Concurrent write MPI/IO benchmarks on File Systems

Prakashan Korambath[*]
Institute for Digital Research and Education (IDRE)
University of California, Los Angeles, CA 90095
ppk@ats.ucla.edu
July 9, 2012

ABSTRACT

In this report we tabulated and plotted concurrent I/O benchmarks on three different file systems namely a NAS, Panasas and Lustre file system. We ran the same executable on two file systems on one cluster because they were accessible on that cluster. The same code was recompiled and similar benchmark was ran on a remote cluster. We discuss the write I/O performance on the three different file systems attached to two clusters along with network topology. In the benchmark each MPI processes is writing their own files. So, the number of files are equal to the number of MPI processes.

Categories and Subject Descriptors

J.0 [**Computer Applications**]: General

General Terms

Performance, Storage, Concurrent write I/O

Keywords

Concurrent I/O, Lustre, pNFS, Parallel I/O

1. INTRODUCTION

The number of nodes and cores on Beowulf clusters [1] are increasing as the price per core is coming down. That also means more applications are now running MPI [2] or threaded jobs that can take advantage of the availability of increased cores. We are also seeing increasing number of commercial and open source applications that support parallel distributed memory MPI jobs or shared memory OpenMP [3] type jobs. As the machine size becomes larger and number of users per cluster increases the data involved in computation is also increasing proportionately and is likely to be in terabytes range for I/O intensive jobs. Only the applications that can fit all data in memory and do very little I/O will be immune to this trend. The disk storage has the lowest data bandwidth in the memory hierarchy starting from CPU registers, L1 cache, L2 cache, RAM to disk storage. The I/O intensive applications need to read data from files on the disk at the beginning of the calculation, distribute them across all processors if it was read by only the master processes and write the data from computation on the disk either by all processes or some of the processes involved. In the MPI world the term processes is more appropriate to use than processors because MPI can spawn multiple threads on the same core although it is not advised due to performance degradation. The I/O could become the bottleneck for those applications that does lot of data reads and writes in a large cluster environment with hundreds of nodes when many jobs are competing for the limited I/O bandwidth. The applications that does frequent checkpoint for restart purpose can also find their performance deteriorated by the limitations on the I/O bandwidth.

The hardware resource to write and read data on disks to and from the memory can come from widely varying architectures. If the data is written on local disks attached to the node, the I/O speed is dependent up on the type of hard disk, which can be traditional spinning magnetic hard drives with varying spinning speeds or NAND based flash-memory solid state drives (SSDs). SSDs have higher data throughput and lower latency. A simple example of this kind of scenario is when computations are done on a standalone desktop. A typical cluster server node on the other hand will have at least a Network File System (NFS) [4] mounted home directory as well as a local disk. Since modern server nodes have 8 to 16 cores per node, this kind of node attached resources will at least be shared by the all the threads of jobs running on that node. Typical measured bandwidth for HDDs are in 100 to 200 MB/s region and in SSDs bandwidth range from 100 to 500 MB/s. On a Linux OS machine commands like "hdparm -tT /dev/sda" or "dd if=/dev/zero of=/tmp/output.dat bs=8k count=256k" may be used to measure the read-write bandwidth.

1.1 File Systems on Beowulf Clusters

Typically most clusters have a home directory connected to a networked file system mounted through NFS protocol on all the nodes. A cross mounted file system is a requirement for running distributed memory parallel job or at least all threads need to have identical path to find the executable. The NFS home directories can be individual hard drives mounted per user or for a group of users in a simplistic sce-

[*]Corresponding author's address: Prakashan Korambath, 5308 Math Sciences, University of California, Los Angeles, CA 90095.

nario, but mostly the NFS home directories are served from storage servers with disks in RAID [5] configuration. RAID is a storage technology to combine multiple disks into a single logical unit. The performance of RAID storage server is dependent on the controller cards, type of hard drives and the RAID level. A simplistic view consists of bunch of disks, controllers and interconnect network to move the data. If all nodes are connected to the home directory of the storage server through a Gigabit switch, which may be the typical scenario in small clusters, the aggregate bandwidth is going to be limited by the Gigabit network speed, which in theory is around 125 MB/s. Some of the clusters have at least 10 Gigabit connections to the I/O node, which will increase the aggregate bandwidth to 1250 MBs/s in theory. Actual available bandwidth will additionally be limited by the buffer size, memory cache and number of NFS threads etc. At the hardware level data is accessed with certain granularity in physical or logical blocks depending on the storage disk array configuration. In addition the write speed and read speed on a file system are going to be different depending on the configuration of the file system. Although NFS is the de facto standard for distributed file sharing in Beowulf clusters, there are few other file systems as well. NFS protocol was originally developed by SUN Microsystems using Sun Remote Procedure Calls (RPC) so that many clients can communicate to a single server node. Global File Systems (GFS) from RedHat is another example of shared disk file system [6].

