Understanding metadata latency with MDWorkbench

Understanding Metadata Latency with MDWorkbench

Julian M. Kunkel[1] and George S. Markomanolis[2]

[1] University of Reading, j.m.kunkel@reading.ac.uk
[2] KAUST Supercomputing Laboratory

Abstract. While parallel file systems often satisfy the need of applications with bulk synchronous I/O, they lack capabilities of dealing with metadata intense workloads. Typically, in procurements, the focus lies on the aggregated metadata throughput using the MDTest benchmark[3]. However, metadata performance is crucial for interactive use. Metadata benchmarks involve even more parameters compared to I/O benchmarks. There are several aspects that are currently uncovered and, therefore, not in the focus of vendors to investigate. Particularly, response latency and interactive workloads operating on a working set of data. The lack of capabilities from file systems can be observed when looking at the IO-500 list, where metadata performance between best and worst system does not differ significantly.
In this paper, we introduce a new benchmark called MDWorkbench which generates a reproducible workload emulating many concurrent users or – in an alternative view – queuing systems. This benchmark provides a detailed latency profile, overcomes caching issues, and provides a method to assess the quality of the observed throughput. We evaluate the benchmark on state-of-the-art parallel file systems with GPFS (IBM Spectrum Scale), Lustre, Cray's Datawarp, and DDN IME, and conclude that we can reveal characteristics that could not be identified before.

1 Introduction

The benchmarking of storage systems fosters understanding of performance behavior in order to identify performance bottlenecks in the storage landscape and to tune applications towards the capabilities of the systems. The benchmarking of metadata operations of a parallel file system characterizes the performance when creating, accessing and searching directories and files. Typical user workloads on HPC systems include the bulk creation of output files from a parallel application, intensive I/O operations on a single file, the post-processing of a subset of files, and the interactive navigation/searching for relevant output files. Since HPC file systems are designed to deal with large files, they typically do not perform well on workloads involving many small files. Such workloads are, however, common and not the exception. This pattern emerges, for example, when home directories and software packages are stored on the shared file system, as

[3] https://www.vi4io.org/tools/benchmarks/mdtest

The final publication is available at Springer via https://doi.org/10.1007/978-3-030-02465-9_5

some sites manage them. Container solutions also suffer from metadata issues. Additionally, some domain-specific workflows, e.g., from genomics involve many small files. This lack of performance has been finally recognized by the vendors and, for example, in the US CORAL pre-exascale initiative, IBM improved the metadata performance of Spectrum Scale (GPFS) significantly. However, there is still the need to drive the effort in the right direction.

Procurement procedures for HPC systems cover metadata requirements often by declaring a certain minimal throughput when running the MDTest benchmark. However, deployed systems still lack mandatory requirements of the users. For example, access to files is sensitive to latency; waiting up to 10 seconds to list 100 files or touch a new file is hindering the interactive data navigation and preparation of scientific software. The high latency is also apparent when hosting software on a shared file system. To overcome the metadata bottleneck, data centers implement workarounds like storing a container or a local file system image on the parallel file system, for example, by serving the software tree as a (read-only) EXT4 image on a Lustre file system. An image can then be mounted (typically read-only) on the clients turning the metadata workload into a well-aligned data workload for the parallel file system. The container solution Singularity[4] uses this approach, too. However, concurrent modifications to such image from multiple nodes are not possible for consistency reasons. Unfortunately, the burst buffer solution IME from DDN and Cray's DataWarp utilize the metadata infrastructure from the backend file system depending on the underlying file system's capabilities.

Besides latency, the phase oriented execution of MDTest leads to several shortcomings: First of all, a production file system will rarely see a phase oriented pattern ever in practice because typically several jobs run concurrently (acceptance testing is an exception). A phase execution of, e.g., file creates or stats in isolation, however, may lead to a different performance behavior as some resources of the parallel file system are not used within a phase, and internal locks are often not utilized. For example, in Lustre, the creation of files does mostly affect the metadata server – the MDT pre-creates a pool of zero-length objects. Moreover, bulk synchronous workloads of many benchmarks are well optimized even for disk-based storage technology, allowing benchmarks to report metrics that are artificially high and not achievable in any production environment as background processes cause interference. Additionally, a storage system that offers separated path for the individual metadata operations (which is desirable) has a disadvantage when performance of each phase is measured individually. Depending on the optimization in place, e.g., the caching strategy, results of a benchmark may vary significantly.

The contributions of this paper are: the introduction of the MDWorkbench benchmark to reveal relevant characteristics of storage systems; and the investigation of latency factors on existing parallel file systems.

