BENCHMARKING FILE SYSTEMS

June - August 2022

AUTHOR(S):
Eshan Bhargava

University of Southern California

SUPERVISOR(S):
Luca Atzori
Krzysztof Michal Mastyna
Joaquim Santos

PROJECT SPECIFICATION

As the scale of processing systems increase, the data access from storage systems needs to increase commensurately. In this project we will report benchmarks on file system technologies used to deliver data under a variety of workflow conditions. We will look at I/O rates, metadata operations, and data integrity and reliability for a number of workflows and access patterns. We will apply consistent tests under similar conditions and scale to get an accurate assessment. Our results will be useful for deployment decisions and will be of interest to a broad community.

ABSTRACT

The file system technologies in discussion for this project include two installations of Intel DAOS and a CephFS production system. We use a variety of benchmarks, such as IOR and fio, to analyze bandwidth of applying load on the file systems. We show that some benchmarks may not be able to be used with certain storage systems. Additionally, we compare two installations of DAOS.

TABLE OF CONTENTS

1 INTRODUCTION 4
- 1.1 DAOS Background . . . 4
  - 1.1.1 DAOS at CERN . . . 4
- 1.2 Ceph Background . . . 5
  - 1.2.1 CephFS . . . 5

2 Benchmark: IOR 5
- 2.1 Background . . . 5
- 2.2 Results . . . 6
  - 2.2.1 Ice Lake Cluster . . . 7
  - 2.2.2 Cascade Lake Cluster . . . 7
  - 2.2.3 Revised IOR Command Results . . . 7

3 Benchmark: DBench 10

4 Benchmark: fio 12

5 Real Use Case: Root LHCb Benchmark 12

6 Conclusion 14
- 6.1 DAOS . . . 14
- 6.2 CephFS . . . 14
- 6.3 Comparison of Ceph and DAOS . . . 15
- 6.4 General Conclusion . . . 15
- 6.5 Acknowledgements . . . 15

7 REFERENCES 15

1 INTRODUCTION

Artificial Intelligence and High Performance Computing Applications have caused exponential data growth. Moreover, this influx of data must be stored and accessed instantaneously. Thus, Data I/O operations become more complex, with a mixture of reads and writes with a wide variety of block sizes. [5] In turn, companies are quickly finding solutions to the increase in data by developing new storage systems.

The purpose of this project to explore file system technologies used to deliver data under a variety of workflow conditions. The two systems being studied include Intel DAOS and CephFS. We will use a variety of benchmarks including IOR, DBench, fio, and more to measure bandwidth and latency.

We start with general purpose performance benchmarks, but the final goal is to evaluate the performance of a High Energy Physics (HEP) specific application. The benchmark for the HEP-specific application was provided by the ROOT team at CERN.

1.1 DAOS Background

DAOS stands for Distributed Asynchronous Object Storage; it is an open source, software-defined, scale-out object store that provides high bandwidth, low latency, and high I/O operations per second (IOPS) storage containers to High Performance Computing (HPC) applications. It is an R&D file system that is the foundation of Intel's exascale storage stack, which has great potential for the use cases at CERN and tech companies. [5]

Most storage systems today are designed for block I/O, with operations going through the Linux kernel using a block interface. While progress has been made in optimizing access to the block device, with techniques like coalescing, buffering, and aggregation, those optimizations are not relevant for Intel's new storage technology. Toward this end, DAOS is designed to optimize access to Intel memory technologies, such as Optane Persistent Memory, which will eliminate the unnecessary overhead associated with traditional storage stacks.[5]

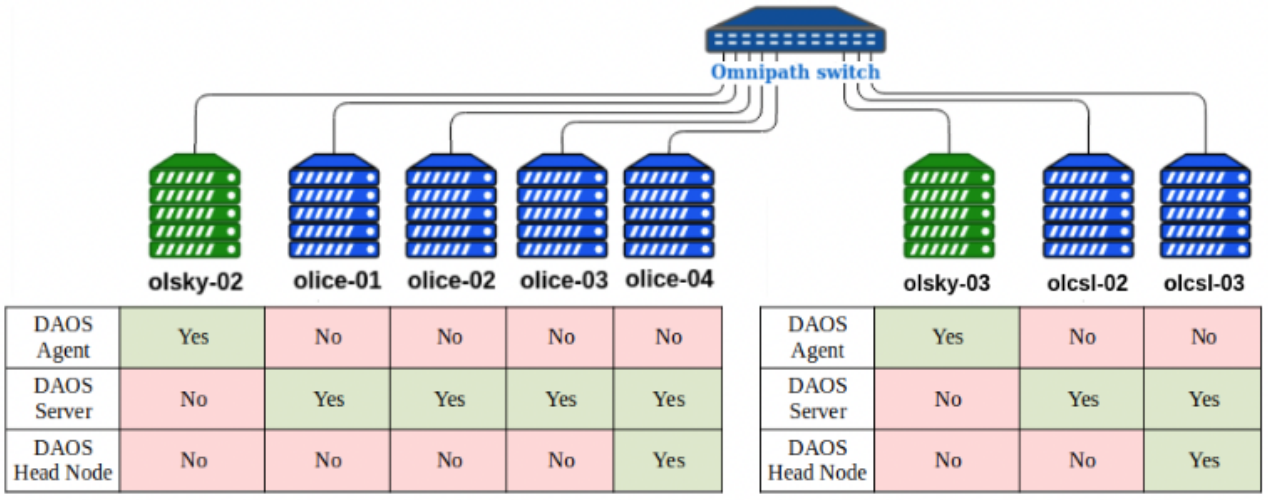
Moreover, traditional storage systems use high-latency peer-to-peer communication. DAOS shifts away from this by using low-latency, high-message-rate user-space communications that completely bypass the Operating System. This allows DAOS to support fine-grained data access and unlock the performance of next-generation storage technologies. [5]

