Hybrid Cloud Storage: Bridging the Gap between Compute Clusters and Cloud Storage

Abhishek Gupta, Rick Spillane, Wenguang Wang, Maxime Austruy, Vahid Fereydouny, Christos Karamanolis

Storage and Availability Business Unit, VMware Inc.

{gabhishek, rspillane, wenguangw, amax, vfereydouny, ckaramanolis}@vmware.com

Abstract

Thanks to the compelling economics of public cloud storage, the trend in the IT industry is to move the bulk of analytics and application data to services such as AWS S3 and Google Cloud Storage. At the same time, customers want to continue accessing and analyzing much of that data using applications that run on compute clusters that may reside either on public clouds or on-premise. For VMware customers, those clusters run vSphere (sometimes with vSAN) on-premise and in the future may utilize SDDCaaS. Cloud storage exhibits high latencies and it is not appropriate for direct use by applications. A key challenge for these use cases is determining the subset of the typically huge data sets that need to be moved into the primary storage tier of the compute clusters.

This paper introduces a novel approach for creating a hybrid cloud storage that allows customers to utilize the fast primary storage of their compute clusters as a caching tier in front of a slow secondary storage tier. This approach can be completely transparent requiring no changes to the application. To achieve this, we extended VDFS [16], a POSIX-compliant scale-out filesystem, with the concept of *caching-tier volumes*.

VDFS caching-tier volumes resemble regular file system volumes, but they fault-in data from a cloud storage back-end on first access. Cached data are persisted on fast primary storage, close to the compute cluster, like VMware's vSAN.

Caching-tier volumes use a write-back approach. The enterprise features of the primary storage ensure the persistence and fault tolerance of new or updated data. Write-back from the primary to cloud storage is managed using an efficient change-tracking mechanism built into VDFS called exo-clones [18].

This paper outlines the architecture and implementation of caching tier volumes on VDFS and reports on an initial evaluation of the current prototype.

1. Introduction

It is indisputable that the IT industry is adopting a model where data centers, whether private or public, are built entirely out of commodity hardware components. This has been the case for many years with x86-based compute clusters and it is now becoming the norm for networking and storage. All functions of the data center are run in software. After all, this is the premise behind VMware's Software-Defined Data Center (SDDC) vision.

However, data center architects still need to optimize the design of their platforms based on the use cases they target. Specifically, *compute clusters* are optimized for application execution. They depend on fast, typically expensive local primary storage (these days almost exclusively solid state) to meet the performance requirements of the applications. Storing large volumes of data on such fast storage is not viable economically.

That's why many organizations depend on separate storage solutions which are designed for cost-effective, bulk data retention. Such solutions may be offered by on-premise deployed products that are often accessible through a networked file system interface. Examples include NFS filers, Data Domain appliances, or dedicated HDFS clusters. Storage services offered by publc cloud providers, such as AWS S3 and Google Cloud Storage are also becoming increasingly popular for these use cases [8]. In either case, storage solutions are decoupled from the compute clusters where the data is generated and/or processed. They exhibit high access latencies and they are not appropriate for direct access by applications. In this paper, we use the term *secondary storage* to refer to any of those solutions, whether on premise or on the cloud.

As a result, customers are faced with a challenge: utilize their compute clusters, such as those built on vSphere and vSAN, while they take advantage of the economics of secondary storage. And especially for VMware customers combining compute resources on-premise (or SDDCaaS in the future) with cloud storage like AWS S3 offers a non-disruptive path for taking advantage of public clouds services.

This paper presents a caching hybrid storage system that addresses this challenge by allowing VMware customers to use their vSphere clusters with their primary storage as a caching tier, and any storage provider of their choice (e.g., Cloud, NFS, HDFS) as a secondary storage tier. We show how this caching hybrid storage architecture can be uniquely tailored to different analytics compute platforms like Spark or Hadoop to further improve performance with application-specific optimizations, e.g., by not persisting intermediate Spark Resilient Distributed Datasets (RDDs) to fault-tolerant storage a la Tachyon [7].

The solution explored in this paper is built on VDFS [16]. VDFS is a hyper-converged distributed file system running on vSAN. It is POSIX compliant, supports millions of volumes, and supports scalable snapshots and clones for both files and volumes. It has been designed to cater to the needs of a cloud-centric storage consumption model.

