SANDIA REPORT
SAND2022-15138
Printed October 2022

Sandia National Laboratories

Viability of S3 Object Storage for the ASC Program at Sandia

Todd Kordenbrock, Gary Templet, Craig Ulmer, Patrick Widener

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

Recent efforts at Sandia such as DataSEA are creating search engines that enable analysts to query the institution's massive archive of simulation and experiment data. The benefit of this work is that analysts will be able to retrieve all historical information about a system component that the institution has amassed over the years and make better-informed decisions in current work. As DataSEA gains momentum, it faces multiple technical challenges relating to *capacity storage*. From a raw capacity perspective, data producers will rapidly overwhelm the system with massive amounts of data. From an accessibility perspective, analysts will expect to be able to retrieve any portion of the bulk data, from any system on the enterprise network.

Sandia's Institutional Computing is mitigating storage problems at the enterprise level by procuring new capacity storage systems that can be accessed from anywhere on the enterprise network. These systems use the simple storage service, or S3, API for data transfers. While S3 uses objects instead of files, users can access it from their desktops or Sandia's high-performance computing (HPC) platforms. S3 is particularly well suited for bulk storage in DataSEA, as datasets can be decomposed into object that can be referenced and retrieved individually, as needed by an analyst.

In this report we describe our experiences working with S3 storage and provide information about how developers can leverage Sandia's current systems. We present performance results from two sets of experiments. First, we measure S3 throughput when exchanging data between four different HPC platforms and two different enterprise S3 storage systems on the Sandia Restricted Network (SRN). Second, we measure the performance of S3 when communicating with a custom-built Ceph storage system that was constructed from HPC components. Overall, while S3 storage is significantly slower than traditional HPC storage, it provides significant accessibility benefits that will be valuable for archiving and exploiting historical data. There are multiple opportunities that arise from this work, including enhancing DataSEA to leverage S3 for bulk storage and adding native S3 support to Sandia's IOSS library.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Modeling and Simulation (ModSim) efforts play a vital role in evaluating the safety, security, and reliability of the nation's nuclear deterrent (ND). Over the last decade the number of simulations that analysts run in a campaign has grown substantially, due to improvements in both the ModSim tools used to explore a parameter space and the speed at which high-performance computing (HPC) platforms can execute parallel workloads. While these advances enable analysts to explore a problem space in greater fidelity, there is concern that our ability to generate data greatly exceeds our ability to archive, digest, and disseminate data products that are of use to the overall community. This gap makes it difficult for analysts to benefit from each others work and results in institutional knowledge being lost to the machinery.

Digital engineering efforts at Sandia are taking steps to help remedy this problem by constructing information systems that can help members of the ND community find all historical data related to a simulated component. Efforts such as DataSEA have constructed metadata search engines that make it easier for users to publish experimental results in a way that creates a lab-wide notebook for others to inspect and supplement. The inherent value of such information systems has created a stampede to create new tools to automatically generate metadata artifacts that can be ingested by DataSEA. We anticipate that these systems will evolve rapidly, and that users will expect to be able to easily perform searches on the metadata and be provided with universal references into capacity data stores that allow them to retrieve the slivers of data they need to their laptop.

In this project we are exploring how new, institutional S3 storage systems can play a role in providing bulk storage for this environment. This report documents our experiences in developing C++ software to interact with S3 storage servers and provides performance measurements for the storage system from multiple perspectives within Sandia's environment.

1.1. Scientific Computing Environments

In an ideal scientific computing environment, users would be able to go to a single computing platform where they could run all their simulations, store all the datasets they ever needed or produced, and be able to locate any other portion of data generated by the institution that might be relevant to their work. This ideal environment is challenging to create for many reasons. First, computing platforms are stood up at different times and locations, and often live in network islands that have limited bandwidth for moving data between platforms. Second, there is a significant cost difference between storage optimized for *performance* versus *capacity*. Thus, it is cost prohibitive to construct a single storage system that performs both roles adequately. Finally, from a programmatic perspective, all projects have finite duration and funding. As such, projects often lack a way to pay for perpetual storage after the work completes.

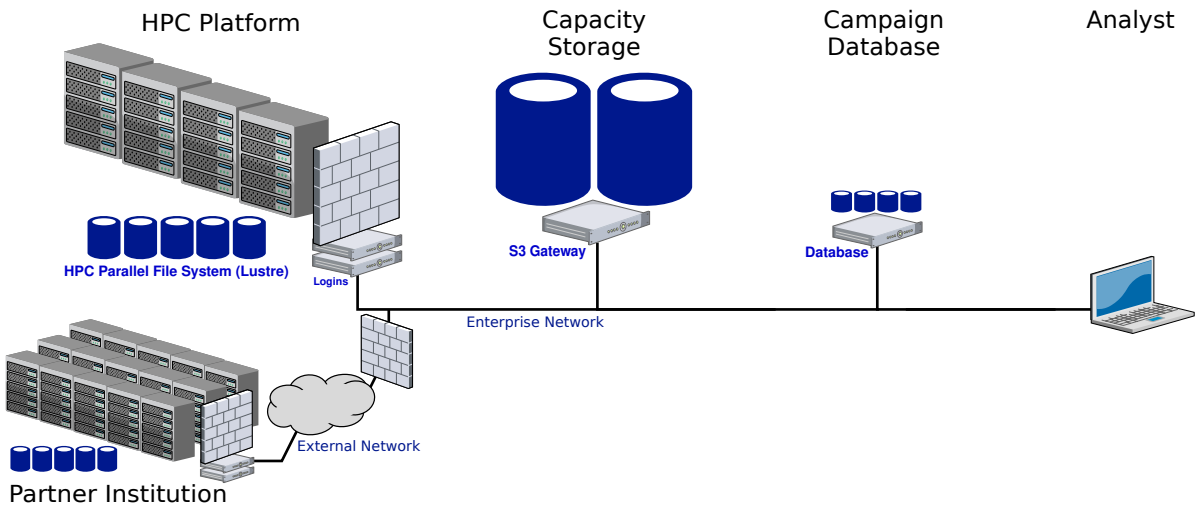

Figure 1-1. HPC platforms and institutional storage systems may be distributed in different locations in an enterprise.

Fortunately, institutions can create productive scientific environments by investing in key resources for different needs, and constructing workflows that allow data to be migrated between different storage systems as a program's lifecycle progresses. As illustrated in Figure 1-1, there are three types of computing resources that an institution may field to support large computing campaigns:

High-Performance Computing Platforms: Scientific computing facilities typically maintain multiple high-performance computing (HPC) platforms in different data centers, and often have partnerships with other institutions or cloud vendors to leverage their resources remotely in specific programs. An HPC platform may contain thousands of compute nodes to process parallel applications efficiently and a parallel file system for holding input and output datasets. HPC storage systems are optimized for performance and generally use a large array of NVMe storage devices and a performant parallel file system such as Lustre to source and sink data for parallel applications. While these storage systems provide a POSIX API for users to read and write files, the storage system is not made available to systems outside of the HPC platform to ensure that external network disruptions do not impede HPC application execution. Given the limited capacity of these systems, users are frequently asked to migrate their result datasets off to other storage systems once parallel analysis tools have harvested the desired information from the results.

Capacity Storage: Scientific computing institutions also provide capacity storage for users to house their data over long periods of time. Capacity storage systems employ thousands of rotational hard drives that favor capacity over performance. These systems can be implemented with proprietary storage devices that are common in non-HPC environments, or through open source file systems such as Ceph [7]. The popularity of cloud-based systems has driven many institutions to choose Amazon's Simple Storage Service (S3) API to allow users to access their data. S3 enables institutions to place a capacity system on the enterprise network, as S3 does not require individual systems to establish long-term

mountpoints with the store.

Database Systems: Scientific computing institutions also field a variety of database-like systems to help users track the state of different information products used in daily operations. These systems can include traditional SQL databases from commercial or open source sources, or NoSQL systems such as Elasticsearch [3], MongoDB [2], and Solr [4]. Given the cost of storing indexed data, database-like systems target the storage of small datasets or metadata about large datasets. The systems are accessible from the enterprise network through standard APIs. For HPC computing users, these systems are commonly used to track information about all the activity taking place in a computing campaign.

1.2. Institutional Storage

In FY2022 Sandia's Institutional Computing procured a new S3 object storage system that can be used by all mission spaces across the enterprise. This system resides on Sandia's restricted network (SRN) and includes deployments at both the SNL/NM and SNL/CA sites with automatic mirroring. These systems were designed in a modular fashion that allows the system to be expanded with additional storage as needed by projects. There are opportunities for additional deployments on both the unclassified and classified networks, provided that the SRN system proves to be a worthwhile investment.