1.1.1 DAOS at CERN

The new DAOS 2.0.2 installation is included on two clusters. One of which has 4 servers (Ice Lake Cluster). The other has 2 servers (Cascade Lake Cluster). The setup can be shown by figure 1. [4] Each of these clusters have 1 client, thereby in theory the Ice Lake Cluster should have better performance.

Not to mention, the Ice Lake Cluster has the new and improved hardware; the Ice Lake microprocessor has a new architecture, allowing the 3rd generation of Xeon to have anywhere from 8 to 40 cores per socket, almost doubling the 4-28 cores available with Cascade Lake microprocessor. Moreover, the L1/L2/L3 caches of each of the Ice Lake cores is 48 KB, 1.25 MB, and 1.5 MB, respectively, compared to the Cascade Lake's 32 KB/1 MB/1.375 MB.

Additionally, Ice Lake features two more memory channels than Cascade Lake with a total of 8, and a peak 3200 DIMM speed, marking an increase from Cascade Lake's peak 2933 speed. Lastly, Ice Lake provides more PCIe lanes, a total of 64, and an improvement to PCIe 4.0 over the previous generations' 48 lanes of PCIe 3.0. [3]



DAOS Agent	Yes	No	No	No	No	DAOS Agent	Yes	No	No
DAOS Server	No	Yes	Yes	Yes	Yes	DAOS Server	No	Yes	Yes
DAOS Head Node	No	No	No	No	Yes	DAOS Head Node	No	No	Yes

Figure 1: CERN openlab DAOS clusters [4]

1.2 Ceph Background

Ceph is an open source, software-defined storage solution designed to address block, file and object storage needs of modern enterprises. [2] The design of Ceph was built around making an object store that is scalable, self-managing, software-based, and has no single point of failure. [7]

Its high scalability and reliability is driving it to be adopted as the new norm for high-growth block storage, object stores, and data lakes. But, the main advantage of Ceph is that it provides interfaces for multiple storage types within a single cluster, eliminating the need for multiple storage solutions or any specialised hardware, thus reducing management overheads. [2]

1.2.1 CephFS

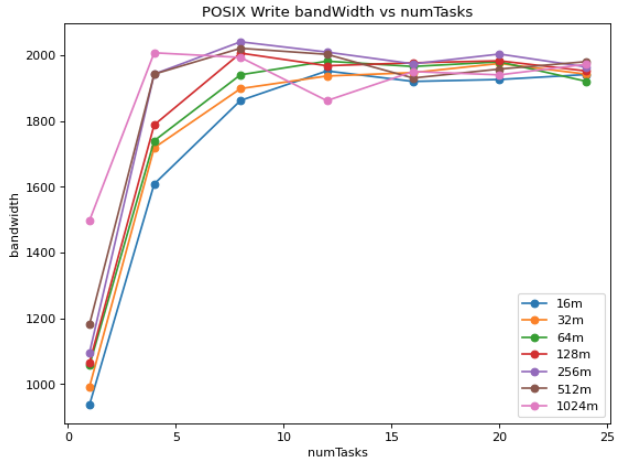
As mentioned earlier, Ceph supports multiple types of storage including objects, blocks, and files. If one wants to interact with a Ceph cluster using files, the client accesses the file system, CephFS. [7] CephFS is a POSIX-compliant, distributed file system with a Linux client that has support for FUSE. [2]

2 Benchmark: IOR

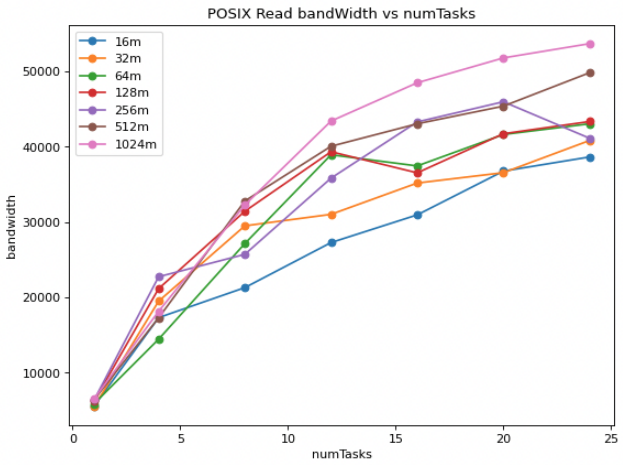
2.1 Background

The main benchmark used for this project is IOR; it is a I/O benchmark that can be used to test the performance of parallel storage systems using various interfaces and access patterns. It uses a common parallel I/O abstraction backend and relies on MPI, or Message Passing Interface, for synchronization. [6]

