Red Storm IO Performance Analysis

James H. Laros III [#1], Lee Ward [#2], Ruth Klundt [*3], Sue Kelly [#4]
James L. Tomkins [#5], Brian R. Kellogg [#6]

[#] *Sandia National Laboratories*
1515 Eubank SE
Albuquerque NM, 87123-1319

[1]jhlaros@sandia.gov

[2]lee@sandia.gov, [4]smkelly@sandia.gov

[5]jltomki@sandia.gov, [6]brkello@sandia.gov

[*] *Hewlett-Packard*
3000 Hanover Street
Palo Alto, CA 94304-1185

[3]rklundt@sandia.gov

***Abstract*— This paper will summarize an IO[1] performance analysis effort performed on Sandia National Laboratories Red Storm platform. Our goal was to examine the IO system performance and identify problems or bottle-necks in any aspect of the IO sub-system. Our process examined the entire IO path from application to disk both in segments and as a whole. Our final analysis was performed at scale employing parallel IO access methods typically used in High Performance Computing applications.**

***Index Terms*— Red Storm, Lustre, CFS, Data Direct Networks, Parallel File-Systems**

I. Introduction

IO is a critical component of capability computing[2]. At Sandia Labs, high value is placed on balanced platforms. Balance between processor network and memory speed is critical, but without an appropriately balanced IO sub-system a severe bottle-neck could be introduced into an otherwise efficient platform. It is, therefore, critical that the IO sub-system, including the parallel file-system, provide performance at a level sufficient to maintain architectural balance (or as close as current technology will permit). The Red Storm[1] platform was carefully architected to achieve this balance. Our stated performance goal for IO is 50000MB/sec, reading or writing, from application to the parallel file-system[3]. We will determine what impediments, if any, might prevent us from realizing this goal.

Careful analysis of the entire path, from application to disk, must be accomplished. Problems in any individual segment or system component can potentially affect performance. The effort described in this paper, while primarily an IO performance analysis exercise, necessarily considered the impact of many system components. The analysis presented in this paper is ongoing. Any change to a system like Red Storm, whether hardware or software, potentially affects how the entire system performs.

We consider the testing described in this paper to be second phase testing. Initial testing was accomplished to establish baselines, develop test harnesses and establish test parameters that would provide meaningful information within the limits imposed such as time constraints. All testing presented in this paper was accomplished during system preventative maintenance periods on the production Red Storm platform, on production file-systems. Testing time on heavily utilized production platforms such as Red Storm is precious. Detailed results of previous, current and future testing is posted online[2].

In section II we will discuss components of the Red Storm architecture pertaining to this performance analysis effort. A complete discussion of the Red Storm architecture is quite involved, therefore, we will limit the details provided in this discussion. We will likewise limit our discussion of Lustre®[3] file-system internals.

In section III we will begin our analysis by examining the IO sub-system architecture first to determine theoretical bandwidth expectations (section III-A), followed by targeted testing to determine demonstrable expectations of what we anticipate to be a limiting factor in our overall performance (section III-B). This evaluation will be followed by parallel performance testing using IO access methods typically employed by High Performance Computing applications. Performance results and analysis of file-per-process IO will be presented in section IV-B, followed by shared-file IO in section IV-C. Throughout this paper we will provide both observations and lingering questions instigated by our findings. Section V will present some final observations based on our testing as a whole. Finally, we will present some future work related to this topic in section VI.

II. Red Storm Architecture

The Red Storm platform is comprised of 12960 nodes each

[1] IO is used throughout this paper, as in the field, to depict Input Output.

[2] Systems designed to support applications that use a significant fraction of the total resource per job.

[3] This requirement resulted from calculating the time necessary to write the entire contents of system memory (in the original configuration) to disk.

