applied sciences

Article

Analyzing the Performance of the S3 Object Storage API for HPC Workloads

Frank Gadban [1,*] and Julian Kunkel [2,*]

1 MIN Faculty, University of Hamburg , 20146 Hamburg, Germany
2 Institute of Computer Science, Faculty of Mathematics and Computer Science, Georg-August-Universität Göttingen/GWDG, 37018 Göttingen, Germany
* Correspondence: frank.gadban@studium.uni-hamburg.de (F.G.); julian.kunkel@gwdg.de (J.K.)

Abstract: The line between HPC and Cloud is getting blurry: Performance is still the main driver in HPC, while cloud storage systems are assumed to offer low latency, high throughput, high availability, and scalability. The Simple Storage Service S3 has emerged as the de facto storage API for object storage in the Cloud. This paper seeks to check if the S3 API is already a viable alternative for HPC access patterns in terms of performance or if further performance advancements are necessary. For this purpose: (**a**) We extend two common HPC I/O benchmarks—the IO500 and MD-Workbench—to quantify the performance of the S3 API. We perform the analysis on the Mistral supercomputer by launching the enhanced benchmarks against different S3 implementations: on-premises (Swift, MinIO) and in the Cloud (Google, IBM…). We find that these implementations do not yet meet the demanding performance and scalability expectations of HPC workloads. (**b**) We aim to identify the cause for the performance loss by systematically replacing parts of a popular S3 client library with lightweight replacements of lower stack components. The created S3Embedded library is highly scalable and leverages the shared cluster file systems of HPC infrastructure to accommodate arbitrary S3 client applications. Another introduced library, S3remote, uses TCP/IP for communication instead of HTTP; it provides a single local S3 gateway on each node. By broadening the scope of the IO500, this research enables the community to track the performance growth of S3 and encourage sharing best practices for performance optimization. The analysis also proves that there can be a performance convergence—at the storage level—between Cloud and HPC over time by using a high-performance S3 library like S3Embedded.

Keywords: HPC; cloud; convergence; storage; AWS S3; S3Embedded

check for updates

Citation: Gadban, F.; Kunkel, J. Analyzing the Performance of the S3 Object Storage API for HPC Workloads. *Appl. Sci.* **2021**, *11*, 8540. https://doi.org/10.3390/app11188540

Academic Editors: Antonio J. Pena and Pedro Valero-Lara

Received: 19 July 2021
Accepted: 9 September 2021
Published: 14 September 2021

1. Introduction

With the increased prevalence of cloud computing and the increased use of the Infrastructure as a Service (IaaS), various APIs are provided to access storage. The Amazon Simple Storage Service (S3) API [1] is the most widely adopted object storage in the cloud. Many Cloud storage providers, like IBM, Google, and Wasabi, offer S3 compatible storage, and a large number of Scale-Out-File Systems like Ceph [2], OpenStack Swift [3] and Minio [4] offer a REST gateway, largely compatible with the S3 interface. HPC applications often use a higher-level I/O library such as NetCDF [5] or ADIOS [6] or still the low-level POSIX API. Under the hood, for the interaction with the storage system, MPI-IO and POSIX are still widely used, while object storage APIs such as DAOS [7] are emerging. If the performance characteristics of S3 are promising, it could be used as an alternative backend for HPC applications. This interoperability would foster convergence between HPC and Cloud [8,9] and eventually lead to consistent data access and exchange between HPC applications across data centers and the cloud.

In this research, a methodology and a set of tools to analyze the performance of the S3 API are provided. The contributions of this article are: (**a**) The modification of existing HPC benchmarks to quantify the performance of the S3 API by displaying relevant performance characteristics and providing a deep analysis of the latency of individual operations; (**b**) the creation of a high-performance I/O library called S3Embedded [10], which can be used as a drop-in replacement of the commonly used libs3 [11], and which is also compatible and competitive with the HPC I/O protocols, and optimized for use in distributed environments.

The structure of this paper is as follows: Section 1.1 presents related work. Section 2 describes the test scenarios and defines the relevant metrics that will be addressed using our benchmarks. We extend the scope of two existing HPC MPI-parallel benchmarks, namely the IO500 [12] and MDWorkbench [13], to assess the performance of the S3 API. Section 3 describes the experimental procedure, the used systems, and the methodology of the evaluation conducted in this work. The tests are performed on the Mistral [14] Supercomputer. Section 3.2.3 analyzes the obtained latency results. Section 4 introduces the S3Embeded library and its possible use. The last section summarizes our findings.

1.1. Background and Related Work

After the release of the Amazon S3 service in 2006, many works were published to assess the performance of this offering. Some of them [15,16] focused only on the download performance of Amazon S3, most of them [15–19] never published or described the used benchmarks, others [15,16,20] are not able to assess S3 compatible storage. The Perfkit benchmarker from Google was used in [21] to compare the download performance of AWS S3 in comparison with Google Cloud Storage (GCS) and Microsoft Azure Storage. However, since the tests were accomplished in the cloud, the obtained results depend heavily on the VM machine type in use, hence, on the network limitation enforced by the cloud provider; adding to the confusion, and in contrast, [19] found in their tests, which were also run in the AWS cloud on different EC2 machines against the Amazon S3 implementation, that "there is not much difference between the maximum read/write throughput across instances".

In [22], the performance of the S3 interface offered by some cloud providers is evaluated. However, the test scenario covered only the upload and download of a variable number of files. Access patterns of typical HPC applications, like the performance of the metadata handling, are not covered.

Gadban et al. [23] investigated the overhead of the REST protocol when using Cloud services for HPC Storage and found that REST can be a viable, performant, and resource-efficient solution for accessing large files, but for small files, a lack of performance was noticed. However, the authors did not investigate the performance of a cloud storage system like S3 for HPC workloads.

The benchmarks we found in the literature for the analysis of S3 performance do not provide a detailed latency profile, nor do they support parallel operations across nodes, which is a key characteristic of HPC applications. As such, the lack of published tools that cover HPC workloads pushed us to enhance two benchmarks already used for HPC procurement, namely IOR and MD-Workbench, by developing a module capable of assessing the performance of S3 compatible storage. The IO500 benchmark (https://io500.org (accessed on 23 April 2021) simulates a variety of typical HPC workloads, including the bulk creation of output files from a parallel application, intensive I/O operations on a single file, and the post-processing of a subset of files. The IO500 uses the IOR and MDTest benchmarks under the hood, which comes with a legacy backend for S3 using the outdated aws4c [24] library that stores all data in a single bucket; as such, a file is one object that is assembled during write in the parallel job using multipart messages. However, since most recent S3 implementations do not support this procedure, the use of a most recent library is required. We also explore the performance of interactive operations on files using the MD-Workbench [13].

2. Materials and Methods

We aim to analyze the performance of the S3 interface of different vendors in an HPC environment to assess the performance potential of the S3 API. To achieve this, a five step procedure is implemented by (1) identifying suitable benchmarks; (2) modifying the benchmark to support S3; (3) defining an HPC environment supporting the S3 API to run them; (4) determining a measurement protocol that allows us to identify the main factors influencing the performance of S3 for HPC workloads; and (5) providing alternative implementations for S3 to estimate the best performance.

2.1. Benchmarks

We extend two HPC benchmarks, the IO500 and MD-Workbench, to analyze the potential peak performance of the S3 API on top of the existing HPC storage infrastructure.

The IO500 [12] uses IOR and MDTest in "easy" and "hard" setups, and hence performs various workloads and delivers a single score for comparison, and the different access patterns are covered in different phases:

- IOEasy simulating applications with well-optimized I/O patterns.
- IOHard simulating applications that utilize segmented input to a shared file.
- MDEasy simulating metadata access patterns on small objects.
- MDHard accessing small files (3901 bytes) in a shared bucket.

We justify the suitability of these phases as follows: B. Welch and G. Noer [25] found that, inside HPC, between 25% and 90% of all files are 64 Kbytes or less in size, as such a typical study of the performance of object storage inside HPC should also address this range, rather than only focusing on large sizes, which are expected to deliver better performance and only be limited by the network bandwidth [21,23]. This is why, using the IOR benchmark, we highlight this range when exploring the performance for files of size up to 128 MB in Sections 3.2 and 4. Large file sizes are also addressed since the performed IO500 benchmarks operate on 2 MiB accesses creating large aggregated file sizes during a 300 s run.

