Lustre at Petascale: Experiences in Troubleshooting and Upgrading

Matt Ezell
High Performance Computing Operations
Oak Ridge National Laboratory and NICS
Oak Ridge, TN
ezellma@ornl.gov

Rick Mohr, John Wynkoop, and Ryan Braby
High Performance Computing Operations
National Institute for Computational Sciences
Oak Ridge, TN
{rmohr,jwynkoop,rbraby}@utk.edu

***Abstract*—Some veterans in the HPC industry semi-facetiously define supercomputers as devices that convert compute-bound problems into I/O-bound problems. Effective utilization of large high performance computing resources often requires access to large amounts of fast storage. The National Institute for Computational Sciences (NICS) operates Kraken, a 1.17 PetaFLOPS Cray XT5 for the National Science Foundation (NSF). Kraken's primary file system has migrated from Lustre 1.6 to 1.8 and is currently being moved to servers external to the machine. Additional bandwidth will be made available by mounting the NICS-wide Lustre file system. Newer versions of Lustre, beyond what Cray provides, are under evaluation for stability and performance. Over the past several years of operation, Kraken's Lustre file system has evolved to be extremely stable in an effort to better serve Kraken's users.**

***Keywords*-Lustre; HPC; Kraken**

I. Introduction

Some veterans in the HPC industry semi-facetiously define supercomputers as devices that convert compute-bound problems into I/O-bound problems. Effective utilization of large high performance computing resources often requires access to large amounts of fast storage. Recently vendors have begun talking about the "big data" problem, where datasets have grown too large to effectively manage with traditional tools and file systems.

The National Institute for Computational Sciences (NICS) operates Kraken, a 1.17 PetaFLOPS Cray XT5 for the National Science Foundation (NSF). Kraken has 112,896 cores, which can easily produce a barrage of data. If not managed effectively, performance can seriously degrade. The most common Lustre-related complaint is about performance; high utilization results in high I/O demands which can sometimes lead to performance degradation. This is not a reflection on the quality of the file system, but rather a statement about the size of Kraken. Years of production experience have taught storage administrators at NICS how to detect suboptimal I/O practices and recommend alternative techniques.

Kraken's primary Lustre file system consists of 2.4PB of DDN hardware capable of over 30GB/sec of streaming bandwidth. Early in 2012, the system was migrated from Cray Linux Environment (CLE) 2.2 to 3.1, thereby upgrading from Lustre 1.6 to 1.8. Bringing the file system to an updated release allows NICS to continue working towards the goal of treating all HPC file systems as center-wide resources. Planning is underway to move Kraken's internal Lustre servers to external servers, at which point the existing Lustre service nodes will be converted to LNET routers. This externalization will allow Kraken's original file system to be mounted on Nautilus, the NICS remote data and visualization resource. Additionally, a second NICS center-wide Lustre file system can be mounted on Kraken. This will increase the total bandwidth available for large science applications.

The Lustre 1.8.4 client, provided with CLE3.1, has known interoperability issues with the latest Lustre 2.X releases. This severely limits NICS' ability to evolve these critical center-wide storage resources to take advantage of newer Lustre improvements. As a result, future work will involve evaluating more recent versions of Lustre beyond those officially provided and supported by Cray.

Over the past several years of operation, Kraken's Lustre file system has evolved into an extremely stable resource which better serves Kraken's users. Ongoing and future improvements aim to provide optimal facilities for scientific discovery.

II. Original Lustre Configuration on Kraken

Kraken's Lustre storage hardware consists of 6 racks, each containing two DDN S2A9900 storage controllers connected to 560 one-terabyte hard drives. File system access is provided by 1 Metadata Server (MDS) and 48 Object Storage Servers (OSSs). Each OSS server provides 7 Object Storage Targets (OSTs), and each OST uses a 8+2 RAID 6 setup to protect against drive failures. The MDS and OSS servers are Cray IO nodes within the XT5 system, and all system nodes communicate with the Lustre servers using the high-speed SeaStar network.

The file system is designed to maximize IO bandwidth. Lustre servers access the OST storage via DDR Infiniband connections directly to the DDN S2A9900 controllers. Based on vendor specs, the storage hardware's peak raw throughput

for sequential reads/writes is 36 GB/s. Benchmarking tests have shown sustained throughput of up to 31.7 GB/s.