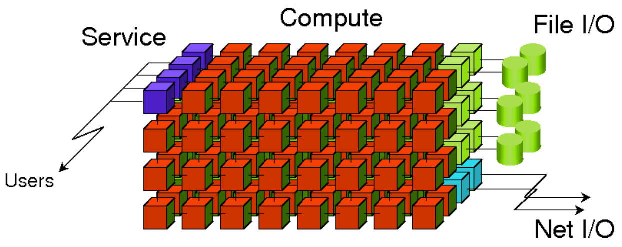

Fig. 1. Red Storm (Logical) Architecture

with a 2.4 Ghz Opteron dual core processor (25920 processors) connected in a 3D mesh topology. The logical partitioning of Red Storm is depicted in figure 1[4]. The minimum bisection bandwidth of the interconnect is 3.6 TB/sec. The bandwidth of an individual link between two nodes is 2.1 GB/sec[4], unidirectional. For the purposes of this paper we will describe nodes as either compute or IO. Each compute node runs a Light Weight Kernel (Catamount[5], [6]) designed to deliver the maximum amount of compute resource to the application, in our case the IO performance analysis application. Each IO node runs a Cray[7] modified version of SUSE[8] Linux$^{\mathrm{TM}}$[9]. It should be noted that the Linux nodes are currently not taking advantage of the available second core. Each IO node hosts a Qlogics[10] 2300 2Gb/sec dual port fiber Host Bus Adapter (HBA). Each port of the IO nodes HBA is individually capable of 2Gb/sec and is connected to a 2Gb/sec port on a Data Direct Networks (DDN)[11] S2A8500 Disk Controller. The DDN controllers are configured into couplets for various reasons including fail-over capabilities that will not be discussed here. Each DDN controller has four available ports which equates to eight ports per DDN couplet. Four IO nodes are connected to each DDN couplet. Figure 4 depicts the connectivity between four IO nodes and a DDN couplet. This configuration is repeated throughout the IO subsystem.

Internally, the DDN controller (detailed in figure 2) has ten disk channels; eight data (A-H), one parity (P) and one spare (S). Each channel is rated at 1Gb/sec. The four external DDN ports are numbered 1-4. Currently we are using Seagate[12] ST373307FC (and similar) fiber channel disk drives. The specified rates for these drives are: minimum - 43MB/sec, average - 66MB/sec and maximum - 78MB/sec[13]. In our configuration there is one drive per data channel per controller port. The eight disks are aggregated and exported as a single LUN[5] per controller port. Ultimately, the LUN is seen as a partition which is used by the parallel file-system. For the sake of completeness figure 2 also depicts the couplet connections used to allow access to LUNs from the other controller in the couplet. This functionality is not exploited in our configuration and will not be discussed in this paper.

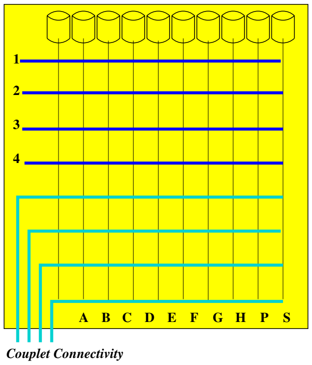

Fig. 2. Disk Controller Internals

The parallel file-system employed on Red Storm is Lustre[6]. In brief, two Lustre file-systems were used in our testing. Both file-systems were configured such that two Object Storage Targets (OSTs) exist on each Object Storage Server (OSS or IO node). By default, OSTs are allocated one per OSS until the maximum number of OSSs are reached. Once one OST on each OSS is allocated the allocation process begins again and assigns the remaining OST on each OSS in the same order. While two different file-systems were used, the OST allocation was carefully controlled to eliminate, or minimize, any potential variability in test results. This information is site specific and is important in evaluating our performance testing[7]. When significant, the maximum number of OSTs, or OSSs that are used will be specified. For the following tests a more detailed discussion of Lustre is beyond the scope of this paper. We will, however, describe additional components or characteristics of Lustre as needed to understand our testing methodology.

III. Available Bandwidth

Based on our description of the Red Storm architecture (as it applies to this effort) we can calculate theoretical limits that we can expect not to exceed. With this knowledge we can also evaluate what segment of the overall IO path we expect to be the limiting factor in performance which will help us interpret the results of our parallel tests.

A. Theoretical

To arrive at theoretical estimates for our configuration we start with a single end to end path that an IO operation travels. If we coarsely represent the segments that an IO operation travels, it would look something like figure 3. As noted in section II and depicted in figure 4, each disk controller will service four IO nodes. We must also determine if any of the internal controller component rates themselves effect the

[4] Physical layout can be found at http://www.cs.sandia.gov/platforms/images/RedStormDiagram1.jpg

[5] Logical Unit Number. Exportable, logical partition. Each LUN is seen by the operating system as one drive.

[6] Cray integrated version of Lustre for the XT3-4 platform, version 1.4.6.8

[7] On our system, each Lustre OST is assigned a single DDN LUN for data storage. Lustre software distributes file data across several OSTs, in portions of equal sizes known as stripes. The number of OSTs used to stripe the file data is defined by the stripe number, which is user configurable. The OSS node is a server where one or more OSTs are attached, and where the software controlling data storage is running. A separate software stack on the MDS (MetaData Server) manages metadata operations, and stores the list of OSTs which hold file data for each file as well as the usual file attributes.

bandwidth that a controller can deliver when all four ports are used in aggregate (ref. figure 2).