1.2 Parallel File Systems

For Linux OS based clusters one of the earliest approaches to mitigate data bandwidth problem was addressed by parallel virtual file system (PVFS) [8] using MPI ROMIO [13] . Other examples of parallel file system include General Parallel File System (GPFS) [7], Lustre [10], Parallel NFS, Panasas [14], Fraunhofer Parallel File System (FhGFS) [15], etc. Some of them are proprietary like GPFS and some are open source software like PVFS. GPFS originally supported AIX only, now it supports Linux as well. In a typical local file system running on a Unix like OS, read or write calls are blocking in nature meaning while a write request is being performed access to that file is locked to protect the integrity of the file and also make sure that the file is not overwritten while it is being read by an active processes. The Portable Operating System Interface (POSIX) I/O interface that is commonly used by application developers for OS compatibility was designed for the local file system with open, close, stat, read and write interfaces is sequential in nature. In order to overcome such limitations during collective I/O calls, the underlying storage file systems have a greater role to play to meet challenges posed by large data-intensive applications. NFS version 3 tries to relax strict POSIX rules by not defining a policy for caching on the client or server, but in practice does not improve the performance much for parallel I/O. Additionally, atomic mode operation is not supported by NFS because it does not guarantee file system level locking.

The simplest scenario of parallel data access in a MPI based application is when each process is independently reading and writing their own files that are not shared with any other processes any time during the computation meaning if they are writing out data they are all writing out to separate files. The parallel access gets more involved when many processes are accessing the same file at certain offsets in a strided pattern or randomly. Concurrent file access and non-contiguous file access are two major challenges for a parallel file I/O operation.

In order for parallel I/O to be efficient their needs to have as many hardware I/O resources as possible, there should be multiple if not dedicated I/O connections between the compute resources and the I/O resources and a high performance network bandwidth to support concurrent data transfer activities. In order for application developers in scientific domain to take advantage of parallel efficiency, they need to have good understanding of high level I/O library API such as MPI-IO API or similar APIs, Parallel File System, storage hardware and network hardware. The parallel file system in a nut shell is a software installed on the hardware to manage the data on the hardware, present the data as folders (directories) and files in hierarchical order, and coordinate the access of these files in a manner that doesn't compromise the integrity of the files. The access to the parallel file system is done usually through MPI-IO interface which is part of MPI-2 implementation. MPI-IO is perhaps the de facto standard for all parallel I/O interfaces that is used in most of the parallel computing that we are aware of. In addition there are higher-level API tools such as HDF5[11], and parallel NetCDF[12], which are available to use in combination with MPI-IO library in some scientific domain. It is best to use MPI-IO library for parallel I/O because the implementers of this API has already put lot of effort in optimizing the collective I/O performance and is expected to be compatible with many parallel file systems [16],[17].

In spite of availability all the API tools, the performance of an application is still dependent on how the application is using the I/O, it could be a simple beginning to end access of a file, or a strided access with certain offset or a random access. There is no one-size fit all strategy to benefit all kinds of data access patterns. Because of these limitations the performance of parallel I/O bandwidth is not guaranteed to be good for all applications.

The developers of parallel file system have used various approaches to get efficiency and file consistency. Virtual block device is one of those approaches where a mapping of logical blocks to physical storage is used to abstract away the physical data location. This virtual approach makes it easier for data migration and duplication meaning data can be migrated off from one device to another when installing new devices or replacing devices.

1.2.1 GPFS

IBM's GPFS file system uses virtual shared disks (VSD), it only assumes block I/O interface and no intelligence at the disk level. GPFS does not use volume management; instead the files are striped across all the disks to give the file system direct control over striping of data across the devices. The individual disks are attached to several I/O server nodes. GPFS uses distributed locking to synchronize access to shared disks to protect the integrity of user data during parallel read-write disk access. When two processes are accessing the same file, read/write atomicity will make sure either all or none of the concurrent writes operations are visible to each processes such that writes to non-overlapping data blocks of the same file proceeds concurrently. Recent GPFS releases also supports data shipping where individual servers can take responsibility for updating changes in data blocks. Clients simply forward data belonging to a particular block to the appropriate node.

1.2.2 PVFS

Some of the parallel file systems distributed metadata on all I/O servers while some store it in a single location. Distributing metadata is a complicated process, while single metadata server is a potential performance bottleneck. The open source parallel virtual file system (PVFS) developed mainly for Linux clusters at Clemson University has two kinds of servers, namely a single metadata server that maintains metadata of all files and many I/O servers to handle storage of the data. File data is distributed in a round robin fashion across I/O servers using a predefined algorithm, which computes the file size as and when required. A RAID configuration may be used to handle disk failures on the I/O servers, however a node failure is not tolerated and the system will become unavailable if a node fails. Also computing the file size each time to list the files in a directory may be a slow process as well.

