A File System Utilization Metric for I/O Characterization

Andrew Uselton, Nicholas Wright
NERSC, Lawrence Berkeley National Laboratory
{acuselton,njwright@lbl.gov}

Abstract

A high performance computing (HPC) platform today typically contains a scratch high-performance parallel file system for data storage. Today, such file systems encompass 10-20% of the purchase price of a HPC resource. Looking forward, it is apparent that the rate of increase of hard drive performance will not keep up with the expected gains in processing, and therefore any effort to keep I/O performance at the same relative level will require a larger and larger fraction of the overall budget. Therefore, it will become increasingly important to understand the I/O usage and needs of HPC workloads in great detail in order to ensure resources are adequately provisioned. Although it is relatively straightforward today to measure the total amount of I/O to and from a file system, and the bandwidths achieved, these metrics reveal only part of the overall file system load, because the size of individual I/O requests and their relation to one another can strongly affect how much data a file system can move in a given interval. In this work we introduce a new metric for file system utilization (FSU) that accounts for both the volume of data moved and the number of disk I/O operations required to move that data. We present a description of our model in which we idealize an I/O transaction to disk (on the server) as requiring time t that includes a small, fixed amount of time (the latency, a_0) and an amount of time proportional to the size of the I/O: $t = a_0 + a_1 * b$. A series of calibration experiments using transactions of various sizes allows us to establish a_0 and a_1, which we do separately for reads and writes. We collect data on the number and size, $N(b)$, of all I/O transactions, which allows us to calculate the File System Utilization (FSU). The FSU metric provides a view of the I/O workload on the file system. We present early results showing that the Hopper Cray XE6 `/scratch` *file system is about four times as busy as we would estimate from a bandwidth metric alone.*

I. Introduction

In a high performance computing (HPC) system there is a strong motivation to achieve the best performance possible [16] and to measure that performance as accurately and honestly as possible [7]. The same holds for the I/O capability of the HPC system, and for its *scratch* file system, in particular. The scratch resource of an HPC system is usually constructed from a parallel file system such as Lustre [8], PVFS [13], or GPFS [14]. This discussion will focus on Lustre, but all parallel file systems are notoriously complex and difficult to fully characterize. The insights and techniques are general and can be applied to any of the file systems.

The I/O and file system infrastructure of an HPC system is often a significant fraction of its total expense [11], and understanding both its workload and how well it handles that workload allows us to gauge how well balanced the I/O resources are to the needs of the system as a whole.

Accurately measuring the peak performance of a file system is not a trivial task. The most common measure of performance is its *peak bandwidth*, which leaves out any consideration of its metadata performance or other architectural limitations*. The peak write bandwidth can be very high when the data is only delivered to an intervening cache, so a fair measure of the peak bandwidth should be for writes that make it all the way to disk. Similarly, the peak read bandwidth can be very high when the data is actually being read from the previously written cache. Again, the fair measure of peak read bandwidth should be for reads that are coming all the way from disk.

Similarly, the most common metric for file system load is the ratio of *observed bandwidth* to peak bandwidth. Using the observed bandwidth as a metric for the load on the file system can easily under-report the actual load. Poorly organized I/O can lead to an observed

*For example, on a BlueGene system each I/O node serves a distinct set of compute nodes, so the full bandwidth is only available to an application running at sufficient scale to drive all its I/O nodes.

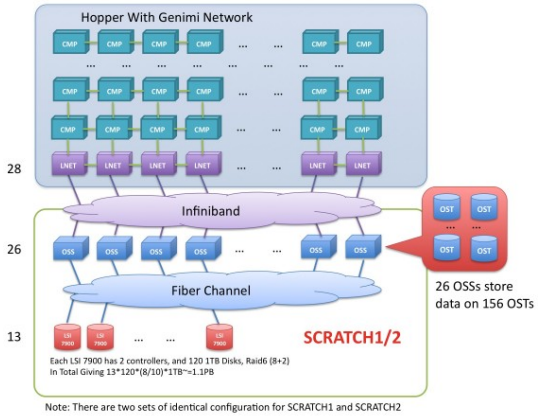


Fig. 1. Hopper compute nodes (CMP) communicate across the Gemini network to LNET Routers and from there by infiniband to the external OSS nodes. Each OSS has six OSTs, and each OST is the service by which the OSS serves a RAID array (LUN) of actual disks. Not shown in this diagram is a controller between the Fibre Channel and the disks. Each controller is responsible for six LUNs, and the six OSTs on an OSS map to six different controllers.

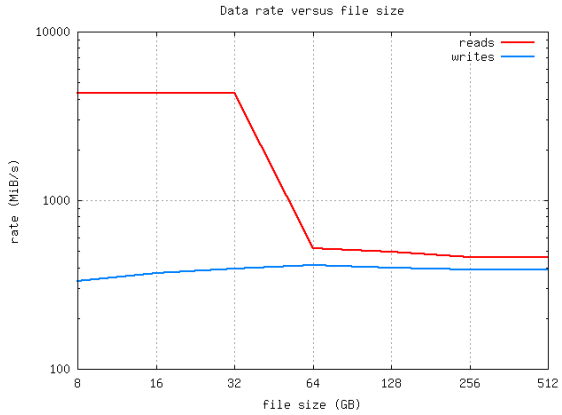


Fig. 2. A graduated sequence of IOR benchmark tests seeks to establish the peak read and write bandwidth. The values (y-axis) for writes (blue) are relatively consistent over the range of file sizes (x-axis), so write caching was not distorting the results. The read rates (red) drop precipitously when the file size exceeds cache (on the server in this case). The fair value is the lower asymptotic limit, not the very large values for smaller files.

bandwidth well below the peak even when there is no additional bandwidth available. One way this can happen is if I/O transactions to disk occur in small fragments with a significant disk seek between each fragment. The *achievable* bandwidth is the peak bandwidth for a given, specific pattern of I/O.