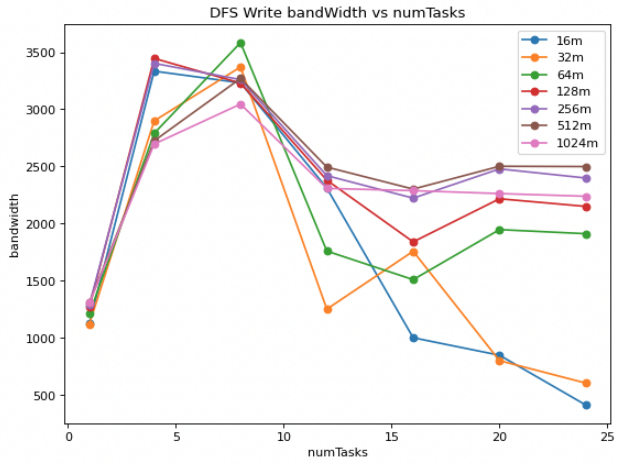
IOR puts load on a system through I/O operations, specifically by executing reads and writes on a file. In this way, it can measure various statistics of a system, including latency and bandwidth. One can specify or change many different flags in order to measure the performance of the system. Some of the flags experimented with for this project are block size, transfer size, and number of tasks.



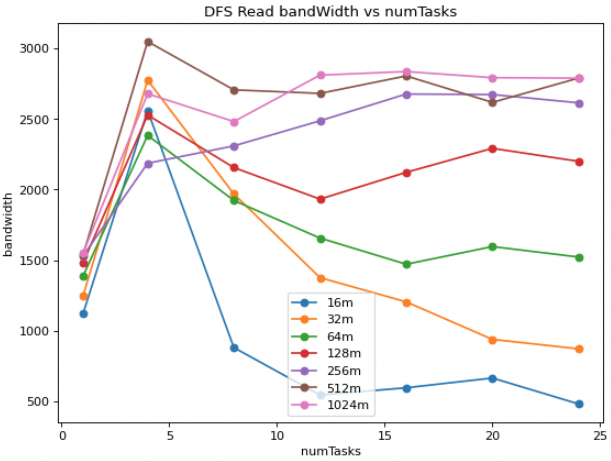
((a)) DFUSE write on Ice Lake Cluster



((b)) DFUSE read on Ice Lake Cluster



((c)) DFS write on Ice Lake Cluster



((d)) DFS read on Ice Lake Cluster

Figure 2: IOR initial trial on Ice Lake Cluster

Another important flag which can be specified is "filePerProc"(F). If this flag is set, each MPI process performs I/O to a unique file. On the other hand, if not specified, MPI processes only write to a single file. The "reorderTasksConstant"(C) flag must also be set in order to avoid the cache.

2.2 Results

The initial flags used with IOR were based on a previous benchmarking report about the first DAOS installation at CERN (DAOS 0.9), where the block sizes varied from 16 Megabytes (MB) to 1 Gigabyte (GB) and the transfer size was 4 Kilobytes (KB). The number of tasks start at 1 and increase to 24 in increments of 4.

Here is an example of that command:

```
$ mpirun -n $i root/bin/ior -a POSIX -b 16m -t 4m \
-F -C -i 10 -o /test1
```

2.2.1 Ice Lake Cluster

One can observe the initial IOR results of the Ice Lake Cluster in figure 2. This figure shows the write and read bandwidth (MB/s) for both the POSIX(DFUSE) and DFS APIs. The top row has the POSIX results and the bottom has the results for DFS.

When comparing the write bandwidth for both the APIs in figure 2, we can see that the POSIX API performs worse than the DFS API in terms of peak performance, but it does have a better shape than the DFS API. In theory, as the number of tasks increase, so should the performance of DAOS, to a certain degree. Therefore, figure 2(a) has a shape which we would expect.

Additionally, we can conclude that as the block size increases, so does the performance in general. A possible reason for this is because with the specific range of block sizes we use, the system can transfer data blocks in similar times with the 1 GB block size as it does with the 16 MB block size, for example. In this respect, each subfigure in figure 2 makes sense because the block sizes with best performance are usually between 1 Gigabyte, 512 Megabytes, and 256 Megabytes. That being said, there are many other observations, excluding figure 2(a), that are not explainable.

Starting with figure 2(b), there is a massive jump in performance. We already rule out the possibility that the IOR program is using the page cache because we specify the "reorderTasksConstant" flag, which reorders the tasks by forcing each MPI process to read data written by its neighboring node.

Looking at figure 2(c), we can see the performance decreases after 8 tasks; this is not the expected behavior because as the tasks increase, the performance should at least stay the same if not increase. If multiple nodes are executing IOPS on a file, such that all nodes are working at the same time, the performance should not decrease, yet it does.

Lastly, when observing figure 2(d), we have a similar discrepancy as figure 2(c) with some block sizes, but not others.

2.2.2 Cascade Lake Cluster

When comparing the set of graphs in Figure 3 to those of Figure 2, we can see that the trends of each graph are very similar, respectively.