1.2.3 Lustre

Another open source parallel file system is Lustre even though it was acquired by SUN and then Oracle and supported commercially. The Lustre architecture calls for (a) a metadata server (MDS) that stores file names, directories, access permission and layout; however, Lustre metadata server is not actively involved in I/O operations, which will ease the load on metadata server; (b) many object storage servers (OSS) that store data on multiple object storage targets (OST); (c) clients to access user data using standard POSIX semantics to allow concurrent and coherent read and write access to the files in the file system. The total capacity of a Lustre file system is the aggregate capacity of OSTs. OSTs can be organized with logical volume management (LVM) or RAID or traditional SAN. OSTs perform their own block operation. When data is stored on traditional SAN resources, OSTs only handle authentication and block allocation. Lustre uses distributed lock manager to protect the integrity of file data and metadata. Applications that do random I/O access are not suitable for Lustre.

1.2.4 Panasas

The Panasas File system or PanFS is a parallel storage similar to parallel NFS (pNFS) [18] protocol to overcome the performance bottleneck of NFS file system by allowing many clients to read and write data in parallel to and from physical storage systems. PanFS is a proprietary file system and it may support pNFS clients in future. The NFS server controls the metadata and coordinate access to the data. The clients query the metadata server to get the location and authentication details of the data and then directly communicates with the storage device. pNFS is part of NFS version 4 protocol. pNFS clients are available in Fedora 16 as well as RHEL 6.2 distributions. Other prominent vendors who contribute to pNFS include IBM, EMC, RedHat, NetApp etc. The PanFS operating system offers three protocols to access files namely DirectFlow, NFSv3 and CIFS. Typical Panasas storage hardware consists of storage blades and director blades. Storage blade contains all applications and user data. Director blade orchestrates all file system activities and virtualize the data across all storage blades.

2. BENCHMARKS

For measuring concurrent parallel I/O benchmark, we used a small program available at an Indiana University site [19]. In order to use this program the cluster should already have an installation of an MPI API either from OpenMPI[20] or MVAPICH2[21]. During run time this executable expects two arguments, which are data file name and block size. The size of the data file written can be increased or decreased by altering the block size. Each processor appends its processor id to the file name. So, if the job is run on 32 processors, there will be 32 files of same size written on the directory where job was run. Checking the name and size of the file is a good indication of the completion of the job.

The program calls `MPI_FILE_OPEN`(comm, filename, amode, info, fh) to open files for writing. The inputs for this API calls are communication world (handle), filename (string), file access mode (integer), and info. The output is a new file handle. This program calls `MPI_COMM_SELF` communicator, which is associated with just the process that called it. So the file name is not shared with any other processes. Then `MPI_File_Write` API is called to write the data to the disk. The `MPI_FILE_Write` calls are enclosed within `MPI_Wtime` call to time the wall clock time to finish the data write call. The program uses `MPI_Reduce` call to find out the longest time required to write the data. The transfer rate is computed by dividing the longest time with total number of bytes written. Additionally, the program does lot of error checking to make sure files are open before proceeding to write step.

We found this program well written for our benchmark purpose to measure the maximum concurrent data write bandwidth on a file system. All MPI processes are independently writing their own files. None of the files are shared among processes. We are aware of few other publicly available benchmark routines as well. We hope to use them in future studies.

2.1 Hoffman2 Cluster

Hoffman2 cluster [22] hosted by IDRE [23] consists of approximately 800 nodes currently. All nodes have either AMD or Intel multi core CPUs with varying amount of RAM ranging from 8 GB per node to 128 GB per node. The communication network fabric consists of both Gigabit and DDR IB network ports and switches. This cluster was initially started with 64 seed nodes from IDRE's predecessor organization, Academic Technology Services (ATS) in 2007. Since then many research groups have contributed nodes to this cluster in a shared cluster concept. The owners of the nodes are given full access to logical equivalent of contributed cores for running their jobs contiguously for many days with the understanding that any unused resources are shared among all users in a 24 hour queue. Due to this kind of arrangement cluster utilization for Hoffman2 is consistently above 85%. Hoffman2 cluster uses Grid Engine scheduler to manage and schedule cluster computing resources. Currently, there are around 1000 users from math and physical sci-

