GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Influence of the file system on the performance of machine learning workloads

Esther Hagenkort

MatrNr: 21876433

Supervisor: Patrick Höhn

Georg-August-Universität Göttingen
Institute of Computer Science

March 30, 2024

Abstract

This work looks into the influence the two different file systems Lustre and BeeGFS can have on the input-output performance when training machine learning models with large amounts of image data. The training of machine learning models costs a lot of time and money and with that also environmental sustainability. With around 90% of the time cost coming from the input-output performance it is an important research area.
Work such as Lackschewitz et al. [Lac+22] and Kunkel et al. [Kun+18] already looked into different storage systems and tools to analyse them.
This work is, in comparison, a more detailed and direct introduction on how the problem can be approached and less result-oriented. It aims to explain specific steps one can take rather than producing extensive benchmarks.
For that the theoretical foundation for the file systems is laid, the data set Conceptual Captions and the characterisation tool Darshan are introduced and their set up and use is explained. Additionally, exploratory tests are conducted and analysed, including an incomplete Darshan log file. Some trends, such as that Lustre seems to be performing worse in the tested work set up, and some possible explanations, such as the performance decrease when reaching a certain filling capacity, are elaborated.

Statement on the usage of ChatGPT and similar tools in the context of examinations

In this work I have used ChatGPT or a similar AI-system as follows:

- ☐ Not at all
- ☐ In brainstorming
- ☐ In the creation of the outline
- ☐ To create individual passages, altogether to the extent of 0% of the whole text
- ☐ For proofreading
- ☑ Other, namely: Deepl and Deepl Write for wording and translation

I assure that I have stated all uses in full.
Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables

List of Figures

List of Listings

List of Abbreviations

AI Artificial Intelligence

CPU Central Processing Unit

GB Gigabyte(s)

GPU Graphics Processing Unit

GWDG Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

HPC High-Performance Computing

I/O Input/Output

MB Megabyte(s)

MiB Mebibyte(s)

MDS Metadata Server

MDT Metadata Targets

ML Machine Learning

MPI Message Passing Interface

NHR National High Performance Computing

ODS Object Data Storage

OST Object Server Targets

SCC Scientific Compute Cluster

STDERR Standard Error

USA United States of America

1 Introduction

Machine Learning (ML) and more specifically Deep learning is growing in importance and performance, which is closely linked to its ability to train on ever-increasing amounts of data. This leads to increasing pressure to improve the I/O performance, which currently is a bottleneck. Enhancing the I/O performance can also enhance the sustainability and cost of resources, such as computational time. This is why this work looks into the influence of the file system on the performance of ML workloads.
Work such as Lackschewitz et al. [Lac+22] and Kunkel et al. [Kun+18] already looked into different storage systems and tools to analyse them. Latham et al. [LRT04] researched the impact file systems can have on Message Passing Interface (MPI)-I/O scalability. Pumma et al. [Pum+19b] worked on scalable deep learning in connection to analysing and optimising their I/O performance.
This work aims to explain specific steps one can take rather than producing extensive benchmarks. For that the theoretical foundation for distributed, parallel file systems in general and Lustre and BeeGFS specifically is laid and the data set Conceptual Captions is introduced as an exemplary ML data set. Exploratory tests examine the variation between different test runs, whether different partitions can influence the workload and for a more in depth analysis Darshan is introduced as a characterisation tool.
The resulting insights include interesting trends that have the potential to be investigated further. For example, the results suggested that in this case Lustre is slower than BeeGFS and that the partitions influence is not significant, while the size of the individual images and presumably the filling level of the file systems do play a role performance wise.
The contributions of this work include the following.

1. Laid the theoretical foundations of the I/O bottleneck
2. Introduced distributed, parallel file systems in general and Lustre and BeeGFS specifically
3. Presented the Conceptual Captions data set and HuggingFace
4. Described Darshan and explained its installation and use
5. Performed and analysed some exploratory I/O tests

This report consists of the main parts introduction, foundation, methods, results, discussion and conclusion. In the foundation the theoretical background of the I/O bottleneck in deep learning is provided and the distributed, parallel file systems are introduced. The Methods section explains the project setup and methodology for analysing I/O performance, including an introduction to the characterisation tool Darshan. The Results section presents the data on average running times, variations, differences in data rates and a Darshan logging attempt. Lastly, these results are discussed and put into perspective. In the conclusion the findings are summarised and an outlook for potential further work is given.