This paper is organized as follows: First, related work is provided in Section 2. Next, the benchmark MDWorkbench is introduced together with its workload

[4] `http://singularity.lbl.gov/`

description in Section 3. In the evaluation (Section 4), we first describe the experimental setup together with four different platforms where the evaluation of the benchmark is conducted, and then present and discuss the results in Section 5. Finally, the paper is wrapped up in Section 6.

2 Related Work

The main reasons for benchmarking a computer system is to understand its performance behavior and compare it with other systems. From the user perspective, benchmarks are of interest that are representative for the applications run on a system. The importance of analyzing and improving metadata performance is illustrated in various studies, for example, in [1] and [2].

In [5], standards for building a good benchmark are discussed. According to Huppler, any good benchmark should have the most of the following criteria: to be relevant, i.e., represent something important, to be repeatable – that the benchmark can deliver same results across various executions, to be fair – not to be optimized for specific platform only, to be able to verify the output, and to be affordable for any user. In [7], an I/O system evaluation is proposed that takes into account both application requirements and the I/O configuration. The methodology is constituted of three phases: In the first one, the characterization of the application I/O requirements is extracted by measuring bandwidth and I/O operations per second (IOPs). In the second phase, the I/O configuration analysis, factors that impact the I/O system performance are identified, such as file system and I/O node connection. Finally, important metrics of application execution are collected in the evaluation phase under different configurations.

Several metadata benchmarks have been used to investigate parallel file systems. MDTest is the most common MPI benchmark for evaluating the metadata performance of a file system. The benchmark runs in phases, measuring bulk-synchronous workloads. It can be executed on any POSIX-compliant file system; the newest version is integrated into the IOR benchmarking suite supporting additional interfaces. The benchmark can create various workloads, file-only tests, and nested directory trees. Fs_mark[5] is a multi-threaded benchmark for measuring concurrent activity on a POSIX file system, e.g., mimicking a mail server. It measures the time for each operation individually and reports an average throughput for each operation type. PostMark [6] is similar but primarily aims to simulate workload of a mail server by utilizing a stochastic model for the operations like creating, reading and deleting of files.

In the web server market, for OLTP workloads and low-latency applications like banks, latency is very important as, e.g., customers tend to leave slow responding web pages. Therefore, the Storage Performance Council[6] designed the SPC-1 benchmark [8]. This specification defines a workload, benchmark, and a test methodology to ensure reproducible results. A standardized workload is

[5] `https://sourceforge.net/projects/fsmark/`
[6] `http://www.storageperformance.org/`

modeled based on real-world applications and the benchmark measures and reports response time and IOPS that are crucial. However, the benchmark is designed to measure a single storage system connected to a client; it is not applicable to an HPC environment.

The Yahoo Cloud Serving Benchmark (YCSB) is widely used to measure the response time of a web application on top of a NoSQL interface offering CRUD operations. YCSB varies the number of requests issued concurrently and measures the latency. However, when responses take unexpectedly long, the threads are unable to create the necessary request rate. As a consequence, it typically reports a too optimistic value. The issue of stragglers hiding latency issues has been well summarized by Schuetze from Azul systems in a presentation[7]. The NoSQLMark [3] is an extended version of the YCSB intended to fix this issue.

3 MDWorkbench

This MPI parallel benchmark[8] mimics the concurrent access to typically small objects. It comes with the following features: deterministic access pattern mimicking interactive users or producer/consumer queues; configurable working set size to fit into a cache of a given size or exceed it; performance regression testing by preserving the working set between runs; support for various storage backends (POSIX, MPI-IO, S3, MongoDB, PostgreSQL), and report of throughput and latency statistics including timing individual I/O operations. Since the benchmark supports object storage, the naming conventions for the pattern are datasets (directories in POSIX) and objects (files in POSIX).

The benchmark executes three phases: precreation, benchmark, and cleanup. The precreation phase setups the working set and the cleanup phase removes it. A precreated 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. The pattern is parameterized with the following variables:

N: The number of MPI processes
D: Working set size: number of datasets to create per process
P: Working set size: number of objects to create per dataset
I: Benchmarking phase – iterations to perform per dataset
S: Size per object
O: Offset in ranks between writer and reader

For the ease of understanding, the created working set is illustrated as directory tree in Figure 2. During the precreation phase the structure is created; each

[7] `https://www.azul.com/files/HowNotToMeasureLatency_LLSummit_NYC_12Nov2013.pdf`
[8] MDWorkbench is available under: `https://github.com/JulianKunkel/md-workbench`

```
for(p=0; p < P; p++){
  for(d=0; d < D; d++){
    dataset = (rank, d) // POSIX namespace is: rank/d/p
    write(dataset, p)
  }
}
```

Fig. 1: Pseudocode: Creation phase

process generates a directory labeled with its rank and with D datasets, each of the datasets is populated with P objects, each of size S. On object storage, the rank directory is virtual – the D datasets are typically prefixed by the rank. The objects are created in order of their number (0 to P-1), see Figure 1. After the benchmarking phase, each process is responsible for cleanup its datasets. Since I objects have been created per dataset in the meantime, the offset of these objects is shifted by I (see the files in Figure 2).