III. Operational Issues

A. Small I/O

One of the most common issues affecting Lustre performance is sub-optimal IO by user applications. In particular, applications which perform large numbers of small read/write requests can have a dramatic impact on the file system. This often manifests itself as high loads on the MDS/OSS servers.

While it is not practical to obtain detailed IO statistics from every node in every batch job, NICS has had significant success in identifying problem applications by looking at only high-level IO trends. NICS has a simple script which obtains the Lustre request history from one or more Lustre servers and, using information from the batch system about currently running jobs, calculates the number of Lustre requests sent by all nodes within each job. The number of requests can be calculated for all MDS/OSS servers, or any subset thereof. An example of the script's output is shown below:

```
kraken# ./lustre_requests
Job       User       Cores   Age     Count
1850782   userA      3072    00:06   85522
1849593   userB      600     09:10   39986
1850042   userC      2628    11:57   22386
1849819   userD      132     05:59   12368
1849929   userD      132     --      9994
1849722   userD      132     05:16   6855
1848293   userE      2160    00:52   6835
1850787   userF      120     --      6481
1849936   userD      132     02:12   5796
1850779   userG      24      00:11   5088
```

While the output does not make a distinction between different types of requests, it is still very useful for identifying anomalies. Experience has shown that the most problematic applications typically have request counts which are 1 or 2 orders of magnitude higher than other jobs on the system. The large number of requests is frequently an indicator that the application is performing small IO.

Once a job outlier has been indentified, admins can take a more targeted approach to investigating the job's IO pattern. A common first step is to login to one or more compute nodes assigned to the batch job and analyze the IO statistics in two of the files in /proc/fs/lustre/llite/{FSNAME}: stats and extent_stats. The first file provides information about system calls like open, close, flock, etc. The second file reports the distribution of sizes for the read/write calls on the node (which can quickly show when an application may be performing small IO).

As an example, a user application was generating a significant number of lustre requests and creating a high load on some of the lustre servers. When contacted about this, the user stated that his application only performed modest amounts of IO about every 5 minutes. The information from extent_stats on one of the compute nodes painted a different picture and showed frequent small write requests:

```
kraken# cat extents_stats
snapshot_time: 1325878779.789272
                read      |        write
 extents   calls % cum% | calls    %   cum%
 0K - 4K:    34  20  20 |  1758  98     98
 4K - 8K:     0   0  20 |     0   0     98
8K - 16K:   135  79 100 |    32   1    100
```

Based on this information, the user was able to identify a debug flag in his code which was unintentionally enabled. After disabling the flag, the IO load from his job dropped dramatically. The user also reported a 5x speedup in his code.

B. Lock Exhaustion on the MDS

The MDS server stores lock information in memory, and when there are large numbers of clients, the amount of memory consumed by locks can become significant. This becomes particularly important for MDS servers containing relatively small amounts of memory, such as the XT SIO service blades with only 8 GB of RAM.

During a recent incident, Kraken's MDS server began encountering OOM errors that recurred after about 4 hours. It was suspected that a large job was acquiring many locks and exhausting the memory on the MDS. NICS began tracking the memory usage of the Lustre locks by capturing the lines for ldlm_locks and ldlm_resources in /proc/slabinfo. The sizes of these slabs steadily increased over time. Just prior to the last OOM condition, ldlm_locks had 1098662 active slabs while ldlm_resources had 6685500 active slabs. At 4K per slab, the memory usage totaled (1098662 + 668550) * 4K = 7,068,848 KB. This is 87% of the available memory.

On several compute nodes associated with this large job, NICS captured lock information from /proc/fs/lustre/ldlm/namespaces/scratch-MDT0000-mdc-ffff880403026c00. Each compute node had 900+ locks cached, although only 1 or 2 seemed to be active. Based on the size of the job, it was clear that these locks accounted for the memory usage on the MDS.

The compute nodes had *lru_size* set to 1200 (100 locks per core) and *lru_max_age* set to 9000000 seconds. The high *lru_max_age* was preventing locks from aging out while the batch job was running, and it was clear that the MDS did not have the memory to accommodate 1200 locks per compute node. A rough calculation showed that the MDS should be able to accommodate 300 locks per compute node. After the *lru_size* was set to 300, the MDS no longer experienced OOM conditions.

C. Client Credit Calculation

Lustre uses a "credit system" as a flow control mechanism between peers. At the Lustre Networking (LNet) layer, two main parameters exist to control this. The "credits" parameter controls how many LNet messages can be sent concurrently over a given network interface (NI), and the "peer_credits" parameter controls how many LNet messages can be sent concurrently to a single peer. For large file systems with many clients, this must be carefully tuned to find a balance between performance and overwhelming the servers.