2 Foundation

2.1 DeepLearning I/O bottleneck

Deep learning has been gaining importance in multiple domains over the last years and is likely to grow more in the following ones. Some popular examples of successful ML models are ChatGPT [Ope22], Stable Diffusion [Tea] and AlphaFold [tea20].
ChatGPT, the chat bot developed by OpenAI, is widely known even to people foreign to machine learning or computer science in general. The tool is used by both students and professionals to assist with presentations or research, as well as by anyone for everyday inquiries.
Stable Diffusion is an image generation model developed by researchers of the Ludwig Maximilian University of Munich and the Interdisciplinary Center for Scientific Computing of the Heidelberg University and Runway[1] [Rom+22]. The model accepts textual prompts as input and produces images that match the provided description. In this manner, the system can not only display images it has encountered during training, but also generate entirely novel ones. [Tea; RE22; Rom+22]
AlphaFold, and now its successor AlphFold 2, is a deep learning model developed by Google DeepMind. Given the amino acid sequence as input it predicts the three dimensional structure of folded proteins. This structure can give insights to the functionality of the proteins and is not easily deducted. With the help of AlphaFold 2 researchers can save hours of tedious lab work and focus on developing, for example, new treatments for diseases. [Cal20; Eis21; tea20]
These and other successes are made possible by three kinds of advancements. First, algorithms were improved and new ones were developed making them not only more efficient but also more capable. Second, specialised hardware, especially processors, made for machine learning has been build. Last, the ML community took advantage of scalable high-performance computing and trained their models in parallel, which highly improved training times and computational power. [Pum+19a]
All these improvements increased the performance of the models and the environmental sustainability of the training process, while decreasing the resource requirements such as money, computational time and electrical power.
To get an idea of the scope: Stable Diffusion v1.4 used 256 40 GB Graphics Processing Unit (GPU)s and trained for 150,000 hours in total, which equals to roughly 24 days per GPU. The carbon emitted were equivalent to 11,250kg CO2. [RE22]
While training these big models, they see billions of high resolution images or other forms of training data such as text or sound sequences. These large amounts of training data are necessary to prepare the models for various situations they might encounter during inference. The better the quality, quantity and variety of data, the better the model can be in the end. Unfortunately, the I/O performance is lagging behind when it comes to the improvements made in recent years. Pumma et al. [Pum+19a] found out, that I/O takes up to 90% of the total training time.
An approach to solve this bottleneck are distributed, parallel file systems, such as Lustre [Lus] and BeeGFS [Thi]. These file systems are developed with a focus on their performance for High-Performance Computing (HPC) and especially also Artificial Intel-

[1] Runway is an applied artificial intelligence research company. Link: https://runwayml.com/, accessed on: 2023-11-05

ligence (AI) and Deep Learning [Fra; Thi; Lus]. But there are still difficulties, such as the *small file problem*, which is for example mentioned by Zhu et al. (2020) in their work to optimise I/O performance of Hadoop distributed file systems [Zhu+20]. It is common for Deep Learning that the huge amount of data needed for training consists of many small files instead of bigger, but less files. This creates a bigger overhead, as every image has its own, and decreases the performance of file systems significantly.
These are reasons why it is important to further look into the influence of the file system on the performance of machine learning workloads. In this work it is specifically looked at the I/O performance when reading and writing an image data set, which could be used for training ML image models.

2.2 Distributed, parallel storage file systems

The data in distributed, parallel storage file systems, as the name indicates,is distributed across multiple servers and clients can access their data in parallel, which is called I/O parallelism. The advantage over storing all the data on dedicated servers is the scalability of the performance and capacity. These file systems in combination with HPC clusters enable scientists to do work that would not be feasible at home or in a small office or research lab because of the extensive required resources. The file systems can be adjusted to the required resources by adding more servers when needed. They also provide additional services such as redundancy for host failure security. [Mar18; Fra]
Their structures are generally designed to include a separation of functionality, as can be seen in Figure 1. The actual data lays on so called Object Data Storage (ODS). The information about how the data is distributed is stored on Metadata Server (MDS) units. Clients can access the storage through multiple Access Servers. These servers retrieve the necessary information from the MDS and provide the requested data from the ODS. Access Servers are for security reasons and therefore optional. As they present an additional bottleneck, they are often omitted when the security requirements permit it. For maintenance reasons all servers are interconnected. [Mar18; Fra]

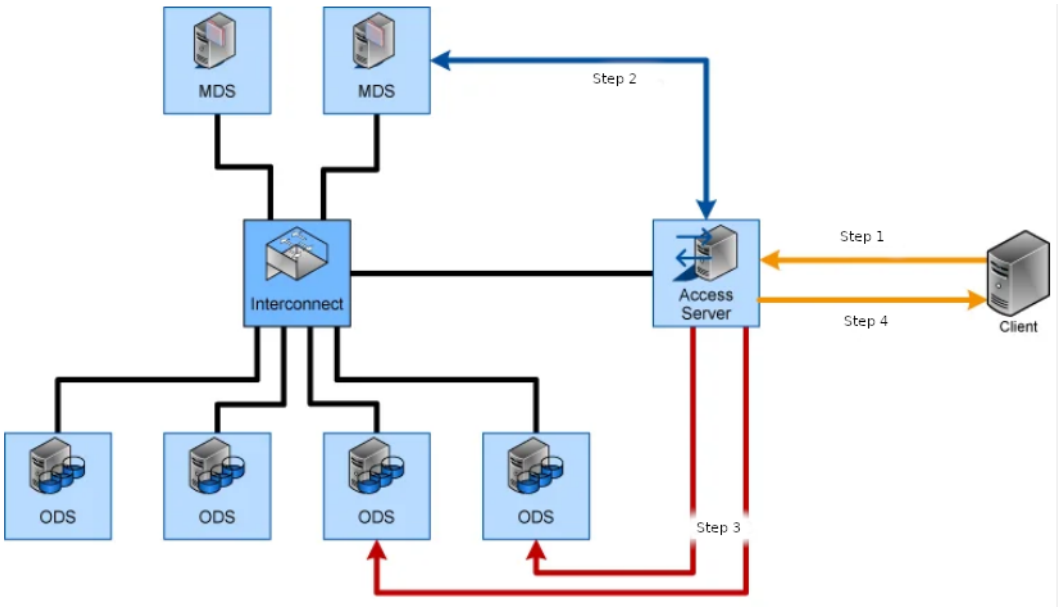


Figure 1: Exemplary storage file system architecture translated from [Mar18].