MD-Workbench [13] simulates concurrent access to typically small objects and reports throughput and latency statistics, including the timing of individual I/O operations. The benchmark executes three phases: pre-creation, benchmark, and cleanup. The pre-creation phase setups the working set, while the cleanup phase removes it. A pre-created environment that is not cleaned can be reused for subsequent benchmarks to speed up regression testing, i.e., constant monitoring of performance on a production system. During the benchmark run, the working set is kept constant: in each iteration, a process produces one new object and then consumes a previously created object in FIFO order.

2.2. Modifications of Benchmarks

For IO500, an optimistic S3 interface backend using the libS3 client library is implemented for IOR in the sense that it stores each fragment as one independent object, and as such, it is expected to generate the best performance for many workloads.

For identifying bottlenecks, it supports two modes:

- Single bucket mode: created files and directories result in one empty dummy object (indicating that a file exists), every read/write access happens with exactly one object (file name contains the object name + size/offset tuple); deletion traverses the prefix and removes all the objects with the same prefix recursively.
- One bucket per file mode: for each file, a bucket is created. Every read/write access happens with exactly one object (object name contains the filename + size/offset tuple); deletion removes the bucket with all contained objects.

As such, the libs3 implementation gives us the flexibility to test some optimistic performance numbers. The S3 interface does not support the "find" phase of IO500, which we, therefore, exclude from the results.

MD-Workbench recognizes datasets and objects and also offers two modes:

- One bucket, the D datasets are prefixed by the process rank.
- One bucket per dataset.

In both modes, objects are atomically accessed fitting directly the S3 API.

The libS3 used in IO500 is the latest one which only supports AWS signatures v4 [26] while the current release of MD-Workbench supports an older version of libs3, which uses the AWS signature v2. As such, it is ideal for the benchmarking of some S3 compatibles systems that only support the v2 signature, like the one found at DKRZ (The German Climate Computing Center).

2.3. Utilizing S3 APIs in Data Centers

Data centers traditionally utilize parallel file systems such as Lustre [27] and GPFS. These storage servers do not natively support the S3 API. In a first iteration, we explore MinIO to provide S3 compatibility on top of the existing infrastructure. MinIO offers various modes, one of which is a gateway mode providing a natural deployment mode: in this mode, any a S3 request is converted to POSIX requests for the shared file system. We show the effect of the object size for the different MinIO modes, and we compare the obtained results to the native REST protocol and Lustre. To achieve the convergence between HPC and the Cloud, it needs to be possible to move workloads seamlessly between both worlds. We aim to allow the huge number of S3 compatible applications, the possibility to benefit from HPC performance while using the same compatibility offered by the S3 interface. Therefore, we create an I/O library called S3Embedded based on libs3, where parts of the S3 stack were replaced or removed to optimize the performance, this library also performs the translation between S3 and POSIX inside the application address space. Our aim is to make it compatible with standard services, competitive with the HPC I/O protocols, and optimized for use in distributed environments. Additionally, the extended library S3remote is introduced, it is intended to provide a local gateway on each node—an independent process similar to MinIO—but uses a binary protocol over TCP/IP for local communication instead of HTTP as HTTP might be the cause of the observed performance issues. In Section 4, we assess the performance of S3Embedded/S3Remote inside HPC.

2.4. Measurement Protocol

We measure the performance on a single node and then on multiple nodes while varying the size of the object and the number of processes/threads per node. To assess the performance of the different modes, we establish performance baselines by measuring performance for the network, REST, and the Lustre file system. Then, the throughput is computed (in terms of MiB/s and Operations/s) and compared to the available network bandwidth of the nodes.

2.5. Test System

The tests are performed on the Supercomputer Mistral [14], the HPC system for earth-system research provided at the German Climate Computing Center (DKRZ). It provides 3000 compute nodes each equipped with an FDR Infiniband interconnect and a Lustre storage system with 54 PByte capacity distributed across two file systems. The system provides two 10 GBit/s Internet uplinks to the German research network (DFN) that is accessible on a subset of nodes.

2.6. MinIO Benchmarks in HPC

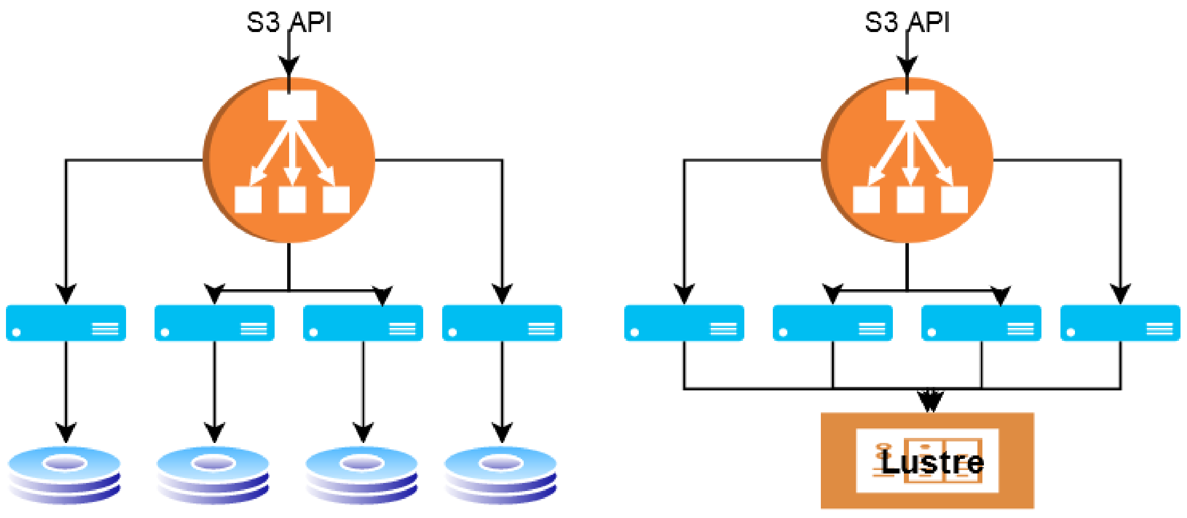
To create a reference number for the performance of S3 and explore the possible ways to optimize performance, we first use the MinIO server (release: 2020-08-18T19-41) to accomplish our tests using the modified benchmarks inside our HPC environment.

MinIO Deployment

MinIO supports the following modes of operation:

- Standalone (*sa*): runs one MinIO server on one node with a single storage device. We test configurations from tmpfs (in-memory fs/shm) and the local ext4 file system.
- Distributed servers (*srv*): runs on multiple nodes, object data and parity are striped across all disks in all nodes. The data are protected using object-level erasure coding and bitrot. Objects are accessible from any MinIO server node. In our setup, each server uses the local ext4 file system. Figure 1a illustrates the deployment.
- Gateway (*gw*): adds S3 compatibility to an existing shared storage. On Mistral, we use the Lustre distributed file system as the backend file system as seen in Figure 1b.

Alongside these three modes, we introduce two modes by inserting the Nginx [28] (v1.18.0) load balancer in front of the distributed and gateway configurations, and we refer to these setups as *srv-lb* and *gw-lb*, respectively. Both variants can utilize a cache on the Nginx load balancer (*-cache*).



(**a**) Distributed Servers mode. (**b**) Gateway mode.

Figure 1. Different MinIO Modes.

3. Results

3.1. MinIO Benchmarks in HPC Environment

3.1.1. Single Client

The first tests are performed using IOR [29] directly. Figures 2 and 3 show the performance on 1 node for a variable object size for the different MinIO modes. We notice that the standalone mode shows the best performance. Gateway mode is effective for reads, but slow for writes. Write performance is 1/3rd of the read performance. Compared to the Infiniband network throughput (about 6 GiB/s), only 7.5% and 2.5% of this performance can be obtained for read and write, respectively. Adding a load balancer has minimal influences on throughput in this setting, except when activating the caching mechanism, which has a tremendous impact on the read throughput.