The Cray CLE install script, by default, sets the number of client credits to 2048. This was quickly determined to be excessive, as Kraken's OSS server were frequently becoming overloaded and unresponsive. Unfortunately, NICS staff were unable to find specific recommendations providing a formula to calculate an appropriate number. The number of credits was slowly reduced to 192 on each compute node. The aggregate performance was not degraded but the maximum load on the OSS servers declined. Single-node performance was slightly limited due to this change.

D. OST Allocation Method

When a new file is created, the MDS server determines which OSTs the file will be striped across. Lustre uses one of two possible allocators when making this determination: Quality of Service (QOS) or Round-Robin (RR). The primary purpose of the QOS allocator is to keep OST utilization as uniform as possible. This allocator employs a weighting mechanism that favors OSTs with the most free space. On the other hand, the RR allocator is designed to maximize IO bandwidth by giving preference to OSTs on different OSS servers.

Lustre's choice of allocator is determined by the *qos_threshold_rr* tunable parameter [2]. This parameter is expressed as a percentage with the default value being 17%. If the difference between the maximum and minimum free space on any of the OSTs exceeds the threshold, the QOS allocator is used in an attempt to rebalance OST usage. If the difference is below the threshold, Lustre considers the OST usage to be balanced and will use the RR allocator to improve IO performance.

The distribution of OSTs allocated to a file can impact IO performance. In an effort to quantify the impact, IOR benchmarks were run when the file system was idle. Each test ran on 300 nodes using a POSIX file-per-process pattern to create files with a stripe count of 1. This configuration was chosen specifically to achieve maximum IO bandwidth. During the test, the *qos_threshold_rr* parameter was manually set to 0 and 100 to ensure use of the QOS and RR allocators respectively. Two IOR tests were run for each allocator method. The results are shown in Table I. Note: These results were run with CLE2.2 which used Lustre 1.6.

Table I
IOR POSIX FILE-PER-PROCESS (CLE 2.2, 300 NODES, 1 STRIPE)

Test	Max Write (MB/sec)	Max Read (MB/sec)
QOS 1	9760	9465
QOS 2	9437	8981
RR 1	29880	18970
RR 2	29987	20486

Table II
IOR POSIX FILE-PER-PROCESS (CLE 2.2, 300 NODES, 4 STRIPES)

Test	Max Write (MB/sec)	Max Read (MB/sec)
QOS 1	7797	11930
QOS 2	8444	12666
RR 1	9969	16886
RR 2	12653	16590

The performance difference is significant. The max aggregate write speed for the RR-allocated files was over 3x the write speed for the QOS-allocated files. The max read speed was more than double.

For each individual process in the IOR test, one can calculate the amount of time spent performing IO and determine which OSS/OST handled the IO requests. The results clearly show that the difference in performance is due to oversubscription. For the RR allocator, OSTs are assigned to files in an optimal manner. All OSS nodes service about the same number of clients (6 or 7), and each OST is assigned to only a single file. On the other hand, the QOS allocator will assign the same OST to multiple files, and some OSS nodes service 10-12 clients while other OSS nodes service only 3-4.

This data does yield one curious observation. The lower read performance for the QOS allocator appears to be due to contention at the OST level. More clients accessing the same OST results in longer read times. However, the lower write performance of the QOS allocator seems to be related to contention at the OSS level. On the same OSS server, an OST written to by a single client exhibited IO times that were nearly identical to an OST on the same server which was written to by 3 or 4 clients.

In addition to the previous IOR tests, another set of identical tests were run using files striped across 4 osts. The results are shown in Table II. In this scenario, there is OSS/OST contention even when using the RR allocator, resulting in IO rates which are closer to those of the QOS allocator. However, the RR allocator equally distributes the contention so we still find that the RR allocator is outperforming the RR allocator.

Based on these results, NICS decided to increase the value of *qos_threshold_rr* from 17% to 55% in order to increase the chance of using the RR allocator.

E. Purging

It is known that Lustre performance may degrade to some extent when the file system usage becomes very high. In

order to maintain performance and ensure adequate free space for running jobs, it is necessary to routinely delete files which have not recently been used. For the purposes of purging, a file is considered to be "recently used" if it has been accessed or modified within the last 30 days.

