User-Centric System Fault Identification Using IO500 Benchmark

Radita Liem* Dmytro Povaliaiev* Jay Lofstead† Julian Kunkel‡ Christian Terboven*
* RWTH Aachen University, † Sandia National Labs, ‡ Göttingen University

***Abstract*—I/O performance in a multi-user environment is difficult to predict. Users do not know what I/O performance to expect when running and tuning applications. We propose to use the IO500 benchmark as a way to guide user expectations on their application's performance and to aid identifying root causes of their I/O problems that might come from the system. Our experiments describe how we manage user expectation with IO500 and provide a mechanism for system fault identification. This work also provides us with information of the tail latency problem that needs to be addressed and granular information about the impact of I/O technique choices (POSIX and MPI-IO).**

***Index Terms*—I/O, workflow, data center, benchmarking**

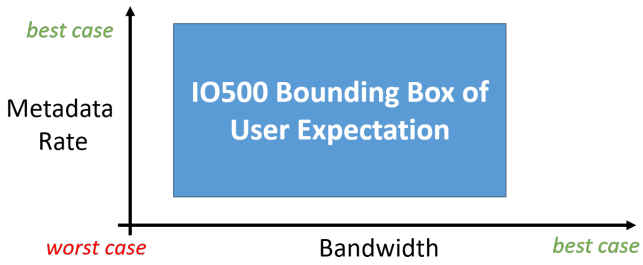


Fig. 1. Illustration of the bounding box of the user expectation.

I. Introduction

LARGE scale high performance computing (HPC) facilities are usually shared by multiple users running their various applications. In this multi-user environment, I/O is one of the resources that are shared among all of the cluster users. This causes performance variability on the running applications since each repeated run might not get an identical share of resources such as storage system and network bandwidth. This makes predicting an application's I/O performance difficult given all of the other various intertwined factors, e.g., the application's structure, interference from other applications, and filesystem characteristics.

The difficulty to predict the I/O performance and its variability leaves users with the inability to know what to expect when running and tuning the I/O performance. In this situation, users tend to rely on their own perception and personal recollection that the application is slowing down or theoretically should be able to run faster. This situation also has been observed by Kunkel and Betke [1] when proposing probing to track user-perceived slowdown. There are several works [2], [3], [4] proposed to track and monitor the I/O performance variability inside HPC facilities, but most of them are focused on the system maintainer as the main stakeholder and leave most users in the dark with what to do to improve their applications.

Long established individual I/O benchmarks, such as IOR and MDTest each offer a glimpse into one factor the user may consider when tuning their code. To get a full view, the user will have to run several benchmarks and compare the combined results against what their application can achieve to assess how well the application's I/O is performing.

Ideally, a benchmark can reveal the best and worst-case expectations. It should also offer more detailed components that reveal different considerations for their understanding of their application's expected performance. For large writes, such as for a checkpoint operation in a scale-up application, looking at I/O bandwidth primarily would be a good guide. For a machine learning application that reads many small files or just many small reads, the basic IOPs a system can offer is a better indicator. With applications having different kinds of I/O phases, a mixed approach that reveals the different aspects of performance represents the workload better. By understanding the access patterns of an application, it is possible to get a rough guess as to how well its I/O calls are performing compared to what should be expected on the system. In this way, a user could use this information as a guide for when and where to tune.

We want to provide users with easy-to-digest and realistic expectation about their application's I/O performance in the specific environment where their applications are running. By having a realistic expectation, users can devise a tuning strategy with a reasonable target to optimize their application. In this pursuit, we propose to use the IO500 benchmark [5], [6] as a way to manage user expectation for a HPC system. The IO500 benchmark is designed to offer a balanced, trustworthy, and representative performance measurement of the typical workloads observed on real systems [5], [7].

The rest of this paper is organized as follows. First is an overview of the IO500 benchmark and why we believe it can be useful as an end user tool in Section II. Next, in Section III, we provide an overview of the proposed workflow for an end user to get their own production-time view of the I/O characteristics for their system and how to use them to tune their application. Section IV offers an overview of the experiments performed with results in Section V. We offer a short discussion and future plan in Section VI.