3.1.2. Parallel IO

We investigate how much parallel IO is able to exploit the available network bandwidth by varying the number of clients accessing the object store. Assuming an ideal scenario, we start MinIO in standalone mode with RAM as backend (sa-shm) on one node. Figures 4 and 5 show the aggregated throughput across another four client nodes, demonstrating the scale-out throughput achieved across many tasks running on various nodes.

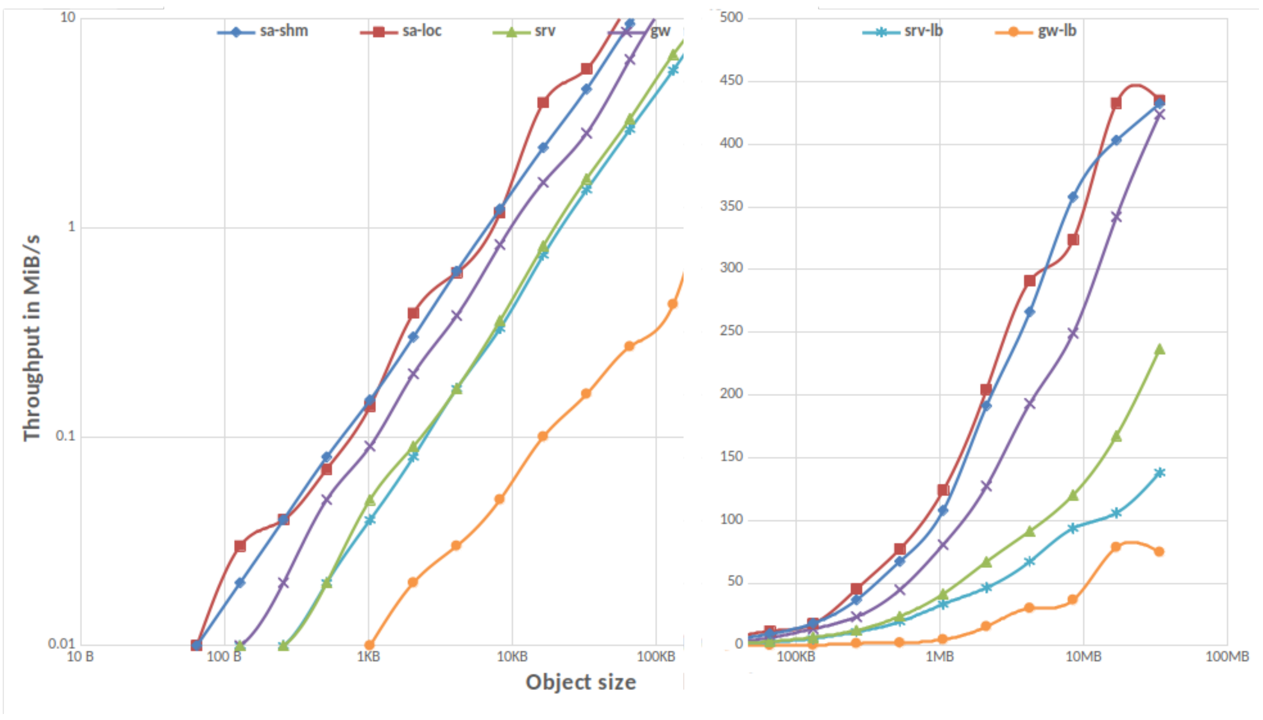


Figure 2. Read throughput for MinIO modes for 1 node and 1 PPN.

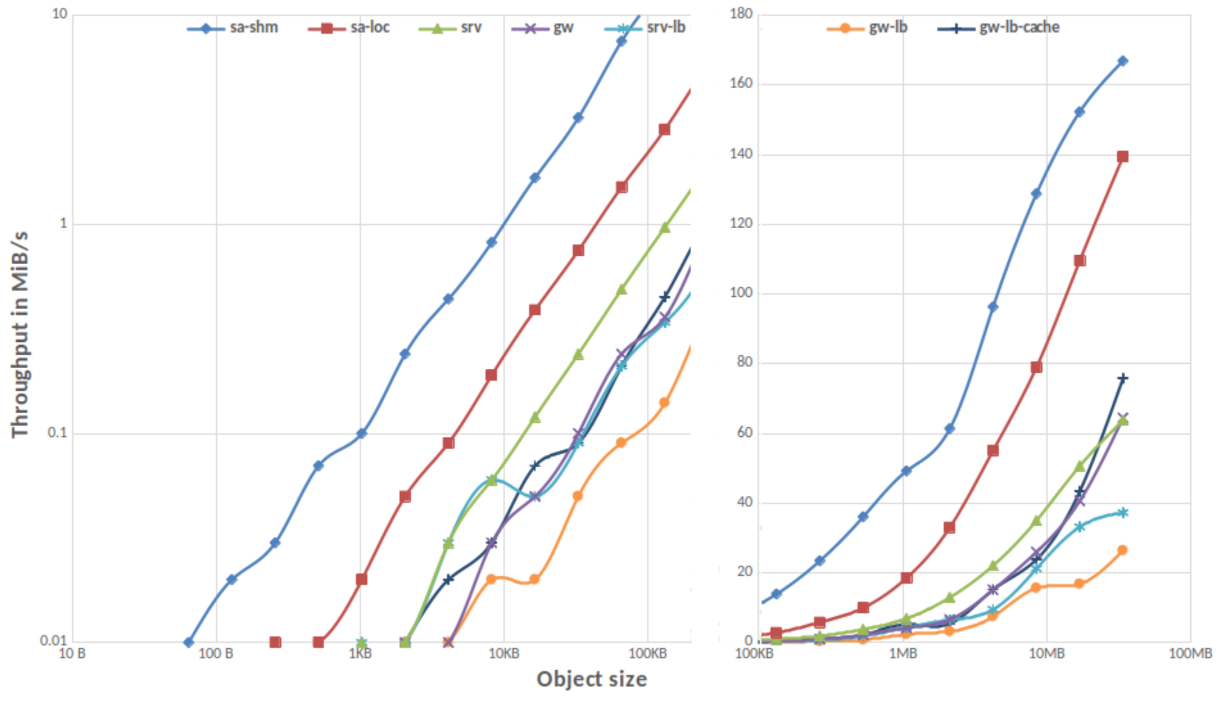
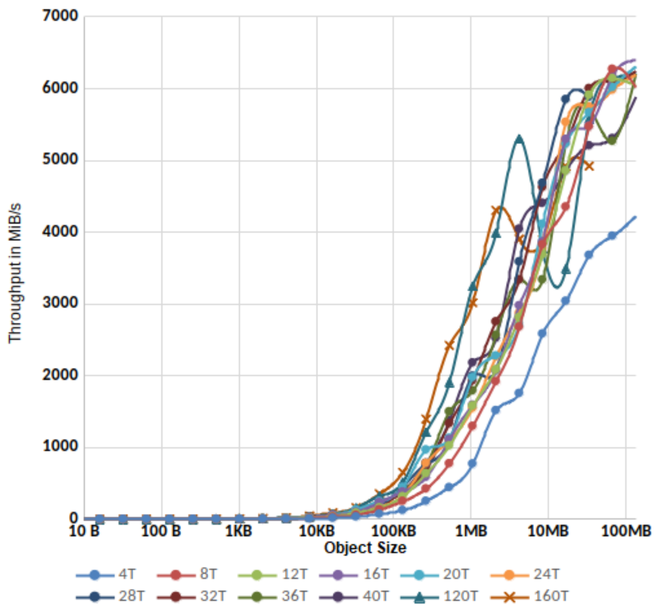
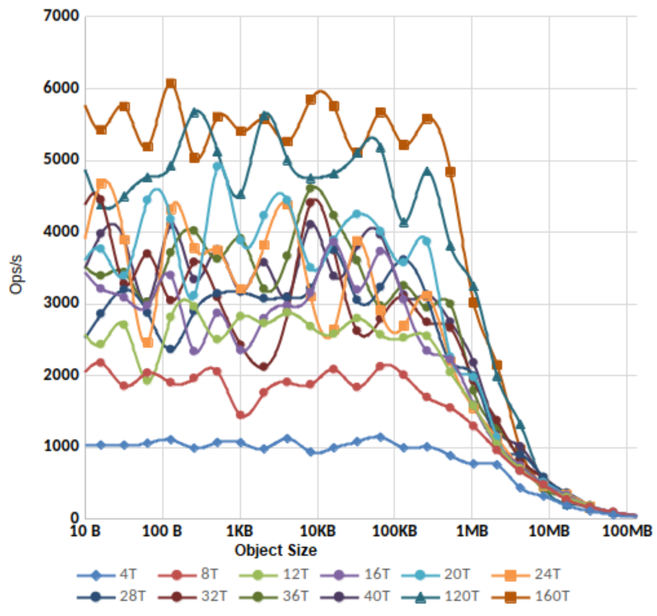


Figure 3. Write throughput for MinIO modes for 1 node and 1 PPN.