In this paper we will exploit a data source in the `/proc` file system that tells us what the pattern of disk I/O is. That extra information will allow us to create an improved metric, the File System Utilization (FSU), that better characterizes the load on the file system. With that metric on the Hopper Cray XE6, we will see that the load on the file system is, in fact, much higher than would have otherwise been apparent. In addition to how busy the file system is, this new data source reveals how well the file system is being used. We will conclude with a comparison of the workload on Hopper's `/scratch` and `/scratch2` file systems.

A. The Hopper Cray XE6

The Hopper Cray XE6 [2] at the National Energy Research Scientific Computing (NERSC) center at Lawrence Berkeley National Laboratory consists of 6384 nodes (153,216 cores) and $212TB$ of memory. Hopper has a peak performance of 1.28 Petaflops. The nodes are connected in a 3D torus by the "Gemini Network". There is an external storage cluster employing the Lustre parallel file system to provide scratch space. Figure 1 shows I/O Nodes on the "edge" of the torus acting as Lustre network (LNET) routers and connecting the compute nodes to the Object Storage Servers (OSSs) via an Infinband network.

The Hopper `/scratch` and `/scratch2` resources are built from LSI RAID units. Each RAID unit has a controller, and the controller is responsible for providing six block devices or Logical Units (LUNs) from the disks in its RAID unit. Each LUN is mounted on one of the OSSs via a Fibre Channel link. The LUNs on one controller are distributed to six different OSSs. A LUN is served by an OSS via an Object Storage Target (OST). That is, the OST is a service running on the OSS. The `/scratch` file system is constructed from 156 LUNs/OSTs on 26 servers and 13 controller pairs for a total of $1.1PB$ of storage. The peak bandwidth of `/scratch` is $35GB/s$. The `/scratch2` file system is identically constructed. There is also an arrangement for fail-over between controllers and servers. Finally, a separate RAID array and MetaData Server (MDS) combination are responsible for the Lustre name space and the mapping of Lustre objects to their OSSs.

B. The IOR Benchmark

The IOR [12] parallel file system benchmark is commonly employed to document peak file system band-

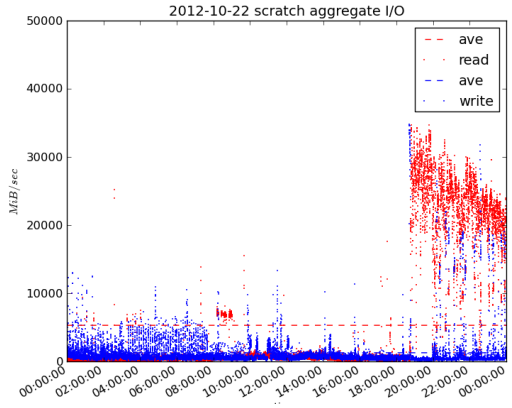


Fig. 3. The Lustre Monitoring Tool (LMT) observes the bandwidth on the Hopper XE6 */scratch* file system every five seconds. 24 hours of data on October 22, 2012, show the bandwidth is not very high until after about 7:00 PM. Prior to that, was the bandwidth low because there was little demand for I/O, or was there a pattern of I/O reducing the achievable bandwidth?

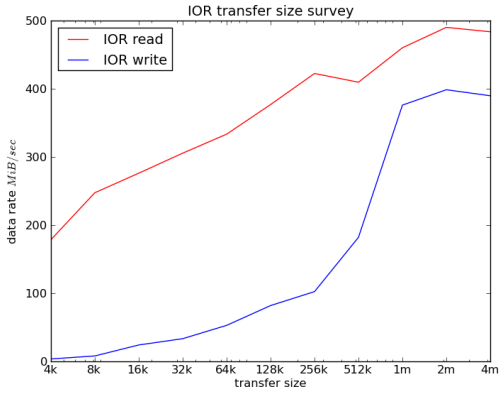


Fig. 4. IOR benchmark tests with varying transfer sizes show a penalty, especially for writes, with smaller transfers. The tests were constructed to have a random seek between each transfer.

width. In doing so, it is important to recognize pitfalls in measuring peak bandwidth and account for details of the system architecture. Figure 2 shows one example of documenting such a feature. In these experiments, an IOR benchmark test writes to, and reads from, files on a single OST via a single OSS. The aggregate file size for the test is increased (x-axis) from $8GB$ to $512GB$. The observed bandwidth (y-axis) is fairly consistently around $400MB/s$ for writes, but falls from over $4000MB/s$ to around $450MB/s$ for reads when the aggregate file size is larger than $32GB$, which happens to be the size of the OSS's memory. The value for read bandwidth that is more representative of what happens with many users is the asymptotic value of around $450MB/s$ reached for large files, not the larger value for smaller target files. All of the results in the rest of this paper are for the asymptotic behavior at large scale.

C. The Lustre Monitoring Tool (LMT)

Each Lustre server (OSS or MDS) maintains a set of `/proc` entries giving information about the internal state and recent performance of the file system. The Cerebro [9], [17] data transfer daemon runs on each server and communicates with an external (to the storage cluster) database server. The Lustre Monitoring Tool [10], [17] (LMT) is a collection of plug-in modules for Cerebro that will harvest `/proc` entries on the servers and save them in the database. Of particular interest is the `/proc/fs/lustre/obdfilter/<OST>/stats` file. In that file are two counters, READ_BYTES and WRITE_BYTES. As counters, the values of these two 64-bit quantities only increase. They measure the number of bytes written (respectively read) on a given OST since the OSS was booted (when they are set to zero). LMT reads these values every five seconds. The differential of those observations produces a time series of observed bandwidth for reads and writes. These time series are a high resolution view of the behavior and performance of the file system components. There is a similar set of counters collected on the MDS and the LNET routers, though this paper will not discuss them further. There are other sources of data on the OSSs not collected by LMT, and that will be the subject of Section IV.