In the benchmarking phase, each process iterates over $D \cdot I$ objects as described in Figure 3. In each iteration, file information from a previously created object is fetched[9], then the object is retrieved and deleted. Then a new object is created. A process reads from datasets created by the D ranks before him and writes to those D ranks after him appending a new object. Objects of a rank's initially created dataset are accessed by $2D$ other ranks. It is assumed that $N > D \cdot O$ to prevent that datasets previously created by a rank are accessed later by itself, but users may test this pattern intentionally.

The outer loop iterates over the object number, thus, if all processes proceed at the same speed, a FIFO pattern of object accesses emerges. Since the suc-

[9] A backend like MPI-IO may implement this operation as NoOp if it is not supported.

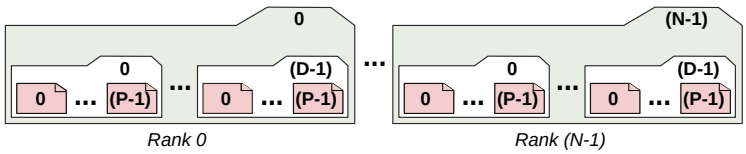


Fig. 2: The content of the working set (directory tree) after pre-creation phase, the ranks responsible for creating/deletion are indicated on the datasets. After benchmark phase all files (red boxes) are offset by I

```
for(i=0; i < I; i++){
  for(d=0; d < D; d++){
      // The "rank" directories owning the files
    read_rank = (rank - O * (d+1)) % N
    write_rank = (rank + O * (d+1)) % N
      // Access previously created data in FIFO order
    dataset = (read_rank, d)
    stat(dataset, i)
    read(dataset, i)
    delete(dataset, i)
      // Append new data to increase the working set size
    dataset = (write_rank, d)
    write(dataset, P + i)
  }
}
```

Fig. 3: Pseudocode: Benchmarking phase

cessful run of the benchmarking phase, the working set looks identical besides that all objects are shifted by I, the benchmark run can be repeated on the same working set simply using an offset to the object number. The pattern can be interpreted as N users working concurrently on shared project directories or as a producer/consumer system where each process retrieves jobs from D waiting queues, processes them and issues them into D further queues.

The benchmark supports an additional *waiting factor* (throttling) parameter $T \geq 0$, that forces a process to wait T times the observed runtime after each operation. Thus, it simulates a processing time and reduces the number of submitted requests; a waiting factor of 1 effectively means that a process performs I/O only half the time. The benchmark can be executed in an adaptive mode that uses this feature several times that scans various waiting factors allowing to investigate the latency of storage depending on the number of concurrent requests (not discussed in this paper). As we will see, this allows identifying overloaded servers.

4 Experimental Setup

Benchmark configuration: The MDWorkbench is executed from a script where several runs are executed individually: 1. Precreation of the working set; 2. Benchmarking phase; 3. Benchmarking phase (this one serves the purpose to verify the results); 4. Benchmarking phase in read-only mode; 5. Cleanup phase. Each benchmark run repeats the measurement three times with a file size of 3901 bytes[10] and reports the individual latencies of each operation. We vary: processes per node (PPN=1 or 10), the number of datasets (D=1 or 10), and the working set size per process (either 10k objects precreated and 2k objects for the benchmarking phase or 1k and 200 objects). Most tests were executed with 10 nodes as these typically suffice to extract near peak performance. In all cases, we run during production, so we cannot exclude interference of concurrent jobs in individual measurements. However, since each measurement is repeated 3 times and the benchmarking phase is repeated several times, outliers can be identified.

Cooley at ACLF: Cooley is the visualization cluster of Mira on the Argonne Leadership Computing Facility. It provides 126 nodes with two Intel Haswell processors (E5-2620v3 @2.4 GHz). Interconnected with FDR Infiniband, the GPFS (v3.5.0) home directory is used.

