Parallel File Systems Assessment

Baricevic Moreno
CNR-IOM DEMOCRITOS
baro@democritos.it

Cozzini Stefano
CNR-IOM DEMOCRITOS
cozzini@democritos.it

Díaz Gilberto
Departamento de Computación and
Centro de Càlculo Científico
Universidad de Los Andes
gilberto@ula.ve

Messina Antonio
International Centre for Theoretical Physics
amessina@itcp.it

Abstract

This paper reports our benchmarking work on available market solutions for parallel filesystems to deal with I/O in small cluster computing environments. Our aim is to present performance assessment of different parallel filesystem products. Different solutions were installed and then benchmarked on a small testing platform. The goal is to identify and implement the best solution both in terms of easiness of use and configuration, and in term of read/write performance. This study, even though limited by the size and the overall peak performance of the hardware setup, can give interesting information to research groups in search of a low cost parallel storage solution for small medium/size clusters.

Keywords Linux Clusters, Parallel I/O, Parallel Computing, Parallel File Systems, HPC, High-Performance Computing.

1. Introduction

Parallel Computing is the most powerful tool, in many areas, for solving problems which have a considerable dimension [1]. This is a form of computation where a large problem can be divided into smaller ones, then, these can be solved concurrently ("in parallel") [2]. People use parallel machines for only one reason: speed. Parallel machines are certainly no easier or cheaper to use than serial machines, but they can be much faster [3].

In the application level, we can classify programs that run on this kind of systems (parallel programs), according to the intensive use of the different resources present in a parallel machine:

- Processor intensive applications
- Memory intensive applications
- Disk intensive applications

In order to obtain high performances in this third category, the parallel machine must have a file system designed to give high performance as well. Inadequate I/O capability can severely degrade overall application performance, particularly in the current multi-teraflops machines [4]. In the same way a large problem is divided into smaller problems which are solved in parallel, a single I/O operation can be executed in parallel, dividing the data into smaller pieces and sending them to different storage devices, using several I/O controllers and several buses. Parallel file systems are file systems capable of performing I/O operations using several I/O devices and several storage devices, at the same time. Additionally, on a cluster, a parallel file systems enable a single-system and consistent view also from the file system perspective.

Linux Clusters are a very suitable platform for parallel file systems because they can provide, at the same time, several I/O controllers, storage devices and I/O buses natively. Furthermore, they are currently, the most popular platform for parallel computing due to their low cost and the availability of open-source parallel applications [1]. Over the last decade, they have predominantly populated the TOP500 list.

A large set of cluster tools for the installation, administration and maintenance of such clusters has been developed. Additionally, parallel cluster file systems are also available. If one has to make a decision about which file system should be chosen, a lot of alternatives have to be considered. There are freely available file systems like PVFS and Lustre, as there are proprietary solutions from

different hardware vendors, like GPFS from IBM, Panasas Filesystem from Panasas and many others. All these file systems allow concurrent access from many clients delivering a significantly superior performance over NFS.

Figure 1 shows how a parallel file system leverage all of these resources to execute a parallel I/O operation.

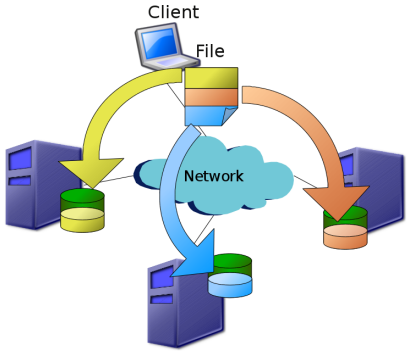

Figure 1. Parallel I/O Operation

In this work we focus on evaluating performance assessment of three parallel file system solutions (PVFS, Lustre, GPFS) widely used in cluster computing.

We were mainly interested in comparing these products in terms of installation and management complexity and on read/write performance on large size files. We did not test other features like fault tolerance or high availability.

Our effort was to address and mimic the typical situation of small/medium size HPC clusters which need a stable parallel filesystem, easy to configure and with decent performance.

The rest of the paper is organized in four sections. The second section describes some basic concepts about parallel file systems and the features of the software products used here. The third section describes the hardware architecture and the tools used to benchmark the performance. In the fourth section we show the results of the work and make the analysis. Finally, we present the conclusions based on the outcomes.