For the ND mission space, the new institutional storage represents an opportunity to modernize the way simulation and experiment data is stored and accessed within Sandia. In addition to providing a common place to house massive amounts of data, the system provides flexible access control and the ability to allow users to make selective retrievals of large datasets. However, S3 is not an interface that any of our existing ASC I/O software can currently access. If ASC I/O researchers do not make the S3 storage systems easy to use, analysts simply will not use the system.

1.3. FY22 Project Objective

This project was established to explore the characteristics of enterprise S3 storage and make recommendations as to how Sandia's ASC software can be modified to make it easy for analysts to take advantage of the hardware. In FY22 this project explored different aspects of working with S3 storage to answer the following research questions:

- *How do C++ developers construct software to interact with S3 stores?*
- *What are the performance transfer characteristics for different S3 systems at Sandia?*
- *How could Sandia's IOSS library be refactored to leverage S3?*
- *How might S3 be incorporated into Sandia's data engineering efforts?*

The remainder of this document provides a summary of the lessons learned while working with different S3 storage systems at Sandia. In Section 2 we summarize how S3 operates and describe how developers can exchange data objects over the network from C++. In Section 3 we describe experiments that examined how fast different HPC systems could transfer data to S3 storage systems that were located outside of the platforms on our enterprise networks. Additional observations made from these experiments is discussed in Section 4. A second set of experiments are presented in Section 5 to explore whether S3 performance can be improved if the storage system is built with high-performance hardware and located within the computing platform's interior. Finally, in Section 6 we discuss options for how Sandia's I/O libraries could be updated to leverage S3, and discuss opportunities for leveraging S3 storage in ongoing modernization efforts.

2. S3: SIMPLE STORAGE SERVICE

Amazon introduced its Simple Storage Service (S3) in 2006 to provide users with a way to store and access large collections of data objects on their storage servers. The S3 protocol was designed to allow users to connect to the store from anywhere in the world, without requiring any special I/O drivers from the client operating system. S3 uses a REST API that runs on top of HTTPS network connections. As such, developers have ported S3 client software to run in a wide array of programming languages. From a user's perspective, an account holder creates a *bucket* on the storage system to hold objects and then uses *put*, *get*, *list*, and *delete* commands to store, retrieve, find, and remove string-labeled objects. Unlike traditional POSIX file systems, S3 stores allow users to place a massive number of items in a single container without performance penalties.

Given the simplicity and usefulness of the S3 API, many vendors besides Amazon have adapted their storage products to have an S3 gateway that S3-based applications can access. For example, the Ceph capacity file system provides a gateway service that allows system designers to host S3 data on Ceph. Unfortunately, this interface is not compatible with Ceph's native RADOS [8] object store (i.e., users cannot store objects with the faster RADOS interface and expect to access them from the S3 gateway). Similarly, commercial storage systems such as IBM Spectrum Scale (GPFS), Dell ECS, MinIO, Zenko, Riak S2, and Triton have S3 interfaces to their products.

2.1. Advantages of S3 Data Stores

There are multiple advantages of using an S3 data store for holding scientific datasets:

Accessibility: The S3 API enables users to connect to the store through any device that can communicate over an IP network. This property makes data accessible from a variety of locations in the enterprise, including HPC platforms, user desktops, containers, VMs, and web services.

Userspace Control: Traditional file systems such as NFS require privileged access to establish a mount point, and make the client computer's stability dependent on its ability to have continuous connectivity to the server. In contrast, S3 operations can be performed in userspace and complete when transfers finish.

On-Demand Access: File-based approaches generally require users to retrieve a large file before they can process it, whether they need all the data or not. S3's *put/get* interface allows an application to pull data on-demand, as the application requires. This property is extremely desirable in analytics where only a subset of the source data is required.

Dataset Composition: The fact that S3 is an object store means that it is easy for developers to decompose a dataset into a large collection of labeled objects that can be exchanged with the store as needed. In contrast, today's file-based approaches require developers to think in three levels: (1) a file-format level for storing data fragments into a single file with a container library such as HDF5 or netCDF, (2) a parallelism level for managing the output files of multiple ranks, and (3) a corpus level where different runs relate to each other. An object-store approach provides a single abstraction where a single key naming strategy covers everything.

2.2. Disadvantages of S3 Data Stores

Similarly, there are multiple disadvantages compared to traditional HPC filesystems:

Poor Performance: S3 stores use a REST API (i.e., commands over HTTPS) that are layered on top of traditional TCP/IP protocols. These protocols are a recipe for poor performance compared to other storage APIs that have been optimized for HPC communication fabrics.

Unfamiliar APIs: Users will always have a difficult time switching from file-based systems, where users can see and touch their files, to object stores, where a dataset may have millions of machine-named objects. The inability for an S3 installation to distill data into a human-observable form is a hindrance to its acceptance.

2.3. Basic Command Line Access to an S3 Store

The easiest way to start interacting with an S3 store is through the Amazon Web Service (AWS) command line interface (CLI) tool. While the `aws` cli is designed as an all-in-one tool for interacting with Amazon's clouds, it can be configured to communicate with other platforms by updating the configuration and credentials files in a user's `.aws` directory. The `aws` tool has an `s3` subcommand that allows users to manipulate and exchange objects with an S3 server. For data management operations that are more sophisticated, we recommend users switch to the Python library *Boto3*. Boto3 provides a more fluid environment for issuing queries to an S3 server, and has workarounds for issues that hinder normal command-line operations.

2.4. S3 Development in C++

As the creator, publisher and maintainer of the S3 specification, Amazon is the primary provider of S3 development tools. The *Amazon Web Service SDK* is a large-scale software package that gives developers the tools needed to create and deploy all manners of cloud-based web services. This SDK covers a large number of languages (e.g., JavaScript, PHP, Python, Ruby, Java, and C++) and multiple platforms (e.g., Linux, Android, Windows, and iOS). Within the larger AWS SDK is the *S3Client* package that provides the discovery, authentication, and data transfer mechanisms that are required for accessing an S3 storage service.

Developing C++ applications with the S3Client is straightforward, as the library follows a request/response pattern that will be familiar to client/server software developers. After initializing the AWS SDK, developers construct an S3Client object that encapsulates network parameters, the object store's location, and sources of authentication credentials. The S3Client is the entry point for S3 and includes synchronous and asynchronous methods for executing all S3 operations. Executing an S3 operation involves (1) creating a new request object for a specific operation, (2) configuring the object's desired settings, (3) passing the object to the S3Client to start the operation, (4) waiting until the operation completes, and (5) extracting and inspecting a response object for the operation. Completion notifications for the asynchronous API methods take place through either *futures* or *callbacks*.

Put/Get methods that transfer objects require that the data be managed through a C++ stream. Streams provide the underlying HTTP transport a common way to transfer data from/to memory, disk, network entity, or any other byte stream that can be wrapped in an C++ `std::iostream`. One issue with this approach is that the C++ standard does not provide a stream class that can wrap a memory buffer without making a copy. For small buffers this may not significantly impact performance. However, for large-scale data from ModSim applications, this overhead may result in processing and memory overheads that disturb the application. Developing a custom stream class that can better serve up bytes from a data array could alleviate this problems (see Future Work in Section 6.1.1).

2.5. S3 Authentication

The S3 authentication system requires each user to have a two-part key composed of an access key ID (i.e., username) and a secret access key (i.e., password). These keys are created by some management tool which could be a web dashboard (NetApp StorageGRID) or programmatic interface (Dell ECS MDSS).

In the Amazon Web Services ecosystem, authentication keys can be stored and retrieved in a variety of system-specific ways. The default authentication module creates a provider chain that is traversed in search of a provider that can authenticate the user. Because a user will typically only have one AWS account, the provider assumes that a user's `default` profile is in use.

Sandia has two corporate S3 services that have separate user databases which creates separate private authentication key pairs for each service. A user of both services will need a profile for each. The name of the profile is not a simple configuration parameter to the default provider chain. Instead, a credentials provider must be created for use by the S3Client object.

2.6. S3 Development on Sandia's SRN

The AWS SDK is designed with the intention that an application will be deployed on the Internet in the Amazon Web Services ecosystem. This implies a number of implementation defaults that are not appropriate for a deployment on a private network such as Sandia's SRN. For example, when an S3 client application initializes, it determines its location (i.e., region) and then tries to

contact an appropriate registry service for the region. These services are not available on Sandia's private network, and will eventually timeout after one second. While this delay may be tolerable in long-running applications, it becomes excruciatingly noticeable in point tools that perform small interactions with an S3 store. It would be beneficial to explore modifications to the SDK to customize the SDK stack for use in a closed network environment.

