HDD-only object storage system

This chart is another measurement of the same test. It shows the accumulation of fragement flush events over the entire test that took over 500ms to complete. Over 35,000 fragment flushes to the storage took longer than 500ms. Another 10% of those (3,100) took longer than 2 seconds. It's likely that these fragment flush operations would result in an object "put failure" or a missing fragment for the specific recorded stream. The reason for this behavior is outlined earlier in this paper; disk drive latency increases dramatically when a burst of random activity is received due to a hotspot or during background scrub operations.

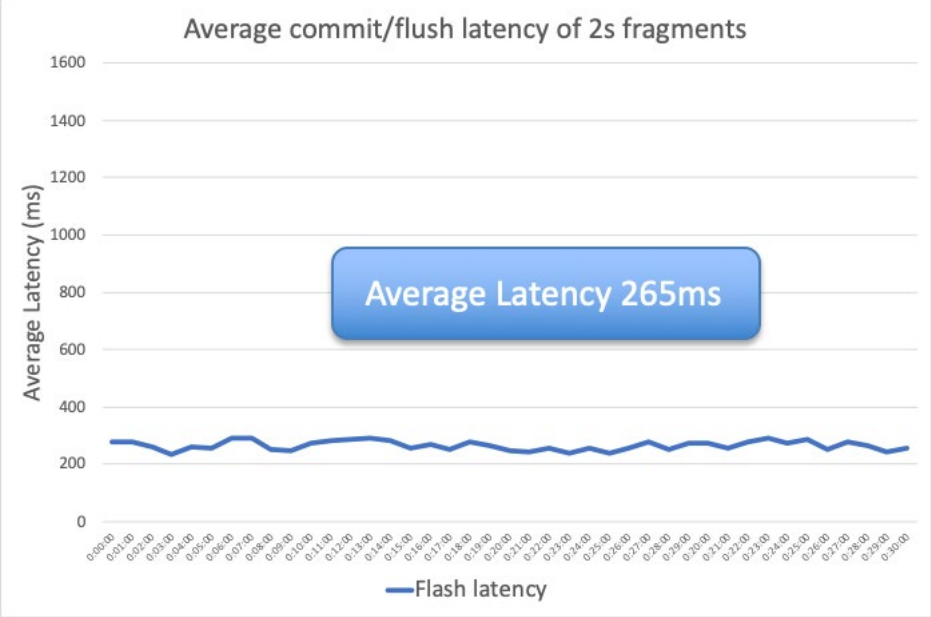

Figure 6 - Average commit/flush latency of a flash-only object storage system

This chart describes the average latency over time during a steady state run using only flash storage devices in the object storage array. As described previously, flash-based storage devices have much lower latency and can handle much higher throughput without exhibing the long-tail latency behavior of hard-disk based storage devices. This chart shows that the average latency of fragement flushes for tens of thousands of simultaneous streams, each flush occurring every 2 seconds. The average latency is 1/5th that of the hard-disk based object storage array.

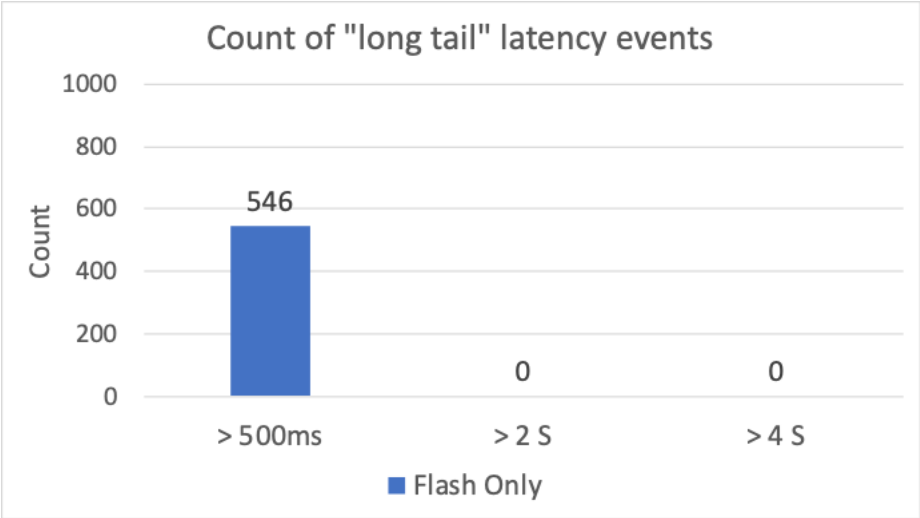

Figure 7 - Count of "long tail" latency events for a flash-only object storage system