TABLE I
IO500 BENCHMARK COMPONENTS

Component	**Tests**	Metric	**Explanation**
IOR 'easy'	ior_easy_write, ior_easy_read	GiB/s	Free to tune IOR parameters. Typically file-per-process, large, aligned chunks to get the best possible bandwidth performance
IOR 'hard'	ior_hard_write, ior_hard_read	GiB/s	Limited options to tune. Forced to use small unaligned I/O to a single shared file for the worst possible bandwidth performance
mdtest 'easy'	mdtest_easy_delete, mdtest_easy_stat, mdtest_easy_write	KIOPS	Free to tune mdtest parameters with zero size files in separate directory per process to represent best case scenario for metadata rate
mdtest 'hard'	mdtest_hard_delete, mdtest_hard_stat, mdtest_hard_write, mdtest_hard_read,	KIOPS	Limited options to tune. Forced all processes to write on a single shared directory. Representing worst case scenario for metadata rate
Find	find	KIOPS	Finding specific subset of files from those created by four scenarios.

II. IO500 BENCHMARK

The IO500 benchmark is a benchmark suite that is designed to capture user-performance experience. The benchmark encourages balanced systems that do not focus on just a single metric, such as bandwidth or IOPs (for IO500, metadata performance). To achieve this goal, it provides five main measurement scenarios using IOR and mdtest as listed in Table I. The tests represent the best and worst possible scenarios for bandwidth and metadata in the form of 'easy' and 'hard' use cases respectively. To avoid weighting any particular metric more heavily than any other, the individual IO500 benchmark components are combined using a geometric mean to find the central tendency among the various metrics.

The IO500 community also provides IO500 list, a ranked list where data center can compete to top the list using their best reported IO500 performance. In the IO500 list, the challenge to get a top score for IO500 is to tune the parameters to balance the 'hard' and 'easy' bandwidth tests against the 'hard' and 'easy' metadata tests. In many cases for Lustre, sacrificing a bit of bandwidth can offer higher metadata performance. While a top score does not indicate that all applications can achieve that performance, the range from the 'hard' to 'easy' on bandwidth and metadata give bounds for users can expect [8], [5] as illustrated in Figure 1.

The IO500 list can help users to determine the latest trend and how their data center performs compared to the other data centers. However, comparing what the user can achieve against what the results in IO500 list is not a direct nor easy comparison because of several reasons:

First, IO500 shows strictly the storage layer performance for the system. Additional overheads introduced by middleware, I/O libraries, data organization and structure, and other factors above the storage layer are not represented. To be fair, many of these are application-specific, and measuring them in a general-purpose, storage-oriented benchmark is not reasonably expected. Despite this, some value for the application users should be available based on these numbers.

Second, the tests in IO500 are carefully designed to eliminate caching effects and, in the case of the 'hard' tests, to be particularly difficult for a storage system to show excellent results. The 'hard' tests are supposed to represent the worst-case scenario a storage system may encounter. The key to this is that the storage system's worst-case scenario may have no relation to the application's worst-case scenario nor will it include factors above the storage layer, such as those outlined above. The IO500 benchmarks are 'pure' tests of each feature rather than the mixed workload experienced by many, if not most, applications.

Lastly, entrants of the IO500 list only report the best results they achieve rather than each set they create. This leads to a slightly to a significantly skewed view of the system for those relying on the published list to know their target system's characteristics. Further, if the dedicated system time is used to run IO500, then interference effects from other running applications would be reduced or eliminated showing an unrealistically rosy picture of the possible performance for the platform. Given these limitations, the potential value of using IO500 for application user's to understand their IO performance is unknown.

III. WORKFLOW

The general overview of the workflow proposed in this work can be seen in Figure 2. The workflow is designed to use IO500 characteristics to help users managing their expectation. It consists of three main steps: 1) Setting up the boundary of expectation; 2) Mapping the application's I/O performance; and 3) Tuning the application with reasonable expectation. An overview of the workflow used is illustrated in Figure 2.

A. Boundary of User Expectation

The first step of the workflow is creating the bounding box of user expectation. We run IO500 with the same configuration multiple times to capture performance variability on the benchmark bandwidth and metadata rate results. In Figure 2, the blue dashed line represents the boundary of user expectation. Multiple dashed lines in this boundary show the performance variability coming from the shared resource nature of the systems that we must take into account. This recorded variability information will keep the information about tail latency in check and get information on the system's health.