2.2.1 Lustre

Lustre is an open source parallel file system, which was originally developed by researchers of the Carnegie Mellon University in the United States of America (USA) [Lus]. Lustre supports many of the requirements of today's state-of-the-art HPC simulation environments [Lac+22]. It can be scaled up to thousands of clients, petabytes of storage and hundreds of gigabytes per second of bandwidth. Its architecture consists of the aforementioned Metadata Servers (MDS) and Object Storage Servers (ODS) and Metadata Targets (MDT) and Object Server Targets (OST). Lustre does provide file redundancy. [Lus; Lac+22]
Students at the Georg-August University of Göttingen can access Lustre through the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), which is a member of the National High Performance Computing (NHR) alliance, via NHR@Göttingen [GWDb]. They provide an HPC Central Processing Unit (CPU) cluster called Emmy and a HPC GPU cluster called Grete [GWDc; Lac+22].
In this work only Emmy is used, as the focus lays on the performance of CPU clusters[2].

2.2.2 BeeGFS

BeeGFS is a shared source distributed, parallel file system developed by the Fraunhofer Institute for Industrial Mathematics. It is software-defined, meaning it is independent of hardware. The developers focused on performance, ease of use and simple installation and management. The performance and capacity can be scaled by increasing the number of servers or disks in the system up to thousands of nodes. The separation of functionality is given by including Metadata Servers (MDS) and Storage Servers (ODS), while avoiding architectural bottlenecks. It also provides file redundancy. [Fra; Thi; Lac+22]
Students at the Georg-August University of Göttingen can get access to BeeGFS through the local HPC resource, the Scientific Compute Cluster (SCC), provided by the GWDG. The SCC includes CPU and GPU nodes, but this work only uses the CPU ones. [GWDd]

3 Methods: Project setup

3.1 Data set: Conceptual Captions

To test I/O performance, the machine learning image data set Conceptual Captions [Sha+18] was chosen, as it represents typical I/O intensive training. It consists of about 3.3 Million image-caption pairs, that were harvested from the web by Google AI [tea; aH]. An intended use was to train ML models to solve an image captioning task and therefore learn to describe images, to for example aid visually impaired people. There is a competition ongoing by Google AI [tea] and all the data can be downloaded freely.
This work does not involve training a model. Instead, the data will be used to test the I/O functionality by loading and saving it to and from storage. Only a subset of the provided image data from the validation set is utilised, as the training data set is too extensive for the intended purpose. That subset is called *images_full* and comprises 12,720 images. It has a total size of approximately 8.2 GB on the SCC and 8.4 GB on Emmy.

[2]Based on experience, training ML models usually involves loading the training data onto CPUs and performing preprocessing there before conducting the actual training on GPUs. Therefore, the decision was made to concentrate on the CPU clusters.

For an additional test a subset of all the images over 1 MB from *images_full* is needed. That data set is called *images_large*. It consists of 1,378 images on the SCC and 1,431 on Emmy, with a total size of approximately 3.5 GB on the SCC and 3.7 GB on Emmy. The different sizes of the data sets and the resulting different subsets on the two clusters can presumably be explained by the difference in overhead. The original data set *images_full* is identical on both clusters, while the subsets were created by the same command, but based on the space the images needed on the clusters individually.

3.1.1 Hugging Face

To download the data set, Hugging Face was used. Hugging Face is a collaboration platform for the machine learning community and provides open-source ML libraries, data sets and pre-trained models [Hug].
For the Conceptual Captions data set Hugging Face provides, among other information, the image_url, the caption and labels. A code example to load the data set and fetch the images with the help of the urls is also provided. [aH]
For this work the code was adjusted to only load the needed subset and save the images as png files without labels or caption. It should be noted that the data sets were saved on the `scratch` file systems and not the `home` file systems on the two clusters. This is because the former are the distributed, parallel file systems that are being tested.

3.2 Darshan

Darshan is an open source scalable HPC I/O characterisation tool. It is used for post mortem analysis, which means for analysing the I/O after running the application instead of during the run. While producing minimal overhead, it gives insights to properties such as the elapsed time, the access sizes and pattern and the file names of each file opened by the application. [Kun+18; car09]
The Darshan source tree is divided into the two parts darshan-runtime and darshan-util. The former is used to generate log files about the I/O performance of instrumented applications on HPC clusters. The latter is for analysing these log files and translating them into human readable or graphically processed formats. [Dar]

3.2.1 Installation

Darshan-runtime can be installed with or without MPI support. As no MPI is required for this work, the configure and build example without MPI support was chosen. The darshan-util installation does not differ depending on the MPI support. An adjusted step by step example when working on the SCC based on the official documentation [Dar] for darshan-runtime and darshan-util can be seen in Listing 1.
The initial steps involve downloading the required files and preparing for installation. The highlighted configuration for darshan-runtime is the crucial aspect. The `- -with-log-path` determines where the log files will be saved. This can also be specifically set later when running Darshan. The `- -prefix` is important as it sets the directory in which Darshan should be installed. As can be seen this is also given when configuring darshan-util. As users are not permitted to install software on the cluster without authorisation, this step cannot be omitted. The compiler to be used is specified by the `CC`. After configuration `make & make install` is used to build and install.

```
$ wget https://ftp.mcs.anl.gov/pub/darshan/releases/darshan-3.4.4.tar.gz
$ tar -xvzf darshan-3.4.4.tar.gz
$ cd darshan-3.4.4/
$ ./prepare
$ cd darshan-runtime/
$ ./configure --with-log-path=/darshan-logs
    --with-jobid-env=SLURM_JOB_ID
    --prefix=/scratch/users/username/darshan/ --without-mpi CC=gcc
$ make & make install
$ cd ../darshan-util/
$ configure --prefix=/scratch/users/username/darshan/
$ make & make install
```

Listing 1: Configure and build example of darshan-runtime and -util based on Darshan's documentation [Dar].