The current purge process is driven by some basic shell scripts that internally use either the "lfs find" or "find" commands to locate files with an atime and mtime older than the purge threshold. These files are then removed from the file system, and the list of files is logged for future reference.

This approach, while functional, has several significant drawbacks. Both the "lfs find" and "find" commands generate extra metadata traffic and are severely limited by the performance of a single MDS. There is also no way to continue a purge process which has been interrupted. It must be restarted from the beginning. This is especially problematic when the purge is interrupted midway through a directory tree containing millions of files, many of which may not even be eligible for purging.

To improve this process, NICS is working to deploy the ne2scan utility developed by Nick Cardo at NERSC. The ne2scan tool reads directly from the MDT device (avoiding additional metadata load) and outputs a list of all files in the file system with the corresponding metadata. This list of files can then be fed into a series of other scripts to purge files which match certain criteria. The ne2scan output could also be used to identify files with potentially undesirable traits (very large files with low stripe counts, directories with excessive numbers of files, etc.) This information could prove useful in targeting performance issues before they escalate into file system wide problems.

F. Poorly Striped Files

Poor file striping choices are a common source of problems for users and administrators. In the majority of cases, this involves setting the file's stripe count too low (typically to 1). The effect on user applications is usually poor IO performance. Once identified, the problem is easily remedied by restriping the file, and this simple solution can have a significant impact on code performance. As an example, a NICS computational scientist recently worked with a user to increase the stripe count on their input file from 1 to 10. The user reported that this simple change decreased the runtime for their application by about 30%.

Unfortunately, these same striping choices cause more painful system level problems. One problem that can occur is high OSS server load. If all the processes in a large parallel application are simultaneously accessing the same stripe count 1 file, the target OST quickly becomes overloaded. Requests to the OST queue up, and the utilization (as reported by iostat) quickly climbs to 100%. Not only does this adversely affect IO by other users to this particular OST, it can choke out IO to all other OSTs on the same OSS server. (It was this symptom that lead NICS administrators to identify the user in the above example.)

Another common system problem caused by poor striping involves filling up OSTs. All too often, users will create large files with small stripe counts, effectively filling up a small number of OSTs. If an OST gets too full, it can lead to IO errors that kill user applications. Having OSTs with high usage can also force Lustre into using the QoS allocator instead of the RR allocator. As previously illustrated, this can result in significant performance degradation for large streaming IO.

To reduce the negative impact of large files with low stripe sizes, we closely monitor OST usage. Any OST with more than a 10% deviation from the average OST usage becomes suspect and needs to be investigated further. While an extremely under-utilized OST may be of concern, the most common scenarios involve extremely over-utilized OSTs.

When an OST has significantly higher utilization compared to the other OSTs, we execute a script designed to query the OST for usage based on a list of users in our LDAP directory for that system. This script, which was developed in house, helps snapshot all users usage on a given OST. The list is compared to our total file system usage statistics and, typically, the offending user is identified as having disproportionate usage on the suspect OST compared to his/her overall file system usage. Once we have identified the offending user, we concentrate on finding the file(s) causing the OST imbalance. A cursory look in the user's scratch directory will usually yield a list of possible problem files (large archive files are usually identified as the culprits). However, 'lfs find' may also be used to generate a list of files for a given user on a single OST.

Once a problem file is identified, its stripe count and size is confirmed using 'lfs getstripe' and 'stat'. If the situation is not critical, we ask the user to re-create the file using a stripe count that is more appropriate to the file size (typically 1 stripe for every 50-100 GB). This balances the load well across multiple OSTs without causing performance penalties for smaller files. Because Lustre does not support dynamic re-striping, the only method for fixing the striping is to copy the original file to a new file with the necessary stripe count, and rename the new file over the original file. This is a labor intensive process for the user and good documentation is important. Nevertheless, often times the user will not respond in a timely fashion and operations staff will intervene and perform the procedure for the user in order to remedy the problem in a timely fashion.

This approach is extremely time intensive and inefficient. NICS hopes to improve the process by using the output from ne2scan to preemptively identify files with sub-optimal striping. This will enable faster detection and remediation of problem files than is currently available.

G. *Monitoring*

Effectively monitoring a large Lustre file system can be extremely difficult. Catastrophic failures are simple to spot, but identifying performance issues may be more elusive.