2. Parallel File Systems

A file system is a set of methods and data structures used to organize, store, retrieve and manage information in a permanent storage medium, such as a hard disk. Its main purpose is to represent and organize resources storage [5].

The term parallel file system describes a file system solution that supports parallel applications. In particular, they are able to leverage existing shared storage resources on a network in order to execute parallel I/O operations. In a parallel file system environment, all the nodes in the cluster may be accessing the same file (or files) at the same time, via concurrent `read()` and `write()` requests. Some few nodes are connected to the storage (known as I/O nodes) and serve data to the rest of the cluster. The main advantages a parallel file system can provide include a global name space, scalability, and the capability to distribute large files across multiple nodes [6].

Some of the most common parallel file systems in use today are:

- Lustre formerly from Cluster File System, then from Sun, now from Oracle (GPL);
- GPFS from IBM (private/proprietary license);
- PVFS2 from ANL (GPL).

2.1. Components

In general, a parallel file system implements a particular form of the following elements which are needed to manage, storage and retrieve objects into a set of shared storage devices.

- **Storage Devices:** these are hardware components which are capable of storing information in a permanent way, such as a magnetic hard disk drive. The special feature here is that this devices are shared through a network.
- **Storage Servers:** these are machines which are in charge of the management of the storage devices and make them available through the network.
- **Metadata Server:** metadata is a general term referring to information that is about something but not directly part of it. For example, the size of a file is a very important piece of information about that file, but it is not part of the data contained in the file itself. The metadata server is in charge of the management of metadata [5].
- **Client:** any machine connected to the network which make use of the information stored in the parallel file system.

The figure 2 shows the architecture of the components that constitute the parallel file system.

2.2. Parallel Virtual File System (PVFS)

PVFS is an open-source, scalable parallel file system. It is designed specifically to scale to very large numbers of clients and servers [7], and for parallel applications which share data across many clients in a coordinated manner.

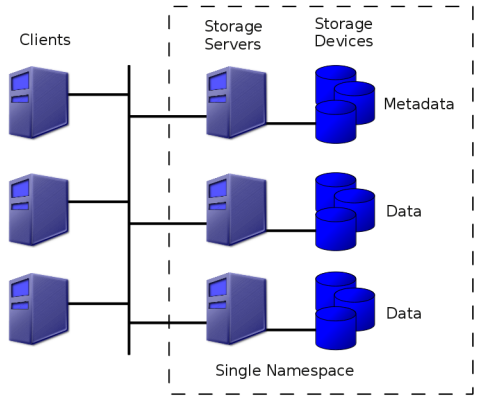

Figure 2. Parallel I/O Operation

PVFS2 servers store data for the parallel file system locally. The current implementation of this storage relies on UNIX files to hold file data and a Berkeley DB database to hold things like metadata. PVFS2 file system may consist of the following pieces:

- **pvfs2-server:** is the server daemon component of the file system. This is the only type of server process, the pvfs2-server, which is also a UNIX process - so one could run more than one of these on the same node if desired. Each instance may act as either a metadata server, an I/O server, or both at once.
- **system interface:** is a low level user space API that provides access to the PVFS2 file system.
- **management interface:** is an API intended for use by system administrators and for maintenance applications and performance monitoring applications.
- **Linux kernel driver:** is a module that can be loaded into an unmodified Linux kernel in order to provide VFS support for PVFS2. This is the component that allows standard Unix applications to work on PVFS2.
- **pvfs2-client:** is a user-space daemon that handles communication between PVFS2 servers and the kernel driver. Its primary function is to convert VFS operations into system interface operations.
- **ROMIO PVFS2 device:** is a component of the ROMIO MPI-IO implementation

2.3. Lustre

Lustre is an open source parallel file system, currently available only for Linux, that provides a POSIX-compliant UNIX file system interface. It is installed on seven of the ten largest high-performance computing clusters in the TOP500 list [1]. Among main features of Lustre we have: **scalability**, Lustre scales up or down with respect to the number of client nodes, disk storage and bandwidth; **POSIX compliant**, most operations are atomic and clients never stale data or metadata; **high availability**, Lustre offers shared storage partitions for data and metadata; **security**, it is possible to configure TCP connections only from privileged ports; **Open Source**, Lustre is developed under GNU GPL; **controlled striping**, it is possible to control the stripe count and stripe size in different ways [8]; **snapshots**, Lustre includes LVM technology to create snapshots.
The main components of Lustre are:

- **Metadata Target (MDT):** is any storage device dedicated to maintain the metadata.
- **Metadata Server (MDS):** server in charge of MDT management.
- **Object Storage Target (OST):** is any storage device where the real data is stored.
- **Object Storage Server (OSS):** these are in charge of the OST management.
- **Lustre clients:** these are machines which mount the file system and use it, for example: computational, visualization or desktop nodes.