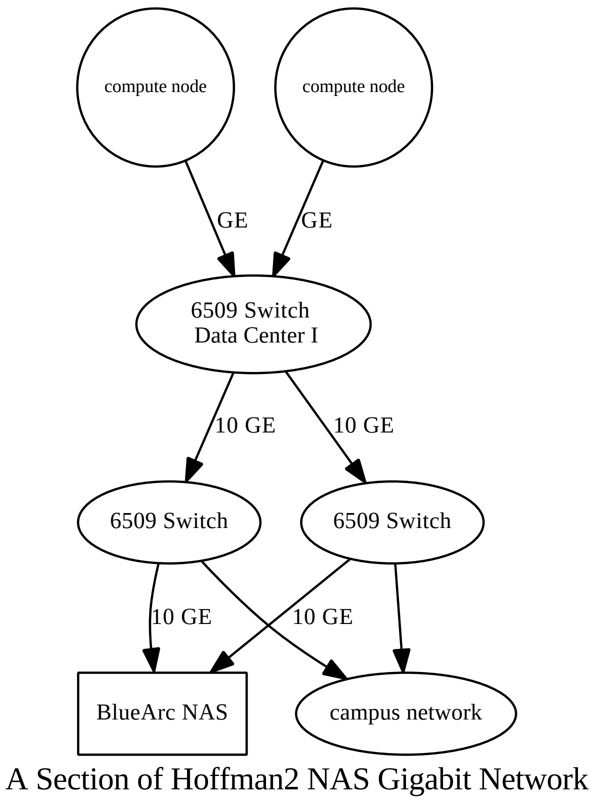

Figure 1: A section of Hoffman2 cluster network showing the data path from compute nodes to NAS storage

ences, engineering, business and social sciences disciplines running various kinds of jobs on the Hoffman2 cluster. The I/O bandwidth on the cluster has risen considerably due to increase in number of nodes and jobs. Initially, the cluster was supported only by a Blue Arc NAS file system, but due to the heavy I/O demand another file system, namely Panasas file system is also added to this cluster. The nodes on this cluster are spread among three data centers in UCLA campus. A single NAS and single Panasas file systems provide home directory and scratch directory for all the nodes through a network of Gigabit and IP over IB connections. All MPI message passing calls in this cluster are handled exclusively by the DDR IB network fabric.

A schematic diagram of network connections from compute nodes in one of the data centers to NAS Blue Arc storage hardware is shown in Figure 1. As shown in Figure 1 the file I/O on this cluster makes three hops before it reaches NAS. The first hop between the compute nodes and a Cisco 6509 switch is through a GE connection, next hop is to a second Cisco 6509 switch through 10 GE connection followed by another 10 GE connection to the NAS. There are two 10 Gigabit up-links from two separate 6509 Cisco switches to the Blue Arc NAS. This is expected to give an aggregated 20 Gigabit bandwidth as well as redundancy in the event one of the 6509 switches fails. The second pair of 6509 switches are connected to other data centers as well as to the campus backbone to have access to login nodes. The nodes and switches in the other two data centers are not shown in Figure 1.

The Figure 2 shows the file I/O path from compute nodes to the Panasas file system. This route uses IP over IB connections through many small and large IB switches. The data from a compute node will have to make at least 6 hops before it reaches the Panasas file system. The IB network traffic uses combination of 24 port Flextronics switch, 288 port SilverStorm 9240 switch and a 36 port Q-Logic 12200 switch before it reaches the Panasas IB router. There are four QDR links from the Q-Logic switch to the Panasas IB router as there are four IB routers in the Panasas hardware shown in Figure 2. The Panasas IB router hardware in turn is connected to a 5548 Switch through two aggregated 10 GE links. The director and storage blades on Panasas file system is connected to 5548 switch through 10 GE links.

The network configurations on the second of the three data centers are almost identical as described above. The third data center, which is actually a performance optimized data center (POD) from HP [24], uses combination of 48 ports and 16 ports switches to connect to the 6509 switches which are in turn connected to the NAS storage file system. The maximum theoretical I/O bandwidth that can be expected on Hoffman2 cluster will be around 2500 MB/s or 20 Gigabits per second.

Since the nodes on this cluster are being added continuously during the last 5 years, not all nodes are identical in terms of architecture and performance. Hence, it is practically impossible to get identical server hardware to measure the bandwidth on Hoffman2 cluster. So the bandwidths that we reported in the tables below should be considered to be the best case scenario considering the fact that this benchmark