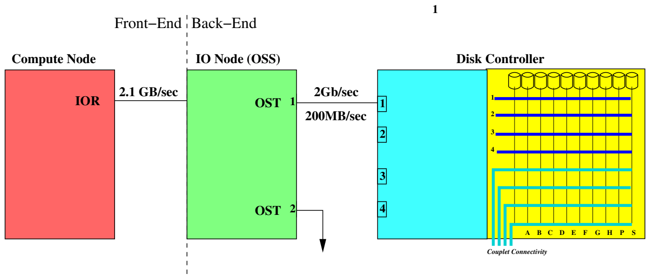

Fig. 3. Single End-to-End IO Path

1) Single Data Path (Theoretical): For the single data path case depicted in figure 3, the link between the compute node and the IO node far exceeds the capability of one port on either the HBA or the disk controller. Assuming the compute node can deliver in excess of 200MB/sec to the IO node we can focus on the bandwidth from the IO node through to disk (back-end). (We will perform tests specifically designed to illustrate that sourcing data from the compute node to the IO node (front-end) is not a limiting factor in our performance in section IV-A). First we should explain the 200MB/sec rate depicted in figure 3. As stated in section II both the Qlogics HBA and the ports on the DDN controller are 2Gb/sec fiber channel ports. Fiber channel uses 8b/10b encoding which requires 10 bits to transmit 8 bits resulting in a 200MB/sec maximum rate ($\left(2 \times 10^9\right) \div 10 = 2 \times 10^8 or\, 200MB/sec$). In the case of a single compute node performing IO over a single port of the HBA we assume that all of the available bandwidth of the controller can be used. The following calculations are used to arrive at theoretical data rates (ref. figures 2 and 3).

- Single port of IO Node HBA @ 2Gb/sec ≈ 200MB/sec max bandwidth
- Single DDN Controller port @ 2Gb/sec ≈ 200MB/sec max bandwidth
- DDN data channels (A-H,P,S) @ 1Gb/sec ≈ 100MB/sec max bandwidth per channel
- 8 DDN data channels (A-H) ∴ 800MB/sec max aggregate data bandwidth

As stated in section II the rate of the disk drives in our configuration range from a minimum of 43MB/sec to a maximum of 78MB/sec. In this single data path analysis all of the aggregate bandwidth of the controller data channels is at our disposal, therefore, even the maximum data rate quoted for the disk drives in our installation throttles the performance of a single data channel path since each channel can only support 100MB/sec. The following calculations show the minimum, average and maximum data rates that can be obtained from disk aggregated over the eight data channels.

- 43MB/sec/disk × 1 disk/channel × 8 channels = 344MB/sec (min)
- 66MB/sec/disk × 1 disk/channel × 8 channels = 528MB/sec (avg)
- 78MB/sec/disk × 1 disk/channel × 8 channels = 624MB/sec (max)

While even the maximum rate the disks can deliver in aggregate (624MB/sec) is less than what the 8 disk channels can support (800MB/sec) the controller port (or the HBA on the IO node) is still the throttling factor in the single end-to-end IO path (theoretically). Since there are two ports per HBA on each IO node we assume the maximum bandwidth we can achieve is 200MB/sec per port.

2) Aggregate Controller Performance (Theoretical): In the tests that follow using a parallel IO application, most involve multiple clients, and controllers. More specifically we use all four ports on a controller in aggregate. We also determine if using all four ports in parallel changes the theoretical limits stated in the previous section. The following calculations are used to arrive at the aggregate controller rates (for a single controller).

- 8 DDN data channels ∴ 800MB/sec max aggregate bandwidth (from above)
- 800MB/sec ÷ 4 ports/controller = 200MB/sec/port/controller

These calculations illustrate the controller data channels are shared (we assume equally) when all four ports of the controller are used. Based on these calculations, the data channels in aggregate support the full port speed of each controller port when all ports are used in parallel. From an individual data channel perspective, the disks are no longer the limiting factor in per data channel performance, even if we use the minimum specified rate. Consider that each controller channel supports 100MB/sec (max). When the entire data channel can be dedicated to a single port, the channel rate is greater than the maximum disk rate specified (per channel). When the data channels are shared, however, each port consumes $\frac{1}{4}$ of the data channel ($100MB/sec/channel \div 4ports = 25MB/sec/channel/port$). Even the minimum specified disk rate of 43MB/sec exceeds the available data channel rate when all four ports of the controller are used in parallel. Again, regardless of the disk speed, the controller port, the HBA on the IO node, or the aggregate data channel capability per port (all now equal) is the limiting factor. Whether we use one port individually or all four ports in parallel the maximum theoretical bandwidth we can achieve between an IO node and a controller should be 200MB/sec.

B. Demonstrable

We feel this testing is necessary for a number of reasons. Documented results are difficult to find that apply to this effort. DDN advertises a performance rate for a couplet ranging from 1.4 - 1.5GB/sec[11]. Based on this figure the per port range is 175-187.5MB/sec. In our informal discussions with DDN about per port performance they stated that scaling from one to three ports should be linear but a small hit is realized when utilizing all four ports in parallel due to various internal controller operations that introduce a small amount of overhead. They also noted the controller should do a good job of balancing this hit across all four ports.