Volumes are created in VDFS to serve as the caching tier of the hybrid system. These volumes are called *VDFS caching-tier volumes*. They cache the working set of application data in a write-back cache on primary storage to hide the latencies of the slow secondary storage. When listing a directory in the caching-tier volume, VDFS creates empty stub files which represent the objects in the secondary storage. It is also much more efficient to list/stat the stub files than talking to the secondary storage on every request. Applications can control the behavior of the caching-tier volumes to implement app-specific optimizations by manipulating the stub files. For example, they can inspect whether files are stubbed, inflate the stubs, or pin inflated files by getting and setting predefined extended attributes.

VDFS caching-tier volumes also use exo-clones [18] and provide a mechanism to aggregate high-churn workloads into bulk updates. This can be used to provide an efficient writable interface, e.g., to Amazon S3, or other high-latency, low-cost object stores.

There are existing caching solutions. However, the most mature and relevant project, Alluxio, is designed to operate high above the platform layer and serves a limited range of applications like Spark and Hadoop. Other solutions that operate at the file system layer only utilize an individual node for caching, rather than utilizing the entire primary storage cluster as the caching tier. At least one enterprise file system product, Compuverde [5], provides a hybrid storage solution where the entire primary storage cluster is used as a cache, but resolving conflicts when multiple primary cluster caches write back to the same secondary cloud store is unsupported. As of this writing, we have found no caching solution outside of VDFS caching-tier volumes that:

- utilizes the entire primary storage cluster as a caching tier;
- provides an application-controllable caching interface through the most widely used storage interface, the file system;
- offers a complete, consistent, and efficient mechanism to resolve conflicts during write-back.

We provide a brief background on VDFS and exo-clones in Section 2, where we also discuss other work related to VDFS caching-tier volumes. We outline the design of VDFS caching-tier volumes in Section 3 and discuss how they can be used for two major use-cases in Section 4. We provide initial experimental evidence motivating VDFS caching-tier volumes in Section 5. Finally we discuss future work and conclude in Sections 6 and 7 respectively.

2. Background

Caching volumes are built in VDFS which in turn runs as a file system atop vSAN. vSAN can provide thousands of objects and VDFS uses those objects to support millions of volumes and billions of files and directories in each volume. Therefore, VDFS is an ideal platform to provide caching for applications demanding object stores that require scalable meta-data capabilities.

VDFS [16] is a fully POSIX compliant scale-out file system being developed by the vSAN team. It allows applications to create multiple volumes where each one can be snapshotted or cloned. VDFS allows parallel data path with readers and writers from multiple hosts by leveraging the parallel data path of vSAN. The policy-driven data reliability and service availability of vSAN are also inherited by VDFS. Before we introduced caching volumes, VDFS had no ability to present volumes that could draw upon more capacious, cheaper storage, from a secondary storage system such as a public cloud or HDFS cluster.

Exo-clone [18] is another building block of a hybrid storage solution. Exo-clone is similar to the send/receive feature of ZFS [10] or Btrfs [15] but is faster and more flexible. Using the exo-clone technique, VDFS can represent the block-level differences between two snapshots, which can be exported as a file and imported by another VDFS cluster efficiently.

Alluxio, formerly known as Tachyon [7], is a memory-centric distributed file system which presents a caching volume like VDFS caching-tier volumes. Alluxio is implemented as a Java framework. Big data applications can link with Alluxio to more efficiently access secondary storage. It does not rely solely on caching to accelerate analytic workloads, as it also uses application-specific optimizations.

Alluxio allows some applications, such as Spark, to avoid writing frequent checkpoints of large distributed computations to fault-tolerant storage (avoiding many replicated writes over network). It achieved this by using application specific lineage which contains information of how to recalculate data if they are lost. Non-big-data applications cannot rely on lineage for such optimizations.

Although every application can benefit from a memory-centric file system that avoids checkpointing, Ousterhout et al. [17] shows that optimizing or eliminating disk accesses can only reduce job completion time by a median of at most 19%. This finding is actually corroborated by Alluxio's own experience with customers like Baidu. In talks with Alluxio, we learned that the 30x speedup on the Baidu workload [1] was primarily because of the elimination of the network latency gap between the secondary storage cluster and compute cluster, not sharing of lineage.