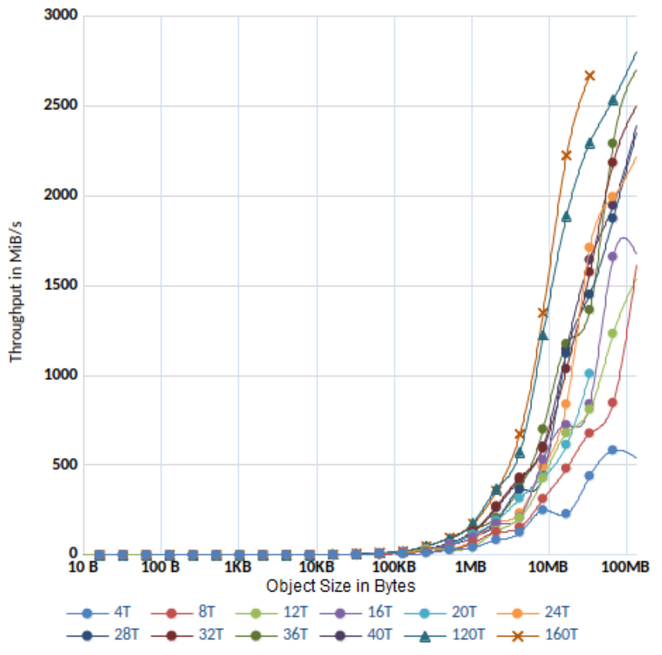


(**a**) Read throughput.

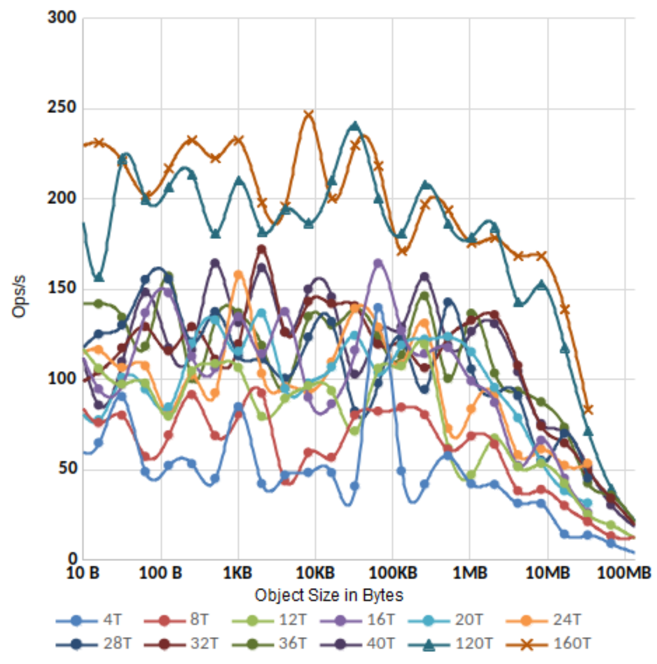


(**b**) Read operations/s.

Figure 4. Aggregated read throughput for N tasks on 4 nodes.



(**a**) Write throughput.



(**b**) Write operations/s.

Figure 5. Aggregated write throughput for N tasks on 4 nodes.

We notice that when increasing the number of tasks per node, we achieve the best throughput (about 6000 MiB/s). The performance per client node is 1.5 GiB/s. For the single server, it is close to the available network bandwidth. The I/O path is limited by latency, while with 4 threads (PPN = 1), about 1000 Ops/s are achieved, with 160 threads about 6000 Ops/s can be achieved. Write achieves about 2500 MiB/s (40% efficiency) and 250 Ops/s, indicating some limitations in the overall I/O path. This also fosters splitting data larger than 1 MiB into multiple objects and using multipart [30] upload/download.

3.1.3. MinIO Overhead in Gateway Mode

A more realistic scenario inside HPC is MinIO running in Gateway mode in front of Lustre. First, we launch the benchmarks on 4 client nodes, and MinIO is started in gateway mode on another set of nodes. We call this setup the disjoint mode.

However, this setup does not scale out efficiently, and this leads us to the introduction of another concept, which we call the local gateway (local-gw) mode, where MinIO is started on the N client nodes in gateway mode and uses the Lustre file system as the backend file system. We launch the benchmarks against the localhost on each node and notice that when increasing the number of tasks per node, we are achieving relatively better performance, compared to the disjoint mode, as shown in Table 1. However, we still achieve only 2% of Lustre performance for various benchmarks.

Table 1. Performance of MinIO Gateway on 4 nodes with 20 PPN.

Benchmark	Metric	Unit	Lustre	MinIO		Local-Gw
				Disjoint-Gw	Local-Gw	% of Lustre
md-workbench	rate	IOPS	18337	37	425	2.3%
	throughput	MiBps	34.100	0.100	0.800	2.3%
IO500	ior-easy-write	GiB/s	18.671	0.153	0.286	1.5%
	mdtest-easy-write	kIOPS	5.892	0.088	0.132	2.2%
	ior-hard-write	GiB/s	0.014	0.003	0.006	45.7%
	mdtest-hard-write	kIOPS	5.071	0.036	0.076	1.5%
	ior-easy-read	GiB/s	11.475	0.693	2.071	18.1%
	mdtest-easy-stat	kIOPS	24.954	1.198	4.092	16.4%
	ior-hard-read	GiB/s	0.452	0.029	0.094	20.7%
	mdtest-hard-stat	kIOPS	18.296	1.281	3.968	21.7%
	mdtest-easy-delete	kIOPS	9.316	0.025	0.023	0.3%
	mdtest-hard-read	kIOPS	6.950	0.449	1.636	23.5%
	mdtest-hard-delete	kIOPS	4.863	0.029	0.025	0.5%

Nevertheless, we notice that increasing the number of clients and the tasks per client leads to an increase in the number of "Operation timed out" errors. An issue that we address in Section 4.

3.1.4. MinIO vs. REST vs. TCP/IP

To understand performance limitations, we plot the throughput of various transfer modes using one node and four processes in Figure 6. The figure includes performance numbers from [23], where the base performance of REST/HTTP is analyzed by emulating a best-case client/server scenario: The throughput of the HTTP GET operation—representing read-only access to a storage server—is calculated for different file sizes. We conduct the same experiments on Mistral, using tmpfs as backend. Since the tests are accomplished against localhost, we can compare the obtained results with the IOR results of the single client MinIO setup described in Section 3.1.1, where MinIO is running in standalone mode with RAM as backend, and using 1Node-4PPN as well. We also include:

- The IOR results of the direct lustre access using 1N-4PPN.
- The TCP/IP throughput measured using `iperf`.
- For reference, the S3Embedded results for a similar setup (1N-4PPN), which we will describe in more detail later in Section 4.

The results show that for objects with size greater than 1 KB, MinIO performance—even in this ideal setup—is significantly lower than the base REST performance and that there is room for improvement, which we address in Section 4.