2.7. S3 Examples

The benchmarks used for this report are available in the CEE Gitlab repository at `https://cee-gitlab.sandia.gov/dsva/s3-examples`. The `README.md` available in that repository describes the structure and usage of the benchmarks, and there are a number of example build scripts available in the `scripts` subdirectory of the repository.

Likely the most interesting of the benchmarks available in the repository are `05_ceph_benchmark` and `06_aws_benchmark`. These perform the same test against a Ceph object store and an S3 object store respectively. Each of these tests connects to the object storage endpoint, performs *N* object puts, *N* object gets, and *N* object deletes. The number and size of the objects is configurable through a command-line parameter to each test.

In addition to the S3 benchmarks, there is `06a_posix_benchmark` which performs the same tests against a POSIX file system. This benchmark opens, writes and closes *N* files, opens, reads and closes *N* files and deletes *N* files.

3. ENTERPRISE S3 EXPERIMENTS

In order to better explore the performance characteristics of S3 storage, we constructed a benchmark program and then ran a variety of tests on different platforms. In this section, we consider the case where compute nodes in an HPC platform attempt to exchange data with an S3 storage system that is hosted *outside* the HPC platform, somewhere on the SRN enterprise network. These tests stress the pathways leading out of the HPC platform as well as through the SRN. For this work we measured performance on four different platforms at Sandia: CEE-Compute, Eclipse, Ghost, and Vortex.

3.1. Enterprise S3 Storage Systems

In these experiments, two different S3 endpoints were used:

CEE-S3: CEE-S3 is an object store managed by the Common Engineering Environment organization. CEE-S3 is a NetApp StorageGRID SG5760 configured with 8 2.2 GHz cores and 4 10 Gigabit Ethernet connections. The raw capacity has not been disclosed. Our tenant account was created with a 5 TB limit.

MDSS (Mission Data Storage Service): MDSS is an object store managed by Center 9300. MDSS is a Dell/EMC Elastic Cloud Storage (ECS) system. MDSS has an installation at both SNL/NM and SNL/CA. A pair of load balancers at each facility distributes client activity to the available storage nodes. The SNL/NM installation is 2 racks of storage nodes with 16x 25 Gbps connections each and a total capacity of 15 petabytes. The SNL/CA installation is 1 rack of storage nodes with 8x 25 Gbps connections and a total capacity of 7.5 petabytes.

In addition to the S3 object stores, each system has access to either local, integrated or shared scratch file systems. cee-compute servers have a locally attached scratch disk and access to the shared centralized GPFS file system at /gpfs. eclipse and ghost nodes have access to the shared GPFS file system at /gpfs and the shared Lustre scratch file systems at /nscratch and /pscratch. Vortex has a integrated GPFS scratch file system at /vscratch1.

3.2. Testing Methodology

The plots below are all reported in the megabytes per second (MBps).

To calculate the throughput of synchronous operations for each object size, each rank records the throughput of its 100 transfer operations, each rank then computes the average throughput of its

100 transfer operations, the average for each rank is summed at rank 0 which calculates the average of the averages.

There were additional asynchronous throughput tests run on a small subset (cee-compute to CEE S3 object store) of the test environment. Due to software upgrades and system availability, the asynchronous tests were not run on the other HPC systems or MDSS Object Store. To calculate the throughput of asynchronous operations for each object size, each rank records the total amount of time needed to complete all the transfers, each rank then computes the average throughput of its 100 transfer operations, the average at each rank is summed at rank 0 which calculates the average of the averages.

The primary focus of our work was to evaluate the corporate S3 object stores as globally available resources. We did have the opportunity to run a few small experiments on the Glinda cluster with an integrated private S3 object store. This object store is a Ceph object store with an S3 interface. We ran both the Ceph benchmark and the AWS benchmark to compare the costs of the two API. The Ceph benchmark shows a considerable boost for put operations while performing nearly identically on get operations.

The plots in this section are organized by client platform; that is, where the benchmark was run to produce the results.

3.3. Benchmarking Parameters

The experiments described below were run on all the described HPC systems using a common set of launch scripts. These scripts execute four tests at four allocation sizes using object sizes from 2 bytes to 512 megabytes. (Table 3-1).

Ranks	Node Count	Processes Per Node	Objects Per Rank	Object Sizes
1	1	1	100	$2^1 - 2^{29}$ bytes
2	2	1	100	$2^1 - 2^{29}$ bytes
4	2	2	100	$2^1 - 2^{29}$ bytes
8	2	4	100	$2^1 - 2^{29}$ bytes

Table 3-1. Experiment Setup

3.4. CEE-Compute

The Common Engineering Environment (CEE) organization at Sandia provides a collection of build servers and compute servers that are cooperatively shared among all users. System components vary from machine to machine and it is the responsibility of users to ensure they do not launch tasks that collide with other users in the system. We selected two compute servers (cee-compute024 and cee-compute26), which feature Intel Xeon E7-4880v2 processors (60 cores at 2.50GHz), 1TB RAM and 2 Broadcom NetXtreme II 10 Gigabit Ethernet connections. Each server has a GPFS file system mount and a locally-attached scratch file system for POSIX file I/O.

3.4.1. CEE-Compute Write Performance

S3 Put performance for the CEE-Compute system is presented in Figure 3-1 for synchronous transfers. The tests were only able to achieve approximately 80 Mbps when communicating with either S3 store. For comparison, CEE-Compute's POSIX write performance for GPFS and scratch is presented in Figure 3-2. The measurements confirm that I/O with the platform's file systems is more than an order of magnitude greater than S3 performance.

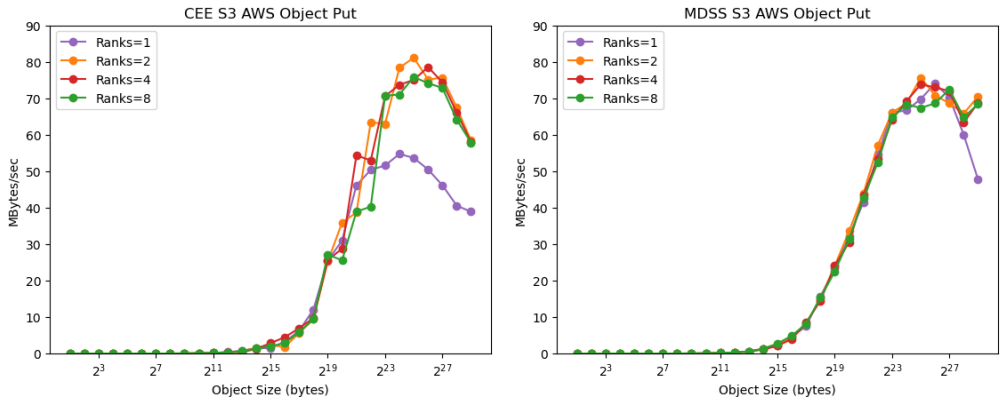

Figure 3-1. CEE-Compute Object Put CEE-S3 vs MDSS

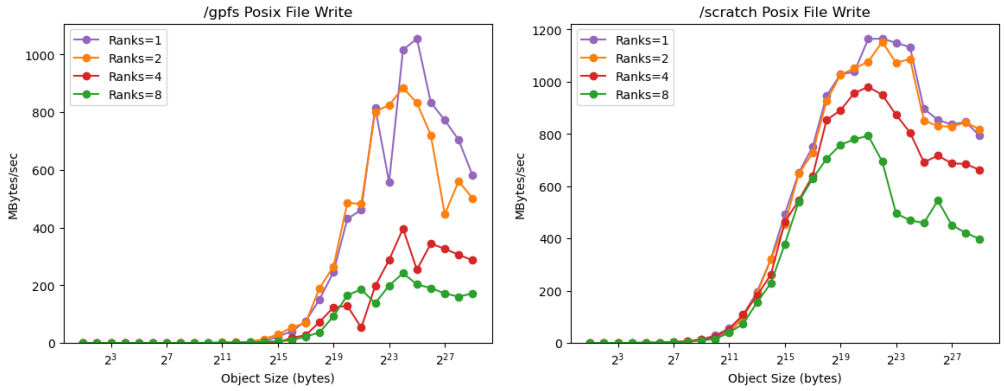

Figure 3-2. CEE-Compute File Write /gpfs vs /scratch

3.4.2. *CEE-Compute Read Performance*

S3 Get performance for the CEE-Compute system is presented in Figure 3-3 for synchronous transfers. While peak performance was higher than it was for reads, throughput is still close to gigabit speeds. For comparison, CEE-Computes POSIX read performance for GPFS and scratch is presented in Figure 3-4. POSIX performance is roughly double that of S3.