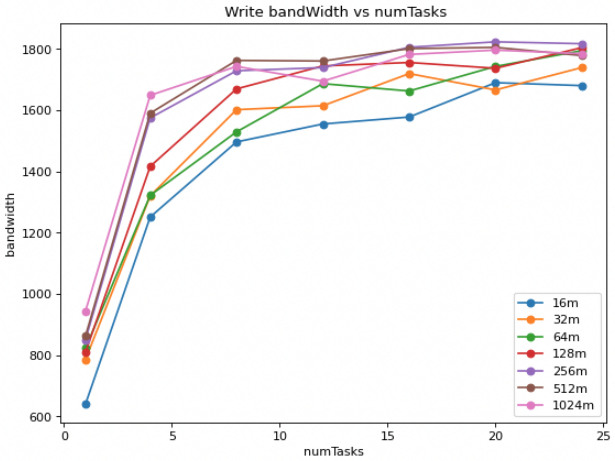
The bigger differences lie in the performance itself. Looking at figures 3(a), 3(b), 2(a), and 2(b), the peak bandwidth for both the write and read IOPS is inferior for the Cascade Lake Cluster in comparison to the Ice Lake Cluster. Therefore, we can make the conclusion that the Ice Lake Cluster performs better than the Cascade Lake Cluster with the POSIX API. But with the DFS API (figures 3(c), 3(d), 2(c), 2(d)), the Cascade Lake Cluster outperforms the Ice Lake Cluster with system writes, but falls behind again with system reads.

However, the caveat with these results is that they are invalid. We were surprised that the POSIX API yielded superior performance in bandwidth than the DFS API. Therefore, we consulted with Intel, and were given a revised IOR command with different flags.

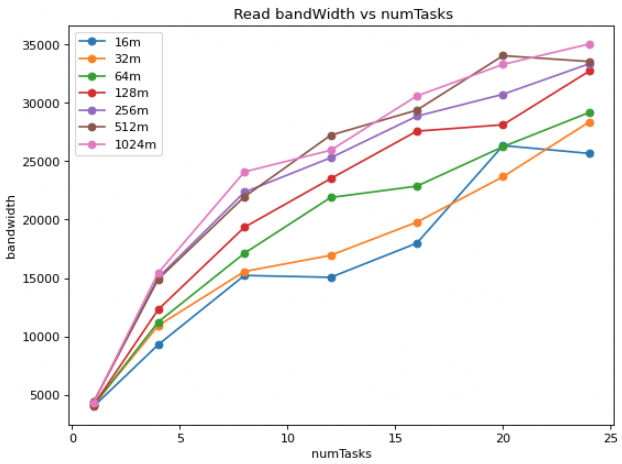
2.2.3 Revised IOR Command Results

The revised IOR command looks something like this with the DFS API:

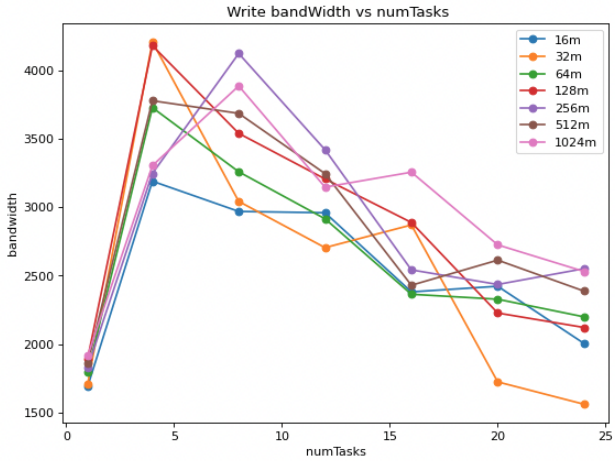
```
$ mpirun -hostfile "$cwd/hostfiles/$nnb-nodes.cfg" -np $np \
--ppn $ppn --bind-to socket "$ior_bin" -a DFS -r -w -t 1MB -b ${bs}G \
-o /testfile -i 5 --dfs.pool=$pool_name \
--dfs.group=daos_server --dfs.cont=$container_name
```



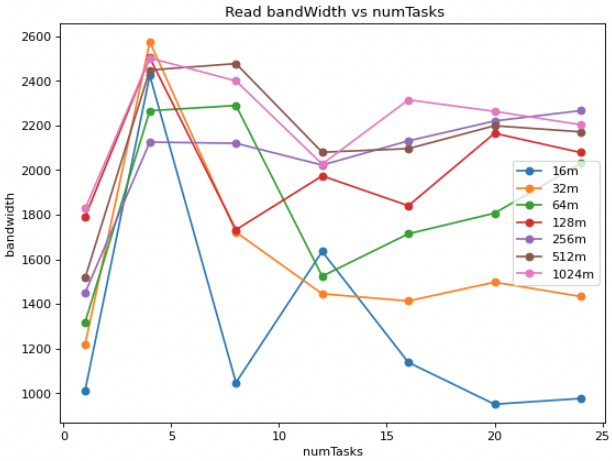
((a)) DFUSE write on Cascade Lake Cluster



