Optimizing Performance of HPC Storage Systems

Optimzing performance for reads and writes.

Torben Kling Petersen, PhD
HPC Storage Solutions
Xyratex Ltd
Langstone Road, Havant, Hampshire PO9 1SA, UK
Torben_Kling_Petersen@xyratex.com

John Fragalla
HPC Storage Solutions
Xyratex International Inc.
46831 Lakeview Blvd, Fremont, California 94538, USA
John_Fragalla@xyratex.com

Abstract — **The performance of HPC storage systems depend upon a variety of factors. The results of using any of the standard benchmark suites for storage depends not only on the storage architecture, but also on the type of disk drives, type and design of the interconnect, and the type and number of clients. In addition, each of the benchmark suites have a number of different parameters and test methodologies that require careful analysis to determine the optimal settings for a successful benchmark run. To reliably benchmark a storage solution, every stage of the solution needs to be analyzed including block and file performance of the RAID, network and client throughput to the entire filesystem and meta data servers. For a filesystem to perform at peak performance, there needs to be a balance between the actual performance of the disk drives, the SAS chain supporting the RAID sets, the RAID code used (whether hardware RAID controllers or software MD-RAID), the interconnect and finally the clients. This paper describes these issues with respect to the Lustre filesystem. The dependence of benchmark results with respect to various parameters is shown. Using a single storage enclosure consisting of 8 RAID sets (8+2 drives each) it is possible achieve both read and write performances in excess of 6 GB/s which translates to more than 36 GB/s per rack of measured client based throughput. This paper will focus on using Linux performance tool, obdfilter-survey, and IOR to measure different levels of the filesystem performance using Lustre.**

Keywords—HPC Storage, Benchmarking,

I. Introduction

Benchmarking system performance has always been an important factor in the HPC industry. This is especially true during procurement phases of a new system. Although benchmarks seldom, if ever, provide any indication of real life performance of an application (unless the application itself is benchmarked at intended scale which is usually hard before the final system is delivered), a benchmark allows an end user to compare an offered solution to others. In a number of RFPs, the customers have written their own benchmarks which is distributed to vendors with specific instructions of how they should be wrong. While this is a prudent approach, these benchmark codes rarely shows the peak performance of a new solution as they often were written on older, less capable solutions. For compute systems on the Top500 list, the benchmark of choice is Linpack [1] that will test a large system floating point performance as a result of the CPU architecture and system interconnect. For storage, however, there is no similar standard and unfortunately no equivalent list. This had led some vendors to state performance of their products in different ways such as aggregated theoretical disk bandwidth that obviously says nothing about the systems actual performance.

In the storage arena, a number of different benchmark suites exist. These can be used to measure a systems I/O performance regardless if the solution is based on direct attached storage, network attached storage (such as NFS based solutions) or a parallel distributed filesystem (such as GPFS or Lustre)[2, 3].

For general filesystem I/O, benchmark tools such as IOR [4], IOzone [5], Bonnie++ [6] as well as meta data performance of said filesystem such as MDtest [7]. For the purpose of this paper, we focused on obdfilter-survey and IOR.

II. Lustre® Parallel Filesystem

While there's a number of parallel filesystems used in HPC solutions today, Lustre currently dominates with 5 of the top10 systems and more that 70% of the Top100 [8]. This is also why Xyratex is basing all solutions on this filesystem and hence this paper will focus on Lustre performance benchmarking.

Lustre is based on the concept of separating meta data from block I/O using a pair of metadata servers (MDS) and a number of object store servers (OSSes) all interconnected with a network fabric (fig. 1).