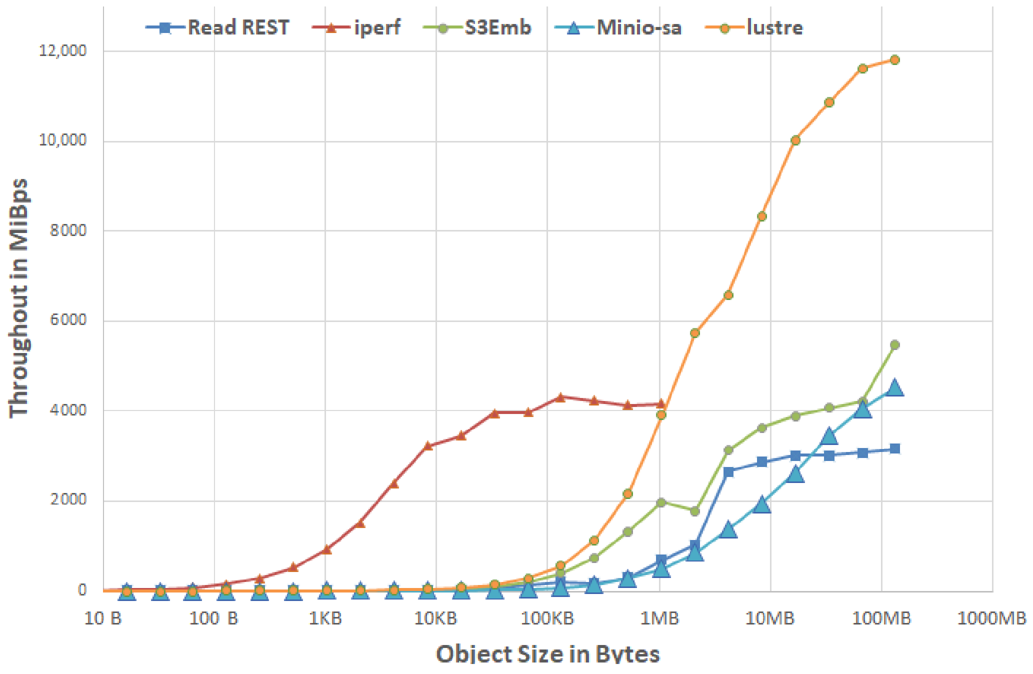


Figure 6. Read throughput MinIO vs. REST using 1N-4PPN.

3.2. *Tests against S3 Compatible Systems*

The next benchmarks are performed on Mistral using IO500 and MD-Workbench.

3.2.1. In-House Tests

Tests are conducted against the OpenStack Swift [3] system already available in DKRZ, Swift version 2.19.2 is used, and the S3 interface is implemented using Swift3 version 1.12.1, which is now merged into swift middleware as the s3api; this is why only AWS signature v2 is available, and as such, the tests were only conducted using MD-Workbench. On four client nodes and with 20 PPN, a rate of 269.5 IOPs and a 0.5 MiB/s throughput are observed during the benchmark phase with MD-Workbench. Table 1 shows the IO500 results for the different systems tested inside DKRZ.

3.2.2. Comparison with Scality Ring

Next, we compare the performance of the MinIO obtained results in Section 3.1.2 to the results of another S3 compatible storage called Scality Ring published in [31]. Scality Ring is a cloud-scale, distributed software storage solution that includes a comprehensive AWS S3 REST API implementation. We are aware that this is not a fair comparison, but it gives us a qualitative comparison that validates that our results are reasonable. In the setup described in [31], RING is deployed on Cisco Networking equipment much similar to the HPC network environment provided by Mistral. For the sake of simplicity, the server CPU capabilities are considered equivalent (Intel Xeon Silver 4110 vs. Intel Xeon E5-2680 v3). The published cosbench [32] results in [31], from the benchmarks launched on 3 nodes with 300 threads are compared to the MinIO Parallel IO results described in Section 3.1, which are also started on 3 nodes with a total of 144 threads (no Hyperthreading). The comparison is shown in Figure 7.

We can see that although the write throughput is relatively similar for files smaller than 32 MB, the read throughput of MinIO is better. Scality has an advantage for writes below 16 MiB because about twice the number of threads is used. Based on these measurements, we presume that Scality does not yield a significant performance benefit over MinIO.

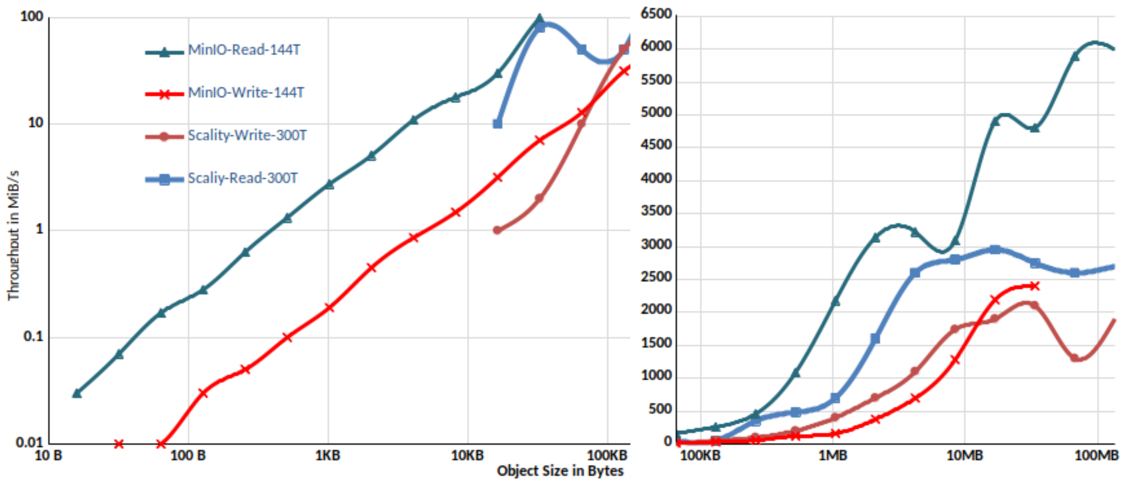
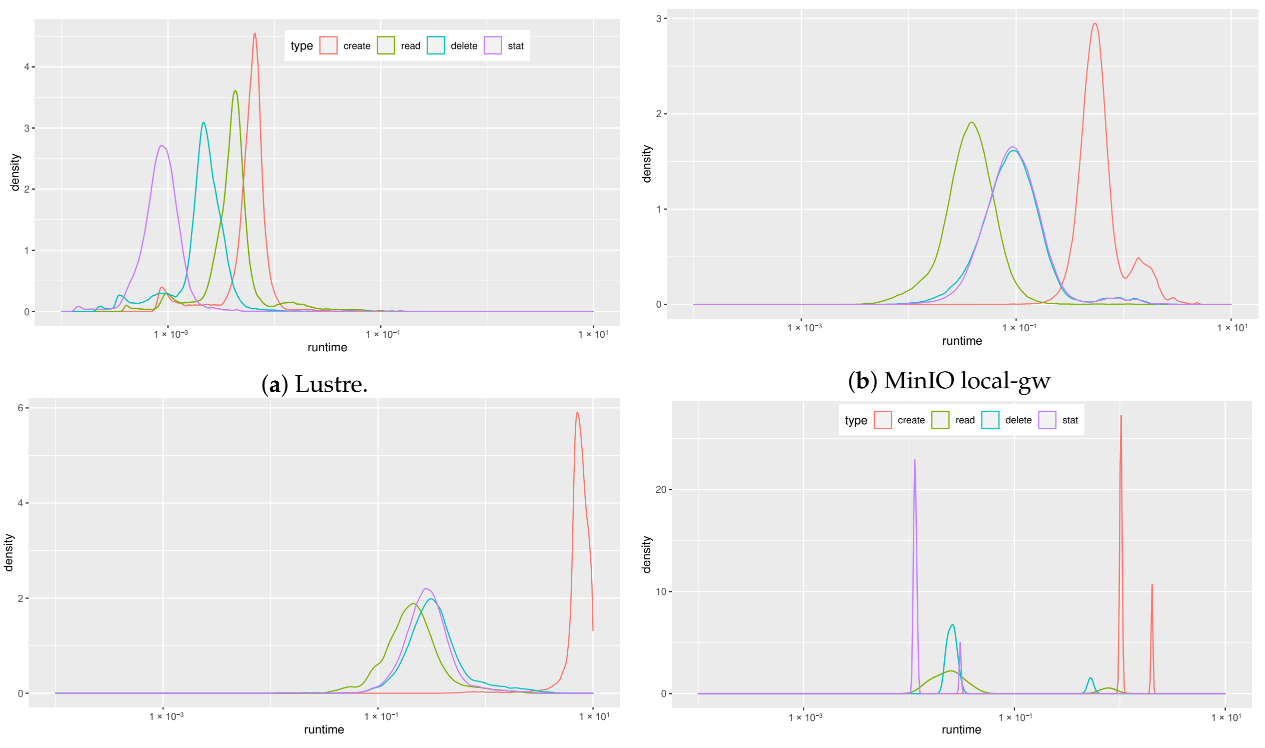


Figure 7. Throughput of Scality Ring and MinIO on 3 nodes.