((b)) DFUSE read on Cascade Lake Cluster

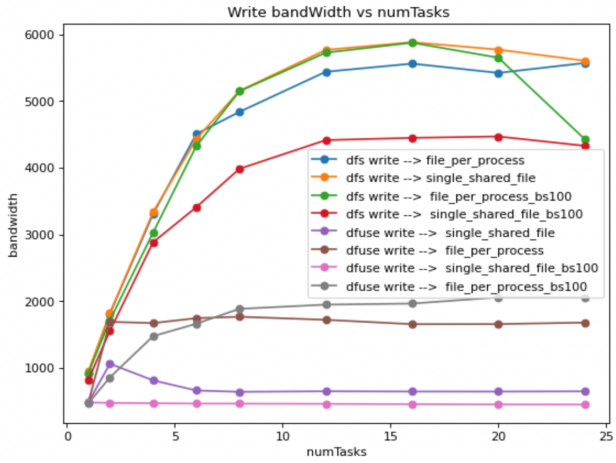


((c)) DFS write on Cascade Lake Cluster

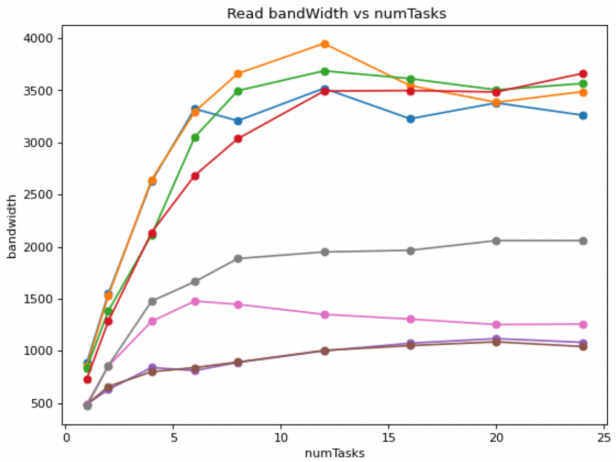


((d)) DFS read on Cascade Lake Cluster

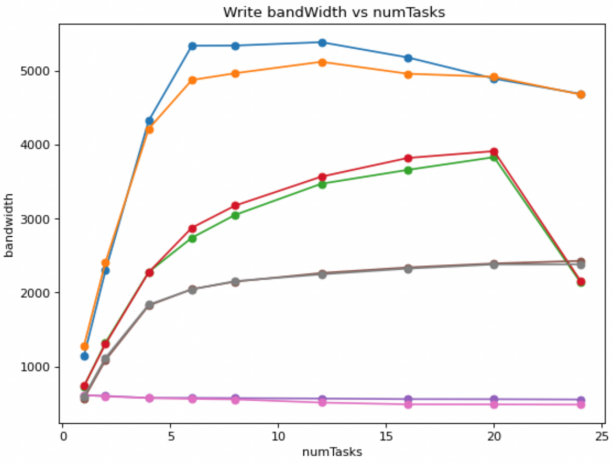
Figure 3: IOR initial trial on Cascade Lake Cluster



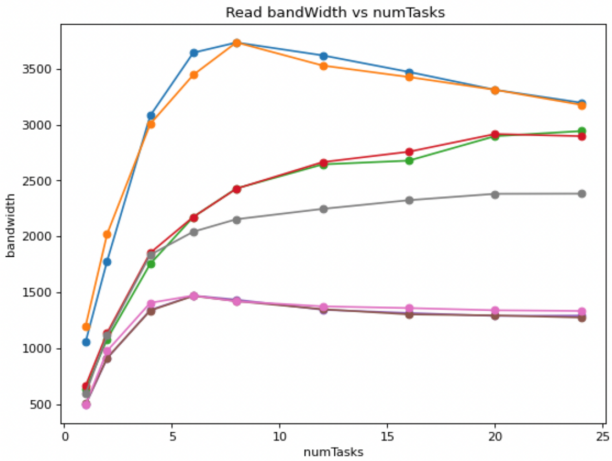
((a)) Intel DFS write on Cascade Lake Cluster



((b)) Intel DFS read on Cascade Lake Cluster



((c)) Intel DFS write on Ice Lake Cluster



((d)) Intel DFS read on Ice Lake Cluster

Figure 4: IOR Intel Results

Whereas, the revised IOR command looks similar to this when using the POSIX(DFUSE) API after mounting the DAOS file system:

```
$ dfuse --disable-caching --mountpoint="$mount_point" --pool=$pool_name \
--container=$container_name

$ mpirun -hostfile "$cwd/hostfiles/$nnb-nodes.cfg" -np $np \
--ppn $ppn --bind-to socket "$ior_bin" -a POSIX -r -w -t 1MB -b ${bs}G \
-o "$mount_point/test" -i 5
```

Evidently, Intel's command has much more flags, which were not present in IOR's documentation. Nonetheless, these flags yielded believable results when put to the test. Figure 4 has the results of the DFS and POSIX APIs of IOR on both clusters.

Firstly, we can observe that the best results, regardless of the cluster, are obtained with the DFS API. The POSIX API performs significantly worse for both IOPS in each cluster. This is a positive result because it shows that the DFS API is indeed using DAOS technologies to perform better, rather than just using the DAOS space which is the case when using the POSIX API with a mount point.

When using the revised Intel command, we can see that the best write performance is achieved with the Cascade Lake Cluster with a bandwidth of a little less than 6000 Megabytes per second at 16 tasks compared to the Ice Lake Cluster, which has a peak bandwidth of around 5200 Megabytes per second at 16 tasks as well.

Looking at the read performance of the Intel command, we can see that the Cascade Lake Cluster outperforms the Ice Lake Cluster yet again. The peak performance of the Cascade Lake Cluster is around 3900 Megabytes per second compared to that of the Ice Lake Cluster, which is around 3700 Megabytes per second.