Alluxio's other contribution is to make a strong case for placing a caching and lineage tracking infrastructure within the storage stack, not specific to any particular application's implementation [7]. However, we note that Alluxio is a Java framework but not a filesystem. It needs FUSE for the filesystem abstraction which is an added overhead. Any performance gains some Java applications obtain by linking directly with Alluxio come at the price of performance degradation for applications running atop FUSE. Despite relying on write-back caching, Alluxio has no efficient snapshot, writable clone, or change-tracking capabilities (e.g., ZFS-send/receive or exo-clones).

Alluxio's idea of having a "caching tier" at the storage layer is beneficial, but their capabilities as a storage layer (i.e., clones, send/receive, fast file system interface) are limited. We argue that their implementation of lineage sharing between applications would be of limited benefit to VMware customers in IoT and big data analytics.

3. Design

Hybrid storage between primary storage on a compute cluster and a public cloud or other cheap secondary storage must tolerate variable and high latencies, unpredictable loss of connectivity, and provide a highly transparent caching and write-back interface to aid trouble-shooting when issues occur. Finally, caching-tier volumes need to expose enough control to the application level that useful application-specific optimizations are possible without requiring changes to the VDFS platform itself.

Based on these requirements, VDFS caching-tier volumes expose properties of the cache to applications through the POSIX extended attributes interface. Furthermore, write-back of changes can either be automic or manually initiated by the application, and what changes are written back can be specifically controlled. In Sections 3.1, 3.2, and 3.3 we discuss how the cache is exposed to applications and how it can be taken advantage of. In Section 3.4 we describe how write-back is accomplished and can be controlled by the application. Finally in Section 3.5 we describe how VDFS makes it possible to use multiple storage backends, including implementation details about our initial Amazon S3 and NFS backends.

3.1 Caching Model

To guarantee full control over the cache, VDFS exposes cache-related properties of files in caching volumes to applications via extended attributes. In our current implementation we support an extended attribute.

- `CACHE_STATE_ATTR`: Used to evict clean files, or fault in files ahead of a read operation.

These attributes can be used by applications to implement a variety of application-specific caching policies or optimizations. For example, by exposing the current cached state of files to the application layer, a job scheduler such as YARN or Mesos, when faced with a decision as to which job to run next, could choose to

schedule the job that has the highest amount of file data already present on the primary VDFS tier.

This optimization in fact was explored in Quartet [12] and was shown to improve the cache hit rate of Hadoop and Spark jobs by up to 54%. Their implementation currently requires kernel modifications to the hosts in the cluster as file cache information is not typically exposed to the application layer, but if running atop a VDFS caching tier volume, this would be unnecessary.

3.2 Caching at the FS Layer vs. App Layer

We decided it was best to expose caching functionality to applications at the file system layer as it allows multiple different compute frameworks to expose and manipulate caching state in a way visible to all other frameworks. For example, Quartet [12] can consult the same caching information that backup or scrubber programs can use to determine which files to backup or scrub next [9]. These programs are not all Java or Spark/Hadoop programs, e.g., in the case of backup or scrubber programs, some are written in C. Regardless, all different types of applications can access the file system and can gain the same visibility to the caching state of the primary tier. Sharing a common cache API with non-Java programs is not the only motivation for tracking cache state in the FS layer. As argued by the authors of Tachyon [7], analytics pipelines are rarely constructed out of a single framework, and being able to share cache-related state can significantly improve performance for the entire pipeline.

3.3 Easy to add future caching features

Using extended attributes to expose new caching functionality at the file system layer also affords the ability to add new features that can be made broadly available to all applications without having to update a plethora of APIs. For example, some new extended attributes we plan to add:

- `CACHE_NOEXOCLONE_ATTR`: Used to mark files that should be skipped when constructing an exo-clone for writeback.
- `CACHE_PINNED_ATTR`: Used to mark files that should not be evicted.

These attributes can be used by applications to implement a variety of application-specific caching policies or optimizations. For instance, Spark stores intermediate computational state in large sets called RDDs. These RDDs can be recomputed based on linage, but this can be time-consuming, so Spark allows users to `persist()` RDDs, and users can persist to RAM, DISK, or a replicated DISK. It would be straightforward to add support to Spark to mark files that RDDs are persisted to as `CACHE_PINNED` and `CACHE_NOEXOCLONE`. This would prevent intermediate files, intended only as computation checkpoints, from being transmitted over the WAN to a public cloud needlessly either due to eviction or write-back.

