WHITE PAPER

Data Center
Accelerating Swift with Intel® CAS

Accelerating Swift with Intel® Cache Acceleration Software

Dramatically improve Swift storage performance. Reduce data loss risk.

Table of Contents

Executive Summary

The explosive growth of unstructured data has resulted in the need to greatly expand safe and efficient storage. Enterprises and cloud service providers (CSPs) are transitioning to software-defined storage (SDS) solutions as a cost-effective way to meet the ever expanding storage demands. OpenStack Swift* is an SDS solution for public and private cloud storage, providing the scalability necessary to manage petabyte and beyond data growth.

As enterprises transition from traditional storage toward more open SDS solutions such as Swift, and use commodity servers to control costs, quality of service and performance requirements continue to be critical. Utilizing a combination of Intel technologies can help significantly boost performance in a Swift environment, enabling enterprises and CSPs to grow their business while continuing to meet demanding service-level agreements (SLAs).

The real world solution presented in this paper shows how an Intel® Solid State Drive (SSD) with intelligent caching software (based on I/O classification) can dramatically improve Swift performance in a hybrid storage environment. This solution is a collaborative effort between SwiftStack and various Intel teams (Intel's Software Solutions Group, Non-Volatile Memory Solutions Group, and Intel Labs).

This caching solution, based on NVMe*/PCIe* Intel® SSD with Intel® Cache Acceleration Software (Intel® CAS), is a scalable and cost-optimized approach that provides significant performance improvements without the need for data migration, or any changes to Swift or Linux* software. In our largest tests, where cluster sizes range from 5 to 15 nodes, we reduced GET latency (read response times) from 53% to 86% (for 1MB and 64KB objects, respectively), with an average latency reduction of 67% across all object sizes. For the small objects (64KB) this latency reduction translates to 7x faster downloads. In addition, Swift replication convergence time is reduced by up to 82%.

- The overall Swift throughput improved by 3x with Intel CAS and NVMe-based Intel SSD vs without.
- The replicator performance data realized a >5x improvement in replica convergence time — reduced from 6 hours (without CAS) to 1 hour and 5 minutes with Intel CAS and NVMe-based Intel SSD.

Swift Architecture and Challenges

The OpenStack Object Store project, known as Swift, is an object-based SDS solution in which the basic unit of stored data is called an object. The type of content in each object is determined by the user (e.g., text, executables, pictures, videos, music, sensor data, etc.). Each object contains a rich set of metadata which enables access, security, and most importantly superior data mining. These objects are durably stored (generally on HDDs) within a cluster of machines.

Swift has two major logical components; Swift's proxy server and Swift's storage server. The proxy server is responsible for communications between the client and Swift's storage servers, and for implementing most of the Swift application program interface (API). The proxy server is also responsible for determining where data lives in the cluster and choosing the right response to send to clients. Swift's storage servers are responsible for storing the data on drives, serving data when requested, and auditing the data for correctness. Swift storage servers use an on-disk layout with a deep directory structure, meaning that there is a relatively large amount of filesystem metadata to be accessed in order to read or write an object.

Swift preferentially flushes object data from the system's page cache in an effort to keep the metadata in memory. In its default configuration, Swift will immediately evict any object larger than 5MB (using a posix_fadvise64 call) after reading the file. In order to combat cache misses, Swift operators often set vm.vfs_cache_pressure to a low value to ensure filesystem metadata is kept in memory, and that object data is flushed instead. If the filesystem metadata is kept in memory, reads and writes are performed much faster than if the system has to retrieve the metadata from the HDD. However system memory is costly and limited by the server hardware.

In order to ensure consistency of the data, Swift's storage servers frequently traverse the filesystems that contain the object data. This filesystem traversal causes a large amount of the filesystem metadata to be loaded into memory. In typical deployments at scale, the filesystem metadata can grow to be much larger than the available system memory page cache! Consequently the Swift storage servers end up accessing file system metadata from HDD in order to perform their tasks. This negatively impacts the performance of these storage server operations, not only for maintaining data consistency, but also for serving data in response to the foreground Swift API requests.

High-Performance Caching Solution

As mentioned above Swift has performance challenges when it comes to accessing file system metadata associated with objects. Processing this metadata is a common performance bottleneck, and as data continues to grow, performance bottlenecks are exacerbated. Intel CAS with its unique I/O classification technology called Differentiated Storage Services (DSS) enables caching of the metadata that does not fit in system memory, onto the SSD, thereby speeding up overall Swift performance.

Employing just one NVMe-based SSD with Intel CAS is an excellent hybrid solution for accelerating access to the HDD data without incurring the high cost of replacing all HDDs with SSDs. This approach enables you to achieve better performance with Swift, and provide acceptable user SLAs in a cost-effective manner.

Figure 1 represents a visual overview of this hybrid solution, where an Intel SSD is added to each object storage node. Intel CAS is also installed on each node and uses the SSD as a cache for all the HDDs in that node.