3.2.2 Log file usage

Darshan-utils offers various methods for analysing log files generated by darshan-runtime, which are described in their documentation [Dar].
A way to create a "complete, human-readable, text-format dump of all information contained in the log file" is the Darshan-parser [Dar]. The output will be printed on the command line or can be saved to a text file. Darshan-job-summary.pl creates a graphical summary of the I/O activity as a PDF file.
Another way to work with the log files is PyDarshan, a Python package providing interfaces to Darshan log files [Dar]. It requires darshan-util, but can easily be installed via pip as can be seen in Listing 2 [PyP].

```
$ pip install darshan==3.4.4.0
```

Listing 2: Command to install PyDarshan via pip.

3.3 Implementation

On both clusters a mamba environment was set up. Mamba is a fast, robust and cross-platform package manager to handle dependencies without interfering with other environments [Qua20].
The benchmark code for reading and writing the images is written in Python, as Python is a typical programming language when working with ML. Important libraries for this work include time, numpy and skimage.io, which provides the imsave and imread functionality to read and save the images. The Python code is made executable with the `chmod +x` command and the Python interpreter of the mamba environment is specifically stated with the hashbang. A code sample can be seen in Listing 5 in the Appendix in Section C.
For analysing the time stopping results numpy and its functionality to calculate the mean

and standard deviation of arrays was used. The diagrams displayed in Section 4 were generated with matplotlib, a library for data visualisation.
A job and all its necessary information is submitted to Slurm, the workload manager of the clusters, via a batch script such as the one shown in Listing 3 [GWDa]. The memory, partition and maximal run time can be set after `#SBATCH` keywords. The mamba environment is activated in line seven and eight with the `source` keyword. Darshan is linked to the application with the `LD_PRELOAD` and in line ten it is specified that Darshan is run without MPI support.
The GWDG website[3] provides information to assist in selecting the appropriate partition for a given task. The Appendix includes a collection of slurm commands that were found useful for this work, along with example outputs in Section A.

run.sbatch

```
1  #! /bin/bash
2
3  #SBATCH --mem 32G
4  #SBATCH -p medium
5  #SBATCH -t 01:00:00
6
7  source /usr/users/username/.bashrc
8  source activate scap_env
9
10 export DARSHAN_ENABLE_NONMPI=1
11
12 env LD_PRELOAD=/scratch/users/username/darshan/lib/libdarshan.so ./code.py
```

Listing 3: Example batch script to start an application linked to Darshan.

4 Results

4.1 Using time stopping

Firstly, basic exploratory tests were conducted. They consisted of reading and writing the whole *images_full* data set described in Section 3.1 and stopping the needed time. The times were averaged over ten runs. To get a better understanding of the impact different partitions can have, the test was run on different partitions on both clusters for comparison. For both clusters a Cascade Lake partition was available. For Emmy a Skylake and for the SCC a Broadwell partition was chosen as well for the tests.

Figure 2 shows the times of the individual runs on all selected partitions for reading and writing the data. It is evident that although there were some deviations between the runs, both writing and reading tests on the SCC were faster than those on Emmy. It is also noticeable that loading the images had more and higher spikes in time compared to writing.

In Table 1 the average times and the standard deviations of the runs for the tests on the different partitions can be seen for reading and writing. Regardless of the partition the

[3] https://docs.hpc.gwdg.de/compute_partitions/cpu_partitions/index.html

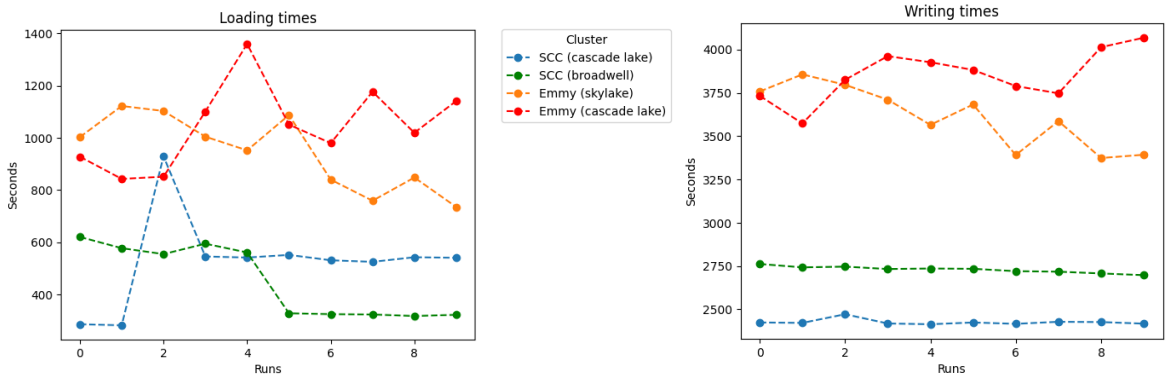


Figure 2: Times for loading and writing roughly 8 GB of image data on different partitions of the SCC and Emmy.

Cluster		SCC		Emmy	
Processor gen.		Broadwell	Cascade Lake	Skylake	Cascade Lake
Reading	avg. times	7.53	8.79	15.75	17.41
	std	2.17	2.80	2.25	2.51
Writing	avg. times	45.48	40.43	60.19	64.20
	std	0.30	0.26	2.82	2.35

Table 1: Times for loading and writing roughly 8 GB of image data on different partitions of the SCC and Emmy averaged over ten runs and the standard deviations of those runs. All times are given in minutes and rounded to two decimals.