2.4. General Parallel File System (GPFS)

GPFS is a commercial parallel file system developed by IBM. It provides file system services to parallel and serial applications. GPFS allows parallel applications simultaneous access to the same files, or different files, from any node which has the GPFS file system mounted while managing a high level of control over all file system operations. This provides global namespace, shared file system access among GPFS clusters, simultaneous file access from multiple nodes, high recoverability and data availability due to native replication mechanisms, the ability to make certain changes while a file system is mounted, and simplified administration that is similar to existing UNIX systems [9]. Its main components are:

- **Administration and configuration tools:** a very powerful set of commands to setup and manage the parallel file systems are provided.
- **Kernel extension:** it's a set of kernel modules which provide the interface to the Virtual File System layer of the operating system.

[1] The TOP500 list is an initiative to assemble and maintain, since 1993, a list of the 500 most powerful computer systems (http://www.top500.org)

- **GPFS Daemon:** it's a Unix-like daemon in charge of all I/O operations and the consistency of the data.

- **GPFS open source portability layer:** for nodes whose operating system is Linux, portable modules for Linux kernel must be compiled in order to have a communication channel between GPFS kernel modules and Linux kernel.

3. Test Platform

Our testbed was based on old hardware recently dismissed from the cutting-edge computational platform at ICTP, but still valid to run such kind of tests. A small Linux cluster was installed using EPICO [10], a flexible and customizable framework for the unattendend deployment of clusters, and it consists of four storage nodes attached to a SAN using Fibre Channel connection, 3 compute nodes and one master node. The SAN is a Sun StorageTek 6140. Compute nodes have two single-core 2GHz AMD Opteron processors and 4GB of RAM while Storage nodes have two single-core 2.5GHz AMD Opteron processors and 4GB of RAM.

Figure 3 shows the hardware configuration.

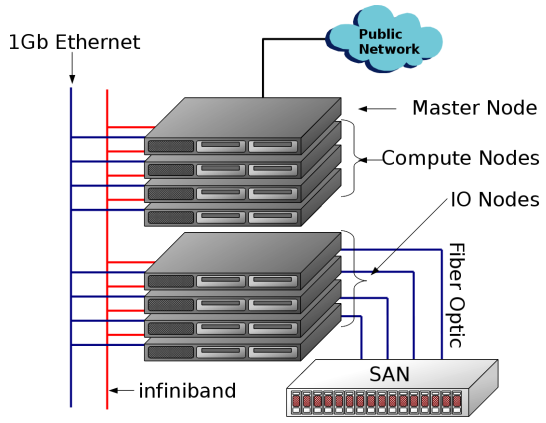

Figure 3. Hardware Testbed

Several different layers (both physical and logical) are involved in such infrastructure. At the lowest level we have the *storage device (SAN)* and right on the top of it there is the *Fibre Channel* bus. We then have the I/O nodes, which are providing a *standard file system* layer and, finally, there is a *network* which is connecting the I/O servers in order to serve the *parallel file system* layer to the clients. Figure 4 shows a graphical idea of this layered architecture. In the next subsection we will briefly present each layer, discussing which kind of peak performance they are able to deliver.

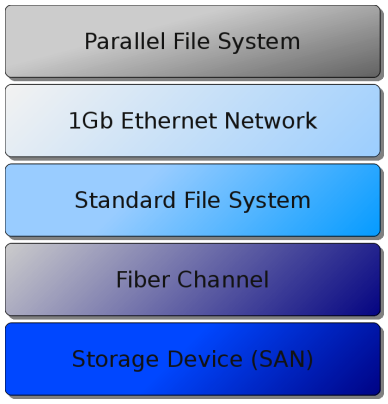

Figure 4. Architecture Levels

3.1. SAN Configuration

The storage device has two I/O controllers with 16 500GB SATA hard-disks. The performance reported by the vendor of such device is in the order of 650 MB/s [11].