3.4 Write-back

Write-back of the cache follows a directed approach where the application has fine-grained control over when updates are written back. The application creates a snapshot, and then can freely write to the caching-tier volume within the workload. Space is allocated from VDFS exactly the same as it would be for any other volume, so management of quota for analytics applications is done the same way as for on-premise file services, object storage, or other storage applications.

As shown in Figure 1, when the application wants to write back, it creates a second snapshot, and constructs an exo-clone [18] between the two snapshots. The exo-clone can then be uploaded to a remote data repository program running from within the public cloud which can apply the exo-clone to the secondary storage. Conflicts are resolved in the same way as they are resolved in exo-clones.

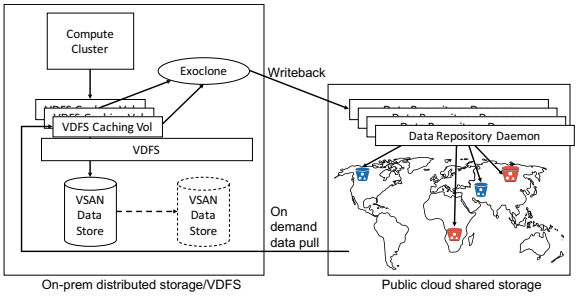

Figure 1. App-controlled write-back of caching-tier volume

As of today we have not yet implemented a remote data repository daemon, but the exo-clone format is an open format and it would be straight forward to write a data repository micro-service within major public cloud providers to write-back object updates, e.g., like the backend implementation of Cloud Array [4].

By supporting write-back in terms of exo-clones, applications are now able to continue making updates and writing on primary storage, even in the face of potential disconnection from the public cloud. Interruptions of write-back are easily resolved: simply re-send the exo-clone file. Since the primary storage caching tier is an entire distributed storage cluster, there is ample capacity to absorb writes while intermittent failures are resolved, without necessitating down-time for the application and the customer.

3.5 Backends

The current implementation of VDFS caching-tier volumes supports Amazon S3 and remote file system secondary storage. When backends supporting new secondary storage services are added, all applications utilizing the extended attribute interface to access the primary caching tier automatically gain the ability to use these new services. For example, if an HDFS or Swift backend is added in the future, all existing applications running atop VDFS caching-tier volumes, including any app-specific optimizations they have made, would immediately work with those new secondary storage protocols and systems.

The Amazon S3 backend takes authentication info, the access key and secret, when the caching-tier volume is created. This allows developers to construct applications that consume S3 storage to operate independent of which S3 account is being used. This is a desirable separation of concerns between vSphere admins and app developers.

Once the volume is created, the application can run all filesystem operations oblivious to the caching state of the file (unless the application explicitly queries its extended attribute) seamlessly spanning the primary caching tier and the public cloud storage provider. A file is never inflated unless it is read or explicitly asked to be inflated by setting the `CACHE_STATE_ATTR` attribute. In our current implementation, once the file is inflated, it is not written back to the public cloud.

To access remote file system secondary storage, the remote file system must be available on a mount point which is available to each host running VDFS. Files are copied across mount points on the host on first access, relying on the host's remote file system driver to access the secondary storage. On ESXi hosts, this would limit remote file system secondary storage to NFSv3 or Plan9 (9P) because these are the only network file system clients currently supported by ESXi.

4. Uses

VDFS caching-tier volumes are motivated by the direction current vSAN customers are taking their primary storage. Some customers want to efficiently analyze streams of data from sensor-equipped factories while ultimately storing the bulk of their data in a public cloud. Other customers are looking for better ways to manage the life cycle of globally developed and deployed applications and storage. We consider several emerging use cases and market opportunities for VDFS caching-tier volumes running on top of vSAN.

4.1 SDDCaaS

SDDCaaS is the enabler of the multi-cloud that provides the ability to utilize the familiar VMware SDDC stack in the public clouds. The main advantage is that the applications could be running on-premise or in the cloud, while the underlying software stack remains the same. This means enterprises can take advantage of both on-premise and cloud infrastructure through the same familiar trusted tools. One of the major implications of this would be the ability of transparently moving applications between on-premise and cloud. This is the area we are investing in, in our roadmap for data management. Another major benefit is that there might be applications that have their compute on-premise while their data could reside in the cloud and vice versa. For instance, imagine a firm with offices at multiple geographical locations having to move an application back and forth between a testing and development team. These deployments can be on-premise or in the cloud. This application can use SDDCaaS to deploy seamlessly anywhere using the same infrastructure enabled by VDFS caching-tier volumes.