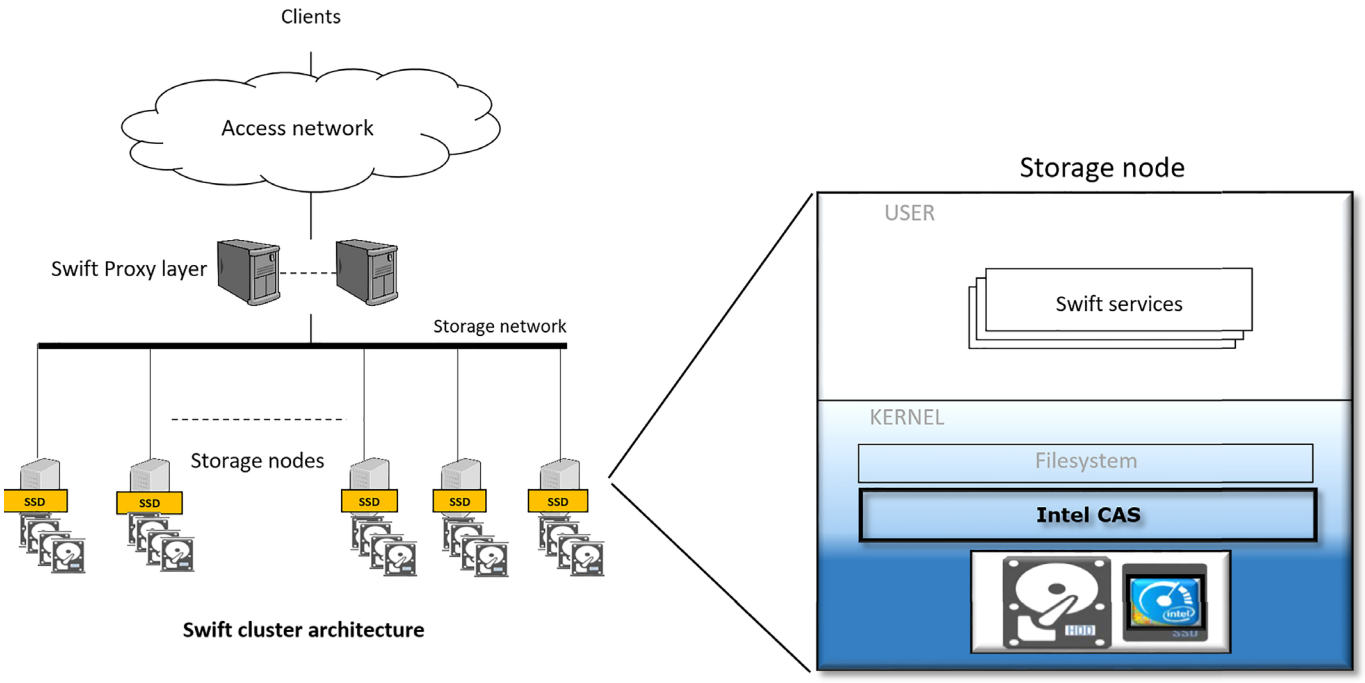

Figure 1: Swift Cluster Architecture With Intel® SSD and Intel® CAS

The resulting performance benefits of adding a single PCIe/NVMe Intel SSD and Intel CAS software, configured with a policy to cache only file system metadata (unless otherwise noted) to each Swift storage node, as shown in figures below.

Figure 2 shows a 3x average increase in overall cluster throughput. These results hold across a range of object and cluster sizes.

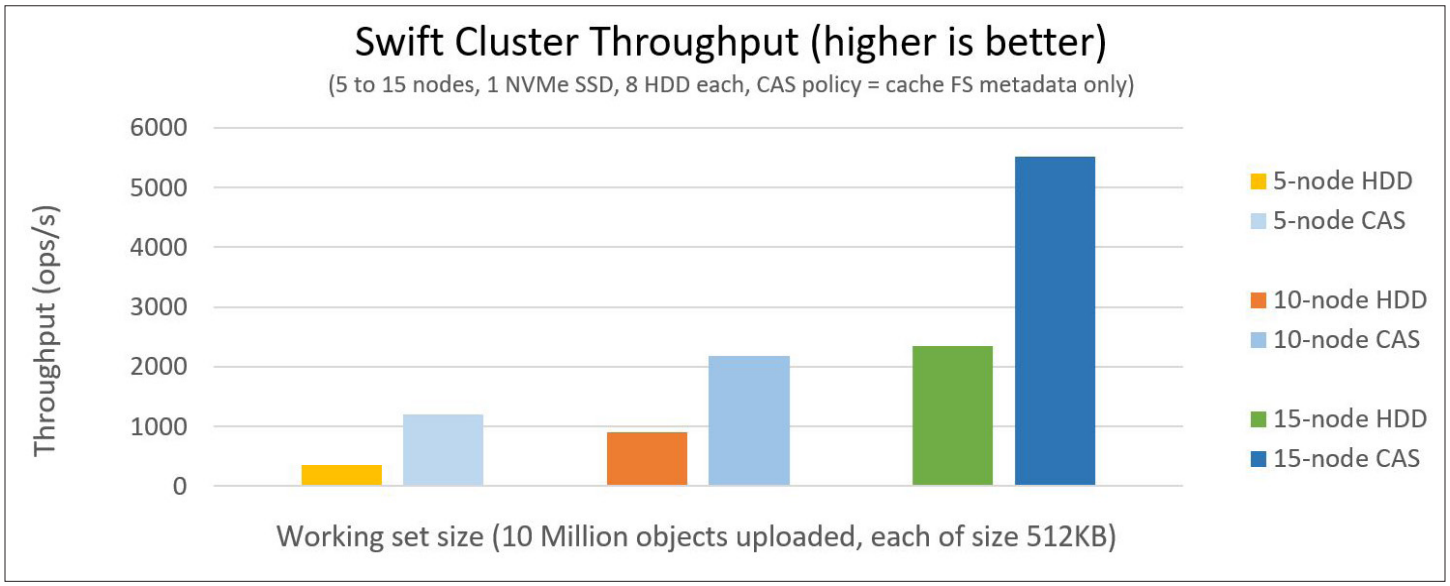

Figure 2: Performance Comparison With and Without Intel® CAS

Figure 3 highlights the improvement in Swift replicator background process performance, compared to an HDD-only solution.