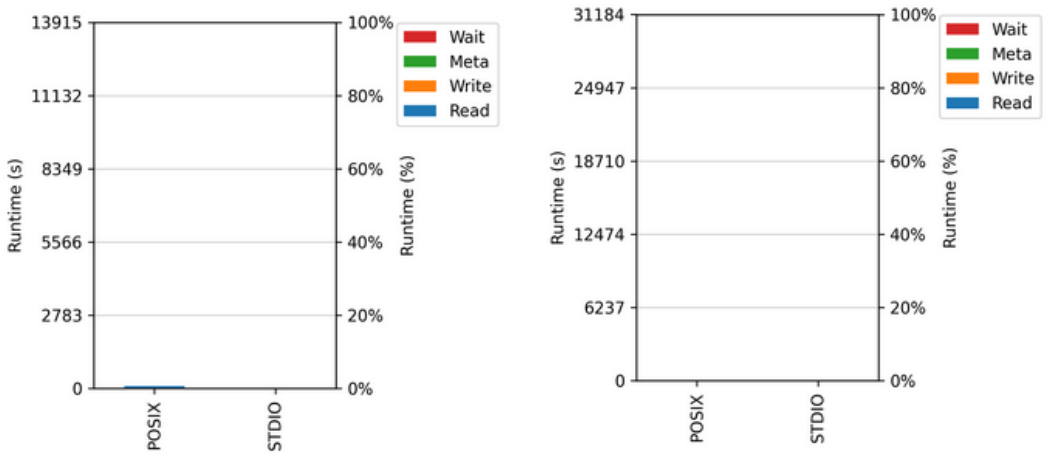
writing times were longer than the reading times. For the SCC, the increase was slightly greater than for Emmy, with the Broadwell and Cascade Lake partition runs taking 6.04 and 4.60 times longer, respectively. For Emmy, the run times for writing were 3.82 and 3.69 times longer for the Skylake and Cascade Lake partitions compared to the reading times. The standard deviation was between two and three minutes for all tests except the writing tests on the SCC. These had standard deviations of less than a third of a minute. When averaging over all runs on both tested partitions on the SCC, the average time for loading and writing are 8.15 and 42.96 minutes. For Emmy the times are 16.58 and 62.20 minutes, which means that the SCC took on average around half the time Emmy needed for loading and a little more than two thirds of Emmy's time for writing.
The same test was run on Cascade Lake partitions on both clusters with the second data subset *images_large*. The result graph can be seen in the Appendix in Section B in Figure 5. The average times and standard deviations can be seen as well in the Appendix in Section B in Table 3. While Emmy took, as expected, longer for the writing, it was surprisingly around four seconds faster than the SCC for the reading. But as can be seen on the graph, there was variation and in four of the ten runs Emmy actually took longer than the SCC. It still means an improvement compared the the SCC being significantly faster before.
The data rates of the tests using Cascade Lake partitions on both clusters with both data sets can be seen in Table 2. For reading both the SCC's and Emmy's data rates improved. Especially Emmy improved by around double as many MB per second compared to the full data set. For writing the improvement was not as significant for Emmy. For the SCC, the data rate decreased slightly for the writing, while it increased slightly for the reading.

Cluster		SCC	Emmy
Processor gen.		Cascade Lake	Cascade Lake
Reading	*images_full*	15.56	8.04
	images_large	17.39	18.72
Writing	*images_full*	3.38	2.18
	images_large	2.96	2.26

Table 2: Data rates for loading and writing the *images_full* and *images_large* image data sets on Cascade Lake partitions of the SCC and Emmy averaged over ten runs. All rates are given in MB per second and rounded to two decimals.

4.2 Using Darshan

The Darshan log files did not yield the intended outcomes. The process of getting Darshan to run on the clusters and linking it to the application took several weeks. It resulted in the instruction described in Section 3. When running the tests with both data sets *images_full* and *images_large* on the SCC, the produced log files looked very similar. Neither of them showed the I/O that would be expected when reading and writing 8.2 GB or 3.5 GB of image data.
The I/O performance estimate was an average of 33.58 Mebibyte(s) (MiB)[4] per second for the test with *images_full* and 8.23 MiB per second for the one with *images_large*.
In Figure 3 the I/O cost of both test runs logged by Darshan is shown. The diagram should display the average amount of run time that each process spent performing I/O, broken down by access type. There are no significant I/O costs displayed.