III. Storage System Architectures

While the choice of filesystem probably has the greatest impact on I/O performance, it is important to remember that the specific architecture of the underlying storage solution is also very important to understand to achieve good throughput. Factors that affect the storage performance ranges from the choice of RAID system (hardware based or software such as MD-RAID), disk protocols such as S-ATA

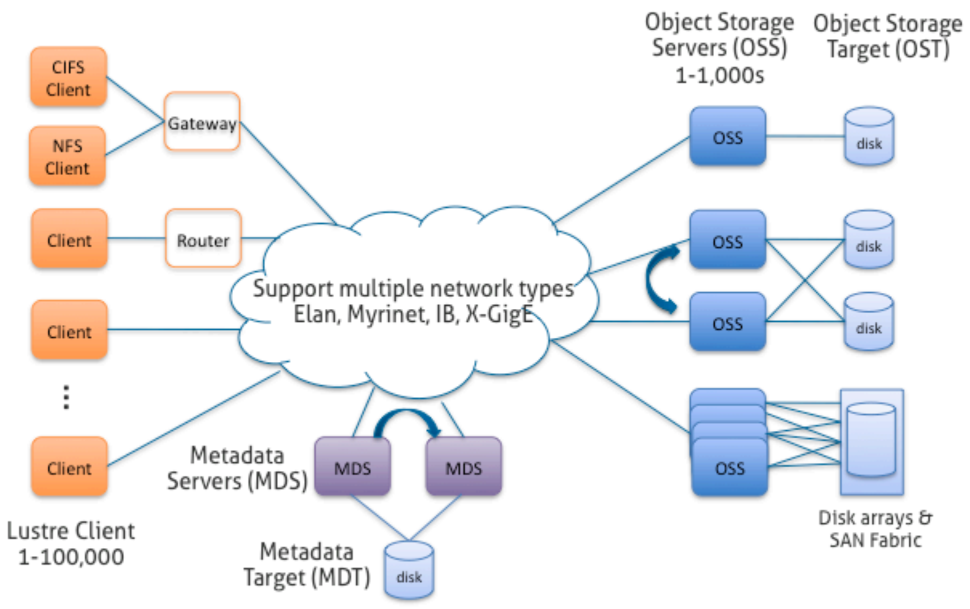

Fig. 1 Schematic overview of active components making up the Lustre filesystem.

and SAS, disk type (even within a range of say 2TB NL-SAS drives from one of the 3 major vendors, has very different I/O characteristics), firmware levels of all components, the components of the software stack, tuning parameters (important even on the client side) and type of interconnect to mention a few.

It is therefore important to benchmark the underlying hardware before starting client based tests to set a baseline of the disk backend subsystem. The most commonly used tool for this (at the filesystem level) is obdfilter-survey[9].

IV. System Architecture

A. *Storage Subsystem*

The ClusterStor 6000 Lustre based engineered solution consist of a pair of active-passive failover servers with shared storage acting as the systems metadata server (MDS) and metadata management server (MGS). Each object store servers consists of a 5 RU enclosure, known as a Scalable Storage Unit (SSU) hosting 84 3,5' disk drive slots and two embedded OSS server modules in the back of the enclosure[10]. The following storage components were used:

- 82 Hitachi Mars-K near line SAS drives (3,5' form factor) with a total capacity of 2 TB each.
- 2 Hitachi Ralston Peak SSDs (2.5' form factor in a 3.5' carrier) with a total capacity of 100 GB each.

The disk drives are configured as 8 RAID 6 (8+2) volumes formatted with ldiskfs rendering them into 8 object store targets (OST) with the remaining 2 drives acting as roaming spares. The 2 SSDs (formatted with RAID 1) was exclusively used for write intend bitmaps WIBs) and for Linux journals.

The embedded server modules (2 per enclosure) are based on a single socket Intel Xeon CPU, 32 GB RAM and an onboard Mellanox CX3 FDR capable HCA. The ClusterStor architecture uses Lustre version 2.1.3 + Xyratex extensions. The ClusterStor solution benchmarked consists of a total of 2 SSUs, 4 OSS servers, and 16 OSTs.

The storage subsystem is configured with the following Lustre parameters: writethrough cache enabled, read cache enabled and the read cache max filesize = 1M. The MDT, MGS and OSTs leverage software RAID and no data caching. All data is stored directly to disk and committed before sending an ACK back to the client.

B. *Clients and interconnect*

The test clients were made up of 64 x86 compute nodes with the following configuration:

- 2x Intel Xeon Westmere Processors
- 24 GB RAM
- QDR interface

All clients as well as the ClusterStor SSUs were connected to a Mellanox based FDR capable core switch fabric.

The clients connected to the storage were configured with Lustre client version 1.8.8. Each client is also tuned according to industry best practice with LRU and check-sums disabled, and the max RPCs in flight set to 32.