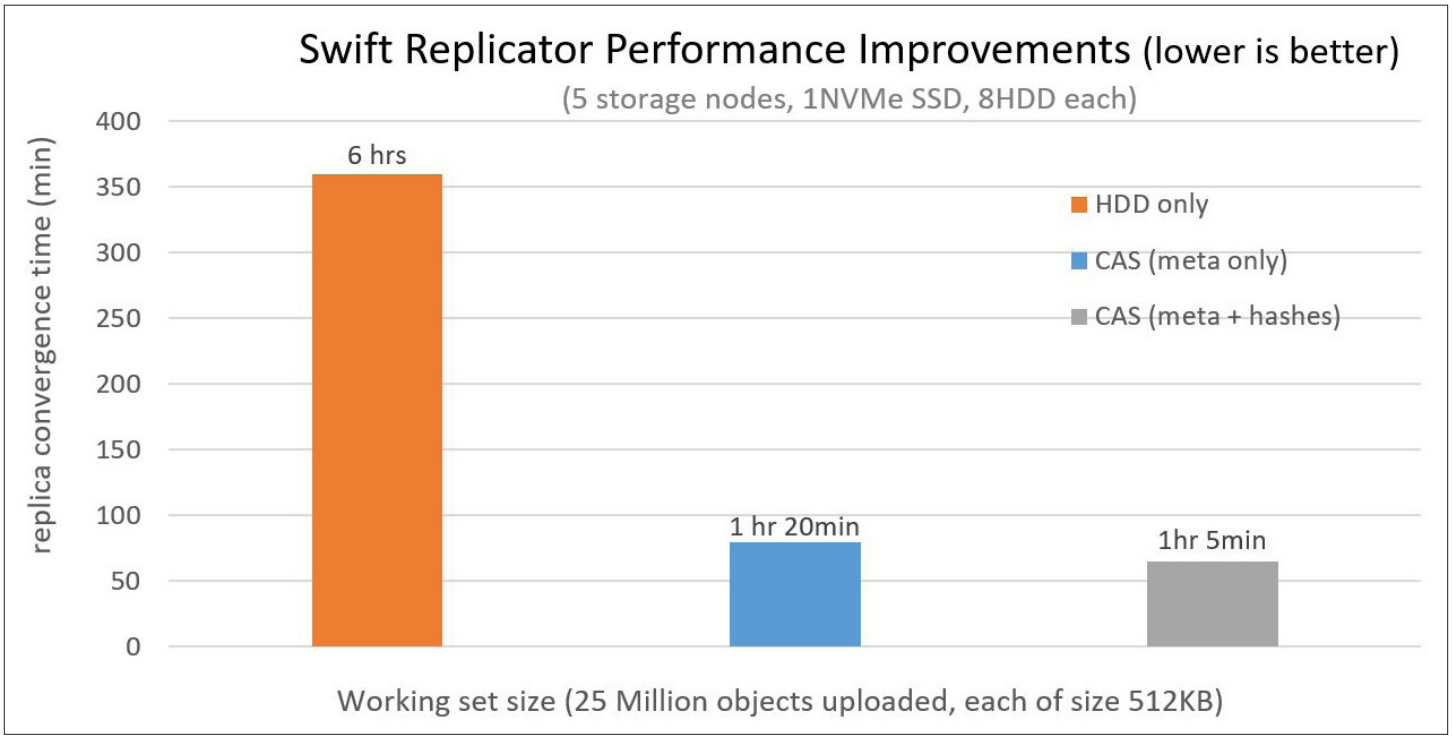

Figure 3: Replicator Performance Improvements With Intel® CAS

Intel® CAS in conjunction with Intel® SSDs are a high-performance solution that can accelerate applications without modification to the existing applications or back-end storage media. Intel® CAS provides unique features tuned specifically for Intel® SSDs, enabling the applications to utilize the SSD for caching the prioritized data, from existing back-end storage media (HDD, SAN). This Intel SSD with Intel CAS combination is a cost-effective hybrid solution that can quickly and easily provide a boost to read and write performance without introducing additional operational costs.

Intel® CAS has a small default memory footprint that can be further reduced by using a feature called Configurable Cache Line Size, which may provide cost saving benefits especially as servers move to higher capacity storage. Other features, such as in-flight upgradability which allows users to upgrade Intel® CAS without interruption, provide further operational efficiencies.

SwiftStack is a complete, enterprise-ready storage product with OpenStack Swift at the core. SwiftStack delivers needed and innovative functionality outside the data path, while OpenStack Swift components are in the data path. This gives enterprises private cloud storage that's easy to deploy, scale, integrate with existing systems, and maintain over time. SwiftStack is a leader in the OpenStack Swift project, with the Project Technical Lead (PTL) being a member of the SwiftStack team. SwiftStack's active involvement in the Swift project is in part due to their commitment to ensure that the functionality that touches your data is open source.

Addressing the I/O Challenges of Swift with Intelligent Caching

Intel and SwiftStack designed and tested a reference solution utilizing Intel® Xeon® processors, Intel® SSDs using high-speed Non-Volatile Memory Express* (NVMe) and Intel® CAS.

The key to the Intel CAS solution is the DSS I/O classification technology (published in ACM's SOSP, 2011); developed by Intel. This allows users to intelligently prioritize caching based on I/O type, size, class (for example, performance sensitive metadata including both Swift and Linux filesystem metadata, such as inodes), and other performance-critical parameters. For example, the filesystem journal can receive a different class of service than regular file data. This selective caching capability enables better tuning of the environment, for specific applications or workloads.

Unlike conventional caching techniques that rely solely on the frequency that data is accessed, Intel CAS can be configured with multiple performance requirements. In this solution the I/O classification capabilities of Intel CAS is utilized to cache performance critical file system metadata onto an Intel SSD. This solution avoids the problems commonly associated with cache inefficiency (cache thrash), and enables cost-effective introduction of high-performance SSDs to an existing HDD-based Swift cluster.

Intel CAS takes advantage of Intel SSDs' robust features to provide the performance gains for Swift. Intel SSDs offer many compelling benefits to enterprise applications; improved reliability, lower power usage, lower latency, greater throughput, and larger concurrent I/O, when compared to HDDs. In cases where primary storage is the performance bottleneck, the faster speed and lower latency of Intel SSDs improve performance by providing a substantial increase in speed and throughput.

Test Environment and Methodology

Comprehensive testing was performed by Intel and SwiftStack on several small Openstack Swift clusters. Several test combinations were executed on five storage nodes, then expanded to ten, and to fifteen storage nodes to observe the true scalability of the solution. Swift clusters were configured with one region, five zones and three data replicas. Each Swift object server had eight hard disk drives (HDD) and one NVMe/PCIe-based Intel® SSD DC P3520. The Swift I/O requests to the HDDs were cached using Intel CAS, caching only the filesystem metadata (unless specifically noted otherwise).

These three key cluster metrics were used to evaluate performance:

- Bandwidth (bytes transferred per second)
- Throughput (GET or PUT operations processed per second)
  "PUT" operations upload objects to a Swift cluster
  "GET" operations download objects from a Swift cluster
- Latency (time to complete a given GET or PUT)

The testing for each cluster was performed in multiple phases. First, a large number of objects of a fixed size were uploaded into Swift (the PUT phase). In order to mimic a system under constant load where a large number of requests result in OS cache thrash, the OS caches were dropped (sysctl.vm_drop_caches=3) after the PUT phase. For the following five minutes, randomly chosen objects from the set of previously uploaded objects were downloaded (the GET phase). The average response latency for individual download operations were measured. These phases were repeated, with each successive phase progressively increasing the number of objects uploaded.

The Swiftstack Controller was used to configure and manage the Swift clusters. The Controller allows operators to conveniently configure Swift policies and push the configured Swift ring information and other Swift parameter tunings to the individual nodes in the cluster. The Swiftstack telemetry plugins were also used to monitor the Swift cluster performance.

The COSBench workload generator was used to generate the client requests to the Swift cluster. COSBench is an open source benchmark tool developed by Intel to test cloud object storage systems, including Swift. It provides fine grained control over different aspects of a workload like Swift API mix, object size (including size distribution), concurrent requests and duration. COSBench enables parallel client requests to be issued to different URLs, to balance the client load on the Swift cluster by spreading requests equally amongst the Swift proxy servers. By utilizing this balancing feature, the need for a separate load balancer is eliminated as it provides the ability to use multiple COSBench worker threads spread across multiple COSBench drivers running on separate machines.

The test environment of the results presented is shown in Figure 4.