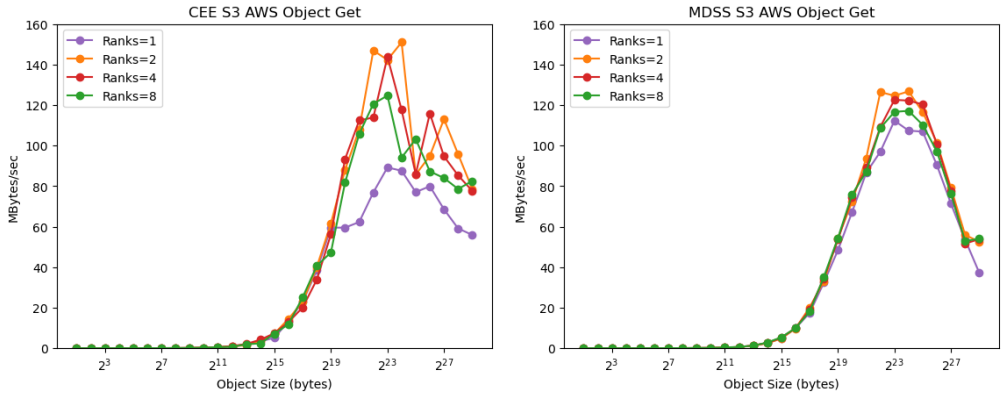

Figure 3-3. CEE-Compute Object Get CEE-S3 vs MDSS

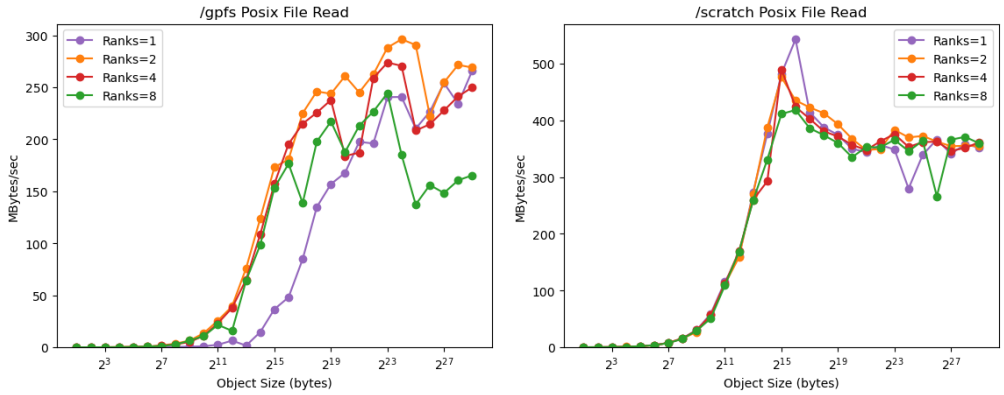

Figure 3-4. CEE-Compute File Read /gpfs vs /scratch

3.4.3. CEE-Compute Asynchronous Performance

The asynchronous put and get performance for CEE-S3 is presented in Figure 3-5. While asynchronous performance gives an order of magnitude performance gain over the synchronous approach in some cases, it is important to note that asymmetric performance degrades significantly as more ranks are employed.

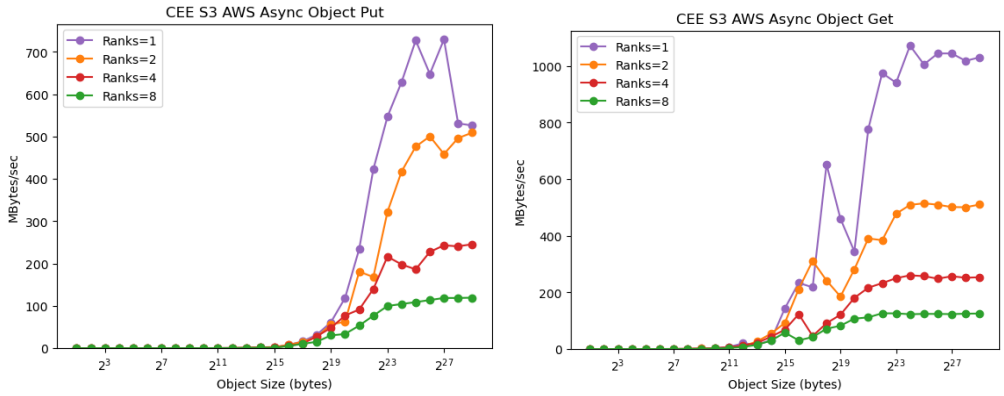

Figure 3-5. CEE-Compute Object Async Put and Get for CEE-S3

3.5. Eclipse and Ghost (CTS-1)

Eclipse and Ghost are members of the CTS-1 family of capacity clusters. CTS-1 clusters are based on the Tundra Open Rack Solution generation of hardware from the TLCC3 procurement. Each CTS-1 compute node has dual Intel Broadwell E5-2695v4 processors, 128GB RAM and 1 Intel Omni-Path 100 Series 100 Gbps (4x25) InfiniBand connection. This gives each compute node an aggregate bandwidth of 25 GBps.

Eclipse and Ghost have three centralized shared scratch spaces available. There is the corporately provided GPFS parallel file system mounted on /gpfs and two corporately provided Lustre parallel file systems mounted on /nscratch and /pscratch. CTS-1 resources are managed by the SLURM suite. For these benchmarks, an interactive session of two nodes was allocated.

3.5.1. Eclipse Write Performance

Write performance is presented in Figure 3-6 for S3 and Figure 3-7 for POSIX. Increasing the number of ranks impeded performance more than it did on the CEE system.

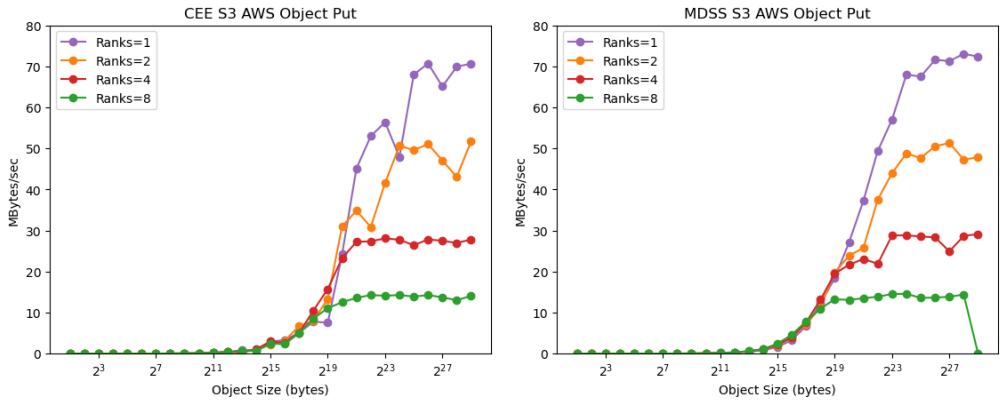

Figure 3-6. Eclipse Object Put CEE-S3 vs MDSS

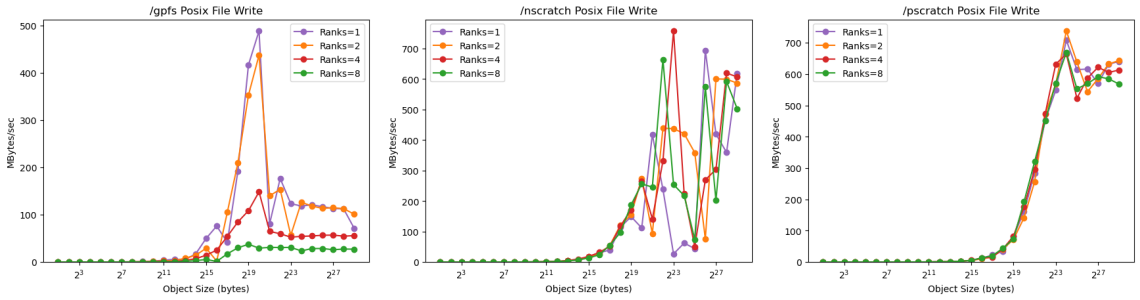

Figure 3-7. Eclipse File Write /gpfs vs /nscratch vs /pscratch

3.5.2. *Eclipse Read Performance*

Eclipse read performance is presented in Figure 3-8 for S3 and Figure 3-9 for POSIX I/O.