In forming the bounding box, we only use the bandwidth and metadata score coming from IOR and mdtest 'easy' and 'hard' without *find* operation. While the IOR and MDTest 'easy' and 'hard' operations are controlled to require open code in those base benchmarks and the configuration used

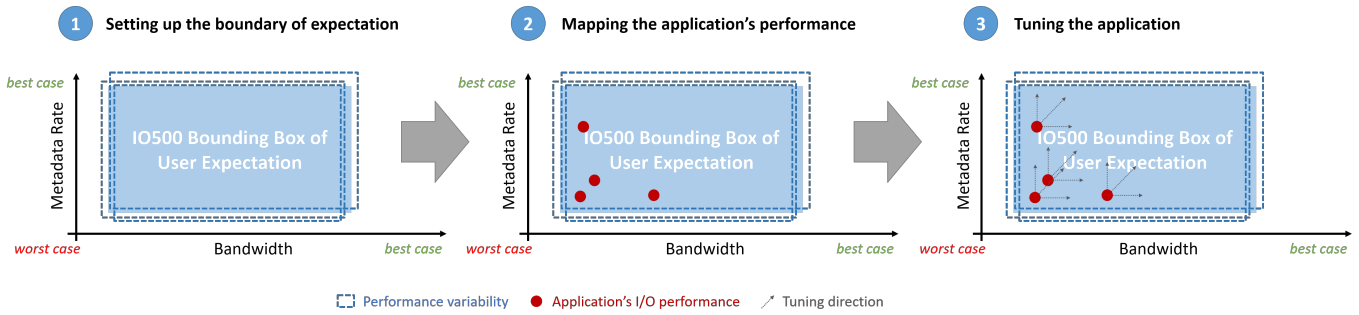


Fig. 2. Workflow steps of the project

published, the find workload is a place for creativity that pushes vendors to identify the most efficient way to walk the namespace for a storage system representing an operation like identifying purge candidates. The original benchmark offered a basic parallel version of find that has since been a place for innovation. For instance, the Intel DAOS [9] team wrote a custom find operation resulting in excellent performance for them. More recently, the MadFS team architected their system to enable very fast metadata search operations, like find, resulting in an astoundingly high value. Given the underlying implementation, the result is still comfortably within both theoretical and practical limits. Since this component is unlike what nearly all applications will do, we choose to ignore this component in favor of those that better represent application activities.

While having the user run the IO500 benchmarks themselves rather than relying on a system provided, production mode standard may seem counter-intuitive. The idea is that as storage systems age, device characteristics can shift due to bad block management or other media issues degrading performance. This has been observed in disk drives [10] and flash-based devices [11]. Having the user run the benchmarks fresh each time will offer a current view of the system rather than a potentially out of date picture.

B. Mapping the Application Performance

For the second step, we run the application multiple times as well, since it will also experience variability, to obtain measurements for the current performance. A tool such as Darshan DXT [12] can offer fine grained details about the I/O operations the application performs giving a view into the performance of different operations, such as bandwidth related writing or reading or metadata operations that would suggest IOPs rates. Using that information, it is possible to map the application's I/O performance onto the same graph. Hopefully, these results appear within the box rather than below or to the left (worse than projected worst case). Given the additional software and hardware layers involved, this is possible, particularly for feature-rich libraries. In Figure 2, the red dots in Steps 2 and 3 represent the observed application performance.

C. Tune the Application

After knowing the position of the application's performance on this graph, the user can either make adjustments understanding where the issues lie or consult an I/O expert and present them with the evidence seeking advice. Once tuned, the user can return to Step 2 and re-measure the application performance and compare the new value locations against the previous measurements. In Figure 2, the small red arrows connected to the red dots in Step 3 represent the desired movement in the observed performance after having tuned the application. The new application measurements are not presented to avoid confusion.

This measure/tune cycle of Steps 2 and 3 is expected to be executed potentially many times until the application reaches acceptable performance for the user. The challenges related to the software and hardware above the storage system must temper expectations. Additional experiments with these layers to determine their overhead contributions could help set more reasonable expectations.

IV. Experiment

The experiment in this project mainly covers the first step of the workflow and demonstrates the challenges in obtaining the bounding box of user expectations when compared to the results reported on the IO500 list itself. We start to explore the tools and methodology to plot the application's I/O performance from the second step using BTIO benchmark. Since the second step is still in the exploratory stage and has strong ties with the third step, step two and three will be part of our future work.