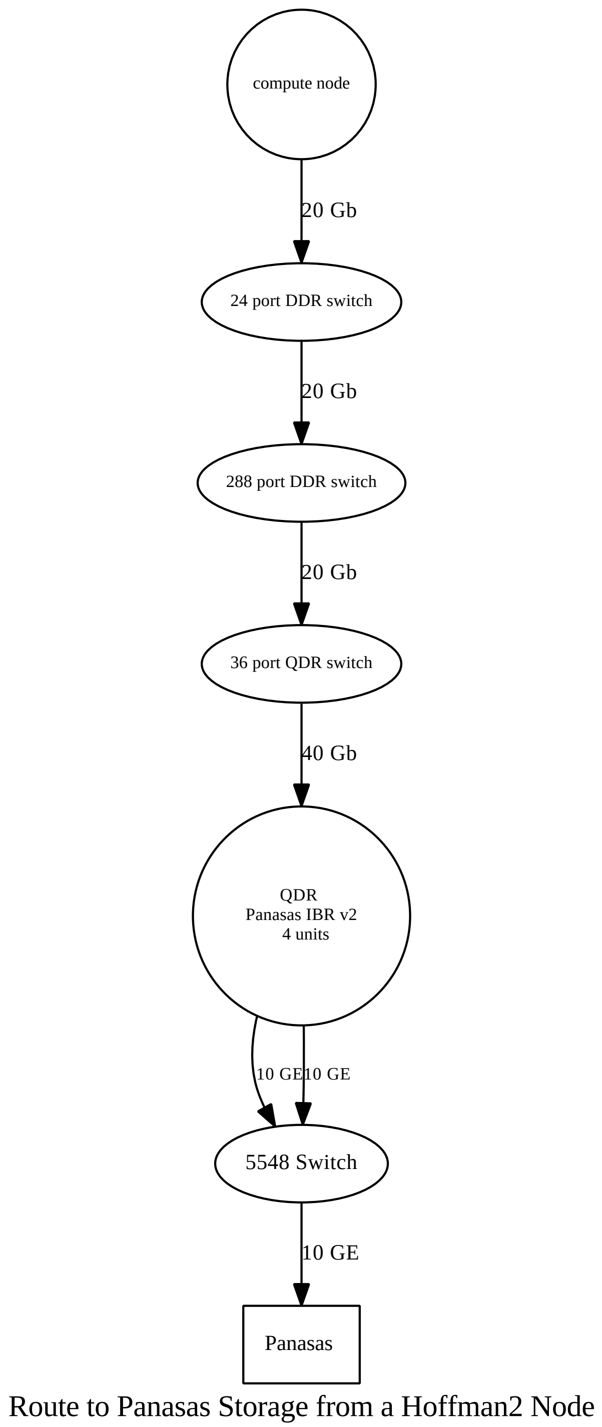

Figure 2: Network configuration for the route to Panasas storage from a node

Table 1: Concurrent write bandwidth on Panasas File System to write 32 MB size files

MPI Processes	Write Speed (MB/s)
32	500
64	581
128	671
256	981
512	960
768	519

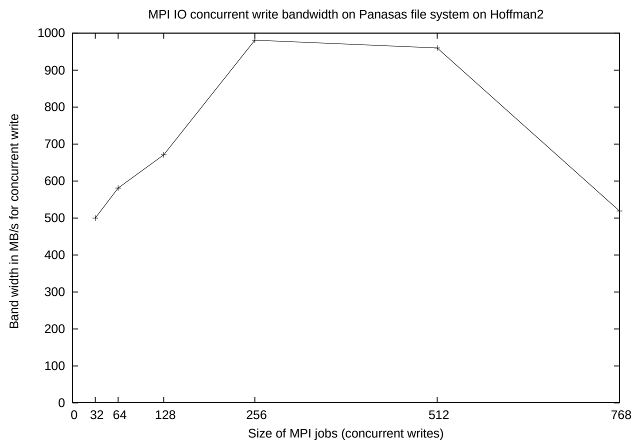

Figure 3: Concurrent file write on Panasas File system on Hoffman2 cluster for varying number of processors

was run when no other users were present on the system in a few hours window during one of the maintenance shutdown schedules. The Hoffman2 cluster currently runs CentOS 6.2 Linux distribution and users Warewulf image provisioning tools to deploy OS on compute nodes.

Table 1. shows the results of concurrent write bandwidth benchmark ran from 32 processors to 768 processors. The nodes used in this benchmark all have 8 cores per node and they have either Intel or AMD dual quad core CPUs. The peak bandwidth that was observed in this run is around 980 MB/s. Although the bandwidth results fluctuated between 50-100 MB/s in repeated runs the peak bandwidth hovered around 980 MB/s. The fluctuation is expected considering the nature of the heterogeneity of the Hoffman2 cluster server nodes. Also as shown in Figure 2 the data has to make 6 hops through different switches to reach the storage unit. That may introduce additional latencies. The Table 1 results are from the Panasas file system which is auto mounted on all compute nodes as scratch directory /u/scratch and has a total capacity of around 70 TB. The size of all files written were 32 MB each for the result shown Table 1. The results are plotted in Figure 3 as well with number of MPI processes on the X-axis and I/O bandwidth in MB/s on the Y-axis.

Table 2. shows a similar concurrent write results as in Table 1, but on a NAS file system from Blue Arc. Again 32 to 512 processors were writing their own individual files to this file system which happened to be the home directory for majority of users at the same time. The nodes used in this

Table 2: Concurrent write bandwidth in MB/s on Blue Arc NAS file system for 32 MB files

MPI Processes	Write Speed (MB/s)
32	394
64	416
128	399
256	300
512	278

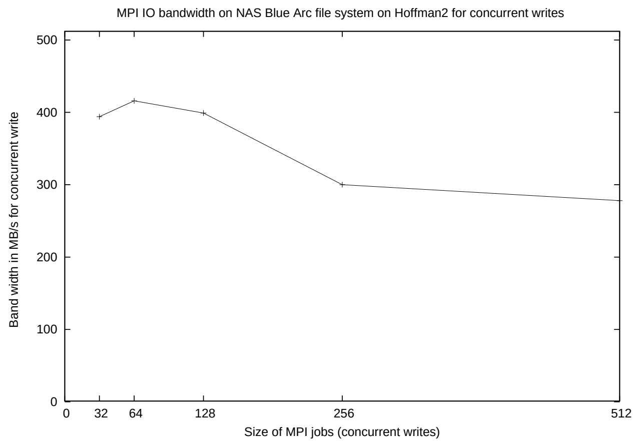

Figure 4: Concurrent file write on NAS Blue Arc File system

run also has 8 cores per node. The peak write capacity was around 416 MB/s. The results are plotted in Figure 4 as well. As can be seen in Figure 1, the number of hops in this Gigabit only network configuration is three between the compute nodes and the NAS storage.