Figure 3 shows 24 hours of data collected by LMT on the Hopper `/scratch` file system. The time series of the read and write observations over the 156 OSTs are summed up and plotted with time on the x-axis. The observed bandwidth (red for reads and blue for writes) is plotted on the y-axis. Each dot is one of the observations taken every five seconds, so there are $17,280$ observations of read bandwidth and $17,280$ observations of write bandwidth in a 24 hour period.

II. I/O Bandwidth as a Proxy for File System Load

There are other factors besides the aggregate amount of data being read or written that determine the delivered performance from an HPC file system. Figure 4 presents a series of IOR benchmark tests that establish the bandwidth, in the asymptotic limit, for a range of

transfer sizes. In this case special attention was given to using IOR options that arrange for each transfer to be followed by a random seek within the file. Two effects have been widely documented (see for example [15]), and are shown in this series of tests. First, that sequential I/O - that is, I/O without random seeks - will perform better, and second, the performance penalty for random seeks is most pronounced for very small transfer sizes. It is this effect we will examine closely in the rest of this paper.

The considerations in running the tests include:

- Do enough I/O so that we are sure that we are not measuring the bandwidth to or from an intervening cache.
- Use enough compute nodes so that we are sure the limit is not on the compute node or network before getting to the server.
- Induce reads from a different compute node from the one(s) that did the corresponding writes, again so we can avoid cache effects.

It is important to understand that when I/O with small transaction sizes and random seeks has a sub-optimal achievable bandwidth, the file system is nevertheless fully engaged. That is, there is no more bandwidth available. Another IOR run during an experiment like those in Figure 4 will compete for file system bandwidth and each will be delayed by the interference from the other.

There is a limit to the maximum rate that data can be moved to disk under ideal circumstances. That is its peak bandwidth. There is also a limit on the frequency of disk transactions that can be carried out. This is the peak I/O Operations per Second (IOPS [3]) rate. The series of experiments in Figure 2 provide an estimate for the former, and the sequence of experiments in Figure 4 provide an estimate for the latter. These experiments, and others like them, will be the basis for our new File System Utilization metric (FSU).

Figure 5 shows the LMT data collected during a sequence of tests like those in Figure 4. The sequence of tests is for five IOR runs. In each run $128GB$ is written via a sequence of transfers of a given size, and each transfer is followed by a random seek. Once the write phase completes the data is read back in using the same transfer size and with each read followed by a random seek. This repeats for five IOR tests. In each test the transfer size is half what it was in the previous test. The tests shown are for $\{4MB, 2MB, 1MB, 512kB, 256kB, 128kB\}$. Additional tests with smaller transfers continued to show the same pattern: Write performance decreases in direct proportion to the transfer size, and read performance decreases, but not quite as badly. We'll return to a discussion of this pattern in Section IV.

The foregoing experiments were all run on a small test system (Grace) and targeted one file system resource (OST/OSS). Similar experiments run on the full Hopper

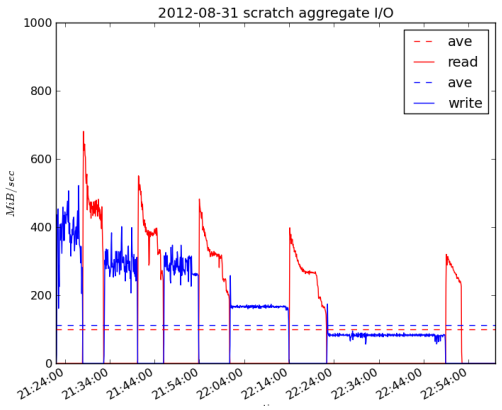


Fig. 5. A time series of observations from the LMT DB show the observed bandwidth, at five second intervals, on the y-axis. Time is on the x- axis. The tests correspond to some of those reported in Figure 4. As the transfer size becomes progressively smaller, the achievable performance decreases.

system [1], [6] documented a peak bandwidth in the vicinity of $35GB/s$. Figure 3 shows LMT data collected on Hopper for the 24 hour period on October 22, 2012. If we use observed bandwidth as a proxy for file system load then we conclude that the file system was relatively idle until around 7:00 PM. On the other hand, if the I/O prior to 7:00 PM had small transfers then the load might actually be high. In order to better measure the actual load on the file system we developed the FSU metric to account for the the pattern of the I/O's activity as well as its volume.

III. An I/O Transaction Model

The simplest way to understand the influence of I/O load on achievable bandwidth is with a model for how I/O transactions take place. Imagine that each I/O transaction requires a small fixed *latency* time as well as a *communication* time that is proportional to the size of the I/O:

$$t = a_0 + a_1 b \tag{1}$$

where:

- t is the time to complete the I/O
- a_0 is the latency
- a_1 is the communication time, per byte
- b is the size of the transaction, in bytes

This very simple model hides many complexities: the small fixed cost is itself going to be a composite of disk

rotational latency, and how often the disk head must seek to a new track. The I/O queuing algorithm (eg. "elevator" [4]) will influence how far the new track is from the current one, from one transaction to the next. Furthermore, an actual disk I/O subsystem will generally post many I/O transactions and have many "in flight" at once. Thus any value for a_0 we may derive is, at best, an "average" value over a range of scenarios. We can rearrange Equation 1 to exhibit the achievable bandwidth B:

$$\begin{aligned} B(b) = b/t &= \frac{b}{a_0 + a_1 b} \\ \hat{B} = B_{(b\to\inf)} &= 1/a_1 \\ IOPS = \left(\frac{1}{t}\right)_{(b\to 0)} &= 1/a_0 \end{aligned} \tag{2}$$

Thus Equation 1 not only gives us the dependence of achievable bandwidth on the transaction size in Equation 2, but also tells us how the coefficients a_0 and a_1 relate to the disk channel bandwidth $\hat{B}$ and the IOPS.