To demonstrate what data rates can actually be achieved on our configuration we will run a number of different tests using the utility sgp_dd[8]. In each case we perform IO directly to the SCSI[9] device layer followed by the file-system layer[10]. Figure 4 depicts the paths that we exercise and connectivity to the DDN controllers. Our first test exercises path A using a single IO node, a single port on the HBA and a single controller. We repeat this test using path B (through the file-system layer). This test demonstrates the (best case) bandwidth for our single data path. We then determine if we can achieve the same rate on both ports of the HBA in parallel from one IO node by using the analogous paths A with D followed by B with C. Finally, we test the bandwidth that can be achieved, at each layer, from a single controller by using a single port on each of four IO nodes connected to the same controller (figure 4). During all of the tests performed we monitor the per port and aggregate performance reported by the DDN controllers themselves and report any applicable observations. The results of the following tests are compared with both our theoretical evaluation and the information we received from DDN.

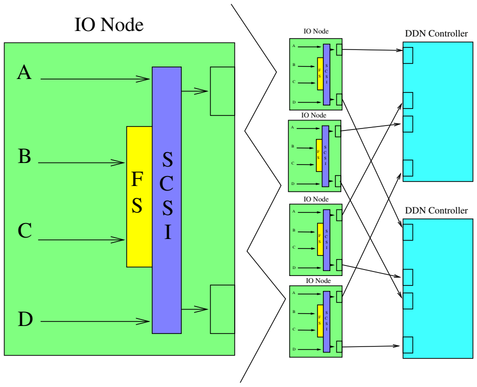

Fig. 4. Back-End Configuration

1) Single path (Demonstrable): The theoretical maximum of our single end-to-end IO path (fig 3) is 200MB/sec (section III-A.1). Executing a single sgp_dd from the IO node through the SCSI device layer, we observe a bandwidth of 196.23MB/sec consistently over three iterations of testing using the command in figure 5.

```
sgp_dd if=/dev/zero of=/dev/sg0 bs=4k count=2621440 time=1 thr=16 sync=1
```

Fig. 5. sgp_dd command through the SCSI layer (path A and D)

If we repeat this operation using both ports of a single IO node in parallel (paths A and D in figure 4) we observe exactly the same rates on each port. (Note each IO node is connected to a separate controller which should allow the maximum per controller bandwidth to be achieved). At the raw device level it appears that sgp_dd reports very near theoretical rates. We also note that monitoring the DDN controller at one second intervals during these operations showed very little fluctuation in data rates.

Next we executed a single sgp_dd using the file-system layer (path B in fig 4). We observed a bandwidth, on average, of 179.84 MB/sec using the command in figure 6.

```
sgp_dd if=/dev/zero of=/mnt/testfile bs=4k count=2621440 time=1 thr=16 sync=1
```

Fig. 6. sgp_dd command through the file-system layer (path B and C)

If we execute this command in parallel on a single IO node using both ports of the HBA (paths B and C in figure 4) we observe a bandwidth of 102.35MB/sec on average per port. We see a notable difference in the bandwidth that is achieved on a single port compared to using both ports in parallel. When performing file-system access, monitoring the DDN controller revealed large fluctuations in data rates during the duration of both the single and dual port tests.

2) Aggregate Controller Performance (Demonstrable): Theoretically, each DDN controller should be able to deliver 200MB/sec/port bandwidth (800MB/sec aggregate). We tested this first at the SCSI device layer by executing the command in figure 5 on each of four IO nodes, in parallel, each to their own SCSI device.

The bandwidth observed, on average, was 195.13MB/sec per port. The individual per port rates were almost identical indicating good balanced performance per port delivered by the DDN controller. This rate is approximately 1MB/sec lower than the single port rate shown in section III-B.1. As with our previous SCSI level testing the data rates observed by monitoring the DDN controllers were very consistent both per port and per one second sample.

Next we executed sgp_dd using the same configuration but at the file-system layer using the command in figure 6.

We observed a bandwidth of 140.82MB/sec on average per port. This test seems to indicate that using all four ports of a controller in parallel at the file-system layer results in less bandwidth per port. As with our previous file-system test we observed great fluctuation in the data rates observed monitoring the DDN controllers both per port and per sample.

C. Available Bandwidth (Conclusions)

From our previous calculations and testing we can begin to make some assumptions and form some expectations. We found that when accessing data at the raw device layer, we achieved very close to theoretical bandwidths whether access was performed over a single path, in parallel over both ports of an HBA, or accessing all ports of a controller in parallel. Additionally, we found that performing the same tests at the file-system layer revealed that parallel accesses, whether utilizing both ports of an HBA or using all four ports of a controller in parallel, achieve less of the theoretical bandwidth. Using both ports of an HBA in parallel resulted in the worst performance observed. In the tests that follow both types of

[8] sgp_dd is a utility that allows raw device IO, in addition to block and file-system layer, and many other expanded capabilities like threaded execution that made this the utility of choice for this effort. We also used dd during this effort.