4.2 IoT

IoT [6] is defined as the inter-networking of physical devices, typically embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data. A majority of IoT implementations send data to the cloud or another central data center, which can be used for crowd sourcing and building statistical models and intelligence to make business decisions, e.g., predicting part failures, deciding when to do maintenance for a part, or proactively fixing performance issues [19].

For many IoT implementations it is critical to be able to make real-time decisions based on the available data at the edge. However, it is typical in remote locations for network bandwidth and stability to be severely constrained. Hence, it is not trivial to analyze IoT data in the cloud as the delay required to upload data, process it remotely, and receive the response would not permit real-time processing at the edge site.

An example of this is Carbon [2, 3], which sells 3D printers for smart factories. Each printer has installed on it over 25 sensors. These 3-D printing devices generate millions of data-points per day. Furthermore these printers are connected, meaning that Carbon can push new software to them every 6 weeks. They collect data from all printer devices their customers use for printing. Using this meta-data, they provide customers *Digital Factory*, a system that logs data on when particular parts were built, operational data, maintenance schedule, and even using printing information to optimize the print operation in real-time. VDFS caching-tier volumes could store the generated printer meta-data, and subsequent accesses to it would be efficient enough to perform real-time analytics. Furthermore, the main body of the collected data would still be stored in the cloud, and VDFS could transparently provide object store or file access to this data with minimal changes to applications.

4.3 Dev-Ops

As of this writing, there is no widely employed design for hybrid storage between multiple geographically distant data centers, e.g., an on-premise primary storage site having one or multiple public-cloud data centers as secondary storage. Morgan Stanley is interested in a hybrid storage solution to replace their aging AFS file system deployment for managing build and tool-chain dependencies around the globe. Although containers seem to be the obvious contemporary choice for maintaining tool-chains, Morgan Stanley has many dependencies utilized by all applications that frequently change, e.g., a DB file that changes within 24 hours. This would force re-building and re-deployment of any containerized app. According to Morgan Stanley, containers are not sufficient to fulfill their requirements.

VDFS caching-tier volumes can be aggregated together into a global namespace, allowing applications to consistently refer to their dependencies, while individual volumes can be accessed independently and in parallel around the globe. Every person and application would get their own local "checkout" of their portion of the overall namespace. This decreases access latency significantly as users doesn't have to go over the wire for every request. The checkout time for the filesystem would be insignificant as every directory and file is initially a stub, which is faulted in on demand. Although, every user will have their personal copy of the filesystem, their filesystem still will be eventually consistent. Total consistency isn't possible without an NFS-like approach. However, an NFS-like approach is slow. Instead, we note that applications typically operate within a partition of the total namespace while ongoing queries access a stale copy, i.e., eventual consistency. Furthermore, for cases where immediate consistency is required, e.g., the DB file our Morgan Stanley customers have to access from every app, these can be accessed directly over 9P as a regular volume.

In fact, being able to aggregate multiple caching-tier volumes into a single namespace addresses another major selling point and use case of the Alluxio product: the ability to present multiple NASes as a single, cached, reliable NAS share for all employees in a firm [7].

Ultimately, we make it easy for the app developer and vSphere admin to organize their data into immediate and eventually consistent segments that are all part of a global namespace, and where eventually consistent segments are automatically cached, and can be updated in batch with exo-clones.

5. Evaluation

We evaluated VDFS caching-tier volumes in terms of whether caching is the right solution for the problem, and then in terms of our caching system's performance. We survey major representative big data and IoT workloads in Section 5.1. We evaluate the performance of VDFS caching-tier volumes compared to a model of a non-caching hybrid solution in Section 5.2.