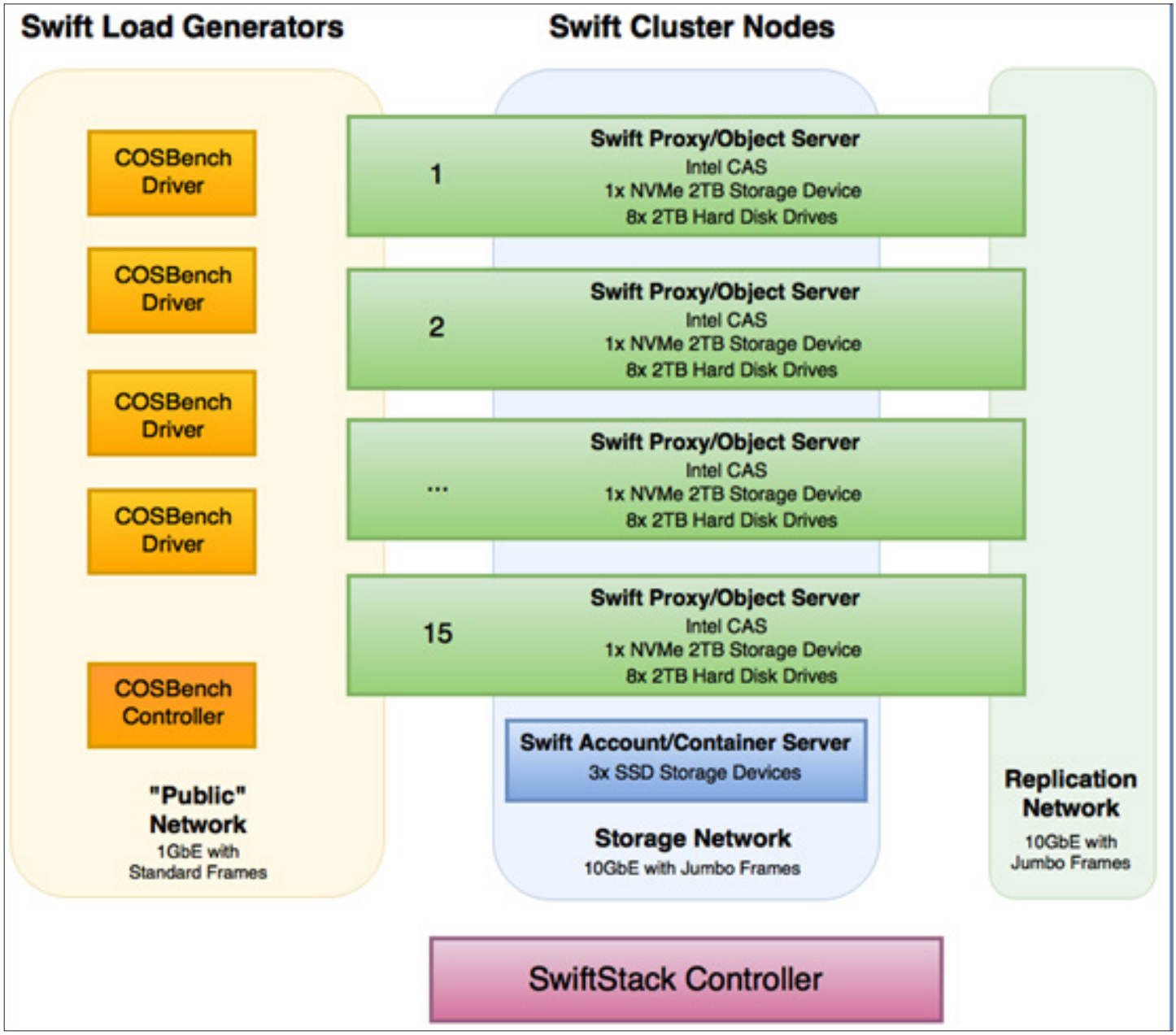

Figure 4: Swift/Intel® CAS Benchmarking Topology

Test Environment Hardware Configuration

Fifteen Proxy/Object storage nodes, each with the following configuration:

- Processors - 2 x Intel® Xeon® E5-2699 (45MB cache, 2.3GHz, 18 cores)
- RAM - 128 GB Memory - 8x 16GB 2133 Reg ECC 1.2V DDR4
- Storage - 8 x 2TB Seagate ST2000NX0403 and 1 x 2TB NVMe-based Intel SSD DC P3520
- Intel RMS3CC080 RAID Controller - SAS 3.0 Mezzanine SAS/SATA 8 Port 1GB
- Network – 2 x Intel X540 10Gbe NICs, 2 x Niantic NIC, 2 x Intel X520-DA2 and 2 x 10G NIC - Intel X540

All other servers (Swift account/container servers and clients), each with the following configuration:

- Processors - 2 x Intel® Xeon® E5-2699 V3 (45MB cache, 2.3GHz, 18 cores)
- RAM - 256GB Memory - 16 x 16GB 2133 Reg ECC 1.2V DDR4
- Storage - 4 x DC S3510
- Network - 2 x Niantic 10Ge, 2x Intel AXX10GBNIAIOM, and 2x Intel X520-DA2

Note: Each server has two separate 10GbE networks configured with jumbo frames, one for typical intra-cluster traffic and the other dedicated for replication between object servers.

Test Environment Software Versions

- Operating Sytem: Ubuntu* 14.04.5
  kernel Revision: 4.4.0-47-generic
- Swift Version: 2.9.0.2-4~trusty
- SwiftStack Controller Version: 4.7.0.1
- Intel® Cache Acceleration Software (CAS) Version: 3.1.1
- COSBench Version: 0.4.2.c4

Performance Tests and Results

Tests Performed with One Storage Node

In the first test scenario Swift is brought up with one storage node containing a single 1TB HDD with no replicas. As described earlier, objects are uploaded to fill up the HDD capacity in successive phases. The test is then repeated with an NVMe SSD as a cache for this HDD. Figure 5 shows the average latency to retrieve individual objects in response to Swift GET requests. The requests are sent by multiple client workload generator threads with 100 outstanding requests at all times throughout the test.

The average GET response latency was higher for the HDD-only test case while the corresponding latency numbers for the Intel CAS test case were much lower. When the filesystem metadata was not found in system memory, each access needed to be made from the HDD. Specifically for each Swift object request, a directory entry and its inode needed to be fetched prior to fetching the actual data for that object. Further, the latency to download objects of the same size increased as the HDD filled. This is because the HDD must seek further as the HDD capacity fills up.

This data demonstrates the impact of HDD seeks for accessing filesystem metadata on the overall object download latency. There was a dramatic reduction in overall object download latency by simply caching filesystem metadata. This is significant since the volume of the data stored per node in typical Swift installations is in the order of several terabytes, rendering traditional approaches of caching data ineffective. Filesystem metadata on the other hand is typically 3-5% of the actual data stored. These tests were repeated across a range of object sizes from 64KB to 1MB, and similar trends were observed for the entire range of object sizes.

In our next test we increased the number of HDDs in the storage node to eighteen. A single SSD was used for caching these HDDs in one Swift object storage node, to evaluate whether the available I/O bandwidth of a single SSD could become a bottleneck in comparison to the parallel I/O bandwidth of a large number of HDDs. The same tests were repeated with a larger number of uploaded objects.

The data in Figures 5 and 6 show similar benefits in latency reduction from caching filesystem metadata only. Caching only the filesystem metadata in the Intel SSD leaves the parallel I/O bandwidth of the HDDs available for serving object data, thus enabling a single Intel SSD for caching a large number of HDDs without becoming an I/O bottleneck.