Kraken's Lustre is currently located inside the Cray, with the servers being XT service nodes. Their only network interface is the internal SeaStar connection. This makes monitoring difficult, as the central NICS monitoring system cannot directly communicate with the Lustre servers. To remedy this, NICS has created scripts that run on Kraken's boot node to aggregate and proxy requests to the central Nagios server. The first script, *check_lustre_ping*, simply makes sure that all of the servers are able to respond to a ping request within a specified time period. The next one, called *check_lustre_load*, is intended to ensure the load is at a reasonable level. High loads may be indicitive a problem on the server or a user performing poor I/O.

To track down poor I/O patterns, the *lustre_requests* script described in Section III-A is used to determine likely candidates for an I/O review.

NICS has enabled very basic monitoring of bandwidth and IOPS for the file system in it's existing configuration by utilizing statistics available from the DDN disk arrays directly. NICS has configured our monitoring system to use the DDN S2A API to collect statistics directly from the DDN controllers. This data is then stored in Cacti, a common Open Source graphing package based on RRDTool. Performance data is polled every five minutes via a Perl script NICS developed in house based on documentation provided by DDN. This script creates a socket connection to each of Kraken's twelve DDN controllers. This data is then stored in an RRD format and aggregated using custom data input methods in Cacti. The data for each controller, as well as the aggregate of all controllers, is displayed in standard Cacti graphs (see Figure 1). Additionally, NICS uses some of the data manipulation functions of Cacti to calculate and graph the average I/O size sent to the DDN controllers. While this does not give instantaneous performance information, mainly due to the large time window between samples, it does allow very broad generalizations of Lustre disk performance for a given time period. This data is helpful for trending and basic health monitoring, but is only a point in time snapshot of the disk subsystem and must be taken in context.

Monitoring the DDN storage controllers is achieved through the use of the DDN application programming interface. This is a socket-based protocol that allows data to be programmatically obtained from the controllers. Scripts have been developed to alert on failed disks, failed power supplies, and other error conditions.

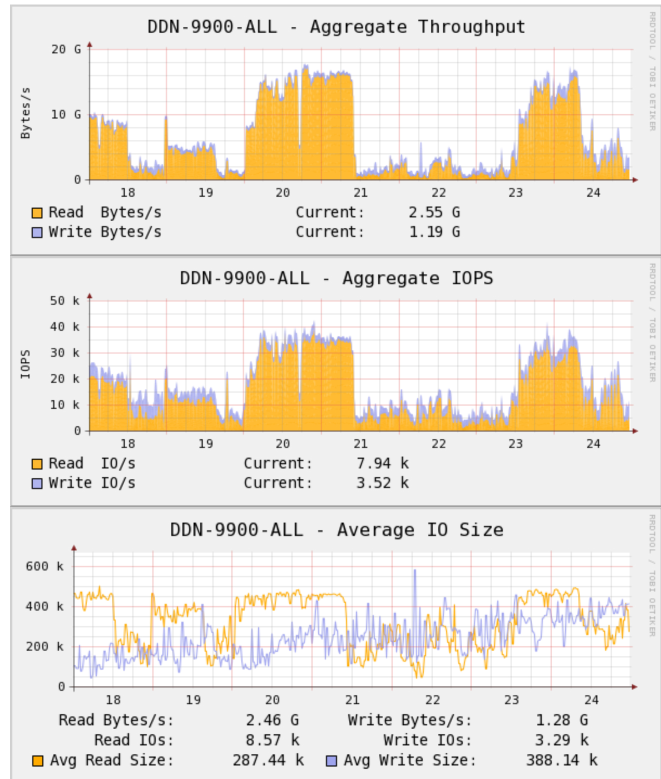

Figure 1. Cacti Graphs

IV. CLE 3.1 Upgrade Experiences

A. *CLE 3.1 Features*

CLE 3.1 is a major software upgrade from CLE 2.2. One major difference is the base operating system version: CLE 2.2 ran on SuSE Linux Enterprise Server (SLES) 10, while CLE 3.1 is based on SLES 11. Additionally, the Lustre client and server versions are upgraded from 1.6.5 (plus Cray patches) to 1.8.4. Due to the large number of changes, a full reinstall is required.

B. *Athena Test Shot*

Athena was a 48 cabinet Cray XT4 system that was co-allocated with Kraken. It had an 80TB Lustre file system served by Fibre Channel RAID devices. Athena was decomissioned to users in 2011, but two cabinets were left powered on to facilitate software testing. Athena was upgraded from CLE 2.2 to CLE 3.1 while preserving the Lustre file system. No significant issues were encountered, giving hope that the upgrade would go smoothly.