5.1 Workloads

We characterize hybrid big data or IoT workloads as following basically a Zipf distribution of random accesses. Our characterization is based on Chen et al.'s [11] study from 2012, and Harter et al.'s [14] study from 2014. Both studies consider big data workloads at major Hadoop or HBase deployments. Hadoop and HBase are compute platforms that rely on a distributed secondary storage backend, HDFS. Chen et al. study five major Cloudera Customer traces across e-commerce, telecommunications, media, and retail as well as circa 2009–2010 Facebook traces. They found access frequency followed a "Zipf-like distribution", that the skew was cache-heavy, and that file access behavior was the same across different industries. Harter et al. performed a more recent study of Facebook Mes-

sages workloads and found conclusively that having a flash caching tier improves performance [14]. Based on these studies, and legacy studies on RDBMS workloads [13] which found also a Zipf access pattern, we designed an experiment that modeled access as Zipf.

5.2 Performance

Our experiment compared the same workload running on either (1) a VDFS caching-tier volume, or (2) a model solution that performs no caching but accesses files on-demand from the data center. The workload we compared was a job that randomly accessed files according to a Zipf distribution. We evaluated the range of Zipf distributions that are cache heavy (s=2) through those that are more uniformly distributed (s=1.1). Our expectation was that the model solution latency would be slightly higher than the average round-trip time to that AWS S3 data center's region: 90–110 ms for all workloads, and that the caching-tier volume's latency would drop as the workload became more cache-heavy. Figure 2 shows our results.

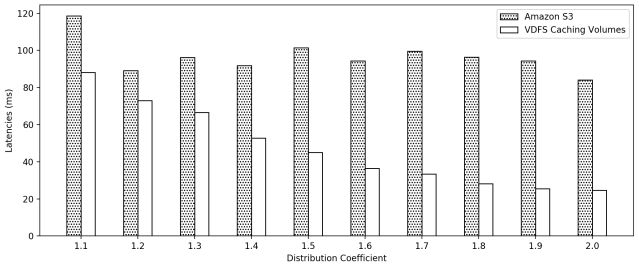

Figure 2. Comparison of average latencies between Amazon S3 and VDFS Caching Volumes accessing randomly selected files according to a Zipf distribution (Friday mid-day)

We see for more uniform workloads (s=1.1, s=1.2) the no-caching solution is 30–50 ms worse than round-trip time. Since the average file size is 17KB, we expected latency to be the bottleneck, but it appears S3 spends additional time actually processing GET/PUT requests within the data center as well. We see that as the workload becomes more cache-heavy that VDFS caching-tier volumes perform better compared to the no-caching solution, as expected, due to more of the workload being accessible without incurring S3 processing time or round-trips back and forth between AWS and the job process.

We ran our caching experiments at different times of the week and found ping times to AWS US west two varied dramatically, from 90–110 ms Friday to 30–50 ms for off-peak time on Sunday night. Figure 3 shows these resutls. Despite significant variation in latency, our caching solution delivered very similar latencies when the workload was cache heavy (s=2), and performance was comparable. In both experiments, the non-caching model solution had no improvement on performance as the workload became cache-heavy. VDFS caching-tier volumes had higher latency than non-caching as the VDFS experiments are run after the non-caching S3 experiments complete and latency can change even in this short period of time.

It is important to note that unlike typical caching solutions, VDFS runs atop distributed primary storage clusters such as vSAN. Those storage clusters typically scale to capacities of 100s of Terabytes or even Petabytes. So, they can act as an effective local cache even for very large data sets. That means that workloads can effectively achieve latencies of less than 1 ms after the cache is warmed up. For workloads, such as IoT workloads, where the data to process was generated recently at the site performing the computation,

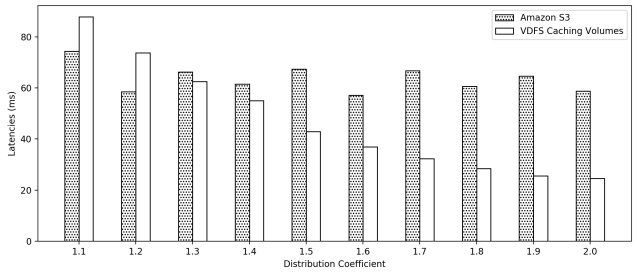

Figure 3. Comparison of average latencies between Amazon S3 and VDFS Caching Volumes during off-peak hours (Sunday midnight)

the cache needs not even be warmed up as the data is already on hand.