We conduct the initial experiments in RWTH Aachen University on the CLAIX-2018 cluster. CLAIX-2018 consists of 1032 nodes each with 48 cores Intel Skylake and 384 GB of memory. There are several attached BeeGFS filesystem [13] nodes, each with a 480 GB SSD on each of these nodes. We are using BeeGFS to minimize interference and reduce the performance variability for this model. The IO500 is also configured using the default parameters to avoid excessive time doing fine-tuning for this proof of concept work.

We run the IO500 test with POSIX-IO and MPI-IO API to get a comparison of these different I/O techniques. Most HPC applications are using these I/O techniques and it will help us build the second step of the workflow in the future. The benchmark runs on four nodes of BeeGFS with the same node

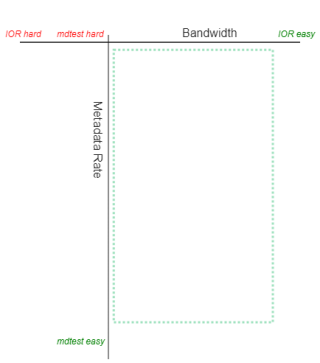
leaf switch to reduce network interference. For these tests, the benchmark is run several times and the results are reported in Section V.

In the exploratory mapping work, we are using BTIO benchmark version 3.4.2. The benchmark is compiled with subtype ”full” class A, B, and C. We run the benchmark on 4, 9, and 16 processes. Darshan with DXT variable on is used to profile the application. We currently only work with the aggregated information from the Darshan log to do the plotting on a single node.

V. Results

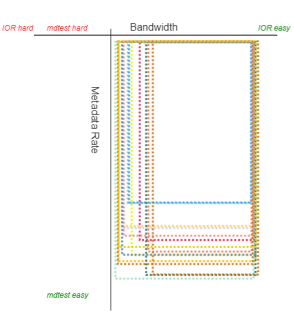
A. Forming Bounding Box of User Expectation

Figure 3 represents an IO500 score for one run of the benchmark and shows a bounding box of user expectation. The aggregated read-write bandwidth and metadata score are plotted into the Cartesian coordinate system that represents the relative position of the bandwidth and metadata score for ’easy’ and ’hard’ scenarios.



IOR hard	IOR easy	mdtest hard	mdtest easy
0.85	1.88	10.97	145.17

Fig. 3. Single run of POSIX-IO IO500 plotted into Cartesian coordinate system made of the aggregated read-write bandwidth score and metadata score

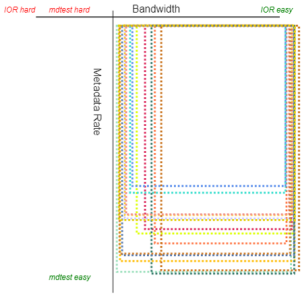


IOR hard (GiB/s)	IOR easy (GiB/s)	mdtest hard (KIOPS)	mdtest easy (KIOPS)
0.85	1.88	10.97	145.17
1.03	1.89	11.26	123.26
1.10	1.87	10.59	129.86
0.87	1.90	10.76	135.27
0.97	1.90	10.64	131.93
0.94	1.86	11.06	102.35
0.95	1.86	10.84	102.06
0.90	1.89	10.98	116.54
0.90	1.89	11.16	115.40
1.08	1.90	11.14	143.09
0.91	1.88	10.80	120.86
0.90	1.90	11.05	131.65

Fig. 4. POSIX-IO results from the aggregated read-write bandwidth and metadata score

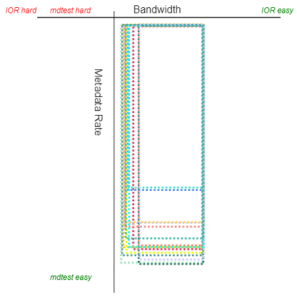
Each run gives four values representing two vertices of the box. We start with plotting the ’hard’ value into the system, and then subtracting the ’easy’ with ’hard’ value to determine the width and the height of the box. Since the premise of IO500 is that ’hard’ scenarios represent the worst case scenario, the score of ’hard’ scenarios should not score better than the ’easy’ scenarios, thus we expect the bounding box will appear in the same sector as figure 3.

We run the same IO500 config file multiple times, as can be see in Figure 4, and observe a performance variability that we already expected between each run.