Mistral at DKRZ: The supercomputer Mistral 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. DKRZ's Lustre version is based on a Seagate fork of Lustre 2.5 with several backports from 2.7. We harness DNE phase I, thus have several metadata servers. However, in

[10] This value is used in the IO-500 benchmark as it prevents inode stuffing; for comparison, we choose it.

the experiments, we use only one metadata server for comparison reasons. The directories are configured with a stripe count of 1; data is stored on 62 ClusterStor L300 appliances providing 124 OSTs. The nodes used for the testing are equipped with two Intel Broadwell processors (E5-2695v4 @2.1 GHz).

IME at Dsseldorf: DDN provided access to their test cluster in Dsseldorf on which 8 nodes could be used for testing. Each node is equipped with two Sandy Bridge processors (8 cores, E5-2650v2 @2.60GHz). They are interconnected with a Mellanox Connect-X-3 card providing 100 Gb/sec (4x EDR). As storage, a DDN ES14K (Exascale 3.1) with 2 metadata servers and Lustre 2.7.19.12 is provided; the storage is complemented by an IME system consisting of 4 servers. The flash native data cache of IME acts as a burst buffer and is drained to the ES14K Lustre, performance reported with IOR is 85 GB/s in write mode. In the conducted tests, IME is used via its FUSE mount.

Systems at KAUST and NERSC: Both KAUST Supercomputing Laboratory and NERSC provide access to Cray XC-40 supercomputers, called Shaheen II [4] and Cori, respectively. Both systems deploy Lustre and Cray DataWarp (DW) technologies. Also, a Sonexion 2000 is installed with a peak performance of 500 GB/s and 700 GB/s of throughput for KAUST and Cori, respectively. Shaheen II is constituted of 268 DW nodes, amounting to 1.5 PB capacity and peak performance 1.6 TB/s, while Cori uses 288 DW nodes with 1.6 PB and 1.7 TB/s peak performance. In both installations, each of DW nodes is constituted by 2 Intel P3608 SSDs. On Cori the latest Cray Linux Environment (CLE v6.0.4) is installed, while Shaheen uses CLE v5.2; this comparison is relevant for us.

5 Results

5.1 Impact of concurrent execution of several metadata operations

This experiment fosters the understanding of the impact when running different operations concurrently vs. the execution in phases. Additionally, it demonstrates how fine grained measurements provide insights into the understanding of behavior. Therefore, we analyze the performance when bulk is creating the files, i.e., starting at an empty directory tree, each process generates 10k files in an explicitly owned private directory. Additionally, the creation rate during the mixed workload of the benchmarking phase is computed with maximum performance and one with a waiting factor of 4. The throughput as computed by overall benchmark runtime is shown in Table 1. Note that during the benchmark phase not only files are created but also read, delete, and their information is queried using stat, so the observed metadata throughput of this mixed workload is 4x the creation rate listed in the table. With a waiting factor of 4, after each operation, we wait 4x the execution time. This, in essence, throttles the load on the metadata server to 1/5, i.e., instead of 100 processes issuing operations, roughly 20 issue requests at any time. Therefore, we compute a corrected (virtual) creation rate by multiplying the measured creation rate by 5.

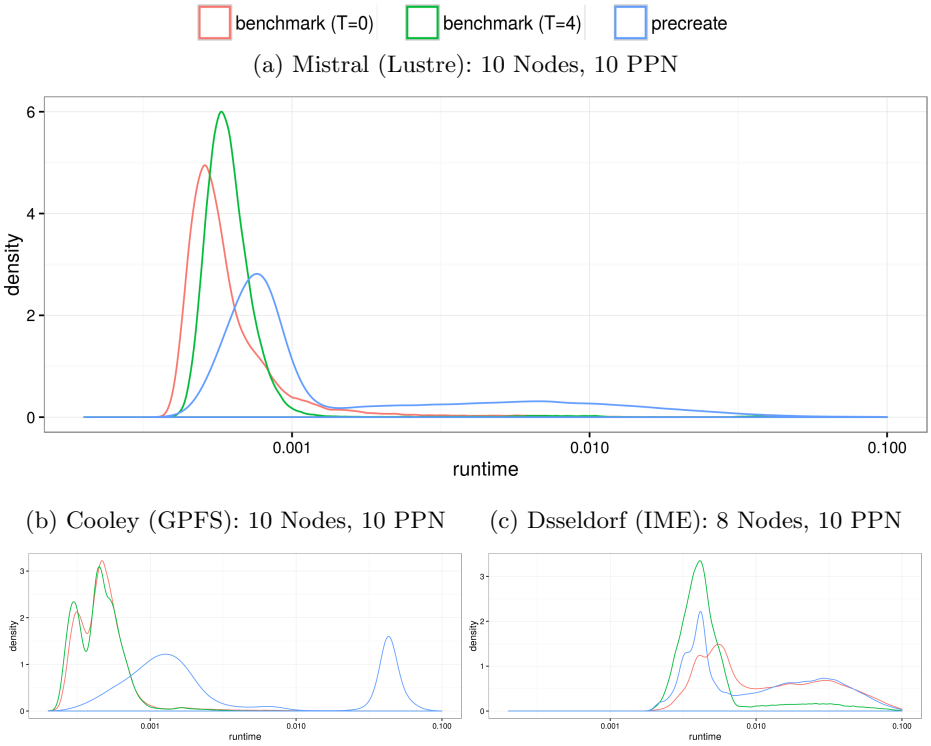
System	Nodes	PPN	D	Creation rate (creates/s) Precreate	Benchm. T=0	Benchm. T=4
ALCF Cooley (GPFS)	10	10	1	6,500	5,640	8,300
Dsseldorf (Lustre)	8	10	1	47,600	12,600	30,700
Dsseldorf (IME+Lustre)	8	10	1	4,500	1,550	4,460
DKRZ Mistral (Lustre)	10	10	1	21,800	2,380	2,220
KAUST (1 DataWarp BB)	10	10	1	3,800	3,390	14,600
KAUST (8 DataWarp BB)	10	10	1	25,600	8,190	32,000
NERSC (8 DataWarp BB)	10	10	1	19,000	8,560	35,100