3.2.3. Latency Analysis

MD-Workbench reports not only the throughput, but also the latency statistics for each I/O operation. The density of the individually timed operations is plotted as shown in Figure 8. A density graph can be considered a smoothed histogram where the x-axis shows the observed runtime and the y-axis represents the number of occurrences.



(**a**) Lustre.

(**b**) MinIO local-gw

(**c**) MinIO disjoint.

(**d**) SwiftS3.

Figure 8. Latency density for the different systems.

The Lustre density figure shows roughly a Gaussian distribution for individual operations, where create is the slowest operation.

Using MinIO, the creation phase takes substantially longer. The local-gw mode, as already seen from the IO500 results, yields the best performance among the tested S3 implementations; however, far from the Lustre performance, which has a latency of around 10 ms and is extremely slow compared to the network base latency of approximately

10 µs. The SwiftS3 system changes the overall behavior significantly, leading to less predictable times.

We conclude that the involved processes behind the S3 implementation are the main cause of latency.

3.2.4. Tests against Cloud Systems

Different S3 vendors were contacted, which either offered a testing account or explicitly allowed us to execute the IO500 benchmark against their endpoints. We follow the guidelines depicted by the performance design patterns for Amazon S3 [33], especially regarding the request parallelization and horizontal scaling to help distribute the load over multiple network paths. Since all the providers offer multi-region storage, we choose the closest storage location to Mistral (located in the EU) to ensure the lowest latency. Due to the limited number of nodes with Internet connectivity on Mistral, the benchmarks are launched on only two nodes with PPN = 1. The results are summarized in Table 2; we scaled down the units by 1000 to better visualize the differences. The results of MinIO launched in local-gw mode—with the same number of nodes and tasks per node—are displayed in the last column.

Table 2. IO500 results comparing S3 Cloud providers.

Benchmark/System	Unit	Wasabi	IBM	Google	MinIO-Local-Gw
Score Bandwidth	MiB/s	0.007	1.642	0.46	12.62
ior-easy-write	MiB/s	2.35	35.00	13.35	46.39
mdtest-easy-write	IOPS	13.04	81.72	21.79	27.96
ior-rnd-write	MiB/s	0.01	0.23	0.07	1.231
mdworkbench-bench	IOPS	5.75	47.23	12.83	15.25
ior-easy-read	MiB/s	1.20	45.37	7.81	73.86
mdtest-easy-stat	IOPS	20.92	145.09	51.10	260.97
ior-hard-read	MiB/s	0.05	5.59	1.38	6.01
mdtest-hard-stat	IOPS	20.74	149.64	49.48	297.62
mdtest-easy-delete	IOPS	10.35	35.02	9.37	81.06
mdtest-hard-read	IOPS	8.54	70.06	18.90	130.36
mdtest-hard-delete	IOPS	10.28	35.25	9.48	94.32

The IBM Cloud Storage provided the best performance in our tests; however, this is far from the performance expected in HPC. Although MinIO in gateway mode provides better performance, this is also below our HPC experience since the network latency, in this case, is minimal compared to the other scenarios. A better solution is needed to leverage existing file systems found either inside or outside the HPC environment, as can be seen in Section 4. Note that some of the mentioned providers might provide better performance when using their native interface instead of S3; however, this is outside the scope of this work. Furthermore, the network interconnection between DKRZ and the cloud provider bears additional challenges.

4. S3Embedded

Because of the scalability limitation of the introduced local-gw mode, we create an I/O library called S3Embedded based on libs3, where parts of the S3 stack are replaced or removed to optimize the performance. By easily linking the S3Embedded library at compile time or at runtime to a libS3 compatible client application, it is possible to use the capabilities of this library. Assuming the availability of a globally accessible shared file system, S3Embedded provides the following libraries:

- libS3e.so: This is an embedded library wrapper that converts libs3 calls to POSIX calls inside the application address space.
- libS3r.so: This library converts the libs3 calls via a binary conversion to TCP calls to a local libS3-gw application that then executes these POSIX calls, bypassing the HTTP protocol.

In Figure 9, we display the results of the **IOR** benchmark while using the libraries mentioned above, in comparison with direct Lustre access and MinIO operating in the local-gw mode already described in Section 3. Note that some values are missing in the MinIO-local-gw results, despite the benchmark being repeated several times. This is because this setup does not scale well with the number of clients, as noted in Section 3.

Using S3Embedded helped us to pinpoint a performance problem in the IOR S3 plugin: we noticed that the delete process in IO500 is time-consuming since when trying to delete a bucket, our developed IOR S3 plugin tries to list the content of the entire bucket—calling `S3_list_bucket()`—for each file to be deleted to clean the fragments; however, since, in case of S3Embeded, all files are actually placed in a single directory, this ought to be very time-consuming. One workaround is to use the option bucket-per-file that effectively creates a directory per file. However, since this workaround does not cover all test workloads in the IO500, we proceed and introduce an environment variable called "S3LIB_DELETE_HEURISTICS", specific to the IOR S3-plugin. It defines at which file size of the initial fragment the list_bucket operation is to be executed; otherwise, a simple S3_delete_object is performed. While this optimization is not suitable for a production environment, it allows us to determine best-case performance for using S3 with the IO500 benchmark.

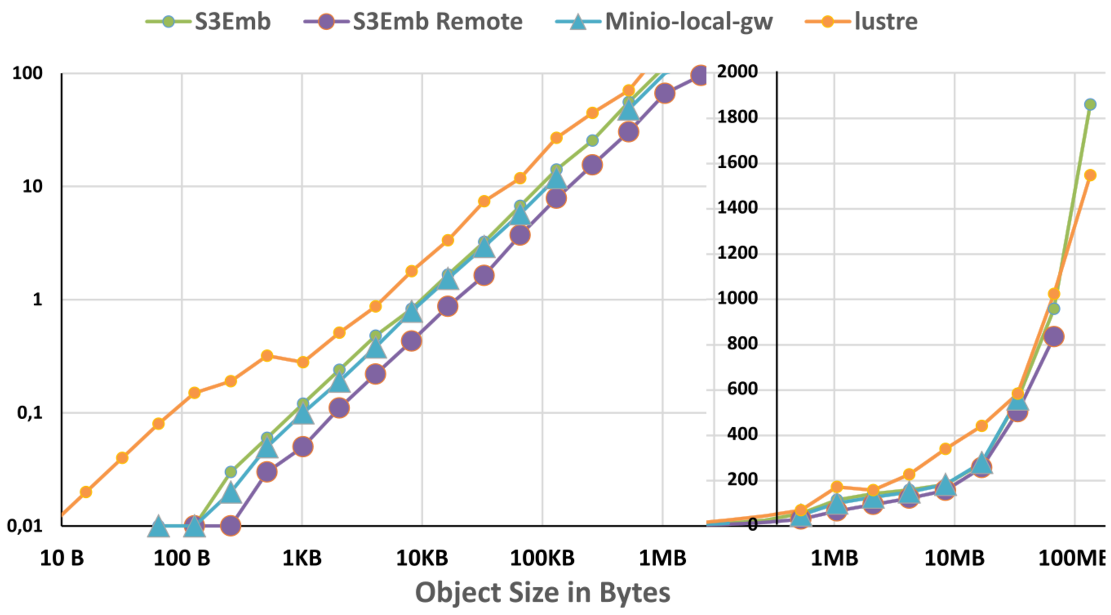


Figure 9. Read throughput of S3Embedded vs. Lustre vs. MinIO for 5N-20PPN.

The results delivered by S3Embedded are very close to the ones obtained for the Lustre direct access—mainly for files larger than 32 MB—far superior to the ones supplied by MinIO-local-gw, they are also free from timeout errors.

Benchmarking with **IO500** reflects the performance improvement delivered by S3Embedded/S3Remote, as shown in Table 3.