The constant and linear coefficients may be different for reads than for writes, so we separate them into Equations 3.

$$\begin{aligned} T_W(b) &= N_W(b)\,(w_0 + w_1 b) \\ T_R(b) &= N_R(b)\,(r_0 + r_1 b) \end{aligned} \tag{3}$$

where:

- $b \in \begin{pmatrix} 4k, 8k, 16k, 32k, 64k, 128k, \\ 256k, 512k, 1M, 2M, 4M \end{pmatrix}$
- $T_W(b)$ is the observed time to completion for the test using transfers of size b, and similarly for $T_R(b)$.
- $N_W(b)$ is the number of I/Os of size b in the test, and similarly for $N_R(b)$.
- w_0 and w_1 are the constant and linear coefficients for write I/Os, and r_0 and r_1 are for read I/Os.

The set of Equations 3 over-determines the coefficients in question and we can perform a simple regression to get best estimates for the individual coefficients: The regression works as follows (for writes): Each combination of $\frac{T_W(b)}{N_W(b)}$ is an estimate for $w_0 + w_1 b$, or equivalently, we have eleven equations: $\frac{N_W(b)}{T_W(b)}(w_0 + w_1 b) = 1$. That system of equations can be organized as a single matrix equation:

$$\mathbf{X}^W \vec{w} = \vec{1} \tag{4}$$

where:

- $X^W_{b,0} = \frac{N_W(b)}{T_W(b)}$, the IOPS observed for writes of size b
- $X^W_{b,1} = \frac{N_W(b)}{T_W(b)} b$, the transfer rate observed for writes of size b
- $\vec{w} = [w_o, w_1]$, the vector of coefficients for writes
- $\vec{1} = [1, \ldots, 1]$, a vector of ones.

We have the values for $\mathbf{X}^W$, so we solve for $\vec{w}$. The same procedure applies for reads.

Using those coefficients in Equations 3 we could calculate the time our simple model predicts for each test. Doing so produces a poor match, though, because the foregoing values for $N_W(b)$ and $N_R(b)$ may not correspond to what actually takes place between the server and the disk.

The Figures 4 and 5 show a definite effect on achievable bandwidth from using smaller transaction sizes, but we need an accurate count of the number of I/Os of each size between server and disk. In the next section we show how to get that information.

IV. Extended Monitoring

The IOR experiments in the previous section were designed to interleave I/O operations with random seeks, so that each experiment would be a distinct combination of $w_0 + w_1 b$ (respectively for reads). The IOR `write()` system calls communicate with Lustre on the client nodes, Lustre gathers together small writes into larger communications from the client to the server (so called *RPCs*) when it can, and sends those to the servers. Those RPCs may contain multiple actual writes, and those writes are then sent to the disk scheduler, which implements an "elevator" algorithm in this case. Thus there isn't a strong guarantee that strict interleaving of, say, $4k$ of I/O with a random seek will be what actually happens. In the case of reads, a $4k$ request will commonly result in a much larger disk I/O operation, as the controller and server both try to anticipate future requests and maintain cached values. If the backlog of writes in the disk scheduler is large enough then small writes can also be combined, though in practice that is unusual. The IOR write test results from Figure 4 do seem to follow the expected pattern, especially for the smaller transfers.

Since the server and controller are actively attempting to read more than the small amount requested in the IOR read test, there can be two effects. First, the server can get lucky, reading a full $1MB$ when, say, only $4kB$ was requested, but then having the remaining $1MB - 4kB$ already in memory when later $4kB$ requests for that data arrive. This improves performance by skipping the disk interaction entirely for the cached $4kB$. Second, memory pressure on the server may cause the $1MB$ read to be evicted before it gets used so that the same $1MB$ has to be read yet again when a nearby $4kB$ is requested. This decreases performance because some disk activity reads bytes that are never sent to the client. From the results of Figure 4 it is evident that the first case dominates.

Without knowing the precise activity taking place on the server to disk communication channel it is difficult to infer precisely what is happening or make any claims about the values of the coefficients, especially r_0 and r_1. Luckily, the details of the disk I/Os are available in the `/proc` entries maintained by Lustre. In particular, the OSS file `/proc/fs/lustre/obdfilter/<OST>/brw_stats` for a given OST has seven histograms governing the "back-end read/write statistics". The histogram called IOSIZE maintains a count of the number of transactions from server to disk in each of the nine size bins $\vec{b} = \{4k, 8k, 16k, 32k, 64k, 128k, 256k, 512k, 1M\}$ bytes.

A new Cerebro module, written by one of the authors [5], gathers the `brw_stats` histograms into the LMT database. That module has been on Grace since August, 2012, and on Hopper since October. Thus, we have a direct observation, at five second intervals, of the values for N_W and N_R from Equation 3.