Table 1: Aggregated performance comparing precreation and benchmarking phase. Benchmarking phase using a waiting factor T of 0 or 4.

From the table, it can be observed that for Lustre based systems the performance of the benchmarking phase with T=0 is $< 1/4$ precreation phase, where for Mistral it drops to 10%. KAUST with one DataWarp node is not able to provide enough metadata throughput while 8 nodes do. The GPFS on Cooley does not suffer much during the benchmarking phase and, thus, allows overlapping the different operation types more efficient than Lustre. The new hardware of the DDN system at Dsseldorf yields the best performance since it uses DNE2 and utilizes two metadata servers. However, the IME performance is significantly lower than the underlying Lustre system. The reason is the FUSE mount and that IME uses a Lustre file system for the metadata handling and needs to manage the flash storage.

Now comparing the benchmarking phase with T=0 and T=4, there are several cases: The virtual creation rate of T=4 is higher, this is caused by a decrease in request latency since the load to the file system is reduced to one fifth – the servers have a small queue of pending operations. GPFS benefits minimal (25%), while other systems the rate increases to 3x. Mistral is different: its performance decreases when the request rate is throttled. The reason for this performance loss of Mistral is that the file system is accessed by many users and background operations issued by other users still lead to waiting queues. The burst buffers at KAUST and NERSC deliver similar results albeit different client nodes are deployed. More details are discussed in the next sections.

This alone is not sufficient to illustrate the difference in precreation and benchmarking phase. Therefore, the density of each individually timed create operation is plotted for selected systems and these phases in Figure 4. A density graph can be considered a smoothened histogram – x-axes shows the observed runtime and the y-axes the number of occurrences, the x-axes uses the same scale for all three examples, the first diagram is printer larger for better exploration. It can be seen, that the overall system behavior changes between precreate and benchmarking phase. For Mistral, the change is minimal, while for the benchmark run they are executed typically below 1ms, the precreation phase shows a right shift in the response time and some operations take now up to 10ms; however, the computed average performance on the benchmarking phase is lower compared to precreate! As we will see, this is due to the fact



(a) Mistral (Lustre): 10 Nodes, 10 PPN

(b) Cooley (GPFS): 10 Nodes, 10 PPN

(c) Dsseldorf (IME): 8 Nodes, 10 PPN

Fig. 4: Density of create operations: creation phase compared to the benchmark phase with waiting factor of 0 and 4. Parameters: D=1, I=2000, P=10000

that delete and read operations of the mix take longer while create speeds up in the mix. The GPFS system behaves totally different between precreate and benchmarking phase. This is presumably caused by the locking strategy that has an issue with shared directories. The throttled benchmark run with T=4 leads to a similar latency distribution than T=0 for Mistral and GPFS. Furthermore, for IME+Lustre the pattern changes, here precreation and benchmark run look similar but T=4 decreases latency of the creates significantly. The reason is again to be examined in the complex interplay with the FUSE client, IME and that some operations cause a delay on IME.

We investigated many cases for 10 and 100 nodes (except for the Dsseldorf cluster which had only 8 usable nodes), and from this analysis, it can be concluded that the impact of the concurrent metadata access pattern is significant. The typically significantly higher throughput of bulk creation demonstrates that preserving a fixed working set by MDWorkbench is able to investigate more real behavior and reveals caching and locking issues. To measure a system, one cannot run a single phase alone as a production system will see a mixed workload.