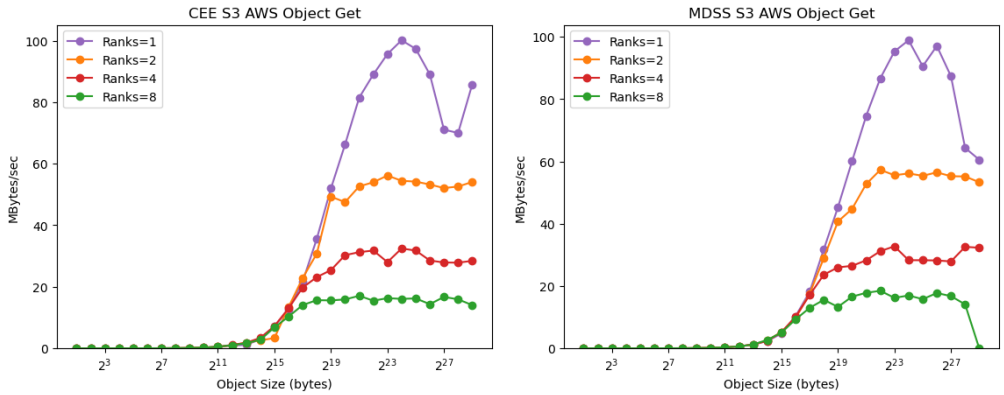

Figure 3-8. Eclipse Object Get CEE-S3 vs MDSS

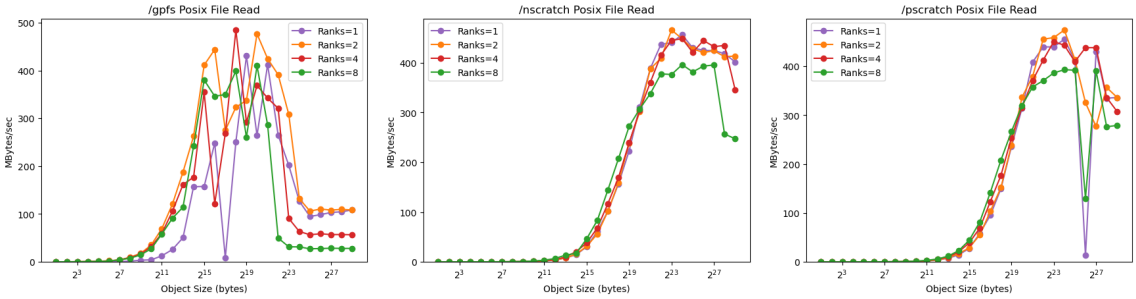

Figure 3-9. Eclipse File Read /gpfs vs /nscratch vs /pscratch

3.5.3. *Ghost Write Performance*

Ghost write performance is presented in Figure 3-10 for S3 and Figure 3-11 for POSIX.

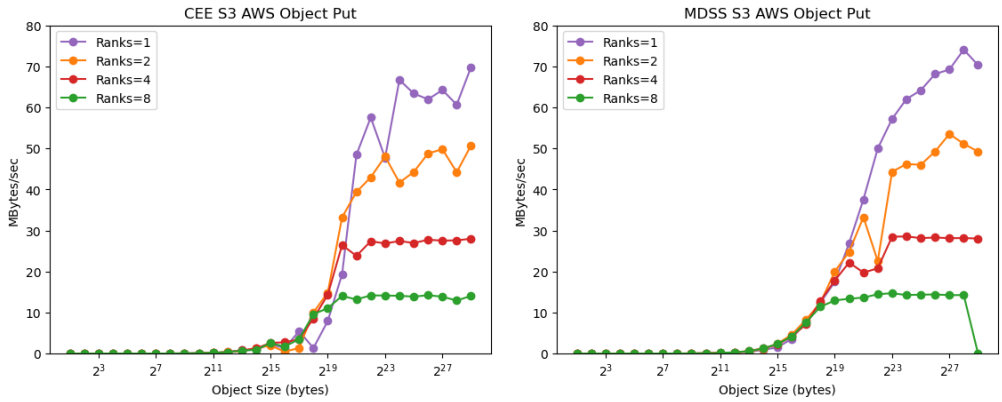

Figure 3-10. Ghost Object Put CEE-S3 vs MDSS

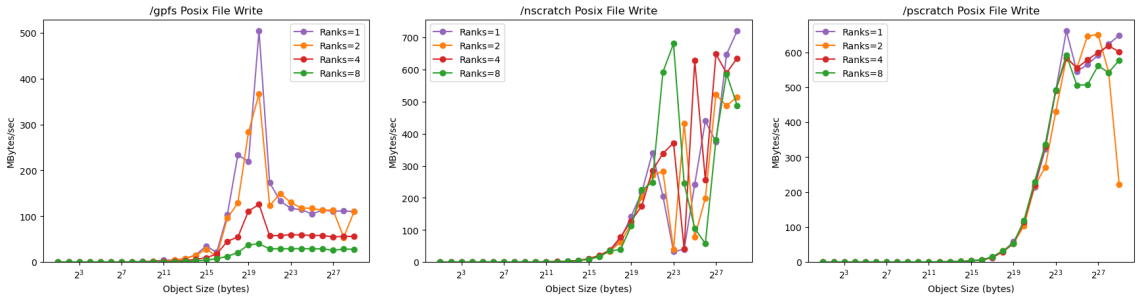

Figure 3-11. Ghost File Write /gpfs vs /nscratch vs /pscratch

3.5.4. *Ghost Read Performance*

Ghost read performance is presented in Figure 3-12 for S3 and Figure 3-13 for POSIX I/O.

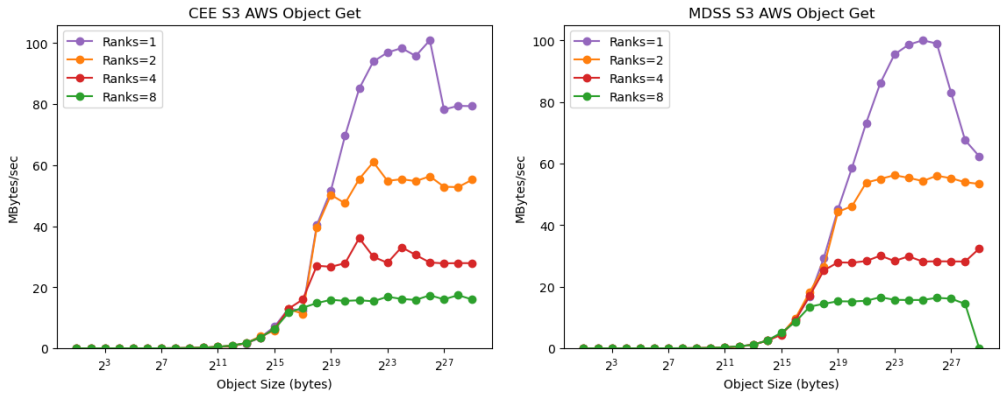

Figure 3-12. Ghost Object Get CEE-S3 vs MDSS

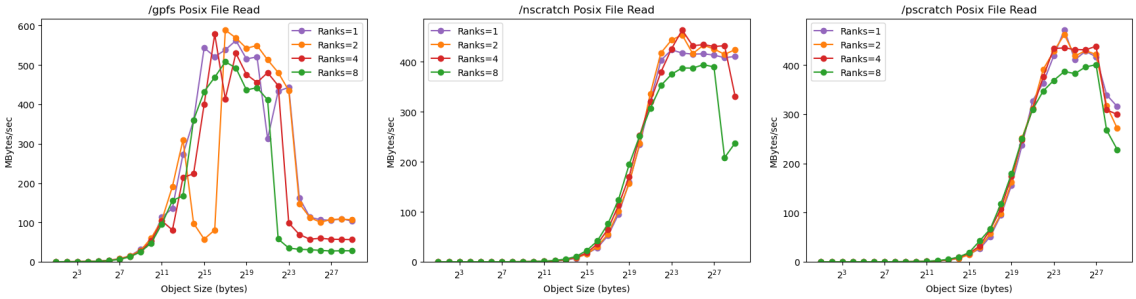

Figure 3-13. Ghost File Read /gpfs vs /nscratch vs /pscratch

3.6. Vortex

Vortex is an HPC testbed platform configured to mimic the LLNL Sierra platform. Vortex is running the CORAL software stack and is for the development, evaluation and benchmarking of codes targeting the Sierra platform. Each compute node has dual Power9 CPUs (22 cores each), 318GB RAM, 4x NVIDIA Tesla V100 GPUs and 4x Mellanox ConnectX-5 Ex at 100 Gbps. This gives each compute node an aggregate bandwidth of 40 GBps. Vortex has an integrated GPFS scratch space mounted on /vscratch1. This scratch space is private to the Vortex cluster, but is shared among all the compute and login nodes. Vortex resources are managed by the IBM LSF suite. For these benchmarks, an interactive session of two nodes was allocated.