Similar to the previous long-tail latency chart, this shows the count of object flushes that took longer than 500ms. The number of operations exceeding 2s is zero. In fact, the bulk of the operations that exceeded 500ms (there were only 546) were much closer to 500ms than 2s. What is evident here is the capacity for flash devices to handle random IO with background tasks and short-term bursts in activity without

introducing "long-tail" latency events. As described in this paper, if the entire storage cluster could be built out of flash devices, this performance would be achieved. However, the capacity required to store tens of thousands (or more) of cloud DVR recordings for months to years would make this cost prohibitive.

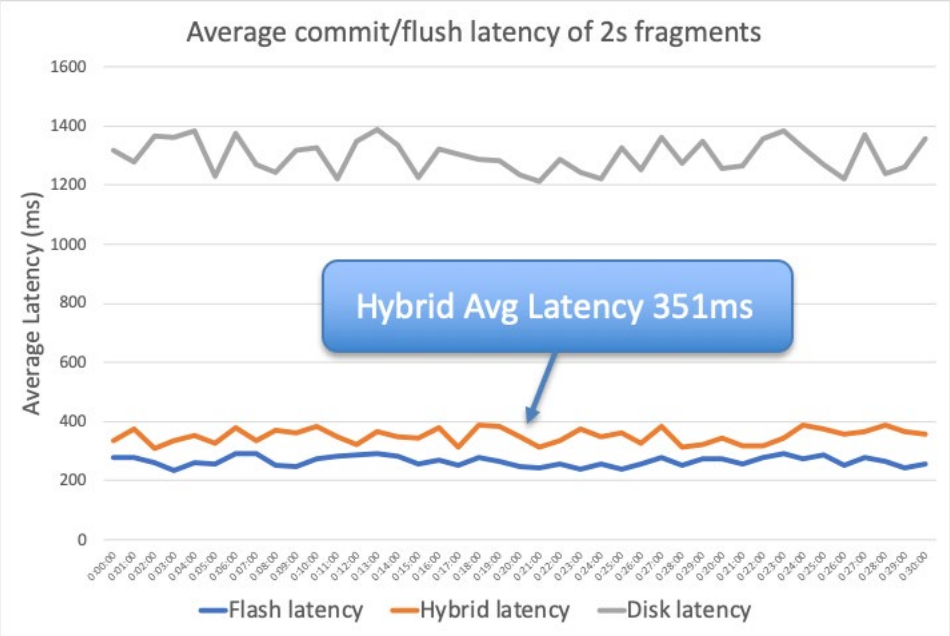

Figure 8 - Average commit/flush latency comparison of HDD-only, flash-only, and hybrid object storage systems

This chart is similar to the previous two charts showing average latency. Here, 3 series of data are overlaid onto the same chart. The "flash latency" and "disk latency" series are identical data to what is shown in the previous chart. They are included along with a new series of data, "Hybrid latency", to illustrate the relative average latency of the 3rd approach, which solves the challenges related to IP-linear TV and private copy Cloud DVR workloads: hybrid object storage. With Hybrid Object storage, a data-plane interface layer is transparently locating data in both disk and flash. Flash is used for writes (As well as recent reads), and an automated distributed task is moving the oldest data from flash to disk. For that reason, there is additional load and overhead. The chart here illustrates that the average latency of this additional abstraction and overhead amounts to less than 100ms to the average fragment flush time. This chart illustrates that hybrid object storage, which enables capacities as large as a traditional hard-disk based object storage cluster, still retains nearly flash-level performance.