The configuration was the following: four independent volumes (two volumes per I/O controller) and four virtual devices with RAID 0 (block-level striping without parity or mirroring, zero-redundancy) for each volume. We assigned four virtual devices of the same volume to each I/O node, and they were recognized as: /dev/sdb, /dev/sdc, /dev/sdd and /dev/sde. Figure 5 shows this configuration.

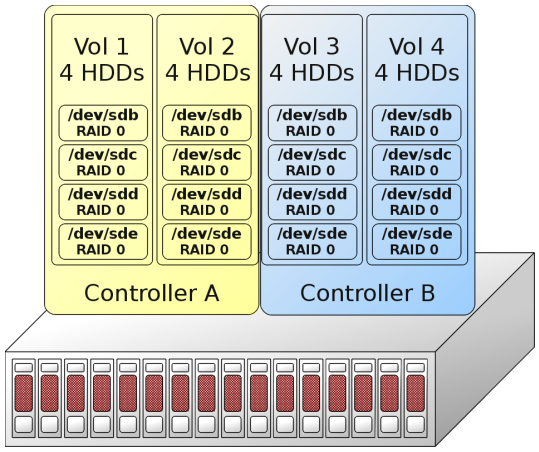

Figure 5. SAN configuration

3.2. FC System

The Storagetek device is connected to the I/O nodes through Fibre Channel. FC Cards on the I/O nodes are Qlogic QLA2344 HBAs (quad-optical 2Gb Fibre Channel

to 64-bit, 133MHz PCI-X HBA) and therefore the peak performance of each channel is of 256 MB/s (200 MB/s maximum speed at half-duplex, 400 MB/s maximum at full-duplex). Only one of the four channels was used for each card/node.

3.3. Standard File Systems

On the top of I/O nodes we evaluated three of the most common standard file systems: **ext3**, **ext4** and **xfs**. They were selected based on their wide spread adoption with respect to other less common choices.

3.4. Network

Two network technologies were used to conduct the experiments: a standard 1Gb Ethernet network and Infiniband SDR cards. Theoretical peak performance are 125 MB/s and 1250 MB/s respectively, however we tested the performance of both networks using iperf [12] between each couple of client and server. The average speeds are presented in table 1:

network	Mbits/s	Mbytes/s
Ethernet	943	117.87
Infiniband	1740	217.50

Table 1. Network performance summary

It has to be pointed out that we did not perform any fine-tuning of the network configuration, in order to keep the same conditions for all parallel file system tests. Moreover, even though the Infiniband cards and the switch used in this test are theoretically capable of 10 Gbps, the overhead of the TCP/IP stack has to be taken in account, as well as the fact that the peak performance of Infiniband can be reached only using native protocols and, possibly, RDMA.

When not otherwise specified, all performance tests were done using Gigabit network. In section 4.4 we present some comparison for Lustre using both networks.

3.5. Parallel File Systems

Three of the most common parallel file systems were assessed: PVFS2 V2.8.2, Lustre V2.0 and GPFS V3.3 (provided by the IBM scholar program).

It has to be pointed out that the configuration process of these parallel file systems requires a considerable amount of time due to their complexity, and that their tuning could take even longer. Therefore, we have evaluated them using their default configuration.

4. Results and Analysis

In this section we describe the benchmarking tools we used, the different kind of measurements we performed, and we discuss/analyze the results obtained. We will try to identify the role, and measure the performance, of each layer composing our testing platform.

4.1. Performance measurement tools

We used the following two well-known standard tools to measure performance values for the logical filesystem layers discussed in the previous section:

- **dd:** a Unix program that executes a low level copy and/or convert raw data. In our context it is actually used as a single benchmark for sequential read and write, but since it does not need a filesystem, as it can operate directly on a block device, it is very useful to get an indication about the raw performance of any storage device seen by the operating system.

  The next step is to put a filesystem on the device being tested, and then thoroughly benchmark the performance with some parallel I/O. For this task we used iozone.

- **iozone** [13]**:** a filesystem benchmark tool able to tests file I/O performance for the following operations: read, write, re-read, re-write, read backward, read strided, fread, fwrite, random read, pread, mmap, aio_read, aio_write. In this assessment we are going to use this tool to test the read and write performance of both standard and parallel file systems.