3.6.1. Vortex Write Performance

Write performance for Vortex is presented in Figure 3-14 for S3 and Figure 3-15 for POSIX. While less impacted by threading than other systems, Vortex operated at roughly half the rate of other systems.

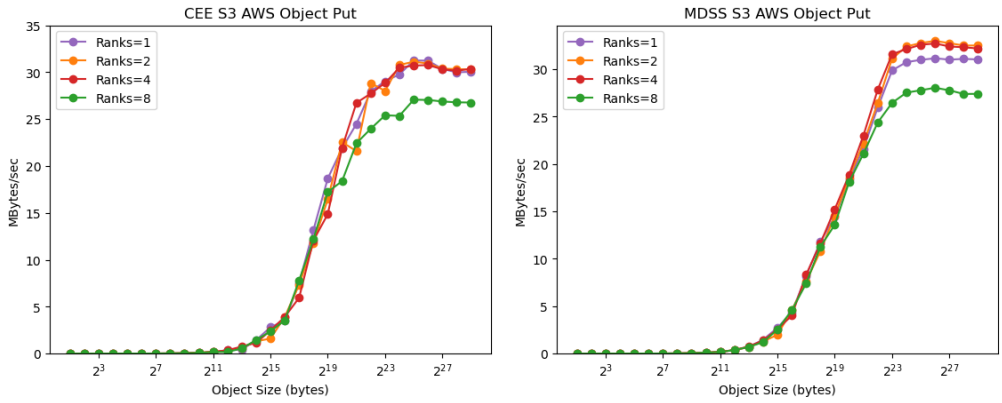

Figure 3-14. Vortex Object Put CEE-S3 vs MDSS

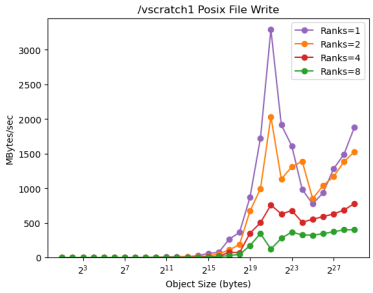

Figure 3-15. Vortex File Write /vscratch1

3.6.2. *Vortex Read Performance*

Vortex read performance is presented in Figure 3-16 for S3 and Figure 3-17 for POSIX I/O.

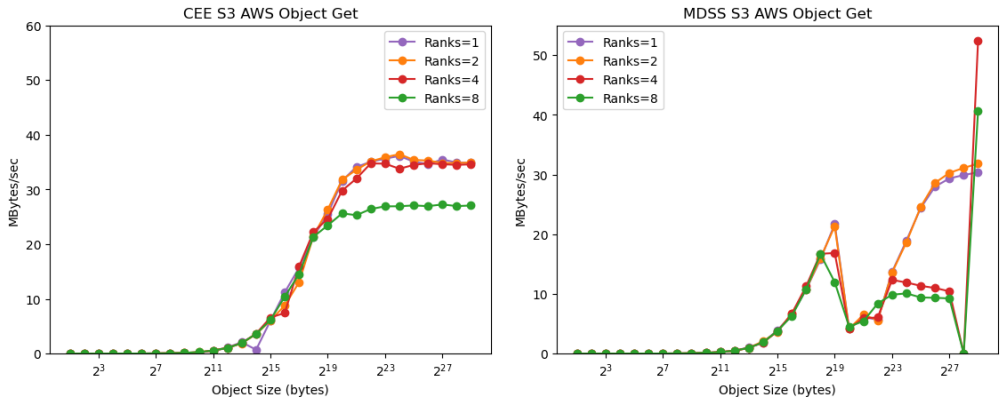

Figure 3-16. Vortex Object Get CEE-S3 vs MDSS

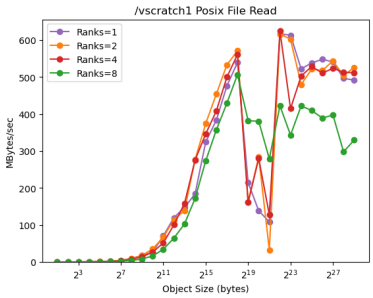

Figure 3-17. Vortex File Read /vscratch1

4. ENTERPRISE S3 DISCUSSION

From our survey, the expected performance of the CEE S3 and MDSS S3 object stores is approximately 80 MBps for synchronous put operations except on Vortex which seems limited to 35 MBps. The throughput of synchronous get operations is not as consistent. Both CTS-1 platforms achieve approximately 100 MBps, while the cee-compute servers are better at approximately 140 MBps and Vortex is lagging at the same 35 MBps.

This type of performance should not be a surprise because the S3 protocol runs over HTTP. Unless there is special routing in the HPC clusters for HTTP traffic destined for the S3 object stores, this traffic is most certainly routed over the SRN. The SRN is designed for desktop network traffic like email, web services and small file transfers not large bursts of ModSim data. Given that, the SRN does surprisingly well on the benchmarks we have run. As the number of ranks scales up, it is hard to say how well it will hold up. There are a few potential bottlenecks that should be considered.

The routers at the edge of the HPC clusters are probably not sized for this amount of the traffic. As the number of ranks increase, so will the size of the simulation. This implies a larger total size of the data written even if the size of the individual writes may decrease. This will be more impactful as traffic reaches the capacity of the routers.

A maximum throughput of 100 MBps is suspiciously close to the throughput of a 1 Gbps Ethernet link. It's possible that there is a slow link in the path that does not show up under typical traffic conditions but becomes an issue when transferring large datasets.

The small sample of asynchronous benchmarks shows more promise. The overall throughput of the entire job is approaching the possible bandwidth of a single 10 gigabit Ethernet NIC. Given that there are two NICs per server, there is still work to be done here to split traffic between the NICs but it may be possible to achieve better application performance if ModSim data can be broken into chunks that can be transferred concurrently.

Eclipse and Ghost show an interesting performance trend. As the number of ranks increases even when the ranks are on different nodes, the average performance with 2 ranks (1 per node) is 30-40% slower than a single rank. The performance continues to drop as more ranks are added. The same is not true for the Lustre scratch file systems which show sustained (if variable) performance as the number of clients and file sizes increase. The GPFS scratch file system shows a somewhat similar performance drop at large file sizes.

4.1. Traditional Scratch File Systems

For comparison, we looked at the performance of the traditional scratch file systems available on each of the systems. The setup is the same as the object stores – each rank writes then reads 100 files from 2 bytes to 512 megabytes. The results show that file systems designed and tuned for use as scratch space do offer better read/write performance. However, the results are not identical across all the scratch file systems.

4.2. Local Scratch

The locally attached /scratch file systems on the cee-compute servers shows stable performance even when the number of ranks increases which leads to more contention. The impact of the contention is best seen in the write performance which drops by approximately 30% when the number of ranks per node is doubled from 1 to 2. The drop is another 30% when the ranks per node doubles from 2 to 4.

The integrated /vscratch1 on vortex shows reliable write performance throughout the entire range of file sizes. There is some signs of contention at the middle and large files sizes. Read performance is similarly good with some significant drops in the middle sizes. This is likely due to contention with other jobs doing significant I/O at the same time.

4.3. Centralized Shared Scratch

The centralized shared /gpfs scratch file system is accessible from both the CTS-1 systems. Both systems show nearly identical and interesting performance. The write performance is reliable but not particularly fast. For writes of 128KB to 1MB inclusive, there is a huge spike to nearly 500 MBps. Outside of the this range the write performance is a predictably smooth curve. In the read case, there is also an interesting performance boost from 32KB to 4MB. This is not a short spike, but a quick rise in performance at the given sizes with a return to the expected performance. /gpfs is tuned for many small I/O operations which could explain this boost.

The centralized shared /nscratch and /pscratch Lustre file systems are accessible from both the CTS-1 systems. While the two file systems do not show the same performance when compared to each other, the performance seen by the two CTS-1 systems is similar. These Lustre file systems are tuned for large I/O operations which shows in the results. /pscratch write and read performance ramps quickly beginning at 512KB and remains high for most of the remaining sizes. There is a small drop in read performance at the 2 largest file sizes. /nscratch read performance matches that of /pscratch, but it's write performance shows very high variability beginning at 512KB. It's unclear if this is contention in the cluster, on the network or at the file system. But it is present in the results on both CTS-1 systems which were collected at different times.