V. Test Methodology

To define the optimal benchmark parameters, a number of settings were tested. This paper will focus on the following subset of all user modifiable test parameters:

```
# pdsh -g oss "TERM=linux thrlo=256 thrhi=256 nobjlo=1 nobjhi=1 rsz=1024K size=32768 obdfilter-survey"
cstor01n04: ost  4 sz 134217728K rsz 1024K obj    4 thr 1024 write 3032.89 [ 713.86, 926.89] rewrite
3064.15 [ 722.83, 848.93] read 3944.49 [ 912.83,1112.82]
cstor01n05: ost  4 sz 134217728K rsz 1024K obj    4 thr 1024 write 3022.43 [ 697.83, 819.86] rewrite
3019.55 [ 705.15, 827.87] read 3959.50 [ 945.20,1125.76]
```

Fig. 2 Results of an optimized obdfilter-survey run on a single SSU. The individual OSS results are highlighted in bold.

A. Obdfilter-survey

A part of the Lustre I/O kit, the obdfilter-survey script generates sequential I/O from varying numbers of threads and objects (files) to simulate the I/O patterns of a Lustre client. While it can run on a network attached client, it is commonly used as a server side tool.

B. I/O mode

IOR supports a number of I/O modes such as Buffered I/O and Direct I/O. In addition, the benchmark suite also supports simulation of several application characteristics such as file per process (FFP) and single shared file (SSF). In this paper we focused used both buffered and direct I/O using the file per process methodology. For the file system, we used the default Lustre stripe size of 1M and stripe count of 1.

C. I/O slots per client

IOR supports a number of client side settings defining how the application is using the hardware and I/O subsystem of the client node. When launching a benchmark run, the node from where the job is issued, refers to a "machinefile" listing the available client IP addresses or hostname and their individual settings. Each line in the machinefile were defined as:

'hostname' slots=4 max_slots=12

For the 64 clients, we define each host with a minimum of 4 slots and a maximum number of slots equal to the maximum number of CPU cores, which in this benchmark setup were 12 cores. Slots can be referenced as number of tasks per client running IOR to measure performance.

The execution of IOR is done using MPI using the --byslot option. One can use the --bynode, but using the --byslot will use 4 slots in the first node than 4 slots in the second node, so on and so forth. If using --bynode, the slots will be filled using 1 slot per node than round-robin back to the first node to fill in the second slots. The benchmarks revealed using --byslot provided evenly distributed benchmarks for the number of slots executing IOR running on all nodes that yielded better per client results than using --bynode.

Under IOR, we defined the block-size per node to be 2X the memory size to ensure no caching effect is being done on the clients to truly measure ClusterStor 6000 performance.

D. IOR transfer size

To assess the importance of IOR transfer size, we ran a number of benchmark runs using different transfer sizes. There have been numerous tests done on other systems where transfer size have been used as a variable but most ranges from sub 1 MB to a max of 4 MB. As our previous performance experiences with the ClusterStor solution have indicated that the system architecture requires significant load to perform well, we decided to vary transfer size (-t) the between 1 and 64 megabytes.

E. Number of client threads

As IOR uses MPI for its execution on multiple nodes, it is possible to control the number of processes each node will use. This is set with the mpirun argument of –np. To assess how much load each client needs to produce to achieve maximum throughput on the storage solution, a number of runs were done varying the total number of processes between 4 and 1024.

VI. RESULTS

As this paper is limited in size, we cannot report all iterations performed to find the optimal performance of the ClusterStor solution. We have therefore chosen to discuss the parameters that produce the best results.

A. Obdfilter-survey

As obdfilter-survey is run on an OSS basis (each Xyratex ClusterStor SSU consists of 2 OSSes), the results (fig. 2) are showing the performance of two OSSes.

The results reveal that a single SSU have a write performance of 6,055 MB/s (75,9 MB/s per disk) and a read performance of 7,904 MB/s (98.8 MB/s per disk).