4.2. Results on I/O server

To evaluate reading and writing performance on the two lowest layers of the architecture (the SAN infrastructure plus the FC channel) we executed the following pair of **dd** commands:

```
dd if=/dev/sdb of=/dev/null bs=512k count=32768

dd if=/dev/null of=/dev/sdb bs=512k count=32768
```

The above commands were executed at the same time on the four nodes, and the results for each node were then aggregated in order to compare them with the results obtained by **iozone** executing the following command lines:

```
iozone -i 0 -i 1 -t 4 -+m machinefile -s 16g \
-r 512k -C -+u -R
```

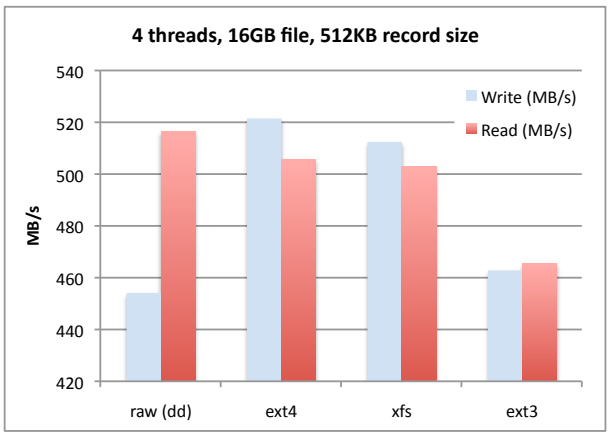

Figure 6. Performance of standard filesystems

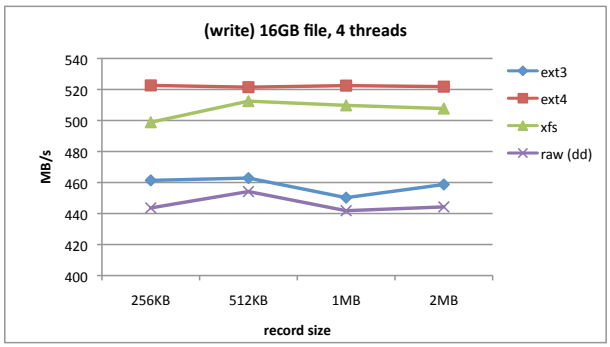

Figure 7. Block size scalability (File size = 16GB, Four threads)

The test was repeated on three different file systems (ext3, ext4 and xfs)

Figure 6 reports the results for 4 threads concurrently and independently writing and reading on the different four devices seen by each node.

Some considerations can be done on the above picture: raw access by means of (**dd**) is delivering the best performance when reading, but this is not true in case of writing, where it is actually delivering the lowest performance observed. Our guess is that this should be related to a better caching mechanism enabled by filesystems when a write operation is performed. Filesystem results indicate that xfs and ext4 are roughly delivering the same performance both in reading and writing (slightly higher than 500 MB/s in both cases) while ext3 is less efficient of about 10%.

We also evaluated the role of different block size in overall performance. The figure 7 shows the write operation performance of the standard file systems used in the *local file system* layer assessment. In this test we used a file of 16GB size, four instances of iozone (one thread per I/O node) and different block size. We can notice that overall performance keeps roughly constant with the block size, this is likely due to the caching mechanism and the asynchronous I/O variation.

4.3. Parallel file systems performance

In this section we report performance obtained on parallel filesystems and we compare them against the aggregate performance obtained on local filesystems. We executed the same iozone test on the three different parallel file systems from the four clients which mounted the parallel filesystem served by I/O servers via Gigabit Ethernet. This is the only, but important difference among the local and the parallel iozone execution. We also note that the default configuration used for the parallel filesystems does not support striping of a single file resulting in no gain in performance when accessing a single file. In some filesystems (e.g. Lustre) this behavior can be modified as discussed later.

The figures 8 shows the results of this test.