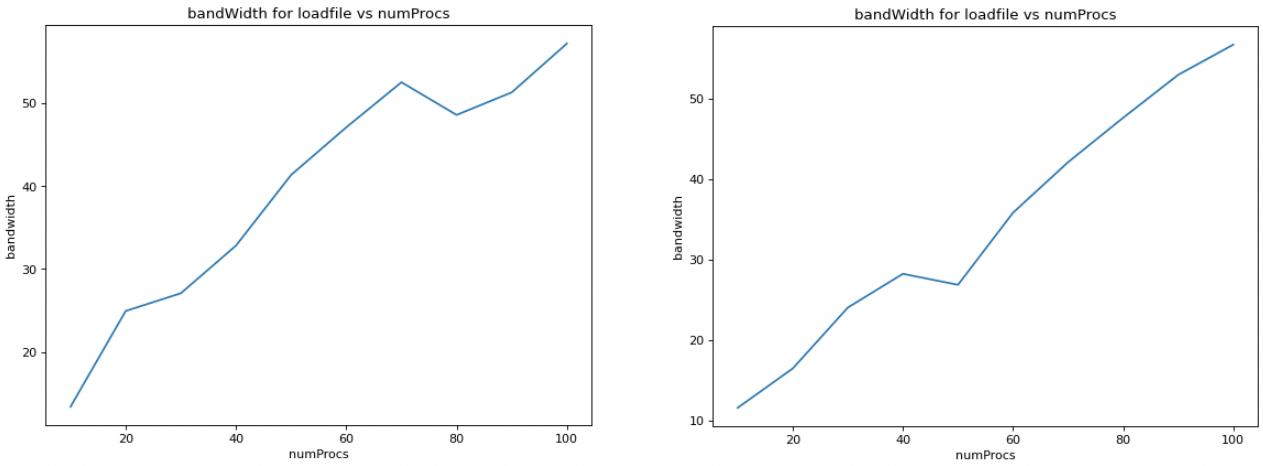
With both I/O operations, the Cascade Lake Cluster beats the Ice Lake Cluster. These results may hint one of two things. Firstly, there is a possibility that there is something wrong with the hardware configuration of the Ice Lake Cluster. Theoretically, the Ice Lake Cluster should have had better performance because it has 4 servers (and improved hardware). When MPI is combined with IOR, tasks should be executed in parallel, and distributed across each server node in the cluster. Thus, IOR should be more efficient with the Ice Lake Cluster, but it is not. Secondly, it could be that the optimal ratio of servers to clients should mimic the Cascade Lake Cluster configuration, 2:1.

However, another thing to keep in mind is that the Cascade Lake Cluster only outperforms the Ice Lake Cluster by a few hundred Megabytes per second for both I/O operations, meaning the performance difference between the clusters is minimal. Therefore, more investigation would probably need to be completed before making a firm conclusion that the Cascade Lake Cluster outperforms the Ice Lake Cluster.

3 Benchmark: DBench

DBench is a tool to generate I/O workloads to stress a file system or a server. Therefore, one can see which workload becomes saturated and can also be used for prediction analysis. DBench is built around the concept of a loadfile, which is a sequence of operations to be performed at a specific time. One can carefully model a load file, such that its content mimics an I/O workload that matches how a particular application performs.[8]

However, since we are not looking for a specific application's performance on DAOS, we let DBench itself create and craft a loadfile. Note that loadfiles generated by each run of DBench



((a)) DBench on Olice Cluster

((b)) DBench on Olcsl Cluster

Figure 5: DBench on DAOS Clusters

fio Results		
I/O Procedure	I/O Type	bandwidth (KB/s)
1 reader	random	201
5 readers	random	1179
1 writer	random	538
5 writers	random	400
1 reader	sequential	7355
5 readers	sequential	37800
1 writer	sequential	1188
5 writers	sequential	1629

Table 1: fio Results

are very similar, but not exactly the same to each other. Therefore, small changes in results across runs are negligible.

Figure 5 has the results of running DBench on the DAOS clusters. The parameter that is changing is the number of processes, while the loadfile generated for each DBench command will have similar amounts and types of I/O operations. When consulting the results, it's hard to differentiate between them, excluding a few small peaks and valleys within each of the subfigures.

Therefore, in the future, we could look into crafting a custom loadfile which puts much more stress on the system, such that the bandwidth difference between both the clusters is much more evident. This could be done by adding more operations within a loadfile generated by DBench and/or increasing the time it runs.

A problem, however, is that DBench may not be aware of the underlying DAOS file system. While it can place load on the mountpoint of DAOS, it may not be able to actually use DAOS servers/hardware to execute operations. Given this fact, DBench may not be a suitable benchmark for DAOS currently, although it is used as a benchmark in the Intel Documentation of DAOS.

Parameters for LHCb Benchmark				
Object Class	OC_SX	RP_XSF		
Compression	none	zstd		
Page Size	65536	524288	1048576	4194304
Cluster Size	52428800			

Table 2: Parameters for LHCb Benchmark

4 Benchmark: fio

As an exploratory analysis, fio was run on Ceph. Fio stands for Flexible I/O, and it is used to run specific workloads on a system. Usually, one would have to carefully tailor a test to simulate a certain workload, but with fio, all one needs to do is change the flags of an fio command. Fio spawns a number of threads or processes doing a particular type of I/O action as specified by the user. It takes a number of global parameters, each inherited by the thread, unless certain parameters are passed in the command that override this setting. The typical use of fio is to write a job file matching the I/O load one wants to simulate.[1]

Looking at the fio results, a conclusion we can make is that sequential I/O operations perform better than random I/O operations. Additionally, the greater the number of workers for a certain I/O task, the higher the bandwidth. Both of these behaviors are to be expected.