5.2 Overview of results for the benchmark phase

An overview of the performance of all systems (and T=0) is given in Table 2. Note, we exceptionally include results of DKRZ's Mistral first file system (phase 1) for comparison. The table describes the parameters for running the benchmark and provides several metrics: The creation rate of the mixed workload – the actual metadata performance is 4x the creation rate, but since so far we discussed the creation rate, we sticked with it; the balance across processes which is (t_{min}/t_{max}) – a value of 100% means all processes finish at the same time; next is the maximum time for any individual operation across all processes; finally, the latency in seconds of the 3rd quartile (Q3), i.e., below which 75% of all observations are – is given for each operation type. Albeit not a density diagram, this enables a quantitative understanding of latency.

First, we look at the balance: Most runs achieve a balance of 90%, i.e., all clients finish at the same time. However, the Mistral Lustre yields worse results, just above 70%; even for 10 client processes (Nodes=10, PPN=1), the balance is worse. The Dsseldorf DDN Lustre also is unbalanced, but significantly faster. Looking into the rates, the DDN Lustre system yields the best results followed by the Lustre systems with 8 DataWarp nodes. IME and the GPFS achieve the worst performance. There are several remarkable observations that can only be made analyzing the latencies: 1) First of all, on several systems a maximum operation latency of one second is observable, which impacts users that work interactively; 2) At Mistral, using Nodes=100 and PPN=10 doubles the overall throughput of the benchmark from 2200 creates/s. But, at the same time, the maximum waiting time increases to 16 seconds and the waiting time of individual operations of the third quartile is increased by an order of magnitude (except for stat which increases slightly). Since in all other cases for Mistral Q3 is similar, this is a clear indicator of an overloaded file system and metadata server. This is expected at some point since the servers need to queue up pending requests increasing their latencies; 3) Q3 for IME system read is worse than for the underlying Lustre system, particularly when running multiple processes per node, so for small files, there is no benefit from the flash based solution, yet. The reason is presumably the overhead in FUSE and particularly the thread handling within FUSE where other operations delay the execution of the read operation; 4) The KAUST burst buffer can deal with a working set of 100k files well, but with 1 million files, the performance drops. This affects all Q3 statistics and the drop is more severe for 8 burst buffer nodes than for one; 5) For all Lustre systems, the number of directories does not influence the performance much, as long as the total working set is the same (D=1, P=10k, I=2k vs. D=10, P=1k, I=200). For GPFS, however, the performance behavior changes significantly, for PPN=1, the case with D=1 is 3x faster than D=10. The Q3 for the stat does not change, but all other operations are one order of magnitude slower. For interactive usage, the importance of latency analysis is apparent when comparing GPFS and Mistral 100 nodes, the creation rate is similar; however, the maximum waiting time for Mistral is 16s! For Lustre, the Q3 is slower by an order of magnitude for most operations, but it also faces extreme slow stragglers (not shown).

