Parallel File System Benchmarking Made Easy [*][†][‡] DRAFT

Andrew Uselton

January 23, 2008

Abstract

This report presents the results of a series of I/O benchmark tests on the *Franklin* Cray XT4 conducted on December 19th, 2008, by Katie Antypas. There is an extended discussion of methodology as well as an analysis of results and recommendations for further testing. This is an early draft with several important sections missing or under development.

1 Introduction

If you want to know the performance of an application on a particular computer, then just run that application on that computer and have a stopwatch handy. With numerous details elided this is how the Supercomputing Top 500 List benchmarks compute performace. The benchmark result is entirely concrete, in that it makes no claims about how some other application might run on that or some other system. Indeed, that is the central reason why the List is criticized. Benchmarking a parallel application's I/O performance to a parallel file system is equally simple, and simple results from such bencharks are criticized for the same reason. One may speculate that there are many more than twelve ways to lie about it.

For a parallel file system benchmark to produce a more widely useful result the benchmark, and its interpretation, require care. This report presents a methodology for the development and interpretation of I/O benchmarks in the context a parallel file system mounted on a cluster-based supercomputer.

A parallel file system mounted on a cluster is a complicated beast of many parts. Before testing may begin it is important to have a clear understanding of all the individual components in the system and how they are connected. This includes at least the cluster and server nodes, the cluster and server networks, and the disk devices. Determining how well the whole system performs begins

[*]acuselton@lbl.gov

[†]LBL-xxxxxx

[‡]This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Berkeley National Laboratory under Contract X-xxxx-Xxx-xx.

with an a priori notion about how the components perform individually and how those components will perform when taken all together. Then, a test can establish if the performance meets the expectation. If the performance is as expected one may be tempted to go no further. Performance below expectation may initiate an investigation into how to tune things to get a better result, or even to change the test to see if that improves the answer. One may hope that performance well above expectation would result in an equally agressive investigation.

> **Claim**: Benchmarking in the absense of an a priori notion of expected performance is a meaningless exercise.

1.1 Why Benchmark?

"Benchmarking" is a method of testing a "system" inteded to foster an understanding of the system and to characterize its performance. Initially, the system may be a poorly characterized collection of components. It is important to establish, from the start, a realistic and predictive model of the system. Thus early tests focus on individual components and how they interoperate. A model of the system is built up incrementally as the model becomes better able to predict the results of tests. An increasing understanding of how the system works may provide feedback to the system's engineers, as the system is debugged, modified, and tuned to improve the measured performance. A well tuned and debugged system with a detailed performance model provides a sound basis for predicting the I/O performance of a given application. Equipped with that analysis a user of the application has a reasonable expectation of its actual performance. Conversely, the application designer has the option of modifying the application to optimize its I/O performance for that system.

Parameter space is infinite and testing time is finite. Parameter space can only be sampled, and one is forced to make assumptions about both interpolated and extrapolated values over the parameter space. Those assumptions, along with issues of sampling error, lead to uncertainty about the meaning of any one test sample. Much of the behavior of the parallel file system cannot be predicted from first principles. Only after some tests have been conducted will testers know what further tests are required.

> **Claim**: The benchmarking evaluation of a computer system is an iterative process

It is usually necessary to test the parts in order to esatablish expectations about the whole.

> **Claim**: The task of benchmarking the components individually is the same as the task of benchmarking the whole.

> **Claim**: Parameter space is of high dimension.

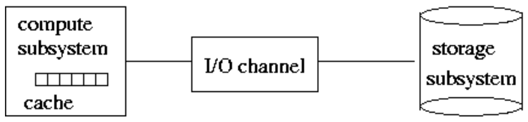

Figure 1: Simple model

2 Methodology: A Simple Beginning

A parallel file system mounted on a cluster is an endlessly complicated system. The task of benchmarking it is equally complicated. As a starting point, and as a way of developing the methodology that will be used throughout this report, this section dispenses with every complexity, making the benchmarking task as simple as possible, but no simpler. Subsequent sections will introduce complexities as needed to solve problems encountered with the simple model.

In this section a system, described in the simplest terms, leads to model that can be compared against the results from a correspondingly simple test.

2.1 Model

The simplest possible I/O model characterizes the system as a *compute subsytem*, a *storage subsystem*, and an *I/O channel* connecting them (Figure 1). In a generic test, a POSIX compliant *write* (or *read*) operation on a buffer of B bytes in the compute subsystem transfers the contents of that buffer to (respectively from) the storage subsytem. A test application notes the time immediately before (t_0) and immediately after (t_1) the I/O operation, and computes a data rate $R = B/(t_1 - t_0)$ measured in bytes per second.

2.2 Parameter Space

In this simple model there is one independent variable: the buffer size B. It is reasonable to test the system for a variety of buffer sizes, so we make B a *parameter* of the test. There is also one dependent variable for the test: the *communication cost* given by the interval $t_1 - t_0$ (equivalently, the data rate R calculated from it).

2.3 Expectations

When confronted with such a simple model the expected performance of the system may come from an analysis of the memory bandwidth of the compute subsystem, the reported speed of the disks in the storage subsytem, and the bus or network speed for the channel connecting them. Memory generally has a higher bandwidth than the network or disk and can be used to cache transmitted data. The network is usually not going to buffer data. Some disks buffer data, but this initial model will assume that data goes to the disk at a single uniform rate.

2.4 Architecture and the Limiting Factor

For small values of B - smaller than the local I/O buffer cache - the memory bandwidth will be the limiting factor determining the data rate, since the write can be completed before the I/O channel and storage subsystem begin to receive data. For larger values of B the I/O channel bandwidth and/or storage subsystem bandwidth will be the limiting factor. This section assumes the disk speed is the limiting factor. The simplest model for this behavior employs a discontinuous function like that in Equation 1.

$$R = [\frac{1}{M} + \frac{1}{N}u(B-b)]^{-1} \tag{1}$$

where:

- R is the expected data rate (MB/s)
- B is the buffer size (MB)
- M is the data rate in the limit of small B (MB/s)
- N is the data rate in the limit of large B (MB/s)
- b is the buffer size at which the transition occurs (MB)
- u is the unit step function, which introduces the discontinuity at $(B-b)$

The expected data rate abruptly changes value at the critical point where buffering can no longer take place, as depicted in Figure 2 in Section 2.7.

2.5 Testable Hypothesis

The expectation produced by the model and the analysis of the architecture constitutes a testable hypothesis about the system. For values of B much smaller than available memory cache b, the data rate R is expected to be about M - the memory bandwidth - and for much larger values should be about N - the speed of the disk.

2.6 Benchmark Application

The application used to test this hypothesis might resemble the following:

```
t1 = time(NULL);
write(fp, buffer, B);
t2 = time(NULL);
interval = t2 - t1;
if( interval > 0 )
  printf(‘‘R(\%ld) = \%f\n’’, B, B/interval);
```

The actual test used was only slightly more sophisticated to allow for sub-second times and detailed error checking.