What we find for the IOR write tests† from Figure 4 is that the number and size of the I/Os to disk are exactly what we expect for the first nine tests. For the tests with $2MB$ or $4MB$ I/O the `brw_stats` histogram also shows $1MB$ disk transactions. There are no histogram bins for larger disk I/Os, and that leads to an ambiguity. The $1MB$ bin could stand for "I/Os $1MB$ and larger". If so, one would expect the count for the $2MB$ IOR write test to be half that for the $1MB$ test, and similarly for the $4MB$ test. In fact, we find is that the counts are the same, thus the larger writes from the client have been broken into $1MB$ writes by the server. There are no I/Os larger than $1MB$ at the disk.

The case for the IOR read tests is very different. Figure 6 shows the histogram of read I/Os from the disk. In the IOR run shown, the test read $8GB$ in $4kB$ `read()` calls with random seeks between. About 5000 of those calls resulted in $4kB$ reads from disk, but most of the disk traffic was for larger sizes, as the server and controller tried to read ahead and or read nearby data in the hopes of having the requested data in memory when subsequent reads arrived. Overall, this worked well and the read test ran about 30 times faster than the corresponding write test. On the other hand, there was actually about $9.5GB$ of disk reads during the test, so almost 20% of the bytes were discarded unused. This effect happened for every read test at every transfer size. The only difference is that request sizes larger than $4kB$ never got subdivided. I.e. the $8kB$ test looks like Figure 6 except with an empty $4kB$ bin, and so on.

With the correct values for $N_W(b)$ and $N_R(b)$ we can

†Detailed results for the seven `brw_stats` histograms from all of the IOR tests on Grace are available at http://portal.nersc.gov/project/pma/grace/ior_test/brw_stats_survey.html

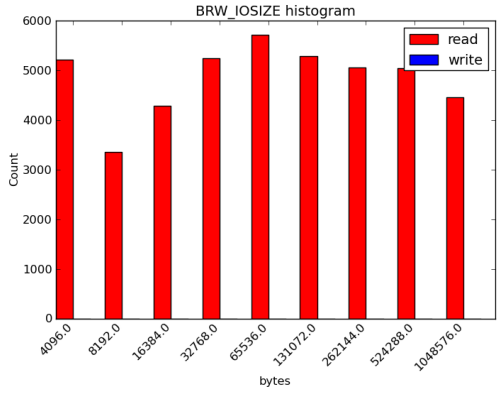


Fig. 6. This histogram shows the relative counts (y-axis) of server-disk read transactions during the IOR test with $4kB$ reads. The x-axis shows the nine bin sizes for which data is collected. The IOR test was supposed to read $8GB$, but the actual amount read was $9.5GB$. Nevertheless, the test ran 30 times faster than the write test. There were no writes during the read test, and the corresponding write test for $4kB$ I/Os did exactly the number of $4kB$ I/Os to disk that would be expected.

	latency (sec)	rate^{-1} ($sec/byte$)	rate (MB/sec)
write	$1.04e^{-03}$	$1.73e^{-09}$	551
read	$8.02e^{-04}$	$1.23e^{-09}$	775

TABLE I. IOR experiments at each of eleven transfer sizes give us estimates for the two coefficients for write transaction time, and similarly for read transaction time.

then return to Equation 4 to get a better estimate of the coefficients. Table I lists the result of this regression. Since there may actually be a I/Os of many different sizes, both read and writes, during an interval, we generalize Equations 3 to give Equation 5 for the estimated time to completion of an arbitrary pattern of I/O:

$$T(\vec{\mathbf{N}}) = \sum_{b \in \vec{b}} \begin{pmatrix} N_b^W(w_0 + w_1 b) \\ +N_b^R(r_0 + r_1 b) \end{pmatrix} \quad (5)$$

where:

- $\vec{\mathbf{N}}$ is all of the $N_W(b), b \in \vec{b}$ and $N_R(b)$

Similarly the achievable bandwidth, given a pattern of I/Os described by $\vec{\mathbf{N}}$, is:

$$B_{achievable} = \frac{Bytes(\vec{\mathbf{N}})}{T(\vec{\mathbf{N}})} \quad (6)$$

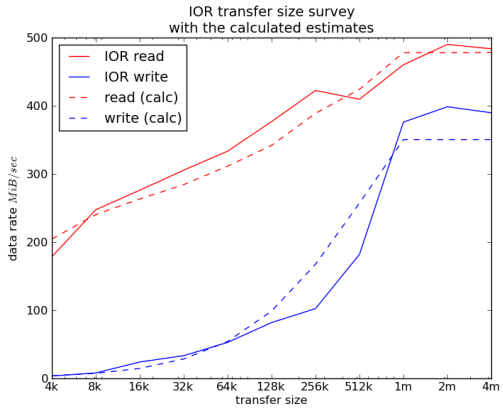


Fig. 7. Solving the regression in Equation 4 gives us estimates for the coefficients w_0, w_1, r_0 and r_1. Using those results, and the observed pattern of I/O for each experiment, in Equation 6 allows us to compare the experimental and estimated values for the achievable bandwidths.

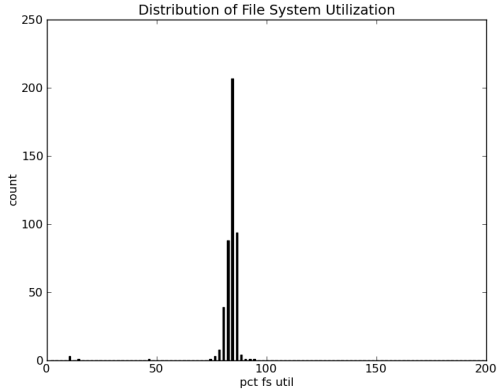


Fig. 8. A histogram of the calculated percent utilization of the targeted OST during the IOR test writing $4kB$ transfers. Note that the peak is a little below 100%. The calibration at this point is still a little imprecise due to the simplicity of the model and the many other factors affecting performance.

where:

- $Bytes(\vec{\mathbf{N}})$ is just the total number of bytes in $\vec{\mathbf{N}}$.