IOR hard (GiB/s)	IOR easy (GiB/s)	mdtest hard (KIOPS)	mdtest easy (KIOPS)
0.52	1.75	2.60	76.12
0.72	1.76	2.56	63.37
0.79	1.72	2.55	67.64
0.55	1.78	2.62	72.92
0.66	1.77	2.56	64.69
0.62	1.71	2.57	52.50
0.63	1.71	2.56	50.49
0.58	1.77	2.54	60.23
0.58	1.76	2.50	59.20
0.77	1.77	2.64	76.67

Fig. 5. POSIX-IO write performance. Meta data range is smaller compared to the aggregated score



IOR hard (GiB/s)	IOR easy (GiB/s)	mdtest hard (KIOPS)	mdtest easy (KIOPS)
1.37	2.02	24.80	360.31
1.47	2.03	26.02	341.16
1.51	2.03	23.60	335.88
1.39	2.03	24.04	338.53
1.42	2.03	23.63	346.27
1.44	2.03	24.96	254.26
1.43	2.03	24.70	257.68
1.40	2.03	25.13	303.84
1.41	2.02	25.57	303.20
1.51	2.02	25.22	362.23

Fig. 6. POSIX-IO read performance. It has large range of meta data performance with much smaller bandwidth performance range and variation

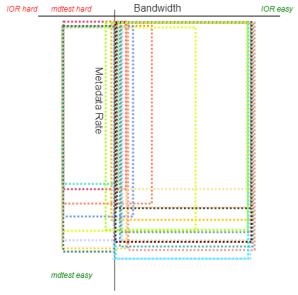
When forming the bounding box of user expectations, besides the aggregated read-write result, we can also look at the read and write value individually. The result for write and read performance can be seen in Figure 5 and Figure 6 respectively. Read and write performance bounding box have different patterns where meta data range of the write bounding box is smaller than the aggregated performance, and read bounding box has much bigger meta data range performance compared to the aggregated value.

With more than 20 runs, we observe that size of the boxes fluctuates but will stabilize within a certain range and will not expand or shrink further. Upon stabilization, we determine that the union of all of the boxes forms our *bounding box of user expectations*, and should represent the boundaries of the performance variability. We then repeat this experiment with MPI-IO API.

B. Anomalous Bounding Boxes

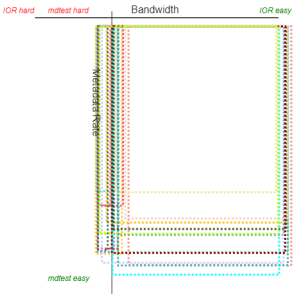
From the previous result with POSIX-IO run in Figures 4, 5, and 6, we can form a bounding of user expectation as it is intended by IO500 benchmark design. However, we see anomalies on several runs with MPI-IO. Figures 7 and 8 show that a few of the IOR ’hard’ scores are better than the IOR ’easy’ score. This challenges the notion that the IO500

IOR 'hard' benchmark represents the worst case scenario for bandwidth. This anomalies are easily seen on the graph as a box laying in the wrong sector, and this can act as a quick sanity check of the cluster/filesystem health.



IOR hard (GiB/s)	IOR easy (GiB/s)	mdtest hard (KIOPS)	mdtest easy (KIOPS)
0.90	1.89	10.74	135.79
0.94	0.47	10.50	106.45
1.14	0.47	12.86	114.73
0.83	1.89	11.12	124.06
1.46	0.47	13.78	130.29
0.88	0.47	13.50	103.47
0.99	0.47	13.24	122.10
0.91	0.47	13.04	135.73
0.95	0.46	13.10	140.41
0.85	0.47	13.02	142.20
0.96	1.90	11.00	141.28

Fig. 7. MPI-IO anomaly from the aggregated score. The Cartesian diagram formation is designed to create figure that will to the direction of 'easy' and 'hard' scenario for identifying such scenario



IOR hard (GiB/s)	IOR easy (GiB/s)	mdtest hard (KIOPS)	mdtest easy (KIOPS)
0.57	1.77	2.59	69.78
0.61	0.45	2.48	56.03
0.52	0.44	2.65	55.88
0.50	1.76	2.54	61.06
0.55	0.43	2.63	64.00
0.54	0.43	2.61	51.70
0.55	0.43	2.54	69.78
0.58	0.44	2.55	68.79
0.61	0.42	2.55	70.35
0.51	0.44	2.54	69.27
0.65	1.79	2.57	73.61

Fig. 8. MPI-IO anomaly from the write score. Aggregated score, write and read score are all showing couple of runs get higher IOR 'hard' score than IOR 'easy' score

Further investigation led to the discovery that a particular node leads to degraded performance. While the details are not clear, this broken node is most likely the reason behind these anomalous results. While the discovery of the broken node is useful, the benchmarks also revealed laggard processes demonstrated by tail latency that will be described in the *Tail Latency Observations* section.