[9] Small Computer System Interface

[10] The file-system, in this case, is a Lustre patched version of the ext3 file-system.

parallel access are used. As outlined previously, each IO node (OSS) in our configuration supports two OSTs. Some of the tests that follow utilize a single OST per OSS (analogous to our single path test). As our tests scale upward we also employ multiple OSTs, one per OSS, that use all ports on their associated DDN controllers in parallel. As we continue to scale we also demonstrate utilization of two OSTs per OSS (utilizing both ports on each IO nodes HBA). By comparing previous results with the results of the following tests we can better evaluate the observed performance.

IV. Performance Analysis Methodology

Using the knowledge gained by the theoretical evaluation of the architecture and subsequent testing we selected tests to further analyze the performance characteristics of the IO subsystem on Red Storm[11]. Our first test was designed to verify our previous assumption that the link between the compute and IO node should not be a limiting factor in performance (section IV-A). IO on capability systems like Red Storm is typically performed in one of two ways. Most applications at Sandia Labs perform file-per-process IO which we define as each process in the parallel application performing IO to a single file. This is simplified further on the Red Storm architecture where only one process executes on a processor. Section IV-B covers file-per-process testing. The other type of IO tested is shared-file IO. An application using shared-file IO performs IO to a single file no matter how many processes (or in our case processors) are employed. Shared-file testing is covered in section IV-C. In both file-per-process and shared-file tests we over-subscribe the IO subsystem in an effort to observe how the file-system performs when stressed. Over-subscription of 60:1 was selected based on the ratio of compute to IO node on the Red Storm platform[12].

The parallel application used to perform both the file-per-process and the shared-file tests was IOR[14]. IOR is well recognized in the community as a capable IO testing application and is instrumented to carefully measure bandwidth at scale even on a large platform like Red Storm[13].

A. *Single Source Node Test*

Our previous analysis showed the back-end (the segment of the IO path from the IO node to the DDN controller, including the disk, see figure 3) is the limiting factor in our IO performance. Additionally, using parallel paths in the back-end at the file-system layer further limits our IO performance. Our assumption, thus far, has been that the link between the compute and IO node would provide far greater bandwidth than the back-end. What we did not consider is how fast the compute node can source data to the IO subsystem. In addition, we have not measured the link between the compute and IO node[14]. In this test we characterize both how fast a single compute node can source data, and whether the network link between the compute node and IO node(s) hinders performance. There is only one process (processor) throughout the test therefore only one file. The factor that will vary in this test is the stripe number of the file[15]. By increasing the stripe number our intent is to simulate a sink that removes our assumed limitation so we can test the performance of other segments in the IO path. Figure 7 shows the results of this test. It should be noted the maximum number of OSTs used in this test is 56. In this test one OST per OSS (or IO node) is allocated. In addition, the stripe number also indicates the number of unique OSTs used, each on an individual OSS.

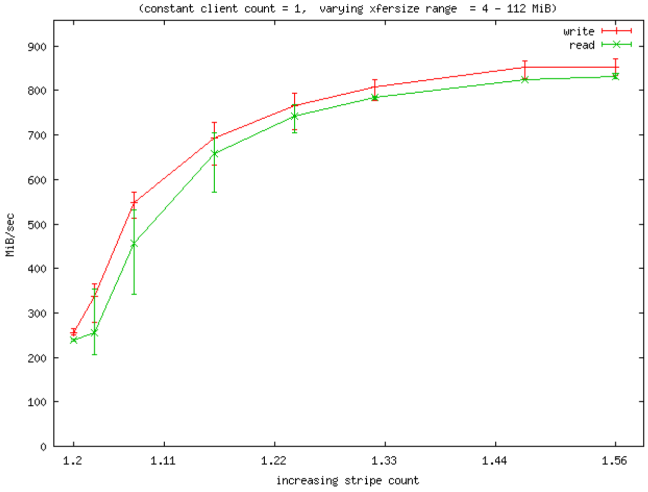

Fig. 7. Single Source Node, Multiple IO Node

The first observation we make is the overall shape of the curve is smooth and regular for both read and write operations. We also observe the error bars, especially for reads, are larger than we would like to see at smaller stripe numbers but tighten up nicely at larger stripe numbers. As the sink increases (as the stripe number increases) the bandwidth delivered by the single compute node increases, quickly at first and flattening as we reach the 56 wide stripe data-point. Without increasing the stripe beyond 56 we cannot guarantee that this is the asymptote but based on the shape of the curve it is likely. Regardless, the purpose of the test was to determine the limits on how much the compute node can source and if there seems to be any performance impediments in the link between the compute and IO partitions. Based on the results of this test, our stated assumption that we are limited by our back-end seems correct. We are left with at least one question: why do we have to scale up to such a wide stripe to sink the bandwidth sourced from

[11] A much larger range of testing was performed. Results can be found at http://www.cs.sandia.gov/RSIOPA

[12] This ratio has changed due to recent upgrade of Red Storm to approximately 75:1. If we consider that the new processors are dual core our ratio increases to 150:1.