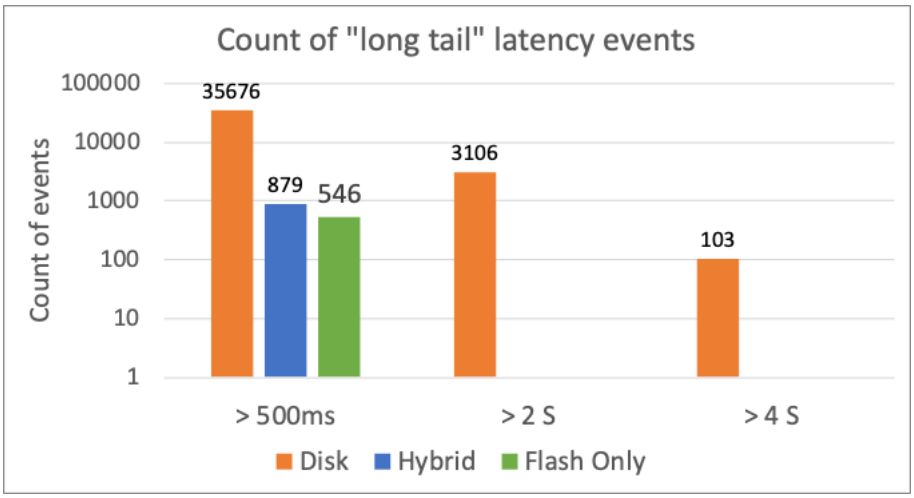

Figure 9 - Count of "long tail" latency events from HDD-only, flash-only, and hybrid object storage systems

This chart descirbes the count of "long-tail" latency flush trasactions. The 3 series of data again repeat the first two tests for hard-disk and flash-only storage arrays. The third series "Hybrid" in blue shows the count of long-tail latency events for hybrid object storage. Similarly to the all-flash storage array, there are zero flush transactions that exceed 2.5s. This is a large improvement over a traditional hard-disk-only object storage array, which exhibits numerous flush IO's that exceed 2.5s and would result in failed recordings or holes (missing data) in the streams that are recorded.

Conclusion

Software-defined object storage has unleashed huge benefits for many digital content use cases. By leaving behind the constraints of legacy hardware as well as legacy file or block access methods, extremely scalable video storage and distribution systems are possible utilizing digital video asset optimized protocols that are designed to efficiently ingest, stream, and store digital content, and run on commodity server technology.

Two emerging use cases, private copy cloud DVR and IP-linear TV, present challenges that must be solved in order to achieve the lowest latency user experience and handle thousands of recordings that occur in bursts. At high scale, large numbers of objects begin to hinder traditional object storage architectures with high metadata and fragmentation overhead. Traditional spinning disk drives, even when deployed in a large-scale object storage distributed environment, cannot keep up with large bursts in IO activity. This leads to latency spikes that cannot be mitigated when there are tens or hundreds of thousands of simultaneous video streams being committed to storage in real-time.

These challenges can be met by employing a new data interface layer to create a software-defined, multi-tier object model that allows cloud-DVR streams to scale in performance and capacity. This means high-performance flash devices can be used to absorb large burst workloads when popular programs are recorded, and that data can be smoothly transitioned to high-efficiency disk drives for long-term archival. This both solves the "small object" overhead issue and also prevents latency spikes that can result in failed recordings. Both tiers -- the flash and disk tier-- can be scaled independently as subscriber counts increase or data retention objectives change.

Abbreviations

CDVR	cloud DVR
SSD	solid State Disk
IOPS	input/output operations per second
SATA	serial attached SCSI
SCSI	small computer system interface
POSIX	portable operating system interface
DVR	digital video recorder
NVMe	non-volatlie memory express

Bibliography & References

[Vasudeva] Object Storage Key Role in Future of Big Data; 2015
https://www.snia.org/sites/default/files/Anil-Vasudeva_Object_Storage_Key_Role.pdf

[Coughlin] Why Hybrid Storage Strategies Give the Best Bang for the Buck; 2014
https://www.snia.org/sites/default/orig/NVM2014_ppt/Coughlin-Hybrid-Storage-v3-012314.pdf

RS-DVR Ruling, United States Court of Appeals, Docket Nos. 07-1480-cv(L) & 07-1511-cv(CON)