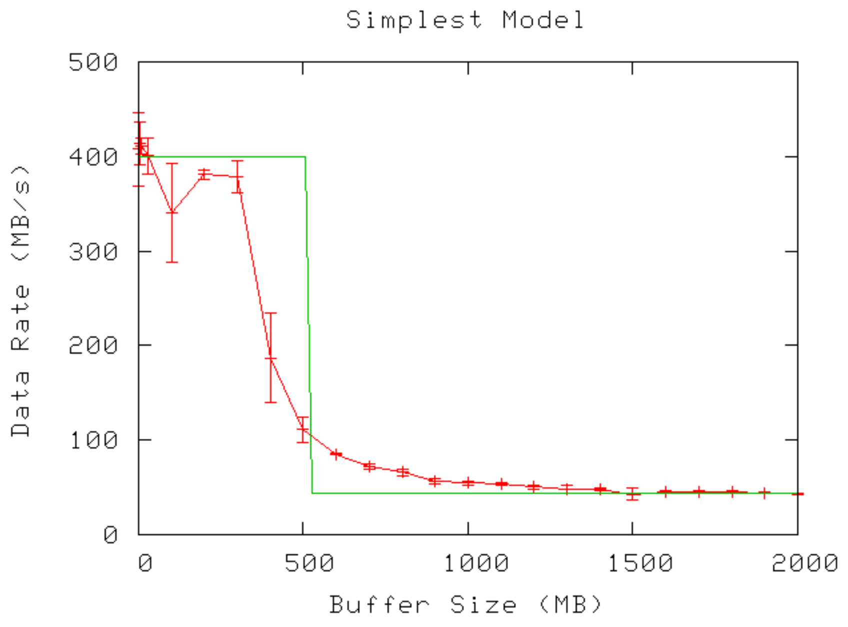

Figure 2: Comparing the model to the results of the simple test run on a laptop

2.7 Test Run

Running the test (on the author's laptop) produces the results depicted as a curve with error bars in Figure 2.

Note that on a *i386* laptop it was not possible to measure for buffers larger than $2GB$. The expected behavior governed by Equation 1 is superimposed on the results.

At this level of detail, the test may be said to have *demonstrated* that the laptop memory badwidth is around $400MB/s$ and the drive subsytem around $50MB/s$. These are the *results* of the benchmark.

2.8 Repeatability

The data points in Figure 2 are actually the averages of 10 runs of the test for each choice of B. The error bars on the data represent the standard deviation of those ten tests about each average. For large values of B the error is very small and gives confidence in the fidelity of the test. For a few tests with low B the error bars are very large. The tests were either subject to wide random perterbations or there may be some other mechanism in play.

2.9 Analysis and Feedback

The measured data rates for small buffers appear to be highly variable, and the large buffer rates appear to be slowly degrading rather than staying constant. At the discontinuity, the data rate does not change abruptly. As the buffer size increases there may be a resource contention issue that operates with increasing probability rather than suddenly turning on.

A sigmoind function (Equation 2) is a convenient way to introduce the increasing probability of resource contention into the simple model. The data rate starts out in the range of typical memory bandwidth and then decends smoothly to a data rate typical of disk speeds at about the point that memory would be full.

$$R = N(1 + (\exp^{((b-B)/\tau)}))/(1 + (N/M)(\exp^{((b-B)/\tau)})) \tag{2}$$

where:

- R is the expected data rate (MB/s)
- B is the buffer size (MB)
- M is the data rate in the limit of small B (MB/s)
- N is the data rate in the limit of large B (MB/s)
- b is the buffer size at which the transition occurs (MB)
- τ is the width of the transition region (MB)

Figure 3 superimposes a sigmoid on the data from Section 2.7. The values for that sigmoid are:

- $M = 400MB/s$
- $N = 50MB/s$
- $b = 512MB$
- $\tau = 50MB$

The correspondence between the data and the sigmoid give some justification for the model.

On the other hand, the correspondence between the data and the sigmoid is imperfect, especially at low buffer sizes. There may be value in refining both the model and the understanding of the architecture in order to give a better explanation and better fit to the data.

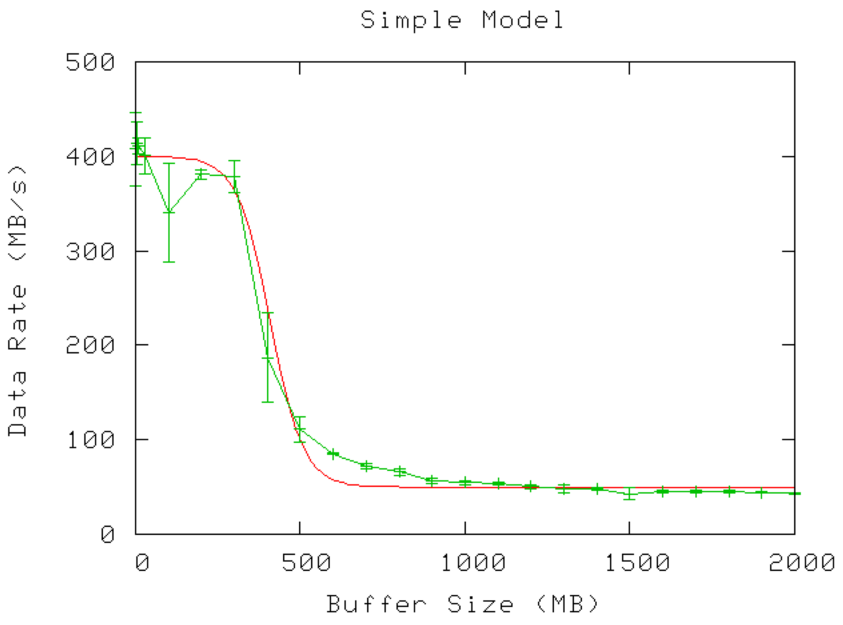

Figure 3: Matching a sigmoid to the data

2.9.1 Additional Testing

Benchmarking is an iterative process. The model does some things well and other things poorly. Further testing is usually warranted. Follow up tests can have any combination of these three goals:

- Test new regions of parameter space looking for new behaviors.
- Test familiar regions of parameter space to confirm previous observations or validate results interpolated from them. This is especially useful if something in the system has changed or is suspect.
- Test with a refined methodology, perhaps using a new tool or significant change to the model.

In proposing follow-up tests it is important to identify the motivation and expected result for each such test. Section 2.10 adds a new dimmension to parameter space and new factor to the model. Interesting things result.

2.10 The First Complication: Transfer Size

Not reflected in the foregoing model is that the duration of a *write* system call involves a *fixed cost* f (seconds) as well as the communications cost B/R. For a large buffer the fixed cost can probably be neglected, and for a small

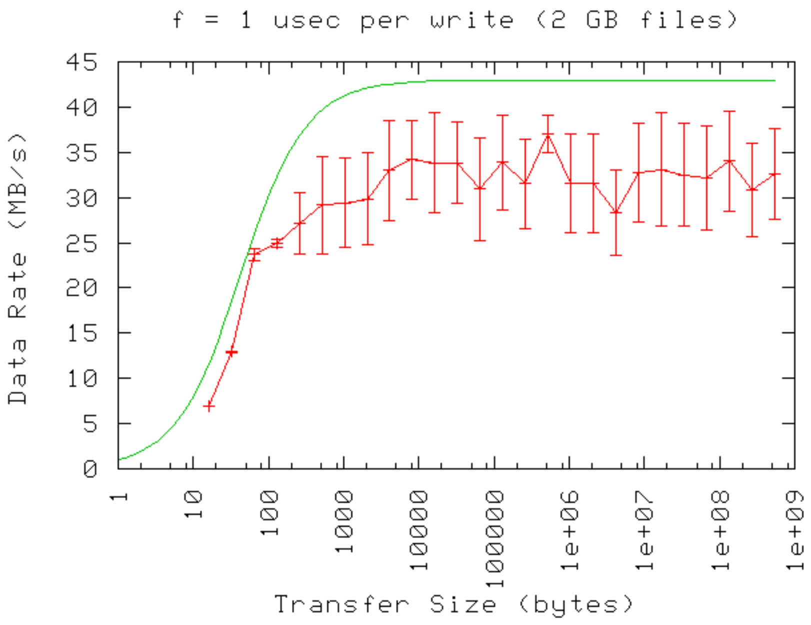

Figure 4: The effect of a fixed cost on data transfers

buffer the higher bandwidth prevaling may make the fixed cost unimportant again. However, some applications do not have a single large write as their I/O pattern. An identical I/O load made of multiple *write* calls may show different behavior. This introduces a second independent variable, the *transfer size* s.