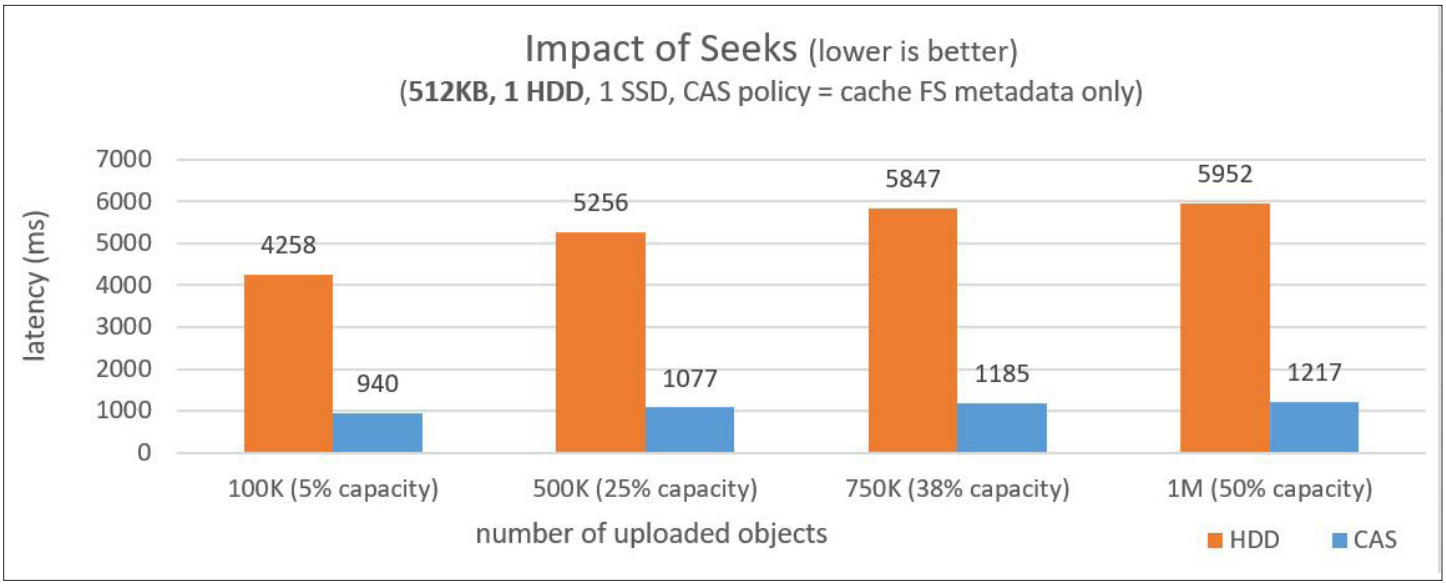

Figure 5: Single Node, 1 HDD Swift Micro-Benchmark

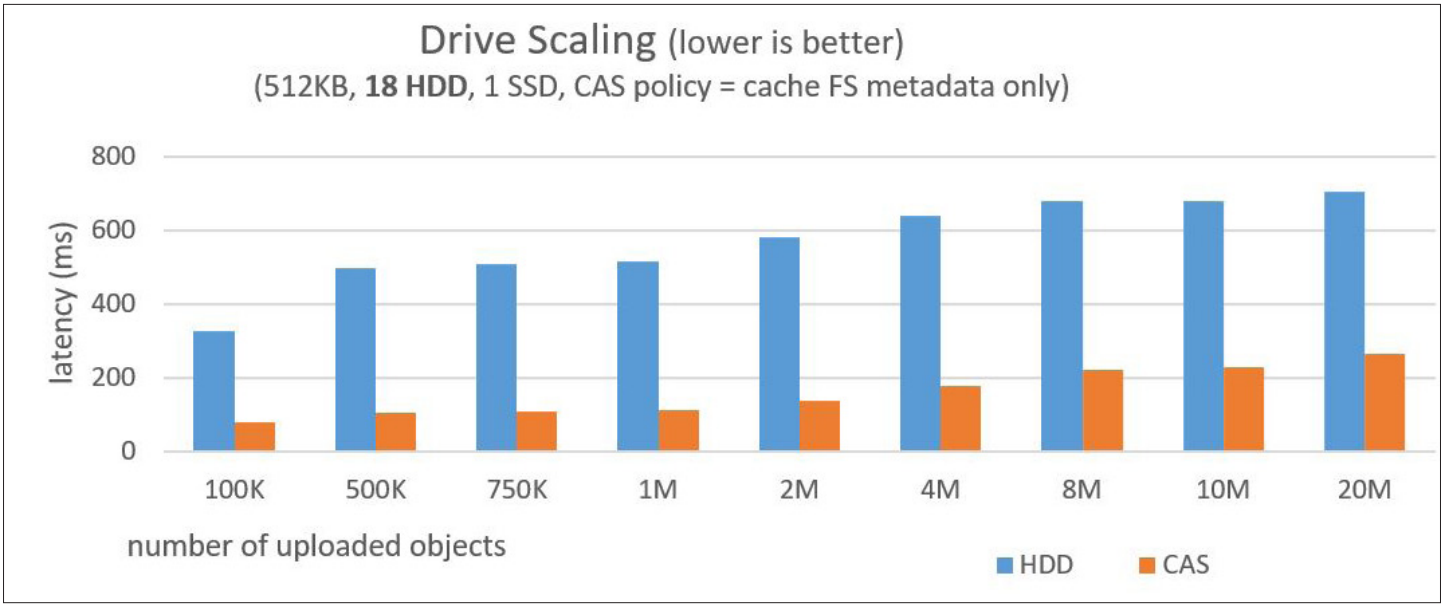

Figure 6: Single Node, 18 HDDs Swift Micro-Benchmark

Tests Performed with Five to Fifteen Storage Nodes

In the following tests scenarios, five to fifteen storage nodes were tested in order to determine if caching filesystem metadata on individual storage nodes translates to a latency reduction in object downloads for a larger Swift cluster. Each storage node was provisioned with 8 HDDs. Each storage node runs Swift object servers as well as Swift proxy servers (to avoid the proxy server becoming a bottleneck as the number of nodes in the cluster is increased). The Swift Account and Container servers were run on separate nodes provisioned with SSDs (based on the commonly accepted guidance to ensure account/container metadata reside on SSDs). The client workload generator distributed the object requests amongst the Swift proxy nodes.

The overall performance, as the number of storage nodes in the Swift cluster is scaled out, is shown in Figures 7 and 8.