The results shown in Table 3. is a test to see how the bandwidth changes with file size. In the previous runs we used a constant size of 32 MB, but in this experiment we varied the data size from 32 to 512 MBs. The results from this experiment which was ran on 512 processors with varied data size did not show any significant change in file transfer rate. Majority of the runs were around 900 MB/s. The results are plotted in Figure 5 as well with file size on X-axis and bandwith on Y-axis. This experiment shows the size of the file in this range is large enough to mask any latency effects.

2.2 Gordon Cluster

Gordon cluster at SDSC [25] is specifically designed for data intensive computing. The cluster consists of 1024 dual socket

Table 3: Concurrent write bandwidth in MB/s on Panasas file system as a function of file size for 512 MPI processors

File Size (MB)	Write Speed (MB/s)
32	917
64	807
128	918
256	899
512	975

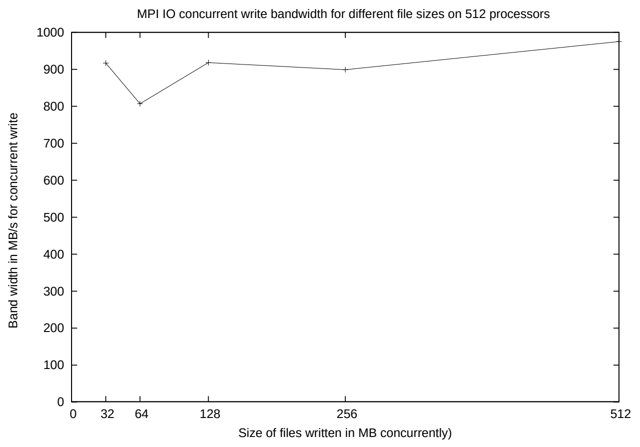

Figure 5: Concurrent file write on Panasas File system on 512 processors vs. file size

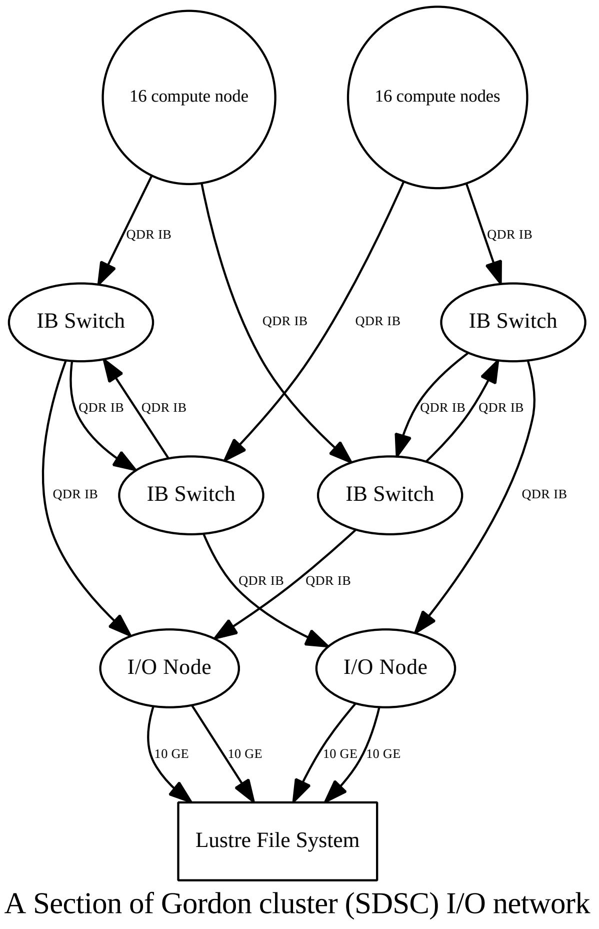

Figure 6: A section of Gordon cluster I/O network showing two 16 compute nodes block and Lustre file system