6. Future Work

Our current study of VDFS caching-tier volumes has some limitations. Firstly, we have not experimented with performance of a data repository server to manage written back exo-clones from within the secondary storage cluster as this has not been implemented yet. Another topic of future work is our current implementation's per-volume granularity. Although we have extended attributes to enable some per-file controls, permissions and authentication is still done at the volume granularity: e.g., the S3 access key and secret is provided when the entire caching-tier volume is created for the workload to run on top of. As we develop VDFS for more immediate use-cases such as vSAN File Services, we will have more time to examine issues like these as we work with customer design partners such as Morgan Stanley to make vSAN and VDFS more useful.

7. Conclusion

Everyone wants to move to cloud for cheap consistent storage. Our experiment shows that keeping the compute cluster on-premise with storage in public cloud is an undesirable setup as it is significantly slower. Now if enterprises move their compute as well to the cloud then not only do they get locked-in to a particular vendor, but also there are new economic and compliance issues they must resolve. There are many caching solutions, but either they are specific to particular compute frameworks, or the caching component is not typically designed as scale-out software, which is a requirement to effectively run on top of vSAN, or other scale-out primary storage clusters used on-premise or in SDDCaaS deployments. This paper introduces a new product concept, VDFS caching-tier volumes, where data mostly resides on inexpensive cloud while compute can be running on faster primary vSAN storage enabling. This enables IoT devices and other use cases to get the best of both worlds. This powerful tool equips enterprises with the ease of management and cost effectiveness of cloud without the burden of vendor lock-in. The most striking benefit of this caching tier is that it runs on top of vSAN which in itself has a multi-thousand install base. This makes adoption for VDFS caching-tier volumes very desirable and economical for our customers.

References

[1] Baidu queries data 30 times faster with alluxio. https://www.alluxio.com/assets/uploads/2016/02/Baidu-Case-Study.pdf.

[2] Carbon 3d. https://www.carbon3d.com.

[3] Carbon 3d. https://www.youtube.com/watch?v=qD1yJDVNkP0.

[4] Cloudarray. https://www.emc.com/storage/cloudarray/index.htm#!resources.

[5] Compuverde technical overview. http://compuverde.com/media/138857/compuverde-technical-overview.pdf.

[6] Internet of things. https://en.wikipedia.org/wiki/Internet_of_things.

[7] Tachyon: Reliable, memory speed storage for cluster computing frameworks. https://people.csail.mit.edu/matei/papers/2014/socc_tachyon.pdf.

[8] What's changing in big data? https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/keynote-address.

[9] George Amvrosiadis, Angela Demke Brown, and Ashvin Goel. Opportunistic storage maintenance. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 457–473, New York, NY, USA, 2015. ACM.

[10] Jeff Bonwick and Bill Moore. Zfs: The last word in file systems. 2007.

[11] Yanpei Chen, Sara Alspaugh, and Randy Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment*, 5(12):1802–1813, 2012.

[12] Francis Deslauriers, Peter McCormick, George Amvrosiadis, Ashvin Goel, and Angela Demke Brown. Quartet: Harmonizing task scheduling and caching for cluster computing. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, Denver, CO, 2016. USENIX Association.

[13] Jim Gray, Ken Baclawski, Prakash Sundaresan, and Susanne Englert. Quickly generating billion-record synthetic databases. Association for Computing Machinery, Inc., January 1994.

[14] Tyler Harter, Dhruba Borthakur, Siying Dong, Amitanand S Aiyer, Liyin Tang, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Analysis of hdfs under hbase: a facebook messages case study. In *FAST*, volume 14, page 12th, 2014.

[15] Justin Jose and Manoj Kumar KV. Solid compression strategy for btrfs snapshot. In *Recent Advances in Electronics & Computer Engineering (RAECE), 2015 National Conference on*, pages 160–164. IEEE, 2015.

[16] Luke Lu, Wenguang Wang, Maxime Austruy, Zhichao Li, George Huang, Ankur Pai, and Christos Karamanolis. VDFS: A cloud-centric virtual distributed file system. In *Proceedings of VMware RADIO 2015*, May 2015.

[17] Kay Ousterhoutand Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. Making sense of performance in data analytics frameworks. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '15, pages 294–307. USENIX Association, 2015.

[18] Richard P. Spillane, Wenguang Wang, Maxime Austruy, Rawlinson Rivera, Ankur Pai, and Christos Karamanolis. Better container runtime image management across the clouds. In *Proceedings of VMware RADIO 2016*, May 2016.

[19] Ben Zhang, Nitesh Mor, John Kolb, Douglas S. Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA, 2015. USENIX Association.