The results in Figure 3 suggest that when writing more than about $B = 600MB$ the data rate r_B will be near the disk subsystem speed. In a test that writes the aggregate volume of data over many separate system calls of size s bytes, the fixed cost of a system call will be significant when s is small enough that f is comparable to the communication cost s/r_B of one transfer. Equation 3 gives the expected rate as a function of transfer size.

$$R_B = r_B(1 + r_B f/s)^{-1} \tag{3}$$

Where:

- R_B is the expected data rate
- B is the aggregate amount written during the test
- r_B is the measured data rate for a single buffer transfer of size B
- f is the fixed cost for a *write* system call
- s is the transfer size - there will be B/s of these

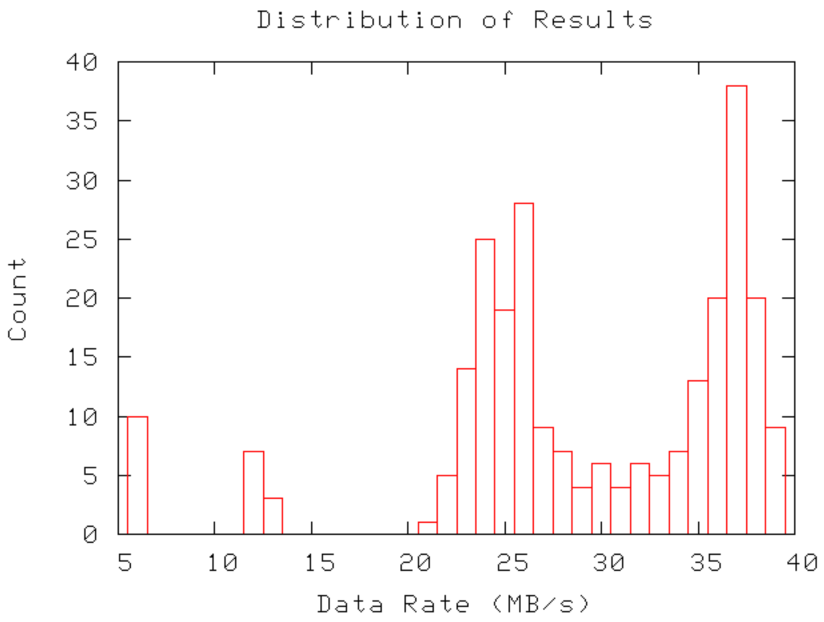

Figure 5: The distribution of data rates is bi-modal

For the system measured in the previous sections, with $B = 2GB$ the measured data rate was $r_B = 43MB/s$. If the fixed cost were about $1\mu s$ then a transfer size of 16 bytes would incur about a 50% performance penalty. Figure 4 superimposes Equation 3 ($f = 1\mu s$, $r_B = 43MB/s$, $B = 2GB$) on the results of a test similar to the one in Section 2.6 except that writes are of s bytes each, carried out in a loop for B/s system calls.

Note that the transfer size is shown on a log scale since all the action takes place in a small fraction of the domain. Also note that the test rarely achieved the expected $43MB/s$ from the previous testing. Figure 5 shows that the distribution of results over the course of the test was bi-modal.

Clearly, something interesting is happening on this laptop. If laptop performance modeling were the subject of this report this would be a significant point to pursue. In later sections, such anomalies in the results will become the primary focus of testing.

2.11 Methodology: Not so Simple

This section has introduced the main themes of this report. Benchmarking requires a model that can produce expectations about the results of a specific test given an understanding of the architecture of the system. A benchmark application produces results that are compared with the predictions of the model. Initially, the model is expected to perform poorly, and the results of testing

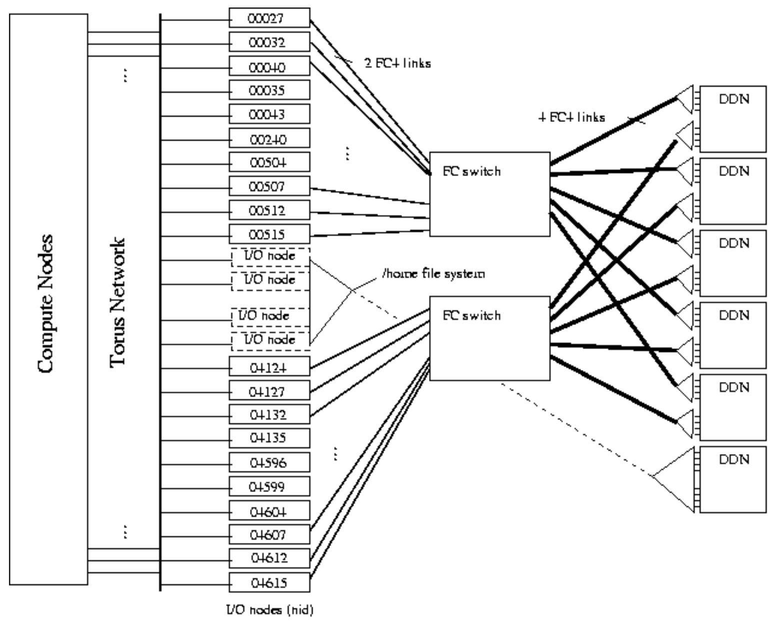

Figure 6: A model of the cluster and its file system

guide the refinement of the model. Many parameters of the test may be varied, and many different kinds of observations may be made. Testing is most instructive when the results are at odds with the model.

3 The Cray XT4: A First Look

3.1 Model

The Cray XT/4 (named *Franklin*) at Lawrence Berkeley National Lboratory has 9660 compute nodes (19,320 processors), 16 login nodes, and 25 I/O nodes, connected together in a *3-D* torus with the Cray Star network interconnect. The I/O nodes provide Lustre parallel file system services to the cluster. Twenty-four I/O nodes act as *object storage servers* (OSSs), and one I/O node acts as a *metadata server* (MDS). The compute and login nodes mount two Lustre file systems, */scratch* and */home*, provided by the servers. Four of the OSSs provide */home* and this report will not say much about that resource. Twenty OSSs provide the */scratch* file system, and on each of these servers there are four Lustre *object storage targets* (OSTs), which mediate the access to the underlying storage. Each OST presents a Logical Unit (LUN) of storage (a $4TB$ block device) from a Data Direct Networks (DDN) model 9500 RAID device. There are five DDNs with 16 LUNs each (another DDN provides storage to */home*), thus the 80 OSTs front 80 LUNs. The Lustre file system has a unified namespace, so a given file may be split amongst several OSTs - the default is four. Figure 6 gives a schematic of some of these details.

A row of equipment racks houses the storage resources for the cluster. A DDN unit is identified by the rack in which it resides. Those are *R2*, *R4*, *R6*, *R8*, *R10*, and *R12* - other racks contain the actual RAID disks. There is also another DDN unit that has not been connected up yet. The unit in rack *R12* is for the */home* file system.

The OSSs connect to the DDNs via $4Gb/s$ Fibre Channel (FC4) links through two Cisco model DSC9513 *switch frames* (or just "switches") in the rack at the end of the row of DDNs. Each OSS has two FC4 links, both of which connect to the same DS-X9112 *switching module* (or just "blade") on a given switch. Each DDN has eight FC4 links divided between two controllers - a *couplet*, in DDN terminology. The four links from one controller go to the same blade on *switch SW1* (bottom) and the other four to *switch SW2* (top). Thus a given blade of a switch connects all four links from a particular DDN controller to the four links from two OSSs[1].

A couplet stacks its two controllers one above the other, labeled *U1* (top) and *U2* (bottom). A controller organizes its four ports in a rectangular pattern with labels *Host-1* (top-left), *Host-2* (bottom-left), *Host-3* (top-right), and *Host-4* (bottom-right).