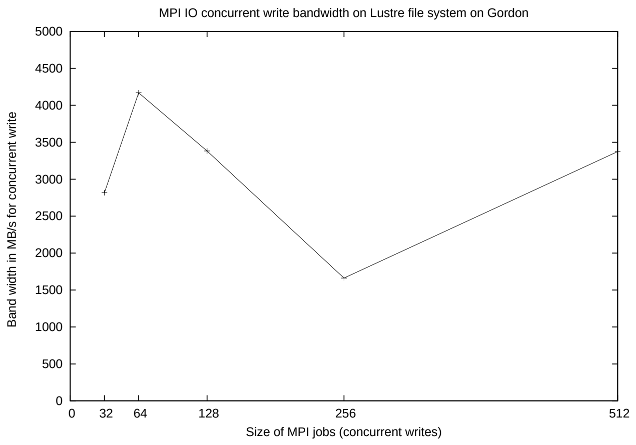

Figure 7: Concurrent file write on Lustre File system on Gordon cluster for varying number of processors

Intel Sandy Bridge nodes (Xeon E5 processors) with 64 GB DDR3 RAM per node. This chip implements Intel Advanced Vector Extensions (AVX) with the capability of 8 operations per clock cycle. Please keep in mind that previous generation chips compute 4 operations per cycle. It also has 300 TB of flash memory SSDs. This cluster employs vSMP software from ScaleMP [26] to aggregate the SMP memory on all nodes for serial and threaded applications. Additionally, a QDR IB network in 3D Torus configuration provides communication interconnect fabric. The frequency of each CPU is around 2.6 GHz. In this paper we used the Data Oasis Lustre file system with 4 PBs of capacity and sustained bandwidth of 100 GB/s. Each of the Gordon nodes have 16 cores. Each of those 16 nodes are connected to an I/O server node and there are 64 of them. For failure redundancy actually two I/O nodes are connected to two sets of 16 compute node blocks. The I/O nodes in turn are connected to the Lustre file system through dual 10 GE links. The Compute nodes are connected to I/O nodes through QDR IB links at 40 Gb/s. The operating system is CentOS Linux version 5.6 with modifications to support AVX and uses Rocks cluster management tools. The MPI implementation on this cluster uses MVAPICH2. The jobs are scheduled on this cluster using Catalina Scheduler and Torque resource manager. The maximum I/O bandwidth from one I/O node to Lustre file system is around 1.6 GB/ write speed. So a careful combination of nodes per core is required to get the maximum throughput bandwidth from this cluster. For this study we used all cores per node so that we have similar comparison to the Hoffman2 cluster bandwidth where all cores on a node is used in the MPI-IO write.

Table 4 and Figure 7 shows results from concurrent write benchmark bandwidth measurement from 32 to 512 processors. The peak value we got hovered around 4 GB/s. The repeated measurements will change the value by few hundred MB/s because our jobs are not the only jobs that are running on this system during this benchmark. So, these results are representative of a typical expected bandwidth on a production cluster. These are not the best case scenario, but more like an expected bandwidth. For highly bandwidth depended applications one can reduce the number of cores

Table 4: Concurrent write bandwidth on Lustre File System on Gordon Cluster at SDSC to write 32 MB size files

MPI Processes	Write Speed (MB/s)
32	2818
64	4169
128	3382
256	1662
512	3374

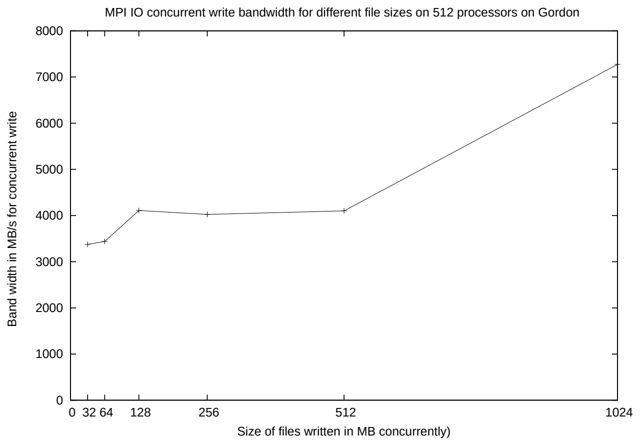

Figure 8: Concurrent file write on Luster File system on 512 processors vs. file size

per nodes inside the submission script. In this experiment we requested all cores in a node.

Table 5 and Figure 8 shows results when processor numbers are fixed at 512 and the file size is changed. The spread in value again displays the expected bandwidth depending on which set of processors are allocated to our job. Due to the careful configuration of File I/O network, Gordon cluster seems to give consistently satisfactory data bandwidth.

3. CONCLUSIONS

In conclusion a parallel file system is a must for any Beowulf cluster of any size because parallel I/O can throttle the computation speed. Jobs that are I/O dominated can be pushed back to CPU dominated if the I/O bottleneck can be well taken care of. It is also important to have as many dedicated I/O servers as possible because it will give the system administrators of the cluster a fine control to prevent a

Table 5: Concurrent write bandwidth in MB/seconds on Lustre file system as a function of file size for 512 MPI processors

File size (MB)	Write Speed (MB/s)
32	3374
64	3439
128	4109
256	4025
512	4102
1024	7273

single application from dominating total I/O bandwidth. In other words by distributing the total I/O bandwidth among many compute servers and allocating compute resources in groups of nodes, which have dedicated I/O servers, one can make sure the data bandwidth from one set of jobs is not affecting the bandwidth for jobs on another set of computational nodes and I/O servers. Also minimizing the number of hops between the compute nodes and the storage unit will improve the performance of applications.