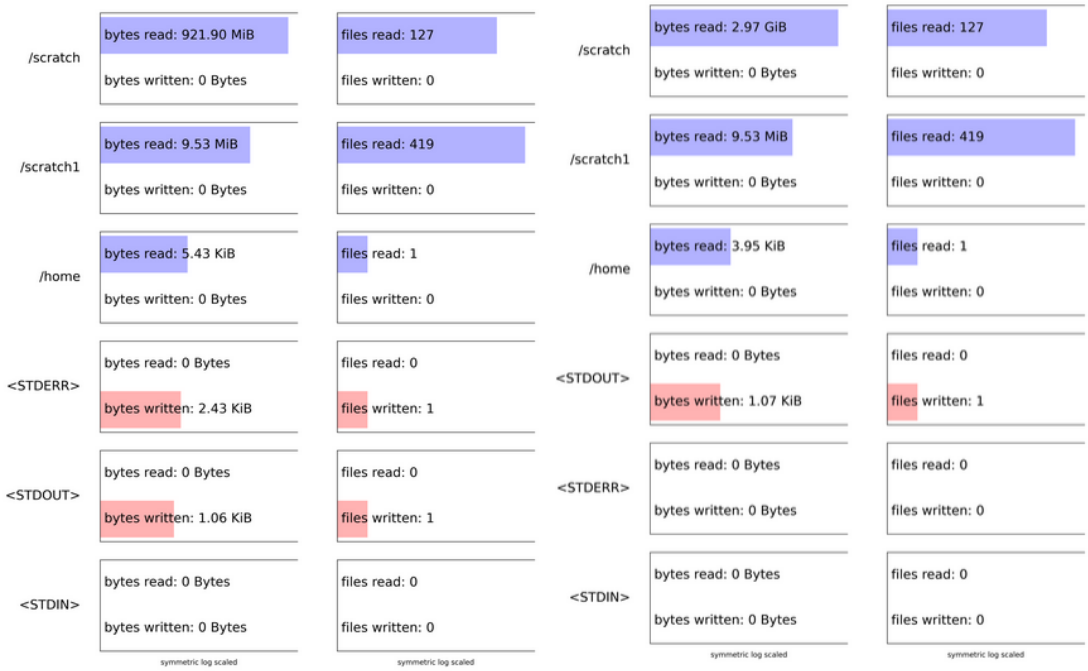
(a) *images_full* (b) *images_large*

Figure 3: I/O cost given by Darshan when running the tests with the two different data sets *images_full* and *images_large*.

As can be seen in Figure 4 the data access by category for both clusters is nearly identical when it comes to the amount of files read or written, even though the sizes differ in some cases. The Standard Error (STDERR) file for the full data set presumably addresses warnings about low resolution images, which are not included in *images_large*. The data

[4]One MiB equals 1.048576 MB.

shown in the figure clearly does not include the test image data. It would have been in the categories `scratch` or `scratch1` and the amount of files would have differed between the two test runs.



(a) *images_full* (b) *images_large*

Figure 4: Data access by category given by Darshan when running the tests with the two different data sets *images_full* and *images_large*.

For both tests the error message shown in Listing 4 was included in the Darshan-parser output.

```
# *ERROR*: The POSIX module contains incomplete data!
#            This happens when a module runs out of
#            memory to store new record data.
```

Listing 4: Error message included in Darshan-parser output when running the tests with the two different data sets *images_full* and *images_large*.

5 Discussion

Although using file systems such as Lustre or BeeGFS takes longer than just using a personal file system at home without the overhead of the server access, they are necessary for HPC. The scalability and performance they offer are essential for training large ML models. The improvement of performance therefore remains an important and interesting topic for both science and industry.
In this work some exploratory tests were made to see some of the influence the two file

systems Lustre and BeeGFS can have on machine learning workloads. First, the two data sets *images_full* and *images_large* were read and written while stopping the times. The results are described in Section 4.
The first thing to note is that it seems like the partitions do not significantly influence the I/O performance in this works testing scope. As Broadwell is an older version than Skylake, which itself is older than Cascade Lake, tests run on the partitions with the older versions should supposedly take longer. However, this was not supported by the time stopping results.
In the tests with the *images_full* data set, it was consistently observed that Emmy with Lustre performed worse than SCC with BeeGFS in terms of reading and writing, regardless of the partition. These tests were not extensive enough to be considered reliable, but the trend is noticeable. Further work could be done to reproduce and understand these results.
It is interesting to note that the standard deviation was not higher for the writing times, even though the writing took longer, which would suggest more opportunities for deviations. An explanation for the variation of times in general could be the overall workload of the clusters changing over time or potentially sharing the compute nodes.
When comparing the writing and reading times of the tests with each other, the SCC's writing times are increasing more compared to its reading times than for Emmy. The performance gap in reading and writing between the two clusters may indicate that the SCC loads faster, resulting in a greater difference between loading and writing times compared to Emmy. Alternatively, Emmy's performance may not decrease as much for writing as it does for the SCC.
As the runs when testing with the *images_large* data set were quite short, the time stopping does not show as clear results. It is unclear which cluster performs better during the loading tests due to the variation, but it is noticeable that the SCC is no longer clearly faster than Emmy. The writing results are similar to those obtained when testing with the full data set and therefore more expected.
The second test to compare data rates when the file systems had to deal with larger images instead of the larger images in addition to the small ones, was to see whether the kind of data visibly influences the I/O performance in this works set up.
Emmy's improvement with the data rate for reading indicates that Lustre might be visibly better in reading large images compared to small ones in this set up. The other data rates are too similar to really draw any conclusions. This indicates that the test set up is not extensive enough for good results, as the research is quite clear on large images being better as they create less overhead, as it is mentioned in Section 2.1.
As became evident, the time stopping results are based on very limited tests. Averaging over ten runs seems not to be enough with all the possible causes for variation and noise. Some trends, such as Emmy being slower than the SCC, seem to be more clear than others. It is important to keep in mind, that the tests were meant as an exploration to get an overview. A deeper look is needed to understand what really affects I/O performance and how it does so. This work is not meant as a guideline to which file system is better.
One of the reasons a file system might perform badly is when it is filled to over 85% of its capacity and degrading rapidly when being filled even more. Warnings were sent for both clusters regarding overfilling, with some reports indicating that they were filled to over 95% capacity. It is not clear how full they were during the tests run in this work and how that exactly influenced their performance. For a better understanding the tests should be run again while monitoring the filling level.

The benchmark code was checked multiple times, but it can not be ruled out that the code itself is flawed, which could lead to the surprisingly long testing times. As both clusters were tested with the same code the comparability of the clusters should not have been weakened by this.
As the image data files are not shown in Darshan's analysis, there must have been an error when logging the application. When looking at the Darshan-parser output an error message indicates, that the POSIX module contains incomplete data, even though it should have had enough memory to record all the data. It is unclear what exactly was at fault.
The data access that is shown by Darshan presumably represents the overhead when running the test application.
Darshan's performance estimate differed significantly between the two test runs. This might indicate a high variation between test runs or could be explained by the error when logging the application.