The *U1* controllers all connect to the switch labeled *SW1*, and the *U2* controllers to *SW2*. Each controller's connections go to one blade of the corresponding switch with the controller ports mapped to the switch ports as follows:

controller	blade
Host-1	*Port-3*
Host-2	*Port-9*
Host-3	*Port-6*
Host-4	*Port-12*

The mapping from controllers to blades (numbered in a switch from the top down) is:

couplet.controller	switch.blade
R2.U1	*SW1.B1*
R2.U2	*SW2.B1*
R4.U1	*SW1.B2*
R4.U2	*SW2.B2*
R6.U1	*SW1.B3*
R6.U2	*SW2.B3*
R8.U1	*SW1.B4*
R8.U2	*SW2.B4*
R10.U1	*SW1.B5*
R10.U2	*SW2.B5*

[1]The intent to co-locate the FC4 links for the OSSs on the same blade as the FC4 links to the DDN controllers for their respective LUNs is actually enforced in the DDN configuration. Since the switches have high capacity crossbars connecting all the blades the mapping from OSSs to LUNs could be arbitrary within the switch itself, possibly without much loss of performance.

The XT4 divides the OSS nodes between two *sides* of the cluster (so the machine could be "partitioned", i.e. made into two separate systems). I/O nodes on one side are all labeled with "(*C0441*)" and on the other side with "(*C0486*)". Some nodes have more than two ports, but all the I/O nodes have two, labeled *s0p1* and *s1p1*. Nodes in the XT4 cabinets are on modules also called "blades". Each blade of the XT4 with I/O nodes in it has two I/O nodes, labeled *b0* and *b1*. I/O blades are indexed by cabinet number *c* and module *m*. Both I/O nodes of an I/O blade have FC4 links to the same blade on the switch. The I/O blades and ports for the */scratch* file system are:

I/O blade	switch.blade
(C0441).c0m6	SW1.b1
(C0441).c0m7	SW1.b2
(C0441).c1m0	SW1.b3
(C0441).c1m1	SW1.b4
(C0441).c1m2	SW1.b5
(C0486).c0m5	SW2.b1
(C0486).c0m6	SW2.b2
(C0486).c0m7	SW2.b3
(C0486).c1m0	SW2.b4
(C0486).c1m1	SW2.b5

On an XT4 blade the four ports form a square with labels *b1s1* (top-left), *b0s1* (bottom-left), *b1s0* (top-right), *b0s0* (bottom-right). The mapping from XT4 blade ports to Cisco blade ports is:

I/O node	blade
b0s0	*Port-1*
b0s1	*Port-4*
b1s0	*Port-7*
b1s1	*Port-10*

I was still trying to understand the above pattern when I first looked, so the above may be incorrect. Furthermore, some of the connections did not appear to be in the designated ports, though again I am unsure. I saw other I/O node to switch connections as follows:

I/O node	switch.blade(.port)
(C0441).c0m0b0s0 (gigE?)	SMW.slot4.Port-C
(C0441).c0m0b0s1p1	SMW.slot4.Port-4
(C0441).c0m0b1s1p1	R1.FSW(qs0).Port-5
(C0441).c0m8b0s0p1	SW1.b5.Port-11
(C0441).c1m3	SW1.b3
(C0486).c1m2	SW2.b4
(C0486).c1m3b0s0p1	SW2.b5.Port-2
(C0486).c1m3b0s1p1	SW2.b5.Port-5
(C0486).c1m3b1s0p1	SW1.b4.Port-5
(C0486).c1m3b0s0p1	n/c

There are quite a few other connections, including some GigE connections, that support the login nodes.

3.2 Parameter Space

In Section 2 the test parameters were the file size and the transfer size. On a cluster with a parallel file system even the simplest test will involve several more parameters, as follows:

Aggregate file size - summed over multiple tasks

Transfer size - of each system call

Tasks - in general, there will be multiple tasks, usually in a parallel application employing the Message Passing Interface (MPI)

Nodes - compute nodes have two processors and usually have two tasks per node, one per processor, but can also be constrained to 1 task or overcommited to more than two tasks

Stripe count - the number of OSTs (equiv. LUNs) used for each file and for the test as a whole, as well as the choice of mapping from files to OSTs

Stripe size - the granularity of placement of file sections on OSTs

I/O pattern - does each task have its own file or do all tasks access a single file system object, or somewhere in between

API - POSIX, MPI-IO, HDF5, others

The initial benchmarking efforts reported here emphasized POSIX *write* system calls with one file per task. The tests employed the default $1MB$ stripe size and used default file system behavior in the selection of which OST recieived which file. There are other characteristics of the file system and of the benchmarks that could be explored, and some of them will be introduced in later sections as the model becomes more detailed.

3.3 Expectations

The storage subsystem has three stages: the OSSs, the Fibre Channel network and the DDNs. The disk consists of multiple independent units, which should scale linearly, i.e. the expected performance is the number of units times the expected performance from a single unit. Similarly, the network between the OSSs and the disks has multiple units that should should scale linearly. Thirdly, the OSSs operate independently and should provide an aggregate performance that scales linearly. The cluster interterconnect is a torus, which will not scale linearly, in general. Finally, the compute nodes are again independent and should again scale linearly. As before, the I/O path must traverse all these components in turn, so the slowest one in any given test will act as the limiting factor. Which element is the limiting factor may be different for different choices of test parameters.

The remainer of this section discusses the details of each component and its contribution to the overall expected performance.

3.3.1 DDNs

The spec. sheet for DDN 9500s reports "streaming" read and write performance of $3GB/s$. DDNs have a small cache (a few GB) for each controller and a limited capacity to aggregate and reorder writes to the individual RAID chains. Small transfer size degrades performance, especially in a random workload where aggregation and reording are not possible. Anecdotal evidence of their performance on a random I/O pattern for transfers above $1MB$ is about $2GB/s$. The sixteen LUNs of a DDN compete for this bandwidth to give an expectation of about $128MB/s$ per LUN to $192MB/s$ per LUN depending on the load.

3.3.2 I/O network

Each of the five DDNs has eight FC4 links giving it a theoretical bandwidth of $4GB/s$. This is balanced on the cluster by two FC4 links on each of the 20 OSSs. Each OSS FC4 link is on the same blade of the Cisco switch with the FC4 link to its corresponding DDN controller. There are four such pairings on each blade, and the DS-X9112 modules's data sheets say they can sustain full bandwidth through all the ports at once. Since the I/O network appears to have higher bandwidth than the DDN disk back-end, with no obvious contention issues, it looks like it will not act as the limiting factor in the storage subsystem.

3.3.3 OSSs

Almost all of the I/O reordering and aggregation takes place in the OSSs, which aggregate aggressively when possible and use standard elevator algorithms to optimize back-end disk performance. The two FC4 interfaces need to be compared to the bus on which they reside (a single PCI-X 66?) to see if the FC4 links can be kept "full". The disk back-end is expected to handle $512MB/s$ on its four LUNs assigned to the four OSTs on a given OSS, or perhaps a little more for a very well-behaved load. The two FC4 links on the OSS have a theoretical bandwidth (together) of about $1GB/s$, so at even "half full" the OSSs' FC4 links would not be the limiting factor.

With small transfers and a random or fragmented I/O load the OSSs would be expected to suffer performance issues as well. As the transfers become very small the CPU load on request processing can become a limiting factor. The Lustre file system attempts to enforce a $1MB$ Remote Procedure Call (RPC) transfer size on communications between file system clients (compute nodes) and servers (OSSs). If clients consistently send smaller RPCs the bandwidth can suffer substantially.