C. *Early Kraken Test Shots*

Since the Cray software infrastructure supports multiple software roots, an alternate root on Kraken was upgraded to CLE 3.1. During several test shots, NICS staff booted into this alternate root to allow the computational scientists to build software packages and libraries under the upgraded

operating system. Because of worries that files created under Lustre 1.8 might not be compatible under Lustre 1.6, the scratch Lustre file system was not mounted.

D. ib_srp with scsidev-emulation

Unlike fibre channel devices that are typically seen without any configuration, Infiniband devices require the system to "log in" before the device is usable. Infiniband uses the SCSI RDMA Protocol (SRP) to allow the use of SCSI devices over Infiniband. Under CLE 2.2, NICS used a startup script to manually login to each device. While this worked, NICS decided to use the more standard *srp_daemon* for the CLE 3.1 install.

With CLE 2.2, Cray supported the *scsidev* program to provide persistent device naming. Under CLE 3.1, *scsidev* was considered deprecated, prompting Cray to develop *scsidev_emulation*. *scsidev_emulation* is a set of *udev* rules and helper scripts that mimicked the behavior of *scsidev*. When the kernel adds new *sd* devices, it will spawn the script */lib/udev/udev_scsidev.sh* to look up the appropriate alias. This script, in turn, uses *sginfo* to determine the serial number of the device. Unfortunately, for Inifiniband SRP devices, *sginfo* does not appear to work from within *udev* when a device is first added. It complains and fails to return a serial number, causing *scsidev_emulation* to fail. As a workaround, a startup script was created to call *udev-trigger* later in the boot process to setup the appropriate persistent device names.

E. MDS Hardware Incompatibility

The metadata server backend storage target (MDT) for Kraken's file system was originally part of an LSI boot RAID device, but it was quickly upgraded to a DDN EF2915 fibre channel device [1]. The DDN MDT had approximately 3.5TB of space available for the file system's metadata. During Kraken's early test shots with CLE 3.1, simple verification was performed to ensure that the block device was present. As mentioned earier, no attempts were made to mount the Lustre devices for fear of creating incompatible files.

NICS determined that Kraken would go live with CLE 3.1 on February 2nd, 2012. After all the service nodes had booted, Lustre began to start. Quickly, NICS staff noticed a problem. While the OSTs were able to mount, they were failing to connect to the MDS. In fact, it did not appear that the MDS had mounted correctly. Looking more closely at the console logs, a set of troublesome messages was found:

```
READ CAPACITY(16) failed
Result: hostbyte=0x07 driverbyte=0x00
Use 0xffffffff as device size
4294967296 512-byte hardware sectors:
                        (2.19 TB/2.00 TiB)
```

Apparently, the 3.5TB MDT was only being seen as a 2TB device. Clearly, this was a major issue that was preventing the file system from coming online. NICS contacted Cray for assistance and began scrambling to understand what was going wrong. Initially, it was believed that the QLogic driver that shipped with CLE 3.1 was incompatible with the DDN EF2915. With no clear path forward to resolving the issue, NICS decided to revert to CLE 2.2 and return to production.

Upon further investigation, NICS staff found evidence that the issue was in the SuSE kernel, not the QLogic driver. In the kernel's *drivers/scsi/sd.c* the function *sd_read_capacity* had changed. Specifically, `cmd[13] = 12;` had changed to `cmd[13] = 13;` and the `scsi_execute_request()` function used length 13 instead of 12 when it was trying to do a *READ CAPACITY(16)*. In debugging, NICS staff noticed the QLogic QLA2xxx driver dropping frames noting that it dropped `0x1` of `0xd` (1 of 13) bytes.

In parallel to this, NICS was working with DDN to determine if a firmware upgrade would improve the situation. DDN quickly determined that there was no firmware update to address this issue. DDN reproduced the problem on a white-box SLES 11sp1 machine connected to an EF2915 running the latest firmware. It was unlikely that any new EF2915 firmware would be created because the device was past its end-of-life.

This left NICS with two options. First, Cray could revert some of the kernel scsi code back to the version present in SLES10. This was considered infeasible because NICS did not want a one-off kernal and nobody at Cray is an expert in this part of the kernel. The other option was to change the hardware on our MDT to a device that understood larger SCSI commands. NICS worked with DDN to acquire a DDN EF3015 device that was setup in RAID 10 mode. During a downtime, the *dd* command was used to copy the data from the old device to the new one.