Table 3. IO500 results for S3Embedded and S3Remote compared to MinIO-local-gw and lustre using 2N-5PPN.

System	Unit	MinIO-Local-Gw	Lustre	S3Embedded	S3Remote
ior-easy-write	GiB/s	0.14	5.47	0.61	0.69
mdtest-easy-write	kIOPS	0.09	7.97	2.42	3.13
ior-easy-read	GiB/s	0.32	2.78	0.48	0.42
mdtest-easy-stat	kIOPS	0.85	13.82	8.02	6.94
ior-hard-read	GiB/s	0.019	0.139	0.046	0.042
mdtest-hard-stat	kIOPS	0.86	5.10	7.25	6.65

We notice that Lustre's performance with POSIX is often more than 10x faster than MinIO-local-gw, and that the error rate increases along with the number of Nodes/PPN. In contrast, the S3 API wrappers deliver much better performance, which is closer to Lustre native performance, and are more resilient to the number of clients, as shown in Table 4.

Even with 10 or 50 Nodes, as seen in Tables 5 and 6, the S3embedded library yields a performance closer to Lustre, but a performance gap remains.

Table 4. IO500 results for S3Embedded and S3Remote compared to MinIO-local-gw and lustre using 5N-20PPN.

Benchmark/System	Unit	MinIO-Local-Gw	Lustre	S3Embedded	S3Remote
ior-easy-write	GiB/s	0.75	23.49	3.40	1.99
mdtest-easy-write	kIOPS	0.39	17.11	7.52	1.19
ior-hard-write	GiB/s	0.01	0.04	0.30	0.05
mdtest-hard-write	kIOPS	0.10	7.25	3.41	0.55
ior-easy-read	GiB/s	2.46	15.87	2.40	1.36
mdtest-easy-stat	kIOPS	5.09	42.59	28.53	0.62
ior-hard-read	GiB/s	0.11	0.38	0.11	0.02
mdtest-hard-stat	kIOPS	4.37	31.49	26.66	0.60
mdtest-easy-delete	kIOPS	-	9.15	5.98	0.41
mdtest-hard-read	kIOPS	-	6.34	3.82	0.29
mdtest-hard-delete	kIOPS	-	6.27	5.04	0.41

Table 5. IO500 results for Lustre vs. S3Embedded using 10N-1PPN.

Benchmark/System	Unit	Lustre	S3Embedded	S3Remote
ior-easy-write	GiB/s	3.202073	0.990504	0.827451
mdtest-easy-write	kIOPS	13.548102	11.819631	0.240857
ior-hard-write	GiB/s	0.015462	0.215551	0.010041
mdtest-hard-write	kIOPS	5.615789	2.226332	0.111696
ior-easy-read	GiB/s	7.483045	0.375538	0.311502
mdtest-easy-stat	kIOPS	21.884679	20.489986	0.123976
ior-hard-read	GiB/s	0.095884	0.022462	0.005101
mdtest-hard-stat	kIOPS	17.428926	6.90457	0.125178
mdtest-easy-delete	kIOPS	8.995095	7.77739	0.10289
mdtest-hard-read	kIOPS	0.184843	1.287055	0.249273
mdtest-hard-delete	kIOPS	8.108844	6.711771	0.249282

Table 6. IO500 results for Lustre vs. S3Embedded using 50N-1PPN.

Benchmark/System	Unit	Lustre	S3Embedded	S3Remote
ior-easy-write	GiB/s	16.262279	5.363676	1.516302
mdtest-easy-write	kIOPS	18.104786	15.952838	1.111315
ior-hard-write	GiB/s	0.030967	0.375326	0.032984
mdtest-hard-write	kIOPS	13.705966	4.337179	0.361589
ior-easy-read	GiB/s	43.390676	2.817122	1.309052
mdtest-easy-stat	kIOPS	46.662299	45.878814	0.620202
ior-hard-read	GiB/s	0.218977	0.128423	0.025549
mdtest-hard-stat	kIOPS	43.834921	44.443974	0.62586
mdtest-easy-delete	kIOPS	9.322632	9.262801	0.585572
mdtest-hard-read	kIOPS	4.079555	6.673985	1.246393
mdtest-hard-delete	kIOPS	8.457431	6.377992	1.246105

Some MDTest results show a better performance in the case of S3Embedded than Lustre, although for both Lustre and S3Embedded, the `stat()` call is used. This might be due to the way S3Embedded implements `S3_test_bucket()`, where the size and rights for the directory and not of the actual file are captured, which seems to be faster.

The radar chart in Figure 10 shows the relative performance of S3embedded and S3remote in percent for three independent runs of all benchmarks. Note that the three Lustre runs are so similar that they overlap in the figure. The graph clearly shows the performance gaps of the two libraries. For the sake of comparison, all relative performance numbers for the run with s3embr-3 are listed. Only for the ior-hard-write phase, the number is close to 100%, while it often achieves 5-10% of Lustre performance. The embedded library also lacks performance for some benchmarks, but is much better.

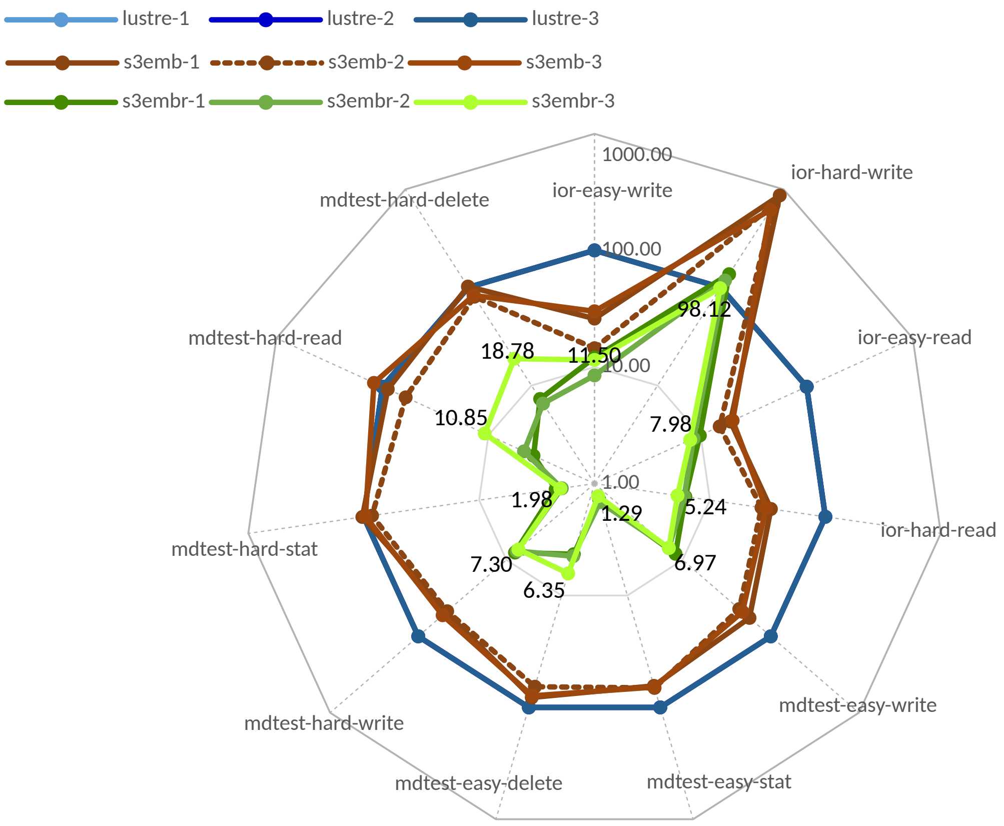


Figure 10. IO500 results of different runs using 5N-20PPN.

5. Conclusions

The S3 API is the de facto standard for accessing Cloud storage; this is why it is the component of choice when building cloud-agnostic applications. By amending IO500 to benchmark the S3 interface, we broaden the scope of the IO500 usage and enable the community to track the performance growth of S3 over the years and analyze changes in the S3 storage landscape, which will encourage the sharing of best practices for performance optimization. Unfortunately, the obtained results in Section 3 indicate that S3 implementations such as MinIO are not yet ready to serve HPC workloads because of the drastic performance loss and the lack of scalability.