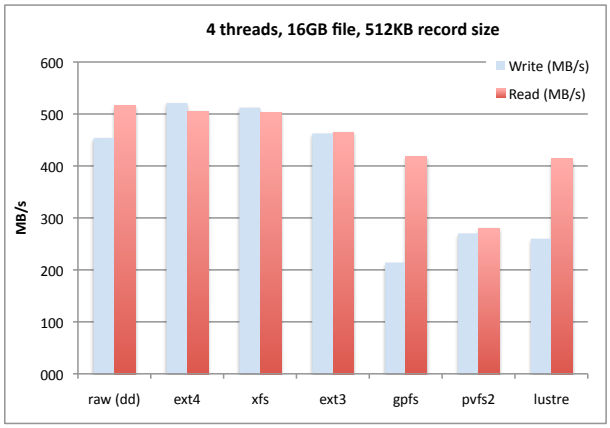

Figure 8. Performance of local and parallel filesystems

From the figure we can observe that parallel filesystems are delivering roughly 20% less of the aggregate performance as measured on local filesystems. The joint effect of the additional Gigabit and the overall metadata management can be responsible of this performance drop. In reading, though, it is presumable that metadata are not accessed/updated often, the 20% gap is thus probably only due to the network, which **has not been tuned at all**.

We also observe that GPFS delivers half of the performance in writing operation with respect to reading, and also Lustre is showing significantly less performance in writing

than reading. PVFS, on the contrary, does not have so much difference in reading and writing operations, even though the overall performance is the lowest among them.

In figure 9 we report the comparison of standard file systems vs. parallel file systems with respect to the size of the file that is written. From the figure, the cache effect played by the RAM of the I/O servers (4GB RAM each) is quite evident for local file systems. Among parallel filesystem, Lustre seems to be the only one that is affected by the RAM size of the I/O nodes. We also include here results obtained with Lustre over Infiniband network (see next subsection for details) (**Lustre (ib)** in figure 11). This setup gave much better results in reading than the same filesystem accessed via Gigabit, but more or less the same in writing. The write operation is affected by a slow down due to the job played by metadata updates, but in reading, we can virtually go as fast as the hardware can.

We again report a gap between local and parallel filesystem for all the file sizes explored outside the cache effect zone. As side observation we note also that ext3 shows a poor cache management in writing operations.

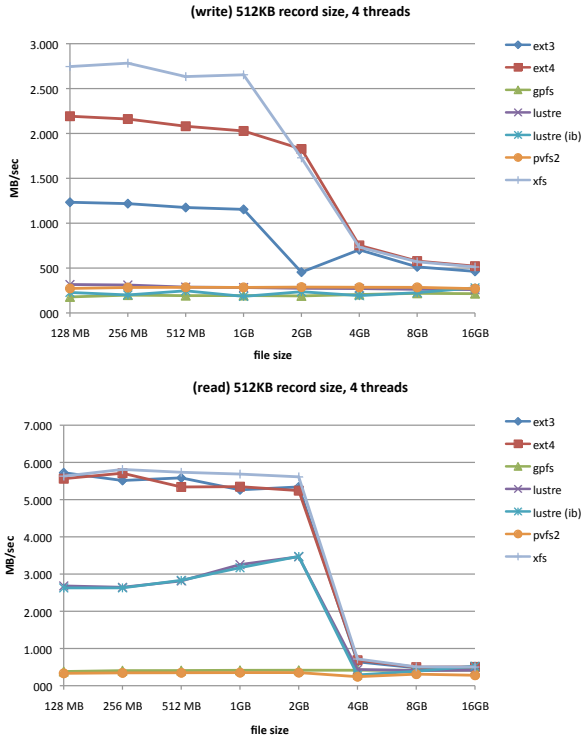

Figure 9. Filesize scalability of local and parallel filesystems

We finally show, in figure 10, how performance increases at the increasing of the number of concurrent files (threads) written on parallel file systems. Performances, both in reading and writing, for all parallel filesystems are there reported - for Lustre we have also included the results obtained using the Infiniband network. All filesystems increases the overall performance both in reading (panel a) and writing (panel b). Scalability behavior in reading is similar for all the filesystems, with Lustre over IB delivering the best performance. Conversely, on writing performance GPFS does not scale at all going from two to four threads, while all the other filesystems show acceptable scalability.