F. Production with CLE 3.1

The final steps to complete the Kraken upgrad to CLE 3.1 were completed on March 8th 2012. Kraken has run in production on CLE 3.1 with Lustre 1.8 since then. So far, no significant Lustre issues have been discovered.

V. NICS Global File System Goals

As NICS has grown through the additions of the RDAV and Keeneland projects, it has become clear that users would benefit from having a parallel file system that is available on all of the HPC resources at NICS. This would ease data management and reduce data replication. The RDAV and Keeneland projects have already taken a significant step towards this with the deployment of Medusa file system at NICS. Medusa is a Lustre file system served off of DDN 10K based storage with external OSS servers connected to a

QDR IB SAN. RDAV and Keeneland systems are connected to this SAN and they share this single parallel file system.

A. Path Forward

Unfortunately, Kraken has so far been unable to share this file system with the other systems. CLE 2.2 on Kraken used a Lustre 1.6 client, whereas Medusa was running 1.8 servers. Providing sufficient bandwidth from Kraken to the SAN while maintaining the Lustre file system served from the SIO nodes is problematic. There are a finite number of SIO nodes, the majority of which are used to serve the current Kraken Lustre file system.

Following the CLE 3.1 upgrade on Kraken, NICS is now running Lustre 1.8 versions on both Lustre file systems at NICS. Also four Infiniband connections have been run from Kraken to the SAN. This has allowed us to begin testing of mounting the Medusa file system on Kraken.

B. Expanding the Medusa File System

NICS is also working on expanding the Medusa file system to reach a peak bandwidth of 40 GB/s and a capacity of approximately 2PB. With this expansion NICS will add more Infiniband connections between Kraken and our SAN, and the mounts of the medusa file system on Kraken will move from testing to production.

C. Externalizing Kraken's File System

NICS is currently deploying external servers that will be used as new Object Storage Servers to externalize the Kraken file system. After the Medusa file system is in stable production on Kraken, NICS will convert the current SIO nodes in Kraken that are running as Object Storage Servers into LNET routers. The new external Object Storage Servers will be connected to the existing Object Storage Targets. The MDS will be moved from an SIO node to an external server and the file system will then be on the SAN and accessible by all the production HPC systems at NICS. The expectation is that all of this can be done without losing any user data on the file system.

VI. Future Work

As usual the future is sure to present many interesting challenges. After finishing moving the existing Kraken file system to an external Lustre file system and mounting both production file systems on all the production compute platforms at NICS, there are interesting tasks ahead. Currently CLE 3.1 is the last planned release of software from Cray to support the SeaStar Interconnect in Kraken. This means our current 1.8 version of Lustre on Kraken will be the last version supported by Cray. Yet, there are features and improvements in the Lustre 2.2 code that we may want for production use. At some point, NICS will need to make a decision to either stay with Lustre 1.8 on all systems or develop a plan for safely upgrading to a Lustre 2.2 client on Kraken.

Managing multiple production Lustre file systems mounted on the mixture of system types at NICS may present interesting challenges. The issues described previously of tracking down poorly behaved user IO will now span multiple production systems. Further, it is fairly clear that with multiple production systems mounting the same parallel file systems that contention issues between users and systems may become a recurring theme. Thankfully, it is expected that this can be mitigated through quality of service settings in the Infiniband software layer.

VII. Conclusion

The Lustre file system on Kraken is a stable, high performance choice for user data storage. While performance issues do occur, NICS staff members actively work to identify and fix these issues. Over time, NICS staff has gained valuable knowledge in tuning and troubleshooting. The result has been continued improvement in file system reliability and speed. With the recent upgrade from CLE 2.2 to 3.1, Kraken's Lustre file system is now at a semi-modern version. NICS is also pursuing the option of moving the Lustre servers off the Cray hardware to external hosts, and repurposing the existing OSS service nodes as LNET routers. By externalizing the file system, Kraken failures would no longer directly impact the Lustre servers. The external Lustre servers would also be better equipped (hardware-wise) than the current Cray service blades, which should further improve stability and performance.

References

[1] John Walsh, Troy Baer, Victor Hazlewood, Junseong Heo, Rick Mohr. Large Lustre File System Experiences at NICS. Cray User Group 2009.

[2] Lustre 1.8 Operations Manual. June 2010.