The MDS doesn't get mentioned much in the early part of the report. One issue that does relate to the MDS is the mapping of files to OSTs. A file is assigned to a number of OSTs determined by the file's *stripe count*. By default, a file will be assigned to be aportioned among four OSTs (stipe count = 4). A *starting OST* is assigned by the MDS and then the remaining OSTs are selected in sequential order. From one file to the next files are assigned to

OSTs in a round-robin sequence. There are also some mechanisms (depending on the Lustre version) for load balancing and distributing starting OSTs more randomly. The stipe count for a file can be set by the user via the *lfs* Lustre utility. For most tests, especially at scale the stripe count will be set to 1, so a particular file will always be entirely on a single LUN.

It is also useful to know that the OSTs are ordered such that they round-robin accross the OSSs. Thus OSS 0 would host OSTs 0, 20, 40, and 60.

3.3.4 Interconnect

The Cray Star interconnect can provide way more than $1GB/s$ into the OSS, though it would need to be compared to the rate available when that data moves across the node from the star network to the PCI-X bus. Since the interconnect itself is a torus it does not scale linearly with the nuber of connections. Some anecdotal evidence suggests there can be traffic flow issues when data converges on a single point in the torus, as it must when all 9660 compute nodes are communicating with the 20 I/O nodes. Nevertheless, the interconnect, as a whole, can provide a lot more data into the OSSs than they can push to disk. I think.

3.3.5 Nodes

The compute nodes resemble the I/O nodes in having individually a very fast connection to the interconnect. In a prallel I/O benchmark the nodes act independently except for coordinating MPI *barriers*. The amount that they can source into the interconnect increases linearly with node count, and is generally way in excess of the *bisection bandwidth* of the interconnect, much less the disk back-end.

An MPI job of N tasks that will run with 2 tasks per node gets $N/2$ nodes assigned essentially at random (the algorithm may be deterministic, I've yet to have it explained to me). For values of N much smaller than $19,320$ the nodes may be nowhere near each other, which would be good for the expected performance if doing so avoided local contention within the interconnect. The path data must travel from one given node to another is deterministic, I'm told, so a poor choice of nodes could have them all lined up competing for the same path into some specific OSS.

3.4 Architecture and the limiting factor

For a single task on a single compute node writing to a file mapped to a single LUN behind a single OST, the test resembles the one in Section 2. Even a very large file should appear to the OST as a streaming load. The expected data rate would be about $192MB/s$, in that case, if the disk back end is the limiting factor. Similarly, two tasks on a single node would employ two files on (it may be hoped) two LUNs behind two OSTs on separate OSSs. If the node can source

$384MB/s$, which it can, then the data rate should scale linearly from one to two tasks.

As the number number of tasks increases, but is still small, the data rate should continue to increase linearly, since more server, network, and disk resources are being used. When the number of tasks is more than 20 all of the OSSs should be involved in the test, and at 40 tasks each OSS should be managing I/O to two OSTs. If there is any resource contention in the OSS the data rate may not be twice at 40 tasks what is is at 20. The same goes for approaching 60 tasks and 80. If all the OSSs can sustain I/O at full bandwidth to all 4 LUNs each (they all still think they are sees a streaming load) the aggregate data rate at 80 tasks would be $15,360MB/s$. All this assumes that the disk back-end has been the limiting factor throughout, and that the files were sensibly laid out in round-robin fashion.

As the number of tasks increases beyond 80 more compute nodes are competing for the same limited bandwidth. The aggregate data rate would remain flat. For 16k tasks (a typical "at scale" test size), with one file per task, there will be 16k files distributed over 80 OSTs (LUNs). The OSTs should have a little over 200 files each: 16 OSTs with 204 files each and 64 OSTs with 205 files each. This will look to the OSSs (and the DDNs) like a random load. The $2GB/s$ that a DDN can deliver to a random load is divided among the 16 LUNs on the DDN to give $128MB/s$ per OST, as previously mentioned. The aggregate data rate at scale is then expected to be about $10,240MB/s$.

3.5 Testable Hypothesis

The first test implied by the foregoing varies the number of tasks from 1 to 16,384. The other parameters should introduce as little complexity as possible:

Aggregate file size - Large enough to prevent cache effects local to the compute node, say $2GB$

Transfer size - large enough to prevent fragmentation and fixed-cost issues, say $4MB$

Nodes - For counts from 1 to 80, step by 1 to explore details of the behavior as OSSs become responsible for additional files. Beyond that, sample parameter space in "powers of two" up to the full-sized test of 16,384

Tasks - two per node (after the first test of 1 task)

Stripe count - set to 1 for all tests

Stripe size - dafault $1MB$

I/O pattern - file-per-task

API - POSIX *write* system calls

Given the forgoing discussion the test results should lie along a curve that begins linear in the number of tasks n, with $R = n * 192MB/s$ up to 80 tasks. Then it should remain flat for a while until the load stops being streaming and becomes random. This would be captured by a sigmoid descending from $15,360MB/s$ to $10,240MB/s$ at about the point where the OSSs and/or the DDNs can no longer reorder the writes. That should happen about when each OSS has no more than one outstanding write per task (at a guess). With $8GB$ of memory,an OSS can cache no more than $2k$ writes of size $4MB$. I seem to recall that the OSSs will actually limit the amount they are willing to cache, and that the limit is a file system tunable. So there would be room for up to 500 outstanding writes per LUN which is about twice as many as there are files that each OST would host. It will be interesting to see if the streaming to random transition occurs at all. Equation 4 has the transition take place at $4k$ tasks. There is no theoretical motivation for that choice, it was chosen to exibit the shape of the curve.

$$R(n)\Big|_{\substack{B=2GB\\ s=4MB}} = \begin{cases} n * r_{\mathcal{S}} & 1 \le n \le \mathcal{O} \\ \mathcal{O} * r_{\mathcal{R}} \frac{1+e^{\frac{T-n}{\tau}}}{1+\frac{r_{\mathcal{R}}}{r_{\mathcal{S}}}e^{\frac{T-n}{\tau}}} & \mathcal{O} \le n \le 16k \end{cases} \tag{4}$$

where:

- R is the expected data rate
- n is the number of tasks
- $\mathcal{O}$ is the number of OSTs: 80
- $r_{\mathcal{S}}$ is the data rate achievable by a LUN under a streaming load: $192MB/s$
- $r_{\mathcal{R}}$ is the corresponding rate for a random load: $128MB/s$
- T is the number of tasks necessary to make the load appear random: $4k$
- τ is the width of the transition region: $1k$

3.6 Benchmark Application

IOR is an MPI application widely used for parallel file system benchmarking. It has features that allow for testing many different dimensions of the parallel file system parameter space. For the purposes of a the basic test described above it does the equivalent of the following:

```
MPI_Comm_rank(Com, &rank);
MPI_Comm_size(Com, &size);
MPI_Barrier(Com);
t1 = MPI_Wtime();
for(i = 0; i < B/s; i++) {
```