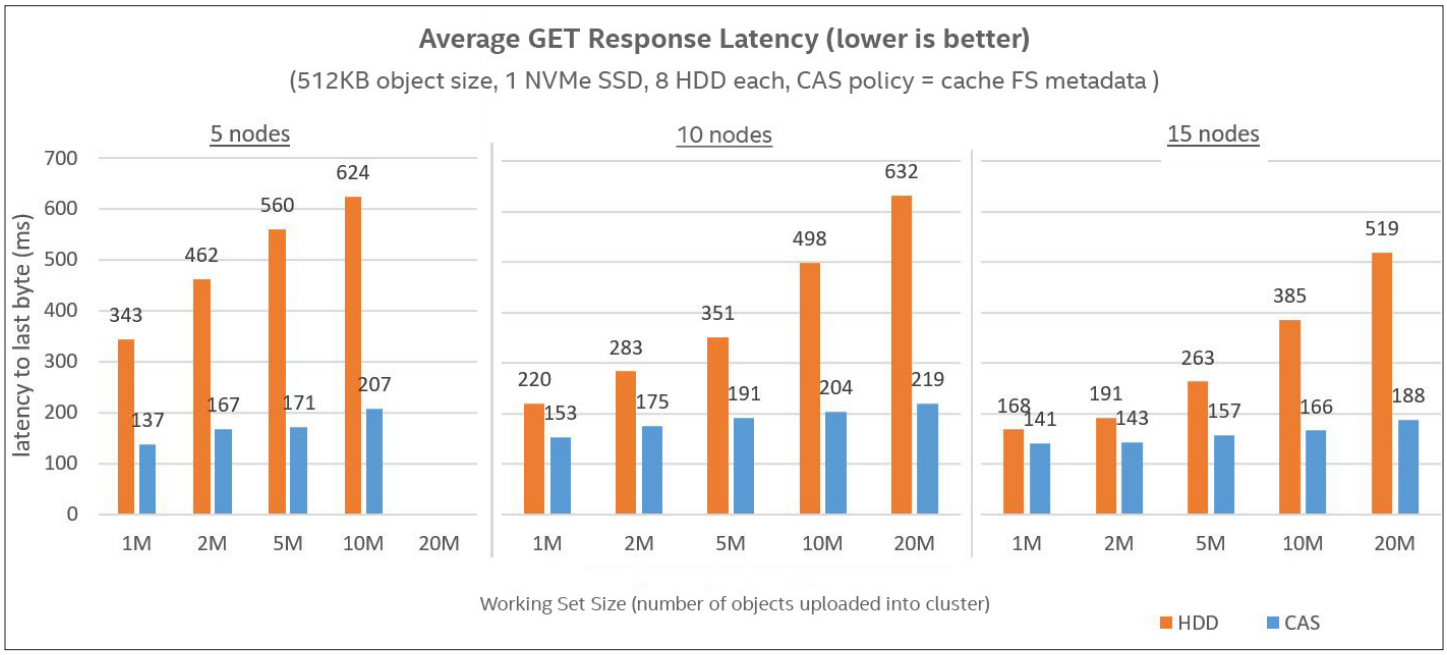

Figure 7: Swift GET Response Latency as Number of Storage Nodes is Increased from 5 to 15 Nodes

For the largest working set across all cluster sizes, the GET response latency with Intel CAS is consistently lower (67% for 5 nodes, 65% for 10 nodes, 64% for 15 nodes), with an average latency reduction of 65%.

The latency to retrieve an object from the object storage node includes, i) Time request waits in queue at object server behind earlier requests ii) time to locate object on drive iii) time to actually read the object off the drive iv) time to transfer the data over the network.

"Why do HDD latency numbers reduce with more nodes?" When more nodes are added, for the same load a) there are fewer requests per object server, resulting in shorter queues b) fewer objects per drive, results in less HDD seek to locate object.

"Why do HDD latency numbers increase with increasing working set size?" A larger working set implies that for a given cluster size, there will be more objects per drive, and hence more HDD seek to locate an object. This HDD seek is what we reduce by caching filesystem metadata on SSD.

The Intel CAS and Intel SSD solution address the HDD seek latencies, providing cost savings by reducing unnecessary storage provisioning.

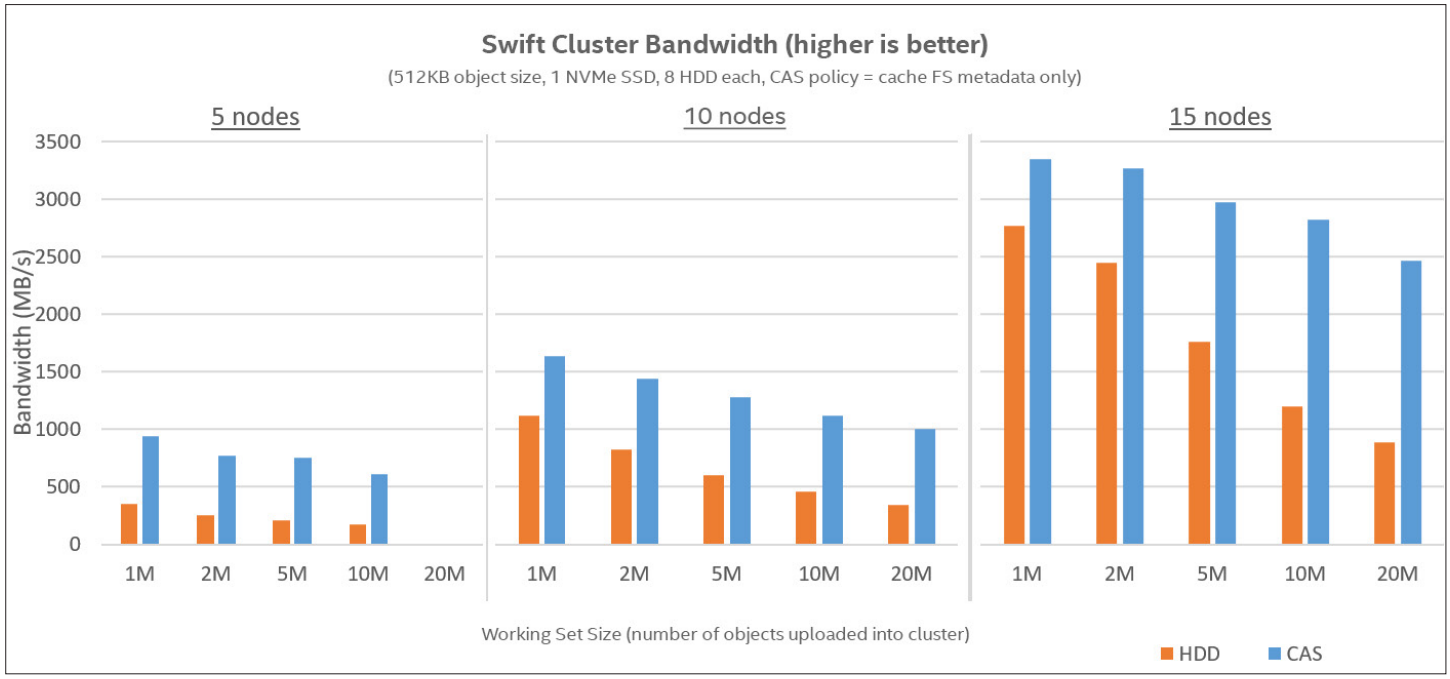

Figure 8: Swift Performance Scaling as Number of Storage Nodes is Increased from 5 to 15 Nodes

The graph in Figure 8 demonstrates the overall cluster bandwidth. The results show that the average cluster bandwidth is 3.1x greater with Intel CAS for the largest working set across all cluster sizes. The overall cluster throughput (not charted here due to space constraints) demonstrates the same improvements. More significantly the trend across each cluster size demonstrates that the relative performance improvements due to SSD with Intel CAS become larger as the working set size increases to numbers closer to more realistic Swift installations.

The above tests were repeated on a 5-node Swift cluster with varying object sizes from 64KB to 1MB to determine the impact of object sizes on the overall performance. For the largest working set size, Intel CAS reduces GET response latency by an average of 67% across all object sizes.