4. FUTURE WORKS

In this report we did not perform any concurrent read benchmark or write and read from many processors to a single file. Those works will be performed and reported in the upcoming reports along with some benchmarks on solid state devices. We will also be investigating application based benchmarks like NAS parallel I/O benchmark routines.

5. ACKNOWLEDGMENTS

We would like to acknowledge UCLA's Cyberinfrastructure grant for the informatics and computational data development for funding the projects under Virtual Computing Laboratory. We also thank XSEDE organization for granting us compute time on SDSC's Gordon cluster. Discussions with other staff members of IDRE has improved the quality of contents presented in this technical report. In particular author like to acknowledge the information shared by Mahidhar Tatineni of SDSC HPC staff on the network topology of Gordon cluster. Contributions from Justin Tea, John Pedersen, and Brian Pape of IDRE in the description of network topology of Hoffman2 cluster is also acknowledged. Shao-Ching Huang of IDRE shared his expertise in plotting graphs and preparing the manuscripts with various latex packages.

6. REFERENCES

[1] William W. Hargrove, Forrest M. Hoffman and Thomas Sterling The Do-It-Yourself Supercomputer Scientific American, 265 (2): pp. 72 - 79 2011

[2] MPI Forum:. `http://www.mpi-forum.org/`

[3] OpenMP: `http://openmp.org/wp/`

[4] Russel Sandberg , David Goldberg , Steve Kleiman , Dan Walsh , Bob Lyon Design and Implementation or the Sun Network Filesystem 1985 USENIX

[5] David A. Patterson, Garth Gibson, and Randy H. Katz A Case for Redundant Arrays of Inexpensive Disks (RAID) UC Berkeley, (1988) SIGMOD, ACM.

[6] GFS: Global File System `http://en.wikipedia.org/wiki/GFS2`

[7] Frank Schmuck "GPFS: A Shared-Disk File System for Large Computing Clusters" In Proceedings of the FAST'02 Conference on File and Storage Technologies. Monterey, California, USA. USENIX 2002

[8] Philip H. Carns,Walter B. Ligon III,Robert B. Ross, and Rajeev Thakur. PVFS: A parallel file system for Linux clusters. In Proceedings of the 4th Annual Linux Showcase and Conference, pages 317-327, Atlanta, GA, October 2000. USENIX

[9] Robert Latham, Robert Ross and Rajeev Thakur Implementing MPI-IO Atomic Mode and Shared File Pointers Using MPI One-Sided Communication International Journal of High Performance Computing Applications 2007 21: 132

[10] Lustre Lustre Parallel File System `http://en.wikipedia.org/wiki/Lustre_(file_system)`

[11] HDF5: Hierarchical Data Format `http://www.hdfgroup.org/HDF5/`

[12] NetCDF: Network Common Data Form `http://www.unidata.ucar.edu/software/netcdf/`

[13] MPICH2:. `http://www.mcs.anl.gov/research/projects/mpich2/`

[14] Panasas: `http://www.panasas.com/`.

[15] FhGFS: High-performance parallel file system from the Fraunhofer Competence Center for High Performance Computing `http://www.fhgfs.com/cms/`.

[16] Prost, J.-P., Treumann, R., Hedges, R., Jia, B., and Koniges, A MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS, in Proceedings of SC2001, Denver, Colorado 2001

[17] Francisco Javier García Blas and Florin Isaila and Jesús Carretero and Thomas Grossmann, Implementation and Evaluation of an MPI-IO Interface for GPFS in ROMIO, Proc. of the 15th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2008) 2008

[18] Parallel NFS: `http://www.pnfs.com/`

[19] Benchmark routines for parallel I/O. `http://beige.ucs.indiana.edu/I590/node86.html`.

[20] OpenMPI: Open Source High Performance Computing; MPI-2 Implementation `http://www.open-mpi.org/`

[21] MVAPICH2: MPI-2 over OpenFabrics-IB, OpenFabrics-iWARP, PSM, uDAPL and TCP/IP `http://mvapich.cse.ohio-state.edu/overview/mvapich2/`

[22] Hoffman2 Cluster:. Hoffman2 general purpose cluster hosted by IDRE. `http://www.ats.ucla.edu/clusters/hoffman2/`.

[23] IDRE:. Institute for Digital Research and Education, UCLA. `http://www.idre.ucla.edu`

[24] POD:. Performance Optimized Data Centers . `http://hp.com/go/pod`

[25] Gordon: Data-Intensive Supercomputing. `http://www.sdsc.edu/supercomputing/gordon/`

[26] Versatile SMP: Aggregating multiple x86 systems into a single virtual x86 system. `http://www.scalemp.com/`