6 Conclusion

This work has shown some comprehensible steps to analyse the influence of the file system when it comes to the I/O performance regarding ML workloads. That included the introduction and usage of the file systems Lustre and BeeGFS and the characterisation tool Darshan.
Some trends, such as that Lustre seems to be performing worse in the tested work set up, and some possible explanations, such as the performance decrease when reaching a certain filling capacity, were elaborated. As this work aims to provide an overview and a possible approach to the problem, further benchmarking is necessary for validation.
Future work should focus on improving the benchmark code and correctly binding it to Darshan. This will ensure the production of accurate log files that can be analysed in detail.
Additionally to the possibilities to focus deeper into the topics mentioned in Section 5, future work could also include testing GPUDirect storage. With GPUDirect storage the data can be loaded directly onto the GPU and therefore forgoes the additional step of loading it onto a CPU and preprocessing it there. This solution has shown benefits performance wise, as Newburn et al. state in their article about accelerating I/O in the Modern Data Center [CJ 21].

References

[aH] @abhishekkrthakur and @mariosasko from Hugging Face. "Datasets: conceptual$_c$*aptions*". In: (). URL: https://huggingface.co/datasets/conceptual_captions (visited on 03/18/2024).

[Cal20] E. Callaway. "'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures". In: *Nature 2020, vol. 588* (2020). DOI: 10.1038/d41586-020-03348-4.

[car09] carns (darshan team). "Darshan project". In: (July 31, 2009). URL: https://www.mcs.anl.gov/research/projects/darshan/ (visited on 03/05/2024).

[CJ 21] CJ Newburn and Kiran K. Modukuri and Kushal Datta. "Accelerating IO in the Modern Data Center: Magnum IO Storage". In: (2021). URL: https://developer.nvidia.com/blog/accelerating-io-in-the-modern-data-center-magnum-io-storage/ (visited on 03/27/2024).

[Dar] Darshan team. "Darshan documentation". In: (). URL: https://www.mcs.anl.gov/research/projects/darshan/documentation/ (visited on 02/01/2024).

[Eis21] M. Eisenstein. "Artificial intelligence powers protein-folding predictions". In: *Nature 2021, vol. 599* (2021). DOI: 10.1038/d41586-021-03499-y.

[Fra] Fraunhofer Institute for Industrial Mathematics ITWM. "Fraunhofer Parallel File System – BeeGFS". In: (). URL: https://www.itwm.fraunhofer.de/en/departments/hpc/fraunhofer-parallel-file-system-beegfs.html (visited on 03/18/2024).

[GWDa] GWDG. "Running Jobs with Slurm". In: (). URL: https://docs.gwdg.de/doku.php?id=en:services:application_services:high_performance_computing:running_jobs_slurm (visited on 03/25/2024).

[GWDb] GWDG NHR-NORD@Göttingen Team. "NHR-NORD@Göttingen intro". In: (). URL: https://gwdg.de/community-pages/nhr-intro/ (visited on 03/26/2024).

[GWDc] GWDG NHR-NORD@Göttingen Team. "NHR-NORD@Göttingen Systeme "Emmy" und "Grete"". In: (). URL: https://gwdg.de/hpc/systems/emmy/ (visited on 03/26/2024).

[GWDd] GWDG Team. "Scientific Compute Cluster (SCC)". In: (). URL: https://gwdg.de/hpc/systems/scc/ (visited on 03/26/2024).

[Hug] Hugging Face. "Hugginf Face Bio". In: (). URL: https://huggingface.co/brand (visited on 03/18/2024).

[Kun+18] Julian Martin Kunkel et al. "Tools for analyzing parallel I/O". In: *High Performance Computing: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers 33*. Springer. 2018, pp. 49–70.

[Lac+22] Nellie Marie Lackschewitz et al. "Performance Evaluation of Object Storages (NHR2022)". In: (2022).

[LRT04] Rob Latham, Rob Ross, and Rajeev Thakur. "The impact of file systems on MPI-IO scalability". In: *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer. 2004, pp. 87–96.

[Lus] Lustre Team. "About the Lustre® File System". In: (). URL: https://www.lustre.org/about/ (visited on 03/26/2024).

[Mar18] Markus Ermes. "DISTRIBUTED, PARALLEL FILE SYSTEMS (translated)". In: (Dec. 3, 2018). URL: https://www.comconsult.com/hochleistungs-dateisysteme/ (visited on 03/18/2024).

[Ope22] OpenAI. "Introducing ChatGPT". In: (2022). URL: https://openai.com/blog/chatgpt (visited on 03/18/2024).

[Pum+19a] Sarunya Pumma et al. "Scalable Deep Learning via I/O Analysis and Optimization". In: *ACM Trans. Parallel Comput.* 6.2 (July 2019). ISSN: 2329-4949. DOI: 10.1145/3331526. URL: https://doi.org/10.1145/3331526.

[Pum+19b] Sarunya Pumma et al. "Scalable deep learning via I/O analysis and optimization". In: *ACM Transactions on Parallel Computing (TOPC)* 6.2 (2019), pp. 1–34.

[PyP] PyPI. "PyDarshan Documentation". In: (). URL: https://pypi.org/project/darshan/3.4.0.0/ (visited on 03/05/2024).

[Qua20] QuantStack mamba contributor. "Mamba's documentation". In: (2020). URL: https://mamba.readthedocs.io/en/latest/ (visited on 03/25/2024).