[13] The exact parameters used in executing IOR for all of the tests mentioned in this paper and additional tests that were performed during this analysis can be found at http://www.cs.sandia.gov/RSIOPA.

[14] We entered this exercise with a fair amount of confidence that the link between the compute and IO node would not be a bandwidth problem based on other testing done on this platform. While this proved to be true we will briefly discuss a problem uncovered during this analysis exercise that reminds us of the necessity of rigorous analysis practices.

[15] Stripe number, in this case, is the number of OSTs used per file. OSTs may be oversubscribed.

a single compute node? Our initial data-point is for a single compute node writing a two stripe file. This process uses two OSTs, each on a separate OSS. The average rate of the three runs for this data-point per OST is approximately 128MB/sec. Based on this initial data-point, seven OSTs should be able to sync in excess of the 850MB/sec rate that we peak at using 56 OSTs. While we do not expect perfect scaling, we have to question why our per OST rate drops as we add additional OSTs. One possible explanation is Lustre overhead, especially considering that we are using a single source node and writing to a single file. We will keep these observations in mind as we proceed with our testing.

B. File-Per-Process Test

The single source node test (section IV-A) added weight to our assertion that the back-end will be the limiting factor in our performance. Our next test(s) are designed to determine how the file-system performs during parallel file-per-process activity. The results of the first test (shown in figure 8) were obtained by increasing the number of clients while keeping the stripe number per client static at one. In this test the maximum number of OSTs used is 56, the maximum number of OSS nodes is 28. For client sizes up to and including 28, one OST is allocated per OSS per client up to 28 clients. Client sizes from 29 to 56 allocate the unallocated OST on each OSS, of the 28, one per OSS per client in the same order[16]. Runs with client counts greater than 56 begin to over-subscribe OSSs in the same order.

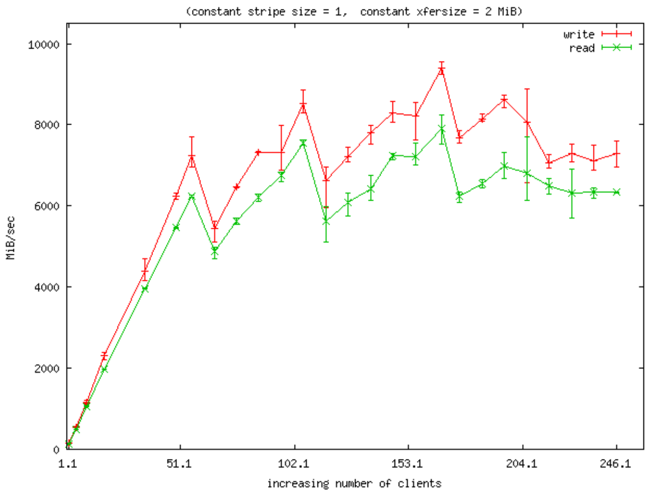

Fig. 8.   Initial File-Per-Process

The graph in figure 8 shows a steep increase in performance which initially peaks when all 56 OSTs are allocated. There is a clear saw-tooth pattern in the remainder of the graph as OST allocation repeatedly fluctuates from unbalanced to balanced allocation. An observation that we can make from the saw-tooth pattern is that unbalanced allocation of OSTs results in poorer performance. This seems logical considering that a portion of the overall number of OSSs that are used have more work to do resulting in the dips observed during uneven allocation data points. We also noticed the peaks are successively higher up to three times the maximum OST allocation after which the available data shows a flattening trend. This trend suggests that to a point over-allocation of OSTs can result in a higher per OST performance when balanced over the number of OSSs. In general, this test seems to show efficient performance. The performance per OST averaged over the entire graph is 134.46MB/sec. This includes data points from uneven allocation. Rates as high as 167MB/sec/OST were observed. Since we are exercising both ports on an IO node in parallel[17] and in many cases all four ports of a controller[18], these bandwidths compare well with previous results from section III-B.

The purpose of the following test is to analyze how the file-system responds to over-subscription during file-per-process access. As mentioned previously, the ratio of compute to IO nodes on Red Storm is approximately 60:1. To be able to test this level of over-subscription sufficiently in a reasonable amount of time the number of OSTs were limited to eight on four OSS nodes. It should be noted that even using a limited number of OSTs the tests using large numbers of clients consume large amounts of time. In some cases we were not able to take multiple data points, however previous tests support these findings.