5. CEPH S3 EXPERIMENTS

As the previous experiments demonstrated, it is challenging for a compute job running on an HPC platform to obtain good performance from an S3 storage system on an enterprise network due the limited bandwidth into and out of the platform. While these issues can be mitigated by providing direct links between the platform and storage system, a fundamental question still remains: how performant can S3 transfers be if the storage system is built with high-performance components?

5.1. Glinda Test Environment

The new Glinda cluster at the SNL/CA site provided a unique opportunity for us to explore this possibility before the platform was turned over for production use. Glinda is a new high-performance data analytics (HPDA) platform at Sandia that features 32-core AMD EPYC Zen3 processors, NVIDIA A100 GPUs, and 100 Gbps InfiniBand networking. An experimental storage cluster was established for Glinda and attached to its InfiniBand storage network. Each node in the storage system includes 64 cores, 1TB of memory, and ten 14TB NVMe storage devices. The SNL/CA is currently evaluating which storage software to deploy on this system and welcomed us to experiment with the hardware.

We installed Ceph on the storage system and enabled it to serve CephFS mountpoints, a Ceph RADOS object store gateway, and an S3 gateway. All network traffic flows over the InfiniBand NICs, using IP-over-IB communication. While we have only conducted basic experiments with CephFS and RADOS, performance has been underwhelming: hardware capable of delivering 12 GBps struggles to deliver 4.5 GBps over the network in single-node, threaded RADOS benchmarks. We theorize that Ceph's use of TCP/IP is an impediment to its performance with high-end hardware, and note that this is ongoing work within the Ceph community to map to improve network throughput. While Ceph may not be selected as the end storage medium for the Glinda storage system, it does provide a high-end S3 store for our testing.

For the performance tests conducted on the Glinda nodes, we extended our benchmark software to use the `libs3` library that was developed by the Ceph team. While this S3 implementation does not have all the features of the AWS SDK S3 library, it is easy to build and use. We report performance numbers for both the Ceph and AWS S3 implementations.

5.2. Glinda Ceph S3 Put Performance

Glinda write performance for the local S3 storage system is presented in Figure 5-1. Ceph's S3 library offered higher performance than the AWS SDK, but still only achieved roughly 200 MBps of performance on hardware that had multi-GBps potential. The number of threads used to write data did not impact performance.

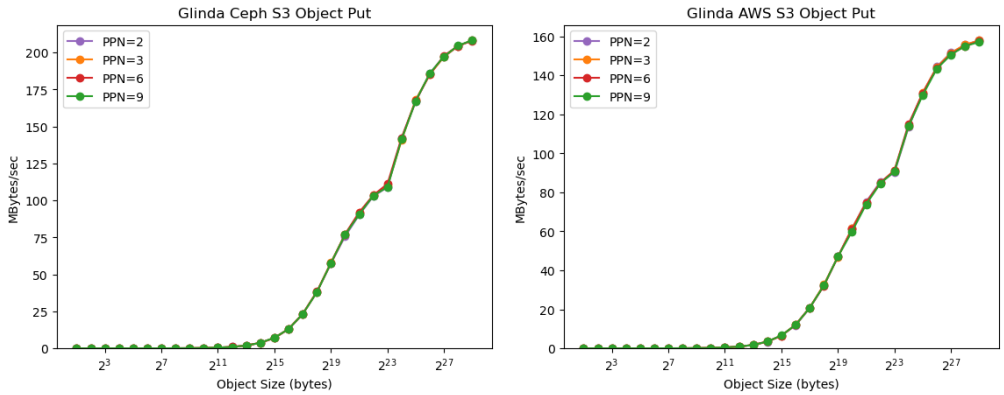

Figure 5-1. Glinda Object Put to Ceph vs AWS

5.3. Glinda Ceph S3 Get Performance

Figure 5-2 illustrates read performance for the local S3 storage system. These plots demonstrate that it is possible to achieve at least a 5x speedup through the use of more performant hardware. However, these performance levels are still an order of magnitude less than what the underlying storage system is capable of delivering. We expect performance is being lost in several places, including Ceph's S3 implementation, the use of HTTP over TCP, and Ceph's native performance limitations.

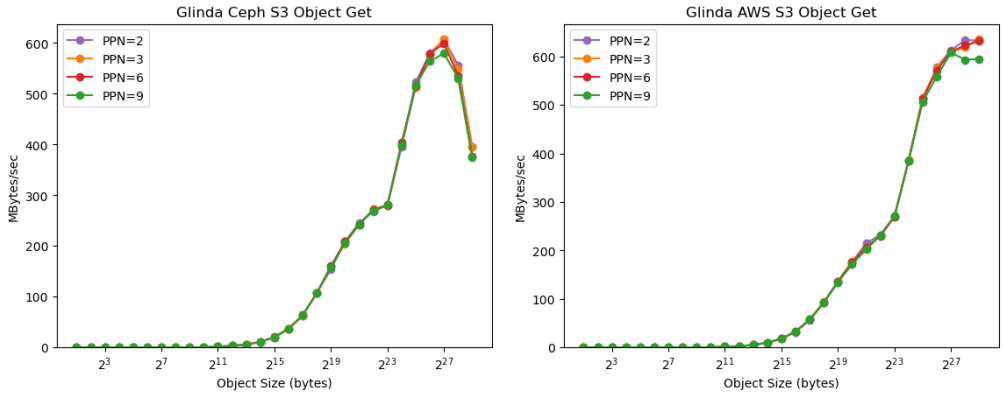

Figure 5-2. Glinda Object Get to Ceph vs AWS

6. CONCLUSION

In this year's work we have explored different options for writing data into S3 storage systems and measured the performance of interacting with enterprise S3 storage systems. The key points from this work are as follows:

S3 Storage Systems do not Replace HPC Storage: Unsurpisingly, S3 storage systems are significantly slower than HPC storage systems. While faster hardware can improve performance, fundamental design characteristics of S3 make it unlikely to be able to compete with HPC file systems. System planners should leverage S3 stores for the role they were designed for: low-cost, accessible capacity storage.

S3 Storage is Highly Accessible: We were able to build and run our storage benchmarks on a wide variety of platforms distributed across the enterprise. We performed this work entirely in user space and never had to request special access on a system to install software or unblock a network path to get to the stores.

Good Performance Over the Enterprise Network: While throughputs were low by HPC standards, Gbps speeds across the SRN network are very promising. From a desktop user's perspective, the new S3 storage systems are responsive and offer significant opportunities for collaboration.

6.1. Future Work

We see many opportunities for expanding the work performed this year.

6.1.1. S3 Client Improvements

While working with the AWS SDK, we discovered that the streaming interfaces were constructed in a way that causes developers to make extra, costly copies of data to send it. From experience, we understand the need for applications to reserve as much memory as possible for computation. Using a *std::stringstream* would duplicate data buffers that would lead to memory "theft". One alternative to a *std::stringstream* is a *std::iostream* class that can stream data in-place to/from the application's own buffers. This would require the application to freeze the data buffer until it has been transferred, but that would also be the case for file writes.

6.1.2. Resolving Network Unknowns

The network infrastructure that connect our HPC systems, storage systems, and desktops is opaque to normal users and certainly has bottlenecks that impede storage performace. Future work involving the S3 stores should focus on working with Sandia's infrastructure teams to ensure that sufficient pathways between key resources are established.

6.2. Adapting IOSS to have Optimal S3 Transfers

We take the position that Key-Value stores like S3, along with well-crafted metadata objects (also stored in S3 as a K-V pair) can manage ModSim mesh data at a more granular level than partitions created for processing by compute nodes. This was demonstrated in [6] which extended IOSS to have a "backend" that can marshal ModSim data to and from FADOEL [5], a distributed Key-Value store developed at Sandia. That work used SPARC as the ModSim example and Catalyst [1] as the downstream consumer of the ModSim data. FAODEL was the data mediator and served the same role that S3 will in work in FY23. Since IOSS provides a common interface to Exodus and CGNS (as well as other mesh file formats) [6] demonstrated this workflow is viable for any mesh format enabled in IOSS. In FY23 we seek to extend this work such that IOSS has a new S3 "backend" in order to investigate the S3 cloud a viable storage option to manage ModSim data.