Nodes	PPN	D	O	P	Rate creat/s	balance %	op_max in s	Latency of quartile 3 read	stat	create	delete
ACLF Cooley GPFS											
10	1	1	2000	10k	1530	95.5	0.13	9.2E-3	8.4E-4	3.9E-4	3.0E-4
10	1	10	200	1k	500	99.8	0.17	1.4E-2	8.5E-4	8.1E-3	4.5E-3
10	1	10	2000	10k	540	100	0.17	1.4E-2	9.8E-4	7.6E-3	4.1E-3
10	10	1	2000	10k	5280	60.7	0.20	1.8E-2	8.3E-4	5.6E-4	2.8E-4
DKRZ Mistral Lustre											
10	1	1	2000	10k	290	86.3	3.60	4.8E-3	6.4E-4	6.4E-4	5.8E-4
10	10	1	2000	10k	2180	68.2	3.50	4.6E-3	4.0E-4	6.7E-4	4.9E-4
10	10	10	2000	10k	2140	78.4	7.90	5.6E-3	4.2E-4	6.3E-4	3.4E-4
100	1	1	2000	10k	1610	72.2	4.60	4.8E-3	6.4E-4	6.5E-4	6.2E-4
100	10	1	2000	10k	4890	77.3	16.00	3.3E-2	6.9E-4	1.1E-2	9.9E-3
DKRZ Mistral Lustre (Procurement phase 1 file system)											
10	1	1	2000	10k	1640	100	0.54	1.0E-3	5.9E-4	7.3E-4	5.3E-4
10	1	10	2000	10k	980	100	3.90	3.8E-3	4.4E-4	7.9E-4	2.9E-4
10	10	1	2000	10k	2660	100	7.40	1.2E-2	8.7E-4	5.4E-3	5.7E-3
Dsseldorf DDN Lustre											
8	1	1	2000	10k	4750	92.4	0.00	5.0E-4	3.1E-4	4.5E-4	3.3E-4
8	1	10	200	1k	4980	95	0.01	5.6E-4	3.2E-4	4.5E-4	3.4E-4
8	10	1	2000	10k	11850	49.5	1.00	1.5E-3	8.1E-4	1.7E-3	1.7E-3
8	10	10	200	1k	10390	40.1	0.10	1.8E-3	9.7E-4	2.0E-3	2.0E-3
Dsseldorf DDN IME											
8	1	1	2000	10k	820	94.9	0.05	7.4E-4	5.5E-4	4.2E-3	4.4E-3
8	1	10	200	1k	820	96.1	0.06	7.3E-4	5.5E-4	4.1E-3	4.4E-3
8	10	1	2000	10k	1540	89.7	0.86	5.4E-3	2.0E-2	2.6E-2	1.2E-2
8	10	10	200	1k	1460	93.4	0.20	8.8E-3	2.3E-2	2.8E-2	1.4E-2
Kaust DataWarp 1 burst buffer node											
10	1	1	2000	10k	3170	99.2	0.03	6.7E-4	3.3E-4	2.2E-3	3.7E-4
10	1	10	200	1k	3130	98.8	0.06	7.5E-4	3.5E-4	2.1E-3	3.8E-4
10	10	1	2000	10k	3340	94.4	0.18	4.7E-3	7.6E-3	1.6E-2	7.9E-3
10	10	10	200	1k	3340	98.2	0.16	5.1E-3	7.3E-3	1.6E-2	8.3E-3
10	10	10	2000	10k	2190	98.4	0.43	5.0E-3	3.9E-3	2.3E-2	2.2E-2
Kaust DataWarp 8 burst buffer nodes											
10	1	1	2000	10k	4650	97.6	0.01	5.3E-4	3.3E-4	1.2E-3	3.0E-4
10	1	10	200	1k	5000	96.9	0.01	4.7E-4	2.9E-4	1.1E-3	2.9E-4
10	10	1	2000	10k	7250	82.1	0.16	1.2E-3	4.8E-4	4.1E-3	1.3E-3
10	10	10	200	1k	6510	91.3	0.16	1.2E-3	3.9E-4	3.9E-3	1.2E-3
10	10	10	2000	10k	1860	91.9	0.43	1.2E-2	1.1E-3	1.8E-2	4.9E-3
NERSC DataWarp 8 burst buffer nodes											
10	1	1	2000	10k	4000	95.3	0.03	5.2E-4	1.4E-4	8.8E-4	9.7E-5
10	1	10	200	1k	6670	93.4	0.02	4.2E-4	1.8E-4	8.2E-4	1.2E-4
10	10	1	2000	10k	8770	84.9	0.15	2.4E-3	2.0E-3	5.3E-3	1.2E-3
10	10	10	200	1k	8730	96.8	0.08	2.7E-3	1.9E-3	6.4E-3	1.3E-3

Table 2: Result overview; several performance metrics for the individual systems

5.3 Understanding latencies

To understand the density diagrams better, the timelines for the individual operations can be analyzed. In Figure 5, we selected timelines to investigate interesting issues. A point represents a measured latency from any of the processes[11].

First, we briefly investigate the difference in creation rate between precreation phase (Figure 5a) and benchmarking phase with T=0 (Figure 5b) for Mistral. One can see a periodic pattern – every 5 seconds, the operation execution stalls and take about a second. This could be explained by the behavior of the underlying LDISKFS (EXT4), e.g., the metadata commit timer. When running the mixed workload of the benchmark phase, intermediate waiting times are reduced and slow operations are less likely. Still, the 5 second commit rate remains. Actually, the waiting time only appears for metadata modifying operations, when running the read-only workload the periodic stalls disappear. Again this shows the importance of running a mixed workload and the complexity of file systems.

For GPFS, two graphs are included: In Figure 5c, the timeline for the precreation phase is shown. One can also see two classes of latency bands, one at 50 ms and one at 1ms. Additionally, in the beginning, with a nearly empty directory, the latency is substantially higher. Operating on multiple directories, i.e., increasing parameter D to 10 reduces performance significantly. For example, with PPN=1 it falls from 1,500 creates/s to 500 creates/s. Figure 5d shows the timings for D=10, here two bands can be observed at 10ms and at 1ms. With same total working set size but D=1, the upper band vanishes (not shown). Presumably, the reason is the locks involved in the directories (tokens in GPFS speech). With D=10, each directory is written and read by 10 processes increasing the looking overhead. During the precreation, each process operates on its directory but periodically flushes the data leading to the higher latencies.