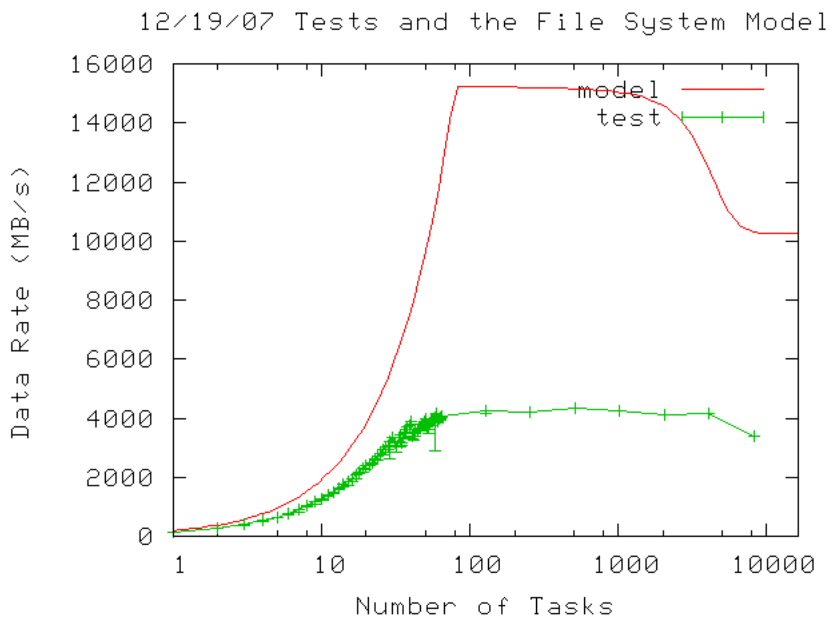

Figure 7: Initial testing does not quite fit the model

```
  write(fp, buffer, s);
}
MPI_Barrier(Com);
t2 = MPI_Wtime();
local = t2 - t1;
MPI_Reduce(&local, &global, 1, MPI_DOUBLE, MPI_MAX, 0, Com);
if( (rank == 0) && (global > 0) )
    printf("R(%d) = %f\n", size, size*B/global);
```

In particular, note that the slowest task determines the reported data rate for the cluster as a whole. A later section will discuss an alternative measurement strategy.

3.7 Test Run

The results from the December 19, 2007, test are in Figure 7 along with the expected performance as given by Equation 4. Testing began by running IOR with every task count from 1 to 64 (rather than 80, see label "phase I" in Figure 8). That was followed by a few tests with the default stripe count set to four ("phase II" in Figure 8), which are not represented in Figure 7 (see Section 3.9.3). Then the scaling study continued for powers of two from 128 to 8192, and with stripe count set back to one ("phase III"). The test for 16384

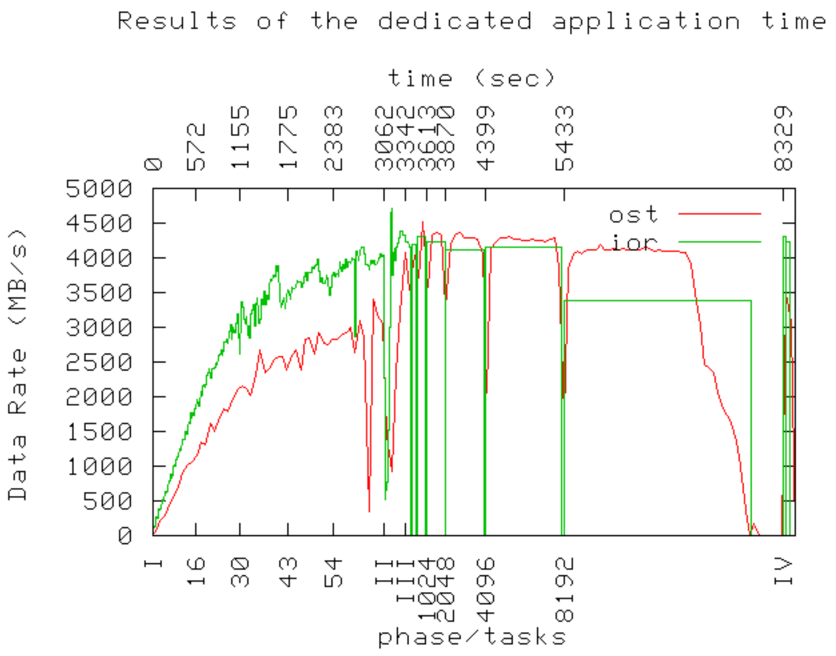

Figure 8: Tracking performance over time on the OSTs

tasks had to be cancelled for lack of time. A few more, smaller tests ("pase IV" - also not included in Figure 7) ran at the very end of the dedicated application time.

In addition to test results produced by the IOR benchmark application the individual OSTs were monitored, minute by minute, for their reported instantaneous data rates. Figure 8 plots the reported data rates, aggregated over all 80 OSTs, during the course of the testing (with label "ost"). Superimposed on that plot is a test by test track of the data rate being reported out (at the end of each test) by IOR (label "ior"). There was a four second quiecent period between each test. For the first tests, in the linear scaling region, the tests themselves were only about eight seconds long, thus the trace labeled "ior" in Figure 8 overestimates the activity on the file system. Beyond the linear region, where the tests took longer the decent to zero is included in the "ior" trace in order to highlight test boundaries.

3.8 Repeatability

In Phase I each test ran three times. Figures 7 and 9 show the error bars, and Figure 8 shows the highest of the three. In the interest of time larger tests were only run once.

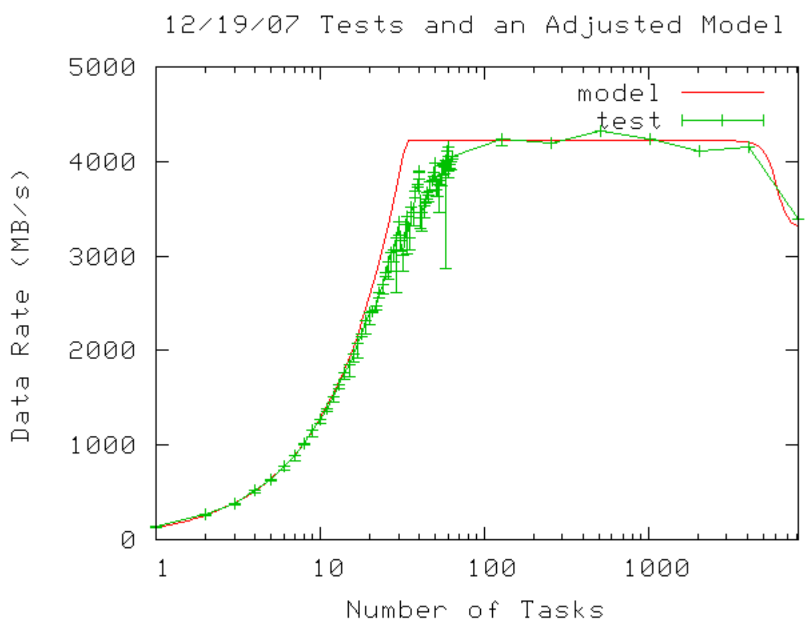

Figure 9: Adjustment of the model to the test data

3.9 Analysis and Feedback

Figure 7 demonstrates that testing does not support the model as initially presented. The values for the constants in the model were chosen based on initial notions of expected component performance. If the model is to be salvaged those constants need to be adjusted. The following values contribute to the revised model depicted in Figure 9.