Figure 7 presents the experiments from Figure 4 and the corresponding estimates for achievable bandwidth we calculate using Equation 6 and the patterns of I/Os observed during the tests.

With the coefficients in Table I and the time series we have collected during these tests we can solve Equation 4 in the forward direction $\hat{y}_W = \mathbf{X}^W \vec{w}$ (respectively $\hat{y}_R = \mathbf{X}^R \vec{r}$). Now, $\hat{y}_W$ is a series of estimates of how busy we expect the file system to be based on the size and quantity of observed I/Os. This sequence of values should be a noisy distribution around the value 1 during a test that we know was running the OST as fast as it could go. For example, $\hat{y}_W$ for the IOR write test with $4kB$ I/Os is in Figure 8. We scale the $\hat{y}_W$ value to 100 so that it resembles a "percent utilization" number and define the File System Utilization (FSU) as:

$$FSU(\vec{\mathbf{N}}, T) = \frac{100}{T} * \sum_{b \in \vec{b}} \begin{pmatrix} N_b^W (w_0 + w_1 b) \\ + N_b^R (r_0 + r_1 b) \end{pmatrix} \quad (7)$$

where:

- $\vec{\mathbf{N}}$ gives N_b^W, the number of write I/Os of size b over an interval, respectively N_b^R for reads.
- T is the length of the observation interval in seconds.

Anecdotal evidence suggests that any time the the FSU value hovers for any length of time at a value well above zero, the file system (or resource) is fully engaged. Thus the presence of a narrow distribution as in Figure 8 is an indication that the file system is fully engaged despite the value being closer to $FSU = 70\%$.

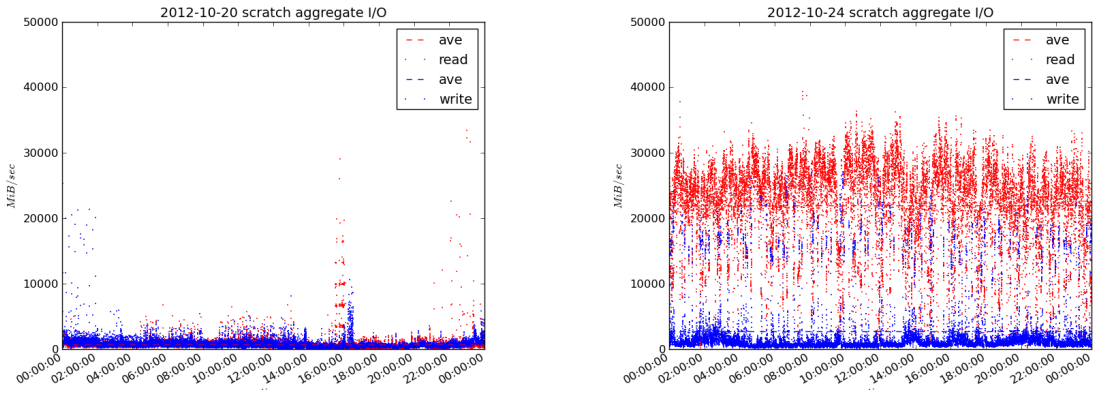
Up to this point the discussion has been centered on presenting and calibrating the FSU metric on a single OST of the Grace test system. In the next section we carry this work to the Hopper system and its two large Lustre file systems.

V. Application of the Model

We ran a similar set of calibration experiments on Hopper OSTs and got a similar set of coefficients. The largest difference was in the $\frac{1}{w_1}$ bandwidth parameter, which was closer to $220MB/s$ (for one OST). This is not too surprising considering Hopper file purge policy tends to be keep OSTs almost always almost full, and full disks have to work harder for write I/O.

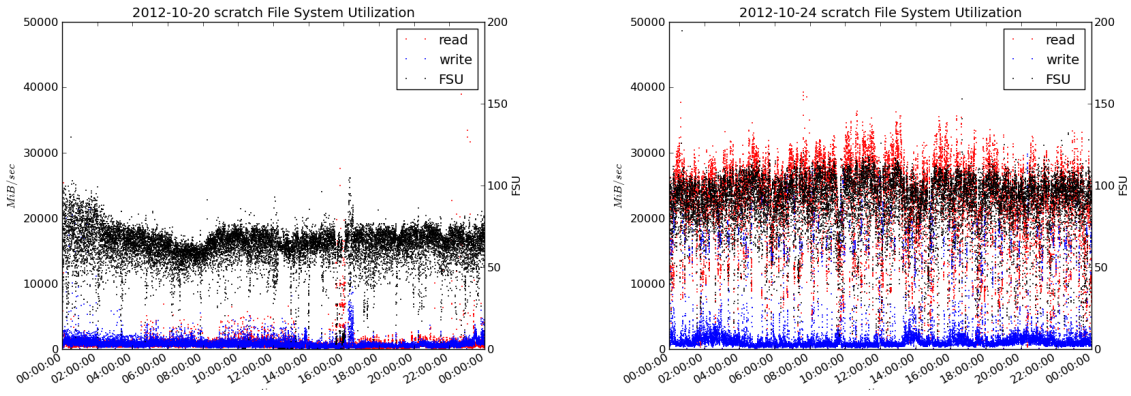
The utilization metric for the file system as a whole is the average of the value across the individual OSTS. The resulting FSU underestimates the actual load on the file system since it does not account for contention for resources between OSTs.