The results of these tests are summarized in Figures 9 and 10 which show that the performance improvements due to Intel CAS were realized across the range of object sizes.

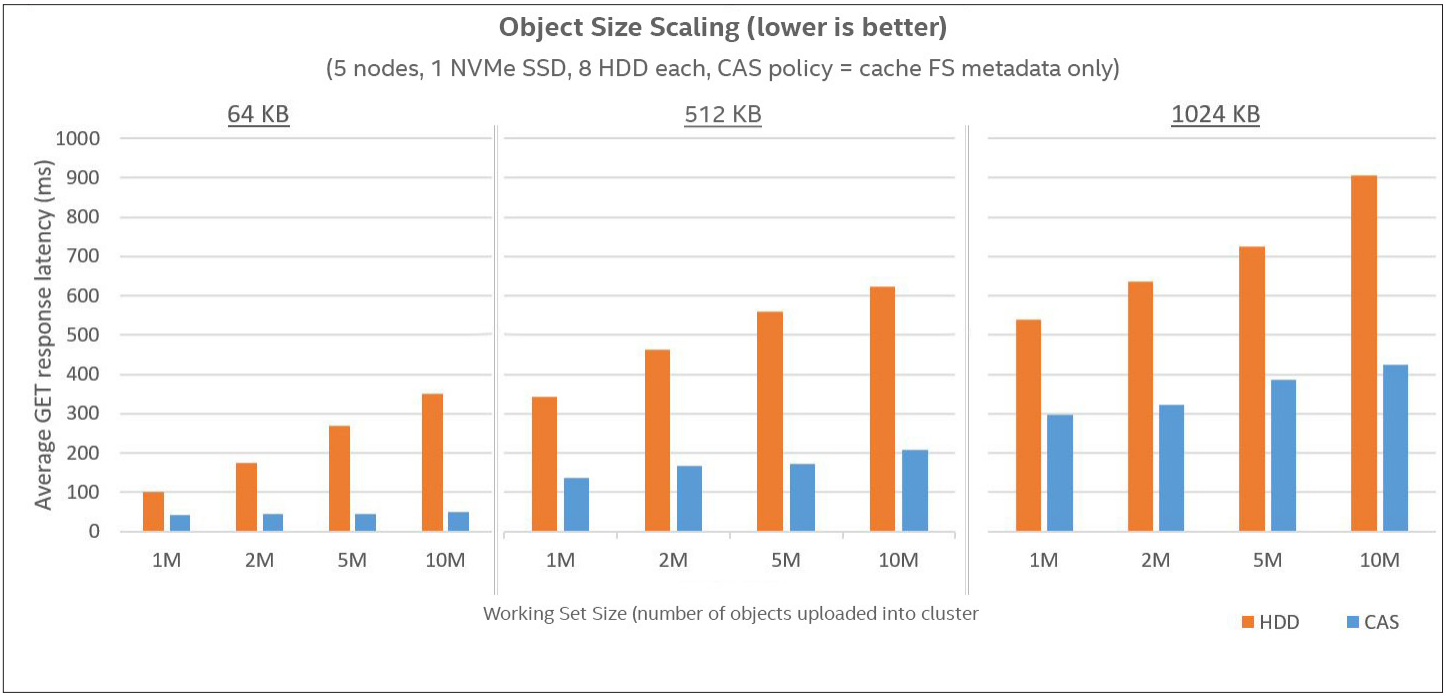

Figure 9: Swift + Intel® CAS Performance Across a Range of Object Sizes

The trend for the performance gains of smaller object sizes is of particular relevance since the small file performance of Swift is a known focus area for improvement within the Swift community. Figure 10 focuses on the average GET response latency for 64KB object size (the smallest object size in our tests). The chart plots the ratio of the latency numbers for HDD compared to Intel SSD with Intel CAS for different working set sizes. For the largest working set size of 10 million uploaded objects, the relative latency improvement of Intel CAS as compared to the HDD-only case is 7x.

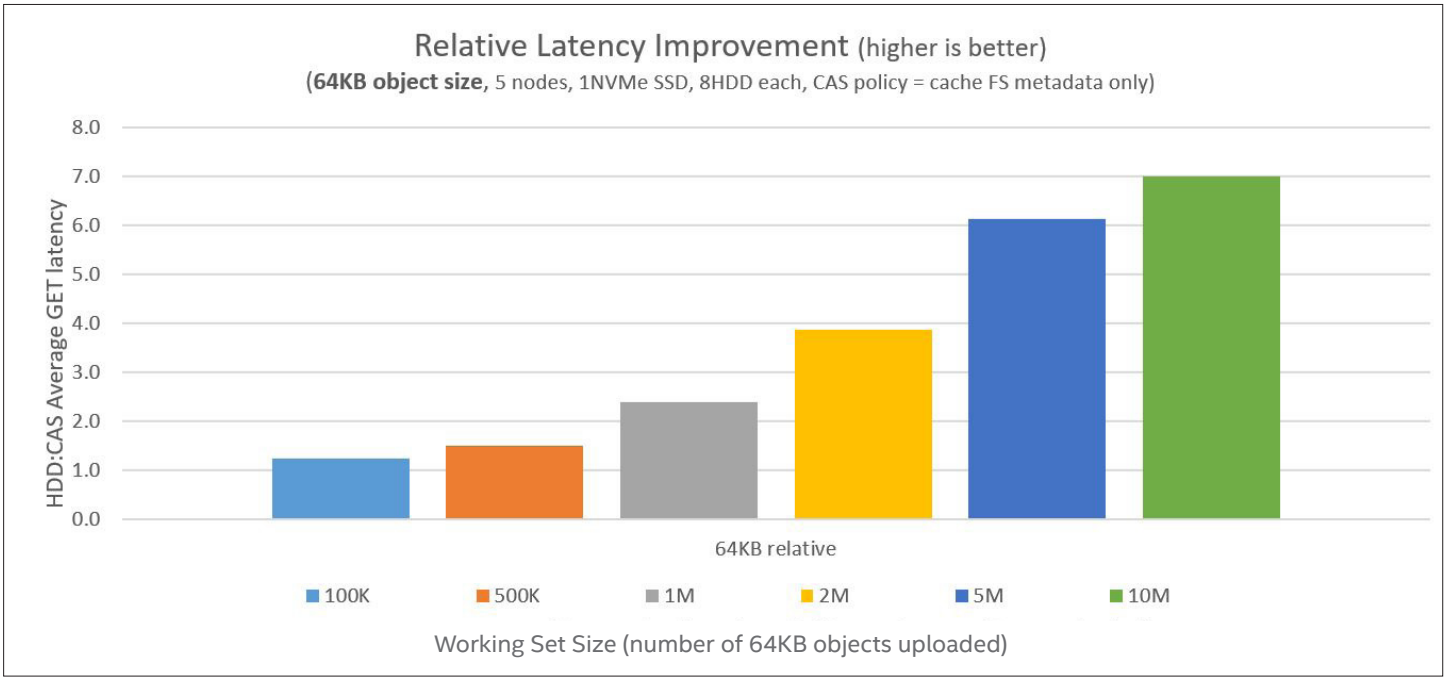

Figure 10: Relative Performance Improvement with Swift + Intel® CAS for Small Objects

Improving Replicator Performance

Swift employs a number of background consistency processes to ensure the consistency of the objects uploaded into the cluster. In this test scenario the focus is the impact of caching on the performance of the Swift object replicator process consistency. The Swift object replicator continuously traverses the filesystem directory hierarchy, generates MD5 hashes of the object contents and compares these hashes with the hashes on the other storage nodes that hold replicas of these objects.