- $\mathcal{O} = 32$
- $r_{\mathcal{S}} = 128MB/s$
- $r_{\mathcal{R}} = 100MB/s$
- $T = 6144$
- $\tau = 512$

Of the adjusted values the streaming data rate for a single LUN is the best justified. Over the first 20 tasks the match appears to be very good. It is apparent that the OSTs do not scale linearly to all 80. The notion that there is a transition from streaming to random load on the DDNs at about 6144 tasks is poorly supported. The one data point at 8192 tasks has an alternative explanation. One aspect of that test, which shows up vividly in Figure 8, offers

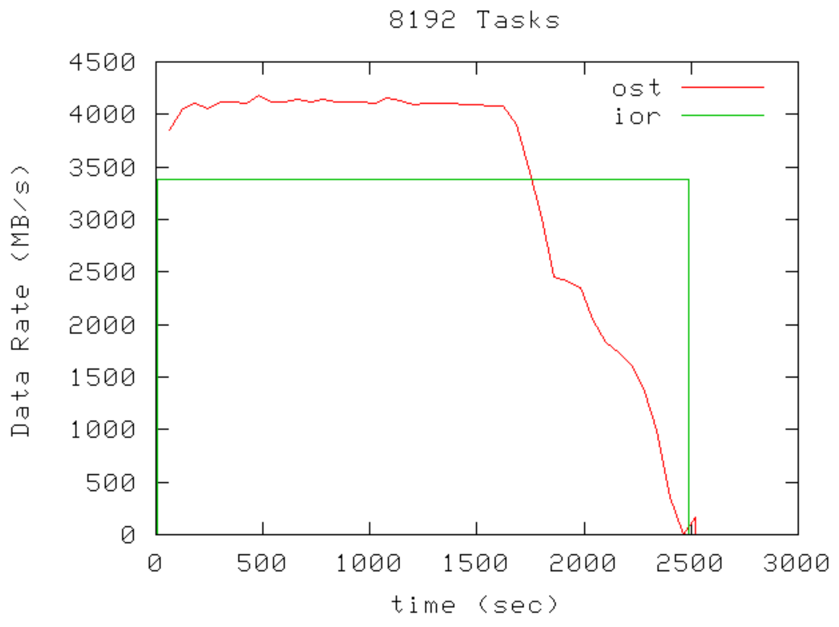

Figure 10: Some tasks straggled in later than the rest

an alternative explanation for the drop in performance. Section 3.9.1 pursues this point.

The file system is only getting full bandwidth up to about 32 OSTs, or about $1 - 1/2$ LUNs per OSS. It would seem that there is some other limiting factor, as yet unexplained. There may be some resource contention at the OSS that prevents the four OSTs from getting full bandwidth. Alternatively, there may be a limiting factor elswere in the I/O path preventing the bandwidth from exceeding about $4GB/s$ aggregate. Section 3.9.2 examines the region of linear scaling more closely.

3.9.1 8192 Tasks

Figure 10 presents the portion of Figure 7 for the 8192 task IOR test. For about the last 40% of the test the OSTs were slowly going idle. The poor performance over that period results in a reported data rate 20% below what the other tests reported.

Figure 11 traces each OST individually over the course of the test. Note that some OSTs finish very early and others very late. Once most of the tasks are complete a few OSTs are able to perform close the previously reported rate of $128MB/s$.

Figure 12 aggregates the rates over the four OSTs on each OSS for a similar set of per-OSS traces. Now notice that no OSS finishes particularly early, but

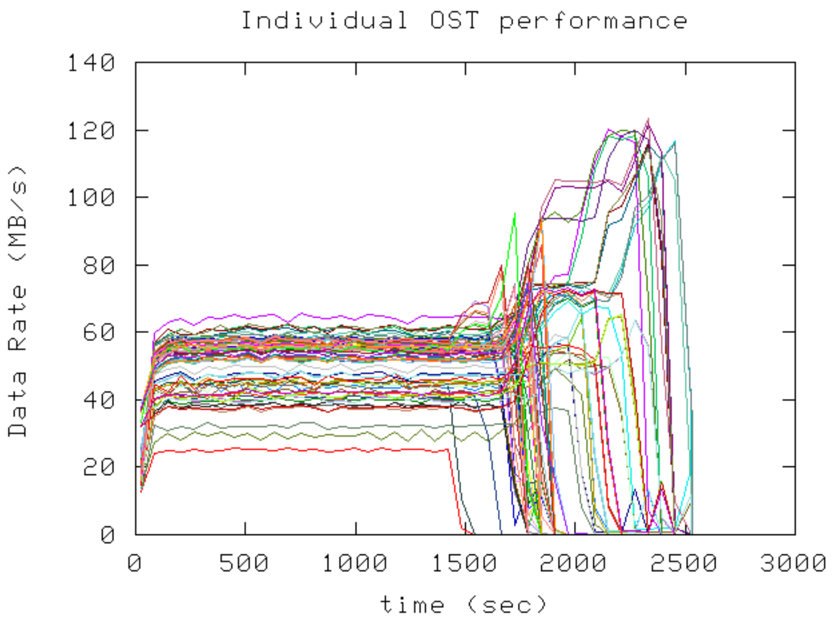

Figure 11: Some OSTs do more work than others

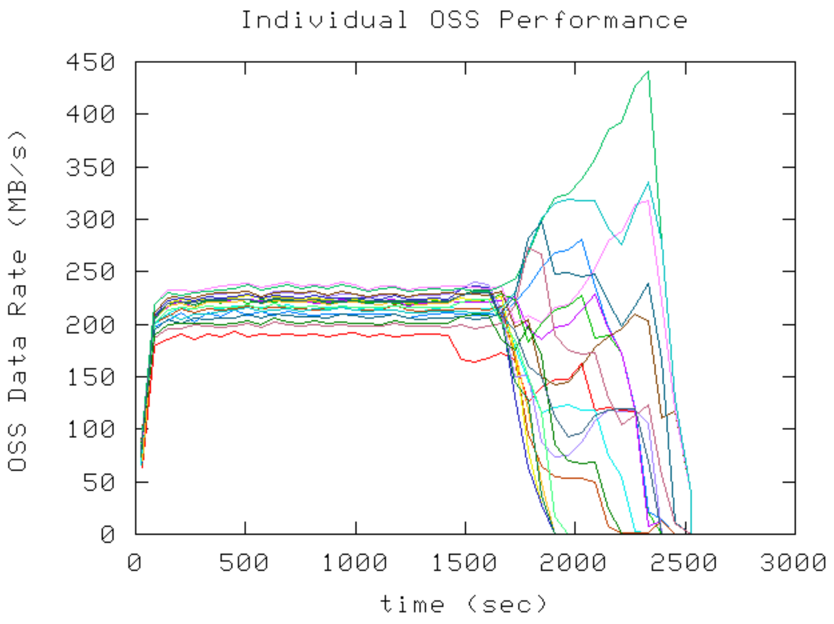

Figure 12: No OSS finishes especially early

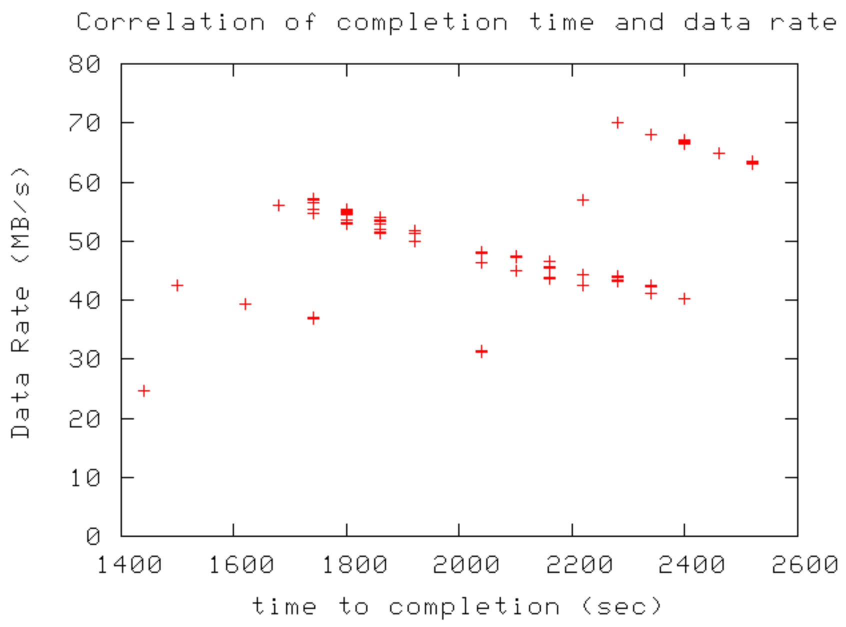

Figure 13: Allocation is quantized at around 30 files

several finish late. One OSS appears to have all four of its OSTs still going. Are some OSSs not getting a fair share of the available bandwidth? The crowded section of the test does not seem to have that much variation in the rates.