Over the six months from October 2012 to March 2013, the Hopper `/scratch` file system averaged about $140TB$ of reads and $70TB$ of writes a day. The median for 24 hours of I/O was closer to $50TB/day$ for reads and writes, both. We think of a median day as being "typical", and Figure 9(a) shows a day in October 2012 where there was a typical amount of I/O. Two days later Figure 9(b) shows an unusually large amount of I/O, mostly reads. Clearly, the file system was very busy that



(a) typical (b) busy

Fig. 9. The long term median for I/O during a 24 hour period on the Hopper `/scratch` file system is a about $100TB/day$ ($600MB/s$ averaged over the day for writes and reads each), and the 24 hours of LMT data on the left is consistent with that "typical" value. The 24 hours on the right shows `/scratch` running at close to its peak bandwidth all day. The graph on the right certainly shows a very busy day, and we might be tempted to conclude that the day on the left was relatively idle.



(a) typical (b) busy

Fig. 10. The same two days are shown here with the FSU values superimposed in black. We were obviously correct that the day on the right was very busy, it turns out the day on the left was also very busy. Considering the way that the FSU value stays consistently high and near a consistent value over the course of the day we may speculate that it shows a file system that had no more bandwidth available.

day, and it would be tempting to conclude that the day on the left shows a relatively idle file system.

In Figure 10 the FSU value for the `/scratch` file system is superimposed in black. Unsurprisingly, the FSU is near 100% all day in Figure 10(b). Figure 10(a) shows the file system was very busy despite the low observed bandwidth. The fact that the FSU both stayed high and stayed near a consistent level suggests that the `/scratch` file system was actually fully utilized during the day. That is, there was no additional bandwidth available.

The two days in Figures 9 and 10 were chosen to illustrate a point, and one may wonder how often is the FSU value high. We have 240 days of data on Hopper, at five second intervals. Figure 11 presents a histogram of all the FSU observations made to date on the `/scratch` file system. Recall that the FSU is intended to gauge the extent to which the file system is fully loaded, but that it can show values well above 100%, and more critically, values well below 100% can still indicate a fully engaged

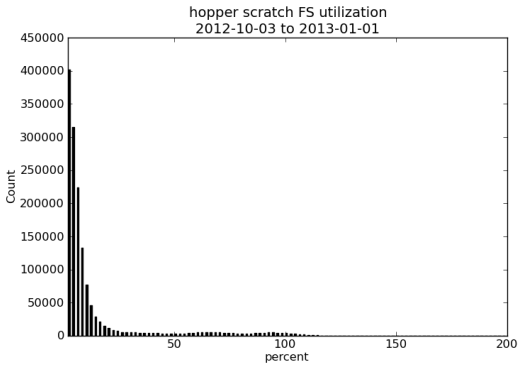


Fig. 11. About a quarter of all FSU observations are in the first bin ($< 2\%$), and about 10% of all observations are high enough to suggest a fully engaged file system. The roughly two thirds of remaining observations fall in between.

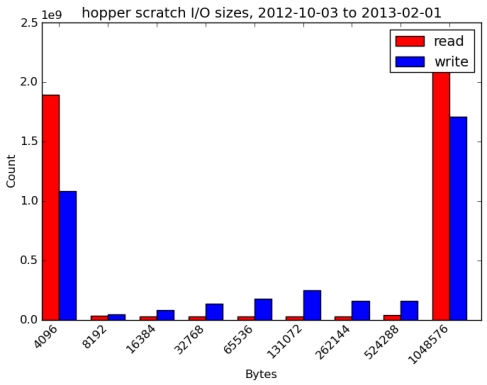


Fig. 12. In the 240 days observation on Hopper `/scratch` the distribution of I/O sizes is nearly evenly split between the optimum $1MB$ transfers to disk and all the other sizes, and those I/Os are dominated by the worst case $4kB$ I/Os.

file system. Each bin in Figure 11 covers 2% of FSU and there are 100 bins from 0 to 200%. The leftmost bin ($< 2\%$) has about a quarter of all the observations. The flat part of the distribution (values above about 30%) accounts for about 10% of all observations, and the remaining $2/3$ of observations fall in between. It is our experience that when the FSU is above around 30% the file system is busy.

Each observation of disk I/O sizes is a histogram over the nine bins from $4kB$ up to $1MB$ by powers of two.

	FSU values	% obs	% opt	% sub
idle	$0 < F < 2$	26	45	55
moderate	$2 < F < 30$	65	70	30
busy	$30 < F$	9	23	77

TABLE II. Divide the FSU observations into the categories *idle*, *moderate*, and *busy*. In each category count how many of the I/Os are optimal versus sub-optimal. `/scratch` is busy about 9% of the time and when it is busy about three fourths of the I/Os are sub-optimal.

Those observations have been taken every five seconds for 240 days, and the composite histogram for that period is in Figure 12, with the count of read I/Os in red and write I/Os in blue. There is a $60/40$ split between I/Os of the optimum $1MB$ size and all other I/Os. Of the sub-optimal I/Os, most are $4kB$, which is the worst case size for performance. As reflected in Figure 7, any I/O below $1MB$ is sub-optimal.

Table II further breaks down the relative impact of sub-optimal I/Os on file system utilization. It presents the count of optimal versus sub-optimal I/Os in each of the previous categories, where *idle* is $FSU < 2\%$, *moderate* is for FSU up to about 30%, and *busy* is anything above that. When the Hopper `/scratch` file system is busy about three-quarters of its I/Os are sub-optimal. Thus the impact of sub-optimal I/O is especially acute exactly when a simple bandwidth metric would not show it.