Since the replicator is one of many consistency processes continuously running in the background, it is challenging to measure its performance. In this experiment, to measure the replicator performance in isolation, the Swift cluster is started with only the Object/Container/Account/Container servers (i.e. without any of the consistency processes). A large number of objects are then uploaded into Swift. Once the upload completes, the Swift replicator is turned on, therefore the replicator is the only consistency process running. The instantaneous performance of the replicator is monitored by the number of partition hash calculations per second. The time taken by the replicator to finish a complete replication cycle (aka replica convergence time) is also measured.

This test was performed in a Swift cluster with five storage nodes and a Swift partition power of 16. The test uploaded 25 million objects of 512KB each, spread across 25,000 containers. The SwiftStack controller telemetry plugin was used to collect the performance data.

Figure 11 charts the number of partition hash calculations per second. The baseline replicator performance is measured with HDD only. Notice that with HDD only, one replication cycle takes nearly six hours to complete.

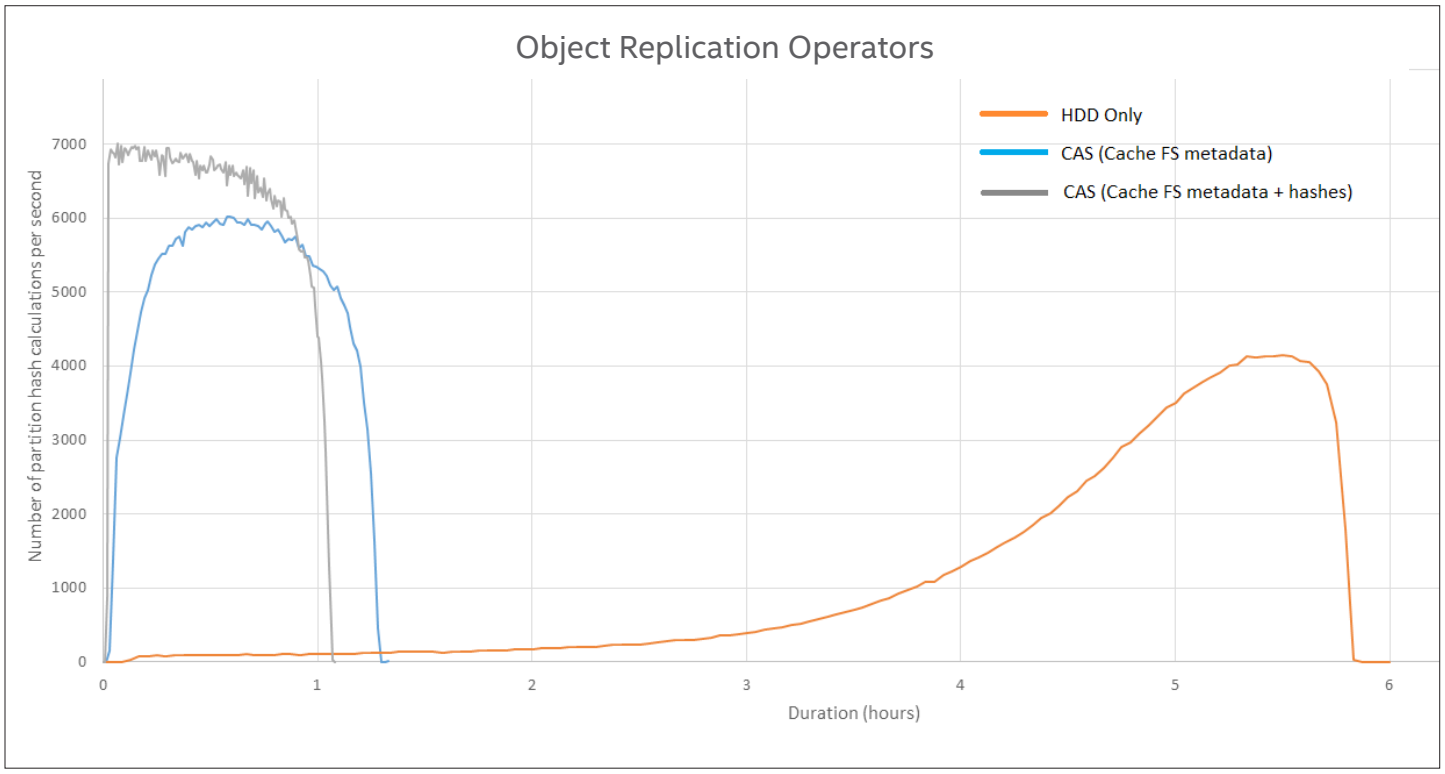

Figure 11. Swift Object Replica Convergence Time

The same test was repeated with Intel CAS configured with a policy to cache only the filesystem metadata, as with all other previous tests. The replicator took 1 hour 20 minutes to complete the replication cycle for the same number of uploaded objects. This test was repeated a third time with an Intel CAS policy to cache the replicator hashes (hashes.pkl files) in addition to the metadata. The reason for doing so, is that the replicator stores MD5 hashes on the stored object files in separate hashes files named hashes.pkl as part of its operations. These hashes files are created per Swift partition and are used heavily as object servers replicate. The results of this test show that the replication cycle time was reduced even further to 1 hour 5 min.

In summary, replicator performance data demonstrates an 82% reduction which corresponds to a >5x improvement in replica convergence time. This is down from 6hrs (for default HDD-only setup) to 1hr 20min (Intel CAS caching metadata only) to 1hr 5min (for Intel CAS caching metadata + Swift hashes files).

Conclusion

Deploying an Intel® Data Center SSD with Intel® Cache Acceleration Software provides superior performance improvements in a Swift cluster. As enterprises face exponential storage growth, they adopt solutions such as Swift, and with a hybrid storage model as presented in this reference solution, they can keep costs down and meet customer SLAs. Enterprises can benefit greatly from this high-performance solution as well as support and documentation from Intel, without the need for application modifications or an infrastructure overhaul.

References

Intel Cache Acceleration Software:

http://intel.com/cas

Intel® SSD product information:

http://www.intel.com/content/www/us/en/solid-state-drives-ssd.html

For Intel® SSD product specifications, please contact your Intel representative.

SwiftStack:

http://www.swiftstack.com

https://www.swiftstack.com/try-it-now

Openstack Swift Project:

https://docs.openstack.org/developer/swift/

Bibliography

Differentiated Storage Services. (October, 2011). 23rd ACM Symposium on Operating Systems Principles (SOSP). Cascais, Portugal: ACM.