[RE22] Robin Rombach and Patrick Esser. "Stable Diffusion v1-4 Model Card". In: (2022). URL: https://huggingface.co/CompVis/stable-diffusion-v1-4 (visited on 10/31/2023).

[Rom+22] Robin Rombach et al. "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 10684–10695.

[Sch] SchedMD. "Slurm documentation". In: (). URL: https://slurm.schedmd.com/documentation.html (visited on 02/01/2024).

[Sha+18] Piyush Sharma et al. "Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning". In: *Proceedings of ACL*. 2018.

[Tea] Stable Diffusion Team. "Stable Diffusion Online". In: (). URL: https://stablediffusionweb.com/ (visited on 03/18/2024).

[tea] Google AI Language team. "Google's Conceptual Captions". In: (). URL: https://ai.google.com/research/ConceptualCaptions/ (visited on 03/18/2024).

[tea20] AlphaFold team. "AlphaFold: a solution to a 50-year-old grand challenge in biology". In: (2020). URL: https://www.deepmind.com/blog/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology (visited on 03/18/2024).

[Thi] ThinkParQ Team. "BeeGFS: The leading parallel file system". In: (). URL: https://www.beegfs.io/c/ (visited on 03/26/2024).

[Zhu+20] Zongwei Zhu et al. "PHDFS: Optimizing I/O performance of HDFS in deep learning cloud computing platform". In: *Journal of Systems Architecture* 109 (2020), p. 101810. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2020.101810. URL: https://www.sciencedirect.com/science/article/pii/S1383762120301028.

A Useful slurm commands

A collection of slurm commands found useful for this work and example outputs is given below. They are based on the official slurm documentation, which can also be referred to for further information [Sch].

- Submit batch script to start job

```
$ sbatch run.sbatch
```

- Review scheduled jobs

```
$ squeue -u username
  JOBID PARTITION       NAME     USER   STATE TIME NODES NODELIST(REASON)
5460973    medium run.sbatch username RUNNING 0:30     1 amp029
```

- Cancel a job with `JOBID`

```
$ scancel 5460973
```

- Review available partitions and nodes

```
$ sinfo
PARTITION AVAIL  TIMELIMIT NODES STATE NODELIST
medium*      up 2-00:00:00     1  idle amp025
```

B Results with *images_large*

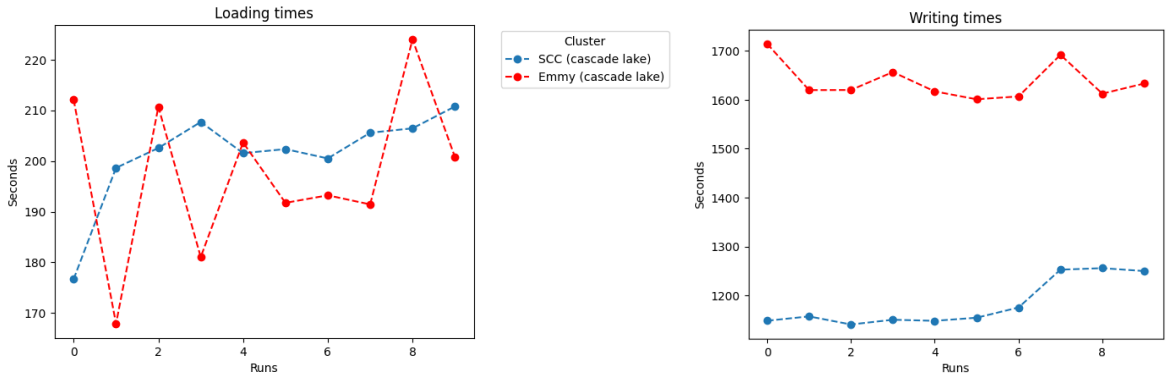


Figure 5: Times for loading and writing roughly 3.6GB of image data on cascade lake partitions of the SCC and Emmy. All images are >= 1MB.

Cluster		SCC	Emmy
Processor gen.		cascade lake	cascade lake
Reading	avg. times	3.36	3.29
	std	0.15	0.26
Writing	avg. times	19.72	27.29
	std	0.30	0.61

Table 3: Times for loading and writing roughly 3.6GB of image data on cascade lake partitions of the SCC and Emmy averaged over 10 runs and the standard deviations of those runs. All times are given in minutes and rounded to two decimals.

C Code samples

io_tests.py

```python
#!/scratch1/users/username/mambaforge/envs/scap_env/bin/python

import os
from glob import glob
from skimage.io import imsave, imread
import time
import numpy as np
import PIL.Image

# get image paths
image_paths = glob(os.path.join("/scratch/users/username/scap/" +
                    "conceptual_captions_data/images_full", "*.png"))
image_paths.sort()
print("Number of images:", len(image_paths))

times = []

for i in range(10): # ADJUST
    start_time = time.time()
    images = [] # reset image list

    # read all images
    for im in image_paths:
        image = imread(im)
        images.append(image)

    time_loading = (time.time() - start_time)
    print("loading_used_time: %s seconds" % time_loading)
    start_time = time.time()

    # save all images
    for i, im in enumerate(images):
        imsave("/scratch/users/username/scap/" +
               "conceptual_captions_data/images_save/%d.png" % i, im)

    time_saving = (time.time() - start_time)
    #print("start_time loading: ", start_time)
    print("saving_used_time: %s seconds" % time_saving)

    times.append([time_loading, time_saving])

np_times = np.array(times)
mean_loading, mean_saving = np.mean(np_times, axis=0)
print("Average time loading\t ", mean_loading,
      "\nAverage time saving\t", mean_saving)
```

Listing 5: Example python code to read and write images while stopping the time.