The area under the curve for a given OSS represents the amount of data that OSS was responsible for delivering to the test as a whole. It looks like some OSSs were having to do more work than others. Figure 13 reorganizes the data from Figure 11 to show, for each OST, the average rate it observed and the time it took to complete its assigned set of 100 or so files. Since all the files were the same size the figure gives estimates for the the number of files each OST was assigned:

$$f = t * r_{\text{ave}}/B$$

where:

- f is the number of files assigned to the OST
- t is the time to completion for that OST
- r_{ave} is the average data rate observed for the OST
- B is the file size used during the test

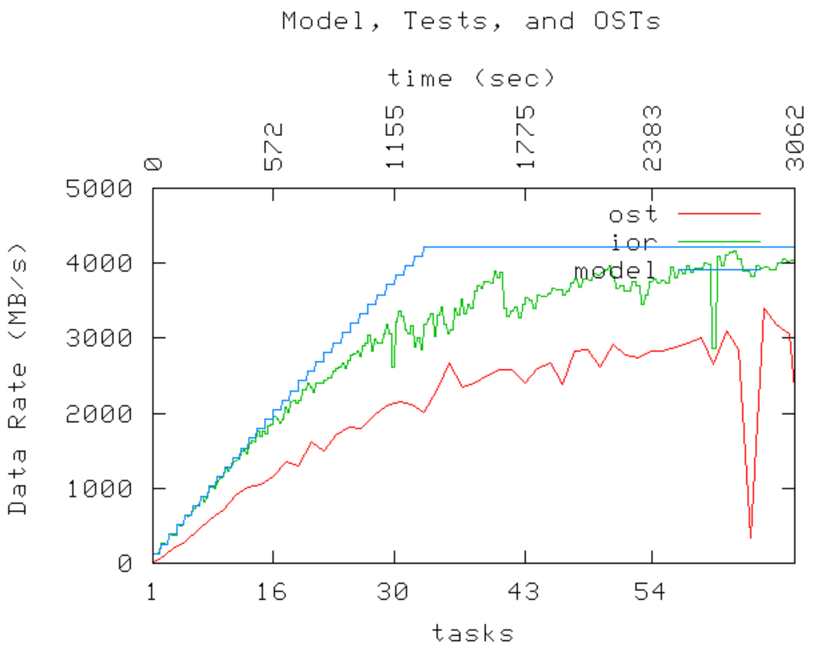

Figure 14: Testing in the region of linear scaling

There are three lines and two isolated points in the figure. The points in a line correspond to approximately the same number of files f, so there were five different loads assigned to the OSTs as follows[2]:

files	OSTs
35	1
62	5
92	66
124	1
156	7

It would appear that the MDS does not hand out files to OSTs individually, but rather does so 30 files at a time. Furthermore, there does not appear to be a strict round-robin alternation amongst the OSTs. Something interesting is happening, the MDS appears to be responsible, and it is disrupting the I/O pattern at scale. The low resolution of the OST monitoring data makes it difficult to determine if the effect is only at scale. The three large tests preceeding the 8192 task test did not seem to be adversely affected.

3.9.2 The linear region

Figure 14 reproduces the portion of Figures 8 and 9 from the region of linear scaling. The correspondence is very good between the observed rates and the expected value of $n * 128MB/s$ up to around 20 tasks. For the tests with 21 up to 40 tasks it looks like the general trend is much more noisy. The second OST per OSS only contributes an extra $73MB/s$ each. By 60 tasks the third OST per OSS has only contributed about $12MB/s$ each. Similarly, the fourth OST per OSS only adds another $12MB/s$ each (based on 128 tasks, since we did not test at 80 tasks). Based on the analysis in Section 3.9.1 the whole notion that testing in the linear region is adding one more OST in each test from 1 to 80 is in question. The fact that it seems to be borne out in the region from 1 to 20 tasks is suspicious, though. It is possible to contrive a test that definitely has all files distributed uniformly in a round-robin fashion. It might be worthwhile to do so.

The saw-ridge shape of the test results may be explained as follows: IOR calculates the reported data rate by dividing the aggregate amount of data transferred by the interval from before the start of testing to end of the last test to complete. Thus if one task is especially slow it can unduely influence the aggregate rate. If there is some resource contention between two OSTs on an OSS then at 21 tasks there will one OSS that is apparently slower than the rest. From the observations above it looks like two OSTs sharing an OSS get about $100MB/s$ each instead of $128MB/s$. The testing interval would be $10.24s = 1024MB/(100MB/s)$. In that case IOR would report:

$$r_{21} = 21 * 1024\text{MB}/10.24s$$

Or about $2100MB/s$ rather than the $2688MB/s$ expected by linear scaling. In fact, the 21 task test got closer to $2400MB/s$. A test of 41 tasks experiences the same penalty compared to 40 tasks. The observed drop is about 10%, though the forgoing analysis would predict closer to 30%. Whatever is happening appears to be rather complicated. Follow-up tests should remove all uncertanty from the mapping of files to OSTs. An explanation for the lack of scaling in the linear region may also reveal why the peak performance at scale is below expectation.

3.9.3 Additional Tests

Additional tests during the same period showed some other interesting results.

- Stripe Count Four

3.9.4 Recommendations

We want to characterize the impact of the non-uniform file distribution. To do that, run one test at scale with uniformly distibuted files. Verify that OSTs

[2]The total of all the files in the table is about 6% fewer than the number of files that were actually in the test, so the results are only approximate.

and OSSs receive a fair distribution of the available bandwidth. This should be at 8192 tasks, since that is the test we have to compare against. A similarly constructed test at 16384 tasks might be revealing. If we can sample more frequently the shorter tests would be useful to verify that the effect is only noticable at scale.

What is the limit that one task (one node) can produce. For one and two tasks, run a sequence of tests at increasing stripe count.

We want to understand what the limiting factor is in scaling up to 80 OSTs.

Examine the maximum possible performance of a single LUN. On a quiecent system can we get full bandwidth from a single OST on a single OSS? Run a series of tests from increasing numbers of nodes, but all writing to files mapped to a single LUN. Does the performance actually exceed the previously observed $128MB/s$ for some loads?

Similarly, run the four OSTs of a single OSS with increasing loads to see if it will deliver full bandwidth to all four. Will it for all 20 OSSs?

We want to characterize the maximum possible performance of a single DDN. Using from 1 to 16 files distributed over the LUNs of a single DDN examine the performance curve. What about 2 files per LUN, and so forth?

Can we find pairs of OSSs that, when fully loaded, appear to interfere with each other? Does it depend on which pairs? Exactly what combinations of fully loaded OSSs are able to saturate the available resources? Do the LUNs of a DDN compete for back-end disk bandwidth? How loaded does a DDN have to be before the limitting factor is on the OSS side rather than the DDN side?

- Preserve files after each test long enough to generate a report on the mapping of files to OSTs.
- Conversely, run a similar series of tests with files in a known even distribution accross OSTs.
- Sample the OSTs more frequently.
- Monitor OSS CPU utilization
- Monitor the iostats across the two FC4 links
- Monitor RPC size
- Use mib to generate client-side performance profiles.

4 Data Management

Benchmarking is a data intensive activity. It is important to organize the recorded information from each test in such a way that it is preserved and searchable.

The following items need to be recorded in a benchmark:

hardware - versions, configuration, and tuning details

computer system software - versions, configuration, and tuning details

file system software - versions, configuration, and tuning details

test application - versions, configuration, and tuning details

test parameters - the set of variables and range of values chosen

test results - the observed quantities and the values for each test

environmental considerations - Is the system otherwise quiescent?

5 Related Work

6 Conclusion

7 Acknowledgments