While FAODEL was the enabling technology in [6], the strategy to create Keys and Values compatible with a K-V store was leveraging the data hierarchy defined in the IOSS mesh interface. Stated simply, to IOSS a ModSim mesh file is a *Region* that maintains a number of *Blocks* and *Sets* each of which maintain a number of *Properties* (meta data, physical properties, etc.) and *Fields* (bulk data, often time-varying). Traversing the IOSS mesh hierarchy (via the IOSS mesh API) from the top-level *Region* to a *Block* or *Set* then to a *Field* or *Property* and aggregating information from each IOSS C++ object instance encountered, a unique string was created for each *Property* and *Field*. Note that time-varying *Fields* also contain time step information in their keys. This same strategy can be employed to develop keys and identify data values for use in decomposing ModSim data to S3.

Taking a post-processing (of computed ModeSim datasets) posture, this finer-grained Key-Value ModSim decomposition has a few advantages overs the traditional approach. First, it enables users to access ModSim data from any language that can interface with the Key-Value store. Python in particular has many resources for AI and ML researchers. Second, downstream consumers of ModSim data may not require the entirety of a ModSim dataset. However, traditional ModSim mesh APIs require the entire dataset be present even if only accessing a small portion of that dataset, inducing local staging of unused data.

In addition to adapting the work in [6], work in FY23 will also focus on developing metadata objects for each dataset stored in S3. These metadata objects will let users identify the K-V pairs needed for a given task, so they can be staged from ModSim dataset decomposed to S3 as described above.

APPENDIX A. Building the AWS SDK and the S3 Benchmarks

The examples shown here use modules specific to the CTS-1 platform, but similar modules are available on most plaforms and will work similarly.

The default modules on the CTS-1 platform are not sufficient to build the AWS SDK or the S3 benchmarks.

```
module purge
module load cmake/3.20.3
module load gnu/10.2.1
module load openmpi-gnu/4.1
module load cde/v2/ninja/1.10.1
```

The AWS SDK builds against the Curl libraries and requires the libcurl headers in addition to the libraries. The libcurl development package that includes the header is not typically installed on Sandia's HPC platforms. Building and installing Curl from source is trivial. After installing curl, be sure to add the bin directory to $PATH, so that later stages of the build will find curl-config.

```
wget --no-check-certificate \
  https://github.com/curl/curl/releases/download/curl-7_80_0/curl-7.80.0.tar.gz
tar xf curl-7.80.0.tar.gz
cd curl-7.80.0
mkdir build
cd build
../configure --prefix=$(pwd)/../install --with-openssl
make install
CURL_CONFIG_PATH=$(pwd)/../install/bin
cd ../..
PATH=${CURL_CONFIG_PATH}:$PATH
```

With all the prerequisites loaded, the AWS SDK can be built and installed. Building the tip of the develop branch can be risky. Instead create a branch at a known good tag, checkout that branch and build it. In this case, tag 1.9.234 is known to be good so it will be used in these examples.

```
git clone --recurse-submodules https://github.com/aws/aws-sdk-cpp
cd aws-sdk-cpp
git branch branch/tag-1.9.234 1.9.234
git checkout branch/tag-1.9.234
git submodule update --recursive
cd ..

mkdir -p aws-sdk-cpp/build
cd aws-sdk-cpp/build
CURL_PREFIX=$(curl-config --prefix)
if [ $CURL_PREFIX != "/usr" ] ; then
```

```
12    CURL_LIBS=$(curl-config --libs)
13    echo $CURL_LIBS
14    CURL_LIB_PATH=$(echo $CURL_LIBS|sed -e 's/-L\(.*\) .*/\1/')
15    echo $CURL_LIB_PATH
16    PKG_CONFIG_PATH="$CURL_LIB_PATH/pkgconfig:$PKGCONFIG_PATH"
17  fi
18  # check if pkg-config can find libcurl
19  pkg-config --libs libcurl
20  result=$?
21  if [ $result -ne 0 ] ; then
22    echo "ERROR: pkg-config couldn't find libcurl."
23    echo "Configuring AWS S3 SDK will probably fail."
24  fi
25  AWS_S3_PATH=$(pwd)/../install
26  cmake -DAUTORUN_UNIT_TESTS:BOOL=OFF \
27    -GNinja -DCMAKE_INSTALL_PREFIX=${AWS_S3_PATH} ..
28  ninja install
29  cd ../..
```

AWS also provides a command line tool as a binary distribution that works on most Linux platforms. The installation script is named `install` which conflicts with name of the installation directory. The script should be renamed to `install.sh`.

```
1  curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" \
2    -o "awscliv2.zip"
3  unzip awscliv2.zip
4  mv aws awscliv2
5  cd awscliv2
6  if [ ! -e install.sh ] ; then
7    mv install install.sh
8    mkdir install
9  fi
10 ./install.sh -b `pwd`/install/bin -i `pwd`/install/aws
11 cd ..
```

References

[1] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. Paraview catalyst: Enabling in situ data analysis and visualization. *ISAV2015: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, 2015, 11 2015.

[2] Kyle Banker, Douglas Garrett, Peter Bakkum, and Shaun Verch. *MongoDB in action: covers MongoDB version 3.0*. Simon and Schuster, 2016.

[3] Clinton Gormley and Zachary Tong. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. " O'Reilly Media, Inc.", 2015.

[4] Trey Grainger and Timothy Potter. *Solr in action*. Manning Publications Co., 2014.

[5] Sandia National Laboratories. FAODEL: Flexible, Asynchronous, Object Data-Exchange Libraries. `https://github.com/faodel/faodel/`. Latest Release: v1.2108.1.

[6] Gary Templet, Matthew Glickman, Todd Kordenbrock, Scott Levy, Jay Lofstead, Jeff Mauldin, Thomas Otahal, Craig Ulmer, Patrick Widener, and Ron Oldfield. Fy20 csse l2 milestone 7186; final update. Technical Report SAND2020-9416 PE, Sandia National Laboratories, 2020.

[7] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D.E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320, 2006.

[8] Sage A. Weil, Andrew W. Leung, Scott A. Brandt, and Carlos Maltzahn. Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07*, pages 35–44, 2007.

DISTRIBUTION

Email—External (encrypt for OUO)

Name	Company Email Address	Company Name
Patrick Widener	widenerpm@ornl.gov	Oak Ridge National Laboratory

Email—Internal (encrypt for OUO)

Name	Org.	Sandia Email Address
Aaron Joseph Moreno	9325	ajmoren@sandia.gov
Aaron L. Brundage	1554	albrund@sandia.gov
Amanda Dodd	8750	ajbarra@sandia.gov
Andrew Younge	1423	ajyoung@sandia.gov
Brenna M. Hautzenroeder	2494	bmhautz@sandia.gov
Camron Proctor	8751	cproct@sandia.gov
Cheryl Lam	8755	clam@sandia.gov
Cory D Lueninghoener	9328	cdlueni@sandia.gov
Craig Ulmer	8753	cdulmer@sandia.gov
Deborah Serna	1355	daserna@sandia.gov
Edward Hoffman	8753	elhoffm@sandia.gov
Elliott Marshall Ridgway	1424	emridgw@sandia.gov
Eric Victor Ho	1424	evho@sandia.gov
Ernest Friedman-hill	8753	ejfried@sandia.gov
Evan Harvey	1424	eharvey@sandia.gov
Greg Sjaardema	1543	gdsjaar@sandia.gov
Jaideep Ray	8739	jairay@sandia.gov
Jay Dike	8752	jjdike@sandia.gov
Jay Lofstead	1424	gflofst@sandia.gov
Kevin Olson	8753	kholson@sandia.gov
Kevin Pedretti	1423	ktpedre@sandia.gov
Marcus Jonathan Gibson	8753	mjgibso@sandia.gov

Name	Org.	Sandia Email Address
Matthew Glickman	1462	mrglick@sandia.gov
Matthew Leon Curry	1423	mlcurry@sandia.gov
Mike Glass	1545	mwglass@sandia.gov
Reed Milewicz	1424	rmilewi@sandia.gov
Ron Oldfield	1441	raoldfi@sandia.gov
Ronald Brightwell	1423	rbbrigh@sandia.gov
Ryan Alberdi	1543	ralberd@sandia.gov
Samuel Andrew Grayson	1424	sagrays@sandia.gov
Sean Hardesty	1543	hardes@sandia.gov
Shyamali Mukherjee	8753	smukher@sandia.gov
Sylvain Robert Bernard	1424	srberna@sandia.gov
Todd Kordenbrock	1441	thkorde@sandia.gov
Tricia Gharagozloo	8753	peghara@sandia.gov
Wesley Poblete Coomber	1424	wpcoomb@sandia.gov
CA Technical Library	8551	cateclib@sandia.gov

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.