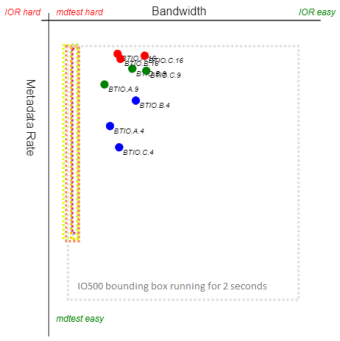
C. Exploration on the I/O Performance Mapping

To test the feasibility of the second step of the workflow we proposed, using one node, we map application's performance into the IO500 bounding box using performance results of BTIO benchmark. From the Darshan log output, BTIO has both POSIX-IO and MPI-IO result in it. We use these results from the Darshan log for plotting.

In Figure 9, the result from the BTIO benchmark falls outside of the bounding box of the user expectation. Our hypothesis is that the application utilizes page caching, whereas the default config of IO500 runs for 300 seconds and this eliminates the cache effect. To test this hypothesis, we run the same IO500 configuration for only two seconds on the 'easy' scenario. The result from the two seconds run is in the grey colored bounding box where we can see that now the application's bandwidth falls within the bounding box of the user expectation for both MPI-IO (Figure 9) and POSIX result (Figure 10).

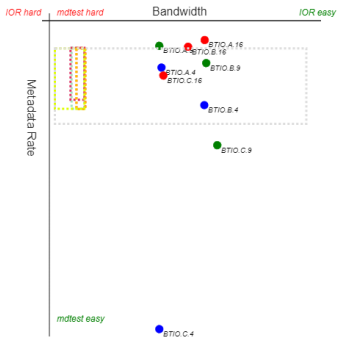
The metadata rate formula that we used in this plot falls within the IO500 result only for MPI-IO API. In Figure 10 however, we can see that the formula that we use does not fit into the bounding box of user expectation. Currently, We define metadata rate (in KIOPs) as a total number of I/O operations divided by the file meta time from the Darshan log output. This formula might not suitable for this plotting. Therefore, We still need to explore tools, metrics, and formula that might be able better represent the I/O performance.

Since the IO500 configuration for the 'easy' parts are not optimized, it is still possible that other application's performance will not fall inside the bounding box of user expectation due to the setup we have. The caching effect also needs to be investigated further, along with performance I/O tools that can give more granular report on the application's performance.



Benchmark	Bandwidth (GiB/s)	Metadata Rate (KIOPS)
BTIO.A.4	0.68	21.67
BTIO.A.9	0.64	14.18
BTIO.A.16	0.73	8.66
BTIO.B.4	0.85	17.08
BTIO.B.9	0.82	11.35
BTIO.B.16	0.75	9.58
BTIO.C.4	0.74	25.50
BTIO.C.9	0.91	11.71
BTIO.C.16	0.90	9.00

Fig. 9. I/O Performance Mapping using BTIO benchmark with Darshan. The bounding box is from MPI-IO API on a single node. Grey colored box is the same config from the boundary formation but running for 2s for 'easy' case



Benchmark	Bandwidth (GiB/s)	Metadata Rate (KIOPS)
BTIO.A.4	0.99	18.42
BTIO.A.9	0.97	5.99
BTIO.A.16	1.27	2.69
BTIO.B.4	1.27	39.90
BTIO.B.9	1.28	15.84
BTIO.B.16	1.16	6.51
BTIO.C.4	0.97	167.96
BTIO.C.9	1.36	62.81
BTIO.C.16	1.00	22.99

Fig. 10. I/O Performance Mapping using BTIO benchmark with Darshan. The bounding box is from POSIX API on the single node.

D. Tail Latency Observations

Several scenarios in IO500 runs can have significant latency that needs to be addressed as see in Figure 11. The runtime duration variability of the figure on the top is taken from the runtime numbers reported by multiple IO500 runs, and the