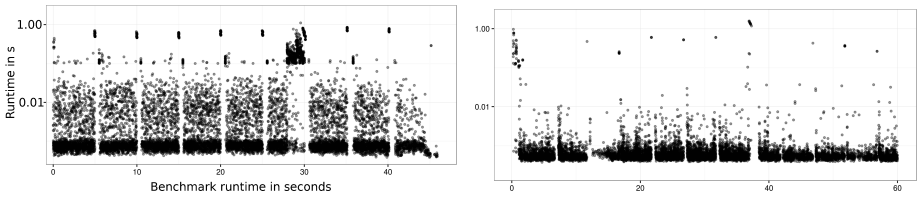
6 Conclusions

In this paper, we discussed several issues for metadata benchmarking and for 10 and 100 nodes showed the importance of latency measurements. Experiments and the introduced methodology shows that with 10 nodes metadata servers of all file systems (except on Mistral) are well utilized. We demonstrated that phase-wise execution of a single operation type is unable to capture certain characteristics like contention caused by metadata changes. MDWorkbench offers an alternative workload emulating the sharing of datasets and objects combined with several features useful for regression testing and data analysis. For example, we identified issues and could understand the relation between observed throughput and latency characteristics on several state-of-the-art systems better. In the future, we will experiment with machine learning to mine interesting patterns.

[11] The plot is sparse, e.g., 100k data points of 1 million creates have been randomly selected. Additionally, all measurements about 0.1s have been added.

(a) Mistral: 10 PPN, D=1, I=2000, P=10k, Precreation phase

(b) Mistral: 10 PPN, D=1, I=2000, P=10k, creates of the benchmark phase



(c) GPFS: 10 PPN, D=1, I=2000, P=10k, Precreation phase

(d) GPFS: 1 PPN, D=1, I=200, P=1000, creates of the benchmark phase

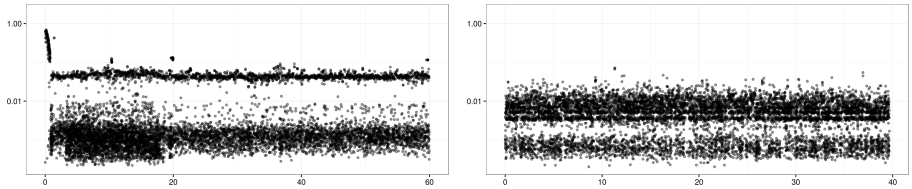


Fig. 5: Timeline of individual measurements for 10 Nodes; sparse plot with random points, but every point above 0.1s is added (qualitative view).

Acknowledgements

Thanks for DDN providing access to their facility and the discussion with Jean-Thomas Acquaviva and Jay Lofstead. This research used resources of the KAUST Supercomputing Core Laboratory, of the Argonne Leadership Computing Facility and NERSC, which are under DOE Office of Science User Facilities supported under Contract DE-AC02-06CH11357 and DE-AC02-05CH11231 respectively.

References

1. Alam, S.R., El-Harake, H.N., Howard, K., Stringfellow, N., Verzelloni, F.: Parallel I/O and the Metadata Wall. In: Proceedings of the sixth workshop on Parallel Data Storage. pp. 13–18. ACM (2011)
2. Carns, P., Lang, S., Ross, R., Vilayannur, M., Kunkel, J., Ludwig, T.: Small-file Access in Parallel File Systems. In: IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing. pp. 1–11 (2009)
3. Friedrich, S., Wingerath, W., Gessert, F., Ritter, N., Pldereder, E., Grunske, L., Schneider, E., Ull, D.: NoSQL OLTP Benchmarking: A Survey. In: GI-Jahrestagung. pp. 693–704 (2014)
4. Hadri, B., Kortas, S., Feki, S., Khurram, R., Newby, G.: Overview of the kausts cray x40 system–shaheen ii. proceeding of Cray User Group (2015)
5. Huppler, K.: The Art of Building a Good Benchmark, pp. 18–30. Springer (2009)
6. Katcher, J.: PostMark: A new file system benchmark. Tech. rep., TR3022, NetApp (1997)
7. Méndez, S., Rexachs, D., Luque, E.: Methodology for performance evaluation of the input/output system on computer clusters. In: 2011 IEEE International Conference on Cluster Computing (CLUSTER). pp. 474–483 (2011)
8. Storage Performance Council: SPC BENCHMARK 1 (SPC-1) – Rev. 3.5 (09 2017)