A. Compare `/scratch` with `/scratch2`

The Hopper scratch I/O resources were designed to have two similarly sized file systems, `/scratch` and `/scratch2`. This was in part to increase the metadata performance, since two file systems would have two metadata servers (MDSs) splitting the load. Another reason for this split was to create a separate resource that was not the default I/O target. It was hoped that applications with especially stringent I/O needs might get better performance in an environment with fewer competing applications. The notion was that heroic I/O was uncommon, so it would be likely that such a job would have nearly the entire resource to itself.

The average read I/O traffic to `/scratch2` is about half that to `/scratch` and the average write traffic is about three-quarters. Table III shows the corresponding breakdown of I/O on `/scratch2` for idle, moderate, and busy times. In this case the I/Os tend to be optimum $1MB$ transactions in every category and that is especially the case when the file system is busy. This reflects the deliberate effort to have heroic I/O jobs use `/scratch2` where they are less likely to interfere with each other. It

	FSU values	% obs	% opt	% sub
idle	$0 < F < 2$	62	78	22
moderate	$2 < F < 30$	34	85	15
busy	$30 < F$	4	95	5

TABLE III. For `/scratch2`, a breakdown of the I/O sizes along the lines in Table II shows that it is busy about half as often and idle twice as often. For every category, but especially when it is busy, the I/O is dominated by optimum $1MB$ I/Os.

also shows that for applications where I/O is important the developers have taken some effort to ensure optimum behavior.

In addition to the file system utilization metric, the histogram of disk I/O sizes provides another dimension for characterizing workload. The LMT observations reveal the volume of data created by a workload, and the LMT extension allows us to see when the work load uses the file system well, as on `/scratch2`.

VI. Conclusions and Future Work

In this paper we have used the IOR benchmark and the LMT data source in a traditional way to establish the peak bandwidth of the scratch file system resources on Grace and Hopper. With those tools we also documented the effect of I/O transfer size on the achievable bandwidth. Those experiments demonstrate that observed bandwidth is not a proxy for file system load. We calibrated a simple model for I/O transactions using the IOR results and saw that effort fail for the same reason we can not use bandwidth alone to characterize workload. We must account for the smaller transactions at the disk.

With an extension to LMT of our own design we began collecting detailed data on the sizes of the I/Os from server to disk. With that new data source we were able to improve the calibration of the model from the experiments. This gave us a detailed view of how disk transaction size actually relates to achievable bandwidth. That lead us directly to a new File System Utilization metric, FSU. With the FSU metric we can then better characterize the workload on the Hopper scratch file system resources. We find that Hopper's `/scratch` file system is busy about 10% of the time, but three quarters of that time the load is from sub-optimal I/Os. The traditional view of system load, based only on observed bandwidth, would underestimate the load by a factor of four. This result has consequences for our ability to properly balance the file system in comparison to the rest of the HPC system (memory, node count, flops, etc).

We also find that the additional detail on the file system workload allows us to better quantify the difference in the workload between `/scratch` and `/scratch2`, which has less total I/O and much less sub-optimal I/O.

There are many ways this work can be extended and improved. A more systematic and thorough set of experiments would improve our estimates of the model's coefficients. A series of such experiments could also document variance between OSTs, and variance over the life time of the OST, especially as it fills. The model itself could be refined to include resource contention at the servers and controllers as well as including additional data sources. Finally we'd like to integrate this work with a similar model for load on the metadata server.

VII. Acknowledgments

We'd like to thank NERSC staff for valuable discussions of these issues and these results.

The authors were supported by the Office of Advanced Scientific Computing Research in the Department of Energy's Office of Science under contract number DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under that contract.

References

[1] File Storage and I/O.
[2] Hopper.
[3] IOPS. http://en.wikipedia.org/wiki/IOPS.
[4] IOPS. http://en.wikipedia.org/wiki/Elevator_algorithm.
[5] Uselton's Extension to the Lustre Monitoring Tool (LMT). https://github.com/uselton/lmt.
[6] K. Antypas and Y. He. Transitioning users from the franklin xt4 system to the hopper xe6 system. In *Cray Users Group 2011*, 2011.
[7] D. H. Bailey. Twelve ways to fool the masses when giving performance results on parallel computers. In *Supercomputing Review*, pages 54–55, August 1991.
[8] P. Braam. File systems for clusters from a protocol perspective. In *Proceedings of the Second Extreme Linux Topics Workshop*, Monterey, CA, June 1999.
[9] A. Chu. Cerebro. http://sourceforge.net/projects/cerebro. Developed at Lawrence Livermore National Lab.
[10] C. M. Herb Wartens, Jim Garlick. LMT - The Lustre Monitoring Tool. https://github.com/chaos/lmt/wiki/. Developed at Lawrence Livermore National Lab.
[11] J. Hick. The 5th Workshop on HPC Best Practices: File Systems and Archives. http://escholarship.ucop.edu/uc/item/35z307bk.
[12] IOR: The ASCI I/O stress benchmark. http://sourceforge.net/projects/ior-sio/.
[13] R. Latham, N. Miller, R. Ross, and P. Carns. A next-generation parallel file system for linux clusters: An introduction to the second parallel virtual file system. *Linux Journal*, pages 56–59, January 2004.
[14] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *First USENIX Conference on File and Storage Technologies Fast02*, Monterey, CA, January 2002.
[15] G. Shipman, D. Dillow, S. Oral, F. Wang, D. Fuller, J. Hill, and Z. Zhang. Lessons learned in deploying the worldâs largest scale lustre file system. In *Cray Users Group 2010*, 2010.
[16] Top500 Supercomputer Sites. http://www.top500.org.
[17] A. Uselton. Deploying server-side file system monitoring at NERSC. In *Cray User Group Conference*, Atlanta, GA, 2009.