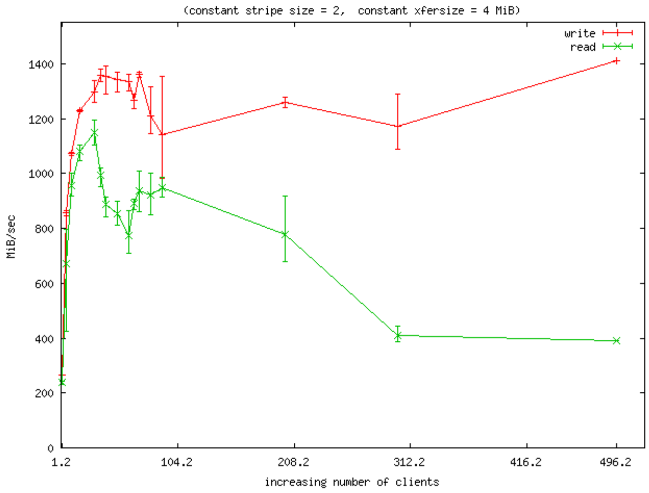

Fig. 9.   Oversubscribed File-Per-Process

While compressed due to the large range of data points, the early portion of the graph, depicted in figure 9, indicates a saw-tooth pattern similar to figure 8. The most remarkable

[16] Some examples to clarify. 2 clients 2 OSTs on 2 unique OSSs. 29 clients 29 OSTs on 28 OSSs (two of the OSTs are on one OSS). 56 clients 56 OSTs on 28 OSSs (two OSTs per OSS). 57 clients begin to oversubscribe therefore they use 56 OSTs on 28 OSSs but one OST is now servicing two clients.

[17] Based on the previously discussed allocation pattern, at client count 29 OSS one OSS node is using both HBA ports in parallel. At 30 clients two OSS nodes are using both ports, at 56 all OSSs are using both HBA ports in parallel.

[18] Dependent on how OSS nodes are connected to controllers.

feature of the graph is the clear drop off of read performance as the over-subscription of the file-system increases. While it is difficult to specifically analyze the exact point the drop begins, the overall trend is clear. At large scale there is a significant problem with read performance. Write performance seems to hold up at large scale in contrast to read performance, using file-per-process IO. As in the test depicted in figure 8, the per OST performance (for writes) compares favorably with previous observations using both ports of an IO node in parallel. Average performance per OST (for writes) calculated over the entire test was 154.15MB/sec with a maximum rate of 176.31MB/sec. We should note that while this test is a useful measure of how the system might perform during over-subscription at least minimal testing should be done at true scale to determine if significant differences are observed.

C. Shared-File Test

While not as common as file-per-process IO, at least at Sandia Labs, shared-file IO is utilized by important applications. Testing how the file-system responds to this method of IO is important. The stripe number used for the single shared-file in this test is eight. Due to the configuration of the Lustre file-system each of the OSTs allocated are on separate OSS nodes. The selection of an eight stripe file was made, as in our previous file-per-process test (see section IV-B), to determine how the file-system responds to over-subscription in the ratio of 60:1. We should note that previous tests performed in this manner proved to be very time-consuming. Due to limitations in the amount of time available on this production platform, fewer data-points were gathered but were chosen based on previous tests to best represent how the file-system responds to shared-file IO.

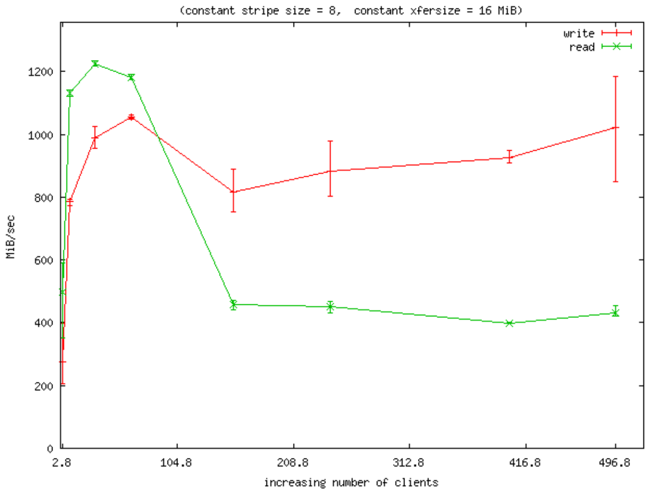

Fig. 10. Oversubscribed Shared-file

The graph in figure 10 illustrates the results of our shared-file IO testing. Our first observation is for small numbers of clients the read performance exceeds the write performance. After the initial rise, the read performance drops sharply. Based on the available data-points the drop appears to begin at about the same level of over-subscription that the file-per-process test in figure 9 indicates. The write performance seems more regular, excluding the fact that for small numbers of clients writing is less performant than reading. After a fairly pronounced dip following the initial peak, the write performance appears to even out through the remainder of the graph. A final observation is that the per OST performance for shared-file IO is far less than file-per-process IO. Average performance per OST (for writes) calculated over the entire test was 105.44MB/sec. The maximum observed rate was 127.61MB/sec.