We believe that the remote access to S3 is mainly responsible for the performance loss and should be addressed. We conclude that S3 with any gateway mode is not yet a suitable alternative for HPC deployment as the additional data transfer without RDMA support is pricey. However, as the experimentation in Section 4 shows, an embedded library could be a viable way to allow existing S3 applications to use HPC storage efficiently. In practice, this can be achieved by linking to an S3 library provided by the data center.

By introducing S3Embedded, a new light-weight drop-in replacement for libs3, we investigate the cause of the performance loss while providing a road toward Cloud-HPC agnostic applications that can be seamlessly run in the public cloud or HPC.

Future Work

In the future, we aim to improve the S3embeded library further and explore the conversion from S3 to non-POSIX calls. We also intend to run large-scale tests against Cloud/Storage vendors on their HPC ecosystems to compare the S3 API performance. Ultimately, our goal is to identify which APIs are needed for HPC applications to gain optimal performance while supporting HPC and Cloud convergence.

Author Contributions: Conceptualization, F.G. and J.K.; methodology, F.G. and J.K.; software, F.G. and J.K.; validation, F.G. and J.K.; formal analysis, F.G. and J.K.; investigation, F.G.; resources, F.G.; data curation, F.G. and J.K.; writing—original draft preparation, F.G.; writing—review and editing, F.G. and J.K.; visualization, F.G. and J.K.; supervision, J.K.; project administration, J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data produced by the experiments conducted in this paper can be found in https://github.com/Frankgad/s3-performance-analysis-paper (accessed on 19 July 2021).

Acknowledgments: The authors thank the DKRZ for providing the CPU hours to conduct the experiments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. AWS. AWS S3. Available online: https://aws.amazon.com/de/s3/ (accessed on 19 July 2019).
2. Weil, S.A.; Brandt, S.A.; Miller, E.L.; Long, D.D.; Maltzahn, C. Ceph: A scalable, high-performance distributed file system. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, Seattle WA, USA, 6–8 November 2006; pp. 307–320.
3. Foundation, O. OpenStack Swift. Available online: https://github.com/openstack/swift (accessed on 19 September 2020).
4. MinIO, I. Kubernetes Native,High Performance Object Storage. Available online: https://min.io (accessed 19 September 2020).
5. Rew, R.; Davis, G. NetCDF: An interface for scientific data access. *IEEE Comput. Graph. Appl.* **1990**, *10*, 76–82. [CrossRef]
6. Lofstead, J.F.; Klasky, S.; Schwan, K.; Podhorszki, N.; Jin, C. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments,Boston, MA, USA, 23 June 2008; pp. 15–24.
7. Lofstead, J.; Jimenez, I.; Maltzahn, C.; Koziol, Q.; Bent, J.; Barton, E. DAOS and friends: A proposal for an exascale storage system. In Proceedings of the SC'16 International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 13–18 November 2016; pp. 585–596.
8. Jamal, A.; Fleiner, R.; Kail, E. Performance Comparison between S3, HDFS and RDS storage technologies for real-time big-data applications. In Proceedings of the 2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, Romania, 19–21 May 2021; pp. 000491–000496.
9. Milojicic, D.; Faraboschi, P.; Dube, N.; Roweth, D. Future of HPC: Diversifying Heterogeneity. In Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 1–5 February 2021; pp. 276–281.
10. S3EmbeddedLib. Available online: https://github.com/JulianKunkel/S3EmbeddedLib (accessed on 9 December 2020).
11. bji. libs3. Available online: https://github.com/bji/libs3 (accessed on 19 August 2020).
12. Kunkel, J.; Lofstead, G.F.; Bent, J. *The Virtual Institute for I/O and the IO-500*; Technical Report; Sandia National Lab. (SNL-NM): Albuquerque, NM, USA, 2017.
13. Kunkel, J.M.; Markomanolis, G.S. Understanding metadata latency with MDWorkbench. In Proceedings of the International Conference on High Performance Computing, Frankfurt, Germany, 24–28 June 2018; pp. 75–88.
14. DKRZ. Mistral. Available online: https://www.dkrz.de/up/systems/mistral/configuration (accessed on 19 July 2020).
15. Garfinkel, S. *An Evaluation of Amazon's Grid Computing Services: EC2, S3, and SQS*; Technical Report TR-08-07, Harvard Computer Science Group: Cambridge, MA, USA, 2007 Available online: http://nrs.harvard.edu/urn-3:HUL.InstRepos:24829568 (accessed on 19 July 2021).
16. Palankar, M.R.; Iamnitchi, A.; Ripeanu, M.; Garfinkel, S. Amazon S3 for science grids: A viable solution? In Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing,Boston, MA, USA, 24 June 2008; pp. 55–64.
17. Bessani, A.; Correia, M.; Quaresma, B.; André, F.; Sousa, P. DepSky: Dependable and secure storage in a cloud-of-clouds. *ACM Trans. Storage (Tos)* **2013**, *9*, 1–33. [CrossRef]

18. Arsuaga-Ríos, M.; Heikkilä, S.S.; Duellmann, D.; Meusel, R.; Blomer, J.; Couturier, B. Using S3 cloud storage with ROOT and CvmFS. *J. Phys. Conf. Ser. Iop Publ.* **2015**, *664*, 022001. [CrossRef]
19. Sadooghi, I.; Martin, J.H.; Li, T.; Brandstatter, K.; Maheshwari, K.; de Lacerda Ruivo, T.P.P.; Garzoglio, G.; Timm, S.; Zhao, Y.; Raicu, I. Understanding the performance and potential of cloud computing for scientific applications. *IEEE Trans. Cloud Comput.* **2015**, *5*, 358–371. [CrossRef]
20. Google. PerfKit Benchmarker. Available online: https://github.com/GoogleCloudPlatform/PerfKitBenchmarker (accessed on 19 July 2020).
21. Bjornson, Z. Cloud Storage Performance. Available online: https://blog.zachbjornson.com/2015/12/29/cloud-storage-performance.html (accessed on 19 July 2020).
22. Liu, Z.; Kettimuthu, R.; Chung, J.; Ananthakrishnan, R.; Link, M.; Foster, I. Design and Evaluation of a Simple Data Interface for Efficient Data Transfer across Diverse Storage. *ACM Trans. Model. Perform. Eval. Comput. Syst. (TOMPECS)* **2021**, *6*, 1–25. [CrossRef]
23. Gadban, F.; Kunkel, J.; Ludwig, T. Investigating the Overhead of the REST Protocol When Using Cloud Services for HPC Storage. In Proceedings of the International Conference on High Performance Computing, Frankfurt am Main, Germany, 22–25 June 2020; pp. 161–176.
24. Korolev, V. AWS4C—A C Lbrary to Interface with Amazon Web Services. Available online: https://github.com/vladistan/aws4c (accessed on 19 August 2020).
25. Welch, B.; Noer, G. Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions. In Proceedings of the 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), Long Beach, CA, USA, 6–10 May 2013; pp. 1–12.
26. bji. libs3 Removes Support for Signature V2. Available online: https://github.com/bji/libs3/pull/50 (accessed on 19 August 2020).
27. Braam, P. The Lustre storage architecture. *arXiv* **2019**, arXiv:1903.01955.
28. Sysoev, I. Nginx. Available online: https://nginx.org (accessed on 19 July 2020).
29. LLNL. IOR Parallel I/O Benchmarks. Available online: https://github.com/hpc/ior (accessed on 19 September 2020).
30. AWS. Multipart Upload Overview. Available online: https://docs.aws.amazon.com/AmazonS3/latest/dev/mpuoverview.html (accessed on 19 July 2020).
31. Walsdorf, O. Cisco UCS C240 M5 with Scality Ring. Available online: https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/UCS_CVDs/ucs_c240_m5_scalityring.html#_Toc15279751 (accessed on 19 September 2020).
32. Zheng, Q.; Chen, H.; Wang, Y.; Duan, J.; Huang, Z. Cosbench: A benchmark tool for cloud object storage services. In Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 998–999.
33. AWS. Performance Design Patterns for Amazon S3. Available online: https://docs.aws.amazon.com/AmazonS3/latest/dev/optimizing-performance-design-patterns.html (accessed on 19 September 2020).