B. IOR – Buffered I/O mode

For maximum write performance, we found a few interesting results with IOR with the buffered IO option and file-per-process. Different IOR transfer sizes provided different but favorable results for Write performance. As can be seen from the results in figure 3, Write performance using buffered I/O is very good and peaks just above 12 GB/s over 2 SSUs. To assess the maximum performance capabilities, we used the results above to define a few specific benchmark runs with optimized parameters for write performance.

Write optimized benchmark 1:

- Number of Clients: 32
- Number of Slots: 128
- Block size (-b): 12g
- Transfer size (-t): 1m
- Buffered IO
- --byslot option for mpirun

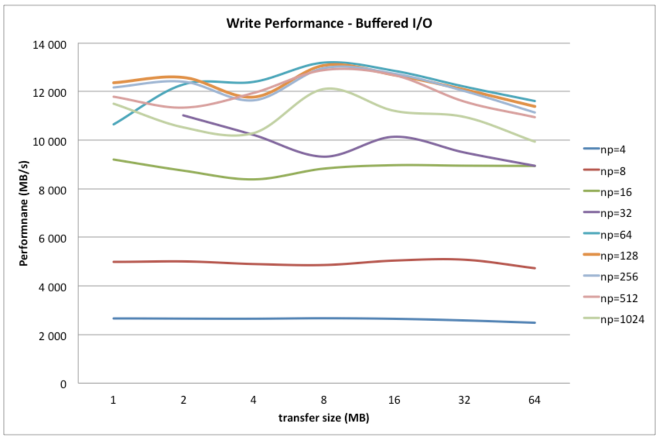

Fig. 3 Write performance using IOR buffered I/O mode.

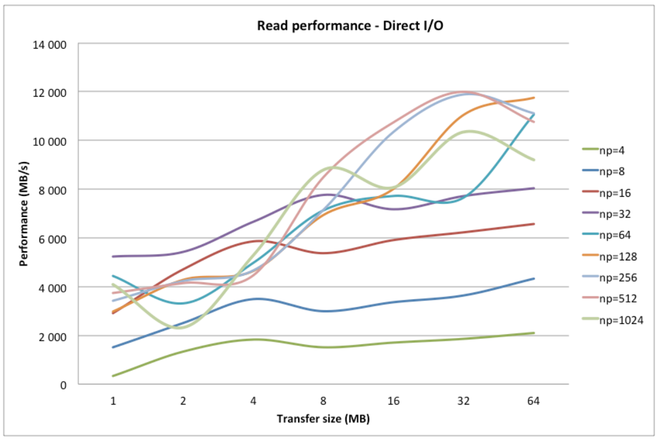

Fig. 4 Read performance using IOR Direct I/O mode.

Benchmark command line:

```
mpirun -machinefile machinefile.txt -np 128 --
byslot ./IOR/src/C/IOR -v -F -t 1m -b 12g -o
/xyratex/test.0
```

```
Write Results:  12,369.32 MB/s
```

Write optimized benchmark 2: In the second benchmark run, we changed the transfer size to 2 megabytes using the following command line:

```
mpirun -machinefile machinefile.txt -np 128 --
byslot ./IOR/src/C/IOR -v -F -t 2m -b 12g -o
/xyratex/test.0
```

```
Write Results:  12,579.55 MB/s
```

Write optimized benchmark 3: In this run, we changed the number of clients, the number of slots and increased the transfer size to 8 megabytes:

- Number of Clients: 16
- Number of Slots: 64
- Block size (-b): 12g
- Transfer size (-t): 8m
- Buffered IO
- --byslot option for mpirun

Benchmark command line:

```
mpirun -machinefile machinefile.txt -np 64 --
byslot ./IOR/src/C/IOR -v -F -t 8m -b 24g -o
/xyratex/test.0
```

```
Write Results:  13,192.72 MB/s
```

C. IOR – Direct I/O mode

For maximum read performance, we found a few interesting results with IOR with the direct IO option and file-per-process As the results indicate that a larger IOR transfer size provide the most favorable results for Read performance, we used the runs to define the best set of parameters to achieve peak read performance. Below are the parameters and the Read results for each of the IOR transfer size the yielded best results.