V. Conclusions

Based on our testing we have determined our IO sub-system is physically configured in a manner appropriate to achieve our performance goals (50000MB/sec, reference section I). Our testing showed that at the device level we can obtain very close to theoretical bandwidth. Consider, if we could achieve the aggregate bandwidth per controller that we demonstrated during device level tests (approximately 195MB/sec per port) our maximum bandwidth for our largest file-system (320 OSTs) would be 62400MB/sec.

In our file-system layer tests we saw performance degrade, especially when parallel paths were employed. The per OST results observed when using Lustre, however, were more promising. If the best of the per OST rates we observed remained consistent at scale our performance goals could be achieved for file-per-process access (for writes). If we calculate the maximum bandwidth achievable using the average rate, for writes, over the entire test in figure 9 we would see 49280MB/sec (154.15MB/sec$\times 320 OSTs$), very close to our performance goals. Alternatively, if we use the maximum rate observed in the same test (176.31MB/sec) we could achieve 56419.2MB/sec (176.31MB/sec$\times 320 OSTs$), which easily exceeds our goal. Unfortunately, all of our tests indicate that read performance suffers at scale. In addition, write performance, while it did not degrade badly like read performance, for shared-file access did not produce sufficient results even using best case figures. Our goal is to exceed 50000MB/sec for IO, reading or writing, whether file-per-process or shared file access is used.

In general, however, this has proven to be a valuable exercise for our site. The close examination of our IO configuration instigated by this effort identified inefficiencies that were corrected and resulted in increased performance. Scaling studies done in conjunction with this effort identified bugs in both Lustre and problems in the high speed network routing algorithm. Our routing problems have been corrected and the Lustre problems identified have been resolved or are in the process of being resolved by CFS[3]. Our lingering concern is performance at large scale, especially for reads. Our results have been shared with both CFS and Cray.

VI. Future Work

This effort is ongoing. Our testing harness is used to verify our IO performance after each upgrade; hardware or software. We continue to test for performance and reliability using the testing procedures developed for this analysis. Our performance goals have not yet been achieved but we are hopeful that continued testing and analysis will move us closer to our goals in the near future. We have also identified multiple opportunities for improvement in our procedure and identified additional tests that should be developed to enhance our analysis capability.

We previously mentioned the necessity to test at true scale. We have performed tests on a Lustre file-system that is configured with 160 OSS nodes with two OSTs per node. In file-per-process, one stripe per client, tests using all 320 OSTs we have seen aggregate transfer rates of up to 54104MB/sec using only 640 clients. This rate is 86% of our device level bandwidth of 62400MB/sec (see section V) and exceeds our goal of 50000MB/sec. In addition, we have seen rates exceeding 50000MB/sec for client counts up to 3200 indicating that high per OST bandwidth rates can be sustained for large client counts. Unfortunately, we have found these results to be inconsistent. We can report, however, that bandwidth numbers like these can be realized and use them to set achievable, repeatable, performance goals in the future.

VII. Acknowledgments

We would like to acknowledge the members of departments 1422 (Scalable Computer Architectures) and 1423 (Scalable Systems Software) for their involvement in this exercise, whether peripherally or directly. We have also worked closely with personnel at CFS and Cray and are grateful for their contributions. On site support was critical to this effort. All members of the Cray system administrative team (Richard Dimock, Victor Kuhns, Barry Oliphant, Roberto Purdy and Jason Repik) supported this effort 24 hours a day. We greatly appreciate their responsiveness and patience. We would also like to thank Robert Ballance and John Noe for their cooperation in scheduling the large amount of system time necessary for this effort to proceed.

References

[1] Red Storm - http://www.sandia.gov/ASC/redstorm.html

[2] Red Storm IO Performance Analysis - http://www.cs.sandia.gov/RSIOPA

[3] Lustre - http://www.clusterfs.com

[4] Red Storm - SeaStar Interconnect - SeaStar Interconnect: Balanced Bandwidth for Scalable Performance, Ron Brightwell, Kevin T. Pedretti, Keith D. Underwood, Trammell Hudson - http://doi.ieeecomputersociety.org/10.1109/MM.2006.65

[5] Catamount (Light Weight Kernel) - Suzanne M Kelly, Ronald B Brightwell, "Software Architecture of the Light Weight Kernel, Catamount," Conference Paper, Cray User Group, May 2005

[6] Catamount (Light Weight Kernel) - Suzanne M Kelly, Ron B Brightwell, John P VanDyke, "Catamount Software Architecture with Dual Core Extensions," Conference Paper, Cray User Group, May 2006

[7] Cray - http://www.cray.com

[8] SUSE Linux - http://www.novell.com/linux

[9] Linux - Linux is the registered trademark of Linus Torvalds in the U.S. and other countries

[10] Qlogics - http://www.qlogic.com

[11] Data Direct Networks - http://www.datadirectnet.com

[12] Seagate - http://www.seagate.com

[13] Seagate Data Sheet - Publication Number: 100195490, Rev. F, Printed in USA

[14] IOR - http://www.llnl.gov/icc/lc/siop/downloads/download.html