However, the bandwidth is very small, which is not expected by a powerful system like Ceph. In fact, the scale of the bandwidth is in Kilobytes per second, rather than Megabytes per second or Gigabytes per second. This means that fio is probably not aware of Ceph and it's hardware. There could be a few reasons for this. The most probable explanation for the discrepancy is that there is a misconfiguration in the way the benchmark was run with Ceph. The less likely reason is that this benchmark is actually not useful for Ceph. In the end, we didn't have enough time to explore Ceph further in order to find the reason for this anomaly.

5 Real Use Case: Root LHCb Benchmark

This benchmark was provided by the ROOT team at CERN. It is based on writing to a DAOS pool and then reading back a partial dataset. This data set is taken from Open Data (LHCb Run 1, B mesons 3 hadrons) and loaded into RNTuple. There are certain parameters that scientists can control, such as compression, redundancy level, size of the stored binary objects, and amount of entries involved in requests at any one time. Our particular experiment varied the parameters in table 2. Tables 3 and 4 have the results of the benchmark on the Cascade Lake and Ice Lake Cluster. Note that since the cluster size parameter is unchanged, I will not mention it in these tables. In each table, for each Page size, Object type, and Compression type, there is a result that is measured in Gigabytes per second.

The gen_lhcb test script reads an existing TTree file and converts it into an NTuple. Then it writes that NTuple to DAOS storage. The lhcb test reads back values from the NTuple and performs some simple analysis on it. In essence, it computes some results based off measurements from different columns and generates a histogram. The read back pattern is heavily columnar-oriented, which is the traditional HEP analysis use case. Thus, values from the same column and roughly neighboring rows tend to be read together.

First, let's compare the write performance with no compression (Tables 2(a) 3(a)). The results indicate that the Ice Lake and Cascade Lake Cluster will yield results between 2 and 3.3 Gigabytes per second, with the Cascade Lake Cluster slightly outperforming the Ice Lake

((a)) No Compression Write

chep21_summary_gen_lhcb-vPSfCS-none		
Page Size	OC_SX	RP_XSF
65536	2.04784	2.09669
524288	2.8775	2.83247
1048576	3.04064	2.92389
4194304.0	2.91011	2.92148

((b)) Zstandard Compression Write

chep21_summary_gen_lhcb-vPSfCS-zstd		
Page Size	OC_SX	RP_XSF
65536	2.15162	2.17508
524288	2.63608	2.56158
1048576	2.82797	2.75375
4194304.0	2.68813	2.73516

((c)) No Compression Read

chep21_summary_lhcb-vPSfCS-none		
Page Size	OC_SX	RP_XSF
65536	1.52846	1.50252
524288	1.95117	1.95548
1048576	1.86271	2.07176
4194304.0	2.15496	2.1344

((d)) Zstandard Compression Read

chep21_summary_lhcb-vPSfCS-zstd		
Page Size	OC_SX	RP_XSF
65536	1.09623	1.28202
524288	1.64154	1.6142
1048576	1.96568	2.01815
4194304.0	2.16757	1.8203

Table 3: Root Benchmark on Cascade Lake Cluster

((a)) No Compression Write

chep21_summary_gen_lhcb-vPSfCS-none		
Page Size	OC_SX	RP_XSF
65536	2.66905	2.04747
524288	2.35451	2.33748
1048576	3.21094	2.36934
4194304.0	2.81844	2.41597

((b)) Zstandard Compression Write

chep21_summary_gen_lhcb-vPSfCS-zstd		
Page Size	OC_SX	RP_XSF
65536	1.80858	1.80315
524288	2.21157	2.2043
1048576	2.27341	2.33801
4194304.0	2.43718	2.43642

((c)) No Compression Read

chep21_summary_lhcb-vPSfCS-none		
Page Size	OC_SX	RP_XSF
65536	0.81567	0.75628
524288	1.05472	1.75792
1048576	1.08523	1.80349
4194304.0	1.2943	1.84193

((d)) Zstandard Compression Read

chep21_summary_lhcb-vPSfCS-zstd		
Page Size	OC_SX	RP_XSF
65536	0.51135	0.59559
524288	0.87178	0.83415
1048576	0.83293	0.84908
4194304	0.94325	1.8319

Table 4: Root Benchmark on Ice Lake Cluster Trial 1

Cluster. For example, when looking at the bandwidth for the page size of 65536, we can see that the Ice Lake Cluster outperforms the Cascade Lake Cluster with the OC_SX type object(2.047 vs 2.669), while slightly underperforming with the RP_XSF object(2.096 vs 2.047). Whereas, with the the page size of 4194304, the Ice Lake Cluster outperforms the Cascade Lake Cluster with both object types. Overall, for this specific setting, the Cascade Lake Cluster outperforms the Ice Lake Cluster 6 times out of 8.

Looking at the rest of the tables, the Cascade Lake Cluster almost always outperforms the Ice Lake Cluster for each combination of page size, object type, and compression type for both the read and write tests.