Read optimized benchmark 1:

- Number of Clients: 64
- Number of Slots: 256
- Block size (-b): 12g
- Transfer size (-t): 32m
- Direct IO (-B)
- --byslot option for mpirun

Benchmark command line:

```
mpirun -machinefile machinefile.txt -np 256 --
byslot ./IOR/src/C/IOR -v -F -B -t 32m -b 12g -o
/xyratex/test.0
```

```
Read Results:  11,889.93 MB/s
```

Read optimized benchmark 2:

- Number of Clients: 64
- Number of Slots: 512
- Block size (-b): 6g
- Transfer size (-t): 32m
- Direct IO (-B)
- --byslot option for mpirun

Benchmark command line:

```
mpirun -machinefile machinefile.txt -np 512 --
byslot ./IOR/src/C/IOR -v -F -B -t 32m -b 6g -o
/xyratex/test.0
```

```
Read Results:  11,996.17 MB/s
```

Read optimized benchmark 3:

- Number of Clients: 32
- Number of Slots: 128
- Block size (-b): 12g
- Transfer size (-t): 64m
- Direct IO (-B)
- --byslot option for mpirun

Benchmark command line:

```
mpirun -machinefile machinefile.txt -np 128 --
byslot ./IOR/src/C/IOR -v -F -B -t 64m -b 12g -o
/xyratex/test.0
```

```
Read Results:  11,754.37 MB/s
```

For Single Shared File benchmarks with IOR, there is still a lot more investigation that still needs to be done to maximize read and write results, reduce the lock contention and using standard IOR parameters with large block sizes to equal 2x the client memory and using smaller block size and the segment option, which reduces locks at much small blocks but the balance between clients and OSTs still needs further investigation. Based on various results for single shared file, we found the number of clients should be a multiple of the number of OSTs to maximize performance, regardless if the number of clients is more or less than the number of OSTs, To optimize performance is done by balancing the block transfer size, using the "segment" flag in IOR as a multiple of the block size, matching stripe sizes to the block size, or having the block size be a multiple of the Lustre transfer size all have factors in maximizing performance and a separate paper is needed to address these type of parameters and results for single shared file.

D. *IOR transfer size*

This parameter showed some significant differences between the 2 I/O modes. When using buffered I/O mode, the smaller transfer size change had some impact on the performance to maximize writes, and with direct IO, there was in increase in read performance when increasing the transfer size, which are illustrated in fig. 3 and fig 4.

E. *Number of client threads*

Increasing the number of client processes clearly had an impact on performance regardless of the mode used. Peak performance however, is reached with different parameters for each type of I/O and I/O mode. Buffered writes peak using 64 processes whereas read peaks using 16 (fig 3). For direct I/O, both read and write seems to peak using 512 processes (fig 4).

VII. Discussion

The results in this paper clearly show that the Xyratex ClusterStor 6000 solution is capable of delivering very high throughput with regards to both read and write performance. In fact the throughput calculated on a per disk drive is very close to maximum single formatted disk performance. The performance of a single HA-pair of object store servers (each using 4 OSTs) peaks in excess of 6 GB/s translating to about 750 MB/s per OST which compared to other published results [11, 12] is significantly higher. While there's several explanations for the differences (different disk drives, older CPU architecture, QDR vs. FDR and PCI-e gen 2 vs. gen 3) the performance gap is remarkable and a result of the system architecture of the ClusterStor system. The use of a balanced solution pushing the bottleneck to the disk backend rather than limiting performance closer to the clients means using fewer disks and maximizing the disk subsystem. Current high end systems suffers from having to use 3-5x the number of disk drives needed from a capacity point of view, to achieve the required performance. With the ClusterStor 6000 solution, this overhead is significantly reduced due to the integration and balanced design of the architecture. This has a number of very important benefits associated with it such as reduced floor space, less power and cooling, and increased reliability. It also has positive effects on supportability, as the number of drives in the solution is fewer, the need to replace and suffer RAID set rebuilds is lessened (assuming similar disk failure rates regardless of vendor and solution).

With the ClusterStor 6000, we found using a combination of buffered and direct I/O option for IOR were needed to maximize results for Writes and Reads. We found IOR writes and reads data differently and reading data is not as sequential as writes. While there might be several reasons for this behavior, Xyratex believes that this is due to buffered IO using pagecache and has all the tools to align pages before sending them to LNET whereas direct I/O does not do that. Buffered I/O directly creates an RPC and run it in synchronized mode. This means that the software is not able to send new chunks of direct I/O data until the previous RPC is processed on client side and a reply is returned.

Based on the initial assessment of the backend disk performance as illustrated by the obdfilter-survey benchmark, one could expect that the read performance would be higher. However, by the same token, the IOR based write performance is actually greater than the obdfilter-survey write results. In essence, while the obdfilter-survey benchmark delivers a good approximation of the disk system backend performance, it does not utilize the entire software stack or the network component. The tool obdfilter-survey is used to measure performance of the OSTs without client and LNET, which has it's own benefits isolating outside factors to provide a base-line performance test.

The analysis of the number of client processes needed to deliver peak performance clearly indicates that storage solutions based on modern architecture (including FDR IB, PCI-E gen 3, CPU architectures with 8 or more cores and hyper threading capabilities) requires a significant load to perform optimally. The current rule of thumb requires a minimum of 8 clients to saturate a single SSU.

Something that sets IOR apart from other benchmark suites and the reason it's been used to benchmark many of the largest parallel filesystems in the world is that it mimic many commonly used applications in that it uses MPI I/O for cluster execution and is bound by most of the limitations normally seem with HPC codes running in large clusters. Within the open source community, efforts are currently underway to define a new and more modern benchmark suite better suited for the storage solution being planned for the near future. For now, we are required to use what is available and while a tool like IOR does require a lot of experimentations and time consuming benchmark runs, with a bit of experience and good understanding of the parameters and what they do to achieve relevant and repeatable performance numbers. This paper really focused on two major modes of I/O (buffered and direct) as well as modulation of two basic parameters (number of client processes and IOR transfer size) with clear and significant impact to the measured results. While not part of the current study, it is clear to the authors that different solutions does require drastically different setup to produce optimal results and that "one size fits all" does not always work. It is therefore very important when writing a RFP, that the benchmark is not being artificially restricted by requiring that an IOR has to be run with pre-defined settings and arguments. Most likely, these settings are derived from systems previously acquired by the customer and therefore based on older technology and software stacks. Forcing a vendor to use suboptimal settings will only result in over provisioning of a solution or choice of an inferior solution. In these cases, a real application, using customer supplied input data is a much better approach, but if that's not doable, traditional benchmarking will have to suffice. As most application suites can be tuned with regards to their data in- and output parameters, properly performed benchmarking can provide good insight into the right I/O tuning.

ACKNOWLEDGMENT

The following people are acknowledged for their contribution to this paper: Alan Poston, Eugene Birkine, Karl Merritts, Colin Faber, and Nathan Rutman.

REFERENCES

[1] "Linpak" http://www.top500.org/project/linpack/.
[2] "GPFS" http://www-03.ibm.com/systems/software/gpfs/.
[3] "Lustre" http://lustre.org.
[4] "IOR" http://sourceforge.net/projects/ior-sio/.
[5] "IOzone" http://www.iozone.org.
[6] "Bonnie++" http://sourceforge.net/projects/bonnie/.
[7] "MDtest" http://sourceforge.net/projects/mdtest/.
[8] "Top500" http://www.top500.org.
[9] "obdfilter-survey" http://wiki.lustre.org/manual/LustreManual20_HTML/BenchmarkingTests.html.
[10] K. Claffey, A. Poston, and T. Kling Petersen, "Xyratex ClusterStor - World Record Performance at Massive Scale" http://www.xyratex.com/sites/default/files/files/field_inline_files/Xyratex_white_paper_ClusterStor_The_Future_of_HPC_Storage_1-0_0.pdf, 2012.
[11] "HP - DDN quickspecs," http://h18000.www1.hp.com/products/quickspecs/13648_div/13648_div.PDF.
[12] W. Turek, and P. Calleja, "High Performance, Open Source, Dell Lustre Storage SystemDell PowerVault MD3200 storage platform and QDR Mellanox Infiniband," http://i.dell.com/sites/content/business/solutions/hpcc/en/Documents/Lustre-HPC-Whitepaper-10082011.pdf, 2011.