However, when consulting with the ROOT team, they found it strange how the read performance (LHCb script) of the benchmark deteriorated on the Ice Lake Cluster, evident by

((a)) No Compression Write

chep21_summary_gen_lhcb-vPSfCS-none		
Page Size	OC_SX	RP_XSF
65536	2.22428	2.22524
524288	2.80022	2.77345
1048576	2.80264	2.79854
4194304.0	2.86110	2.90832

((b)) Zstandard Compression Write

chep21_summary_gen_lhcb-vPSfCS-zstd		
Page Size	OC_SX	RP_XSF
65536	2.12981	2.12560
524288	2.63672	2.60660
1048576	2.76273	2.76357
4194304.0	2.87566	2.89805

((c)) No Compression Read

chep21_summary_lhcb-vPSfCS-none		
Page Size	OC_SX	RP_XSF
65536	1.30397	1.26878
524288	1.94374	1.94246
1048576	2.07695	2.01569
4194304.0	2.13248	2.06476

((d)) Zstandard Compression Read

chep21_summary_lhcb-vPSfCS-zstd		
Page Size	OC_SX	RP_XSF
65536	1.18197	1.23968
524288	1.67102	1.77391
1048576	1.97760	1.94331
4194304	2.13033	2.09163

Table 5: Root Benchmark on Ice Lake Cluster Trial 2

Tables 3(c) and 3(d). Therefore, we ran the benchmark once more on the Ice Lake Cluster, and it yielded better results, which can be found in Table 5. Although, the Cascade Lake Cluster still outperforms the Ice Lake Cluster, the difference in the performance is much smaller. This performance discrepancy between both the benchmark runs on the Ice Lake Cluster is most likely due to the possibility that another user could have been putting heavy load on their DAOS pool, such that the whole system was stressed. If this situation indeed came to fruition, then it could very well have deteriorated the performance of this benchmark.

6 Conclusion

6.1 DAOS

The conclusion that we have to make about DAOS when comparing the two clusters is actually that the Cascade Lake Cluster outperforms the Ice Lake Cluster. This is an unexpected result because the Ice Lake Cluster has more servers than the Cascade Lake Cluster. As mentioned earlier, there could be something wrong with the hardware configuration of the Ice Lake Cluster or the optimal ratio of servers to clients should mimic the Cascade Lake Cluster configuration, 2:1.

6.2 CephFS

We tried to run many benchmarks on Ceph, but were hindered by the fact that we did not have access to sudo. This was a major issue because we need sudo privileges to deploy CephFS. Moreover, many benchmarks that don't need sudo privileges do need to be aware of the underlying file system. In the case of IOR, there are APIs that allow MPI/IOR to register the presence of and use an underlying file system technology, such as the DFS API for using DAOS. However, there is no such API for CephFS. We stuck with fio as a benchmark to run on the Ceph mountpoint. Fio is also not aware of Ceph, so it is simply used as an exploratory analysis.

6.3 Comparison of Ceph and DAOS

Unfortunately, during my internship there was no way to compare Ceph and DAOS because of the lack of time and the fact that we did not have sudo privileges. If we had such privileges, we would be able to deploy a CephFS instance and thereby match the pool sizes, and other resources. Then, depending on which file system does better, CephFS or DAOS, we could possibly make a conclusion that the respective File system is better because of the hardware that it utilizes.

6.4 General Conclusion

All in all, this work was useful to set a first baseline for benchmarking different file systems at CERN. Although, this project was mainly exploratory there is value in having tried different options because the framework has been set to try other options and experiments in the future.

6.5 Acknowledgements

I would like to thank the CERN openlab for giving me the opportunity to work on this project. I would also like to thank Ian Fisk from the Flatiron Institute for providing access to Ceph and the ROOT team for providing a benchmark we could use for a real HEP use case. Lastly, I would like to thank Luca Atzori, Krzysztof Michal Mastyna, and Joaquim Santos for mentoring me during this project.

7 REFERENCES

[1] Jens Axboe. *fio - Flexible I/O tester rev. 3.30*. 2017. URL: https://fio.readthedocs.io/en/latest/fio_doc.html#overview-and-history (visited on 08/11/2022).

[2] Canonical. *openlab DAOS version 2.0.2*. 2022. URL: https://ubuntu.com/ceph/what-is-ceph (visited on 08/04/2022).

[3] Zach DeMeyer. *Comparing Ice Lake to Previous Generations of Xeon Scalable Processors*. 2021. URL: https://gestaltit.com/tech-field-day/zach/comparing-ice-lake-to-previous-generations-of-xeon-scalable-processors/#:~:text=Ice%20Lake%20also%20features%20two,Cascade%20Lake's%20up%20to%202933. (visited on 10/17/2022).

[4] CERN openlab-systems Docs. *openlab DAOS version 2.0.2*. URL: https://openlab-systems.web.cern.ch/daos/daos/ (visited on 08/04/2022).

[5] Intel. *DAOS: Revolutionizing High-Performance Storage*. 2021. URL: https://www.intel.com/content/www/us/en/high-performance-computing/daos-high-performance-storage-brief.html (visited on 08/02/2022).

[6] *ior*. 2018. URL: https://ior.readthedocs.io/en/latest/ (visited on 08/04/2022).

[7] Sniper Network Ross Turk. *Ceph Intro Architectural Overview*. 2015. URL: https://www.youtube.com/watch?v=7I9uxoEhUdY (visited on 08/04/2022).

[8] *Welcome to the DBENCH web pages*. URL: https://dbench.samba.org/ (visited on 08/10/2022).