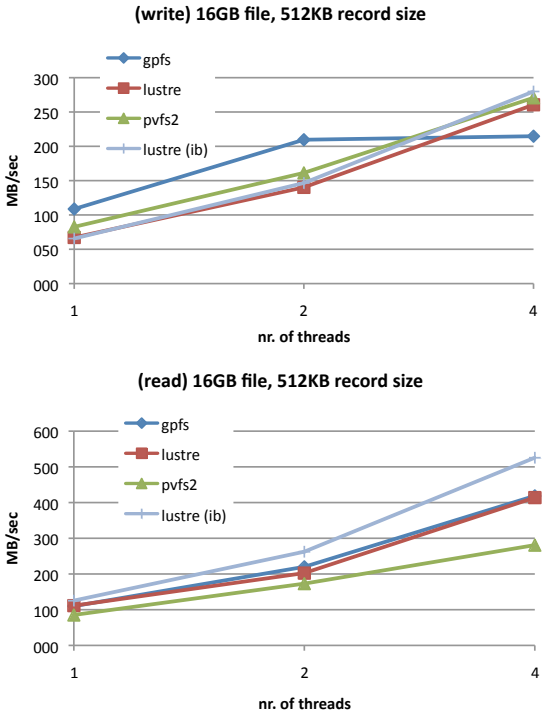

Figure 10. Parallel File System Thread scalability

4.4. Further testing with Lustre

Since Lustre gave quite good results in previous tests and since it is highly configurable, we made a few more tests in order to better understand the impact of the TCP/IP stack on the filesystem performance and how the filesystem behaves with different striping configuration.

At first, we just configured the Lustre `net` module in order to use the Infiniband network instead of the standard Gigabit network. In figure 11 we compare the performance of a single thread of iozone, with file size of 16GB and block size of 512KB.

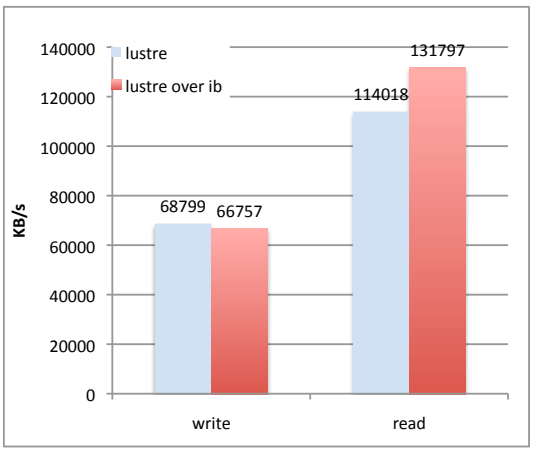

Figure 11. Comparison between Lustre over Gigabit and Lustre over Infiniband network

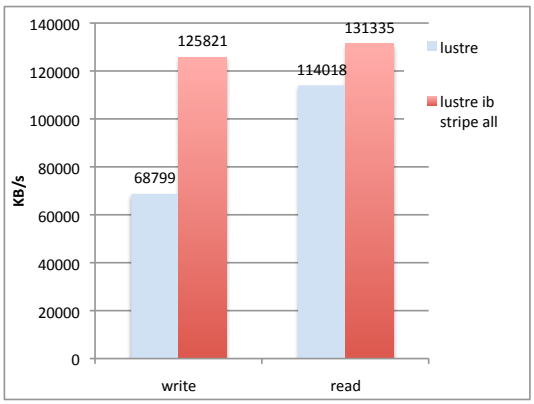

Figure 12. Comparison between Lustre (gbe) and lustre (ib) with every file striped over all OSTs

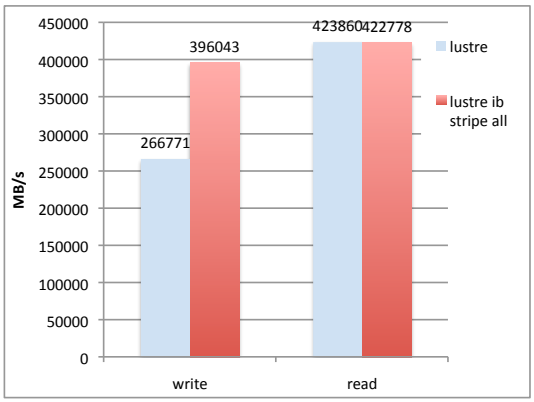

Figure 13. Running 4 clients against Lustre (gbe) and Lustre (ib) with every file striped over all OSTs

The results show an improvement in performance (around 10%) in reading, and almost no improvement in writing, which is a bit surprising since we would expect more or less the same improvement both for reading and writing.

As the reader can easily verify, with a single iozone thread we are always getting performance which are lower than the performance of a single raw device. This is because Lustre, by default, writes a single file on a single OST, which means that when a single file is written on the filesystem, this will be actually written on a single LUN of our SAN.

This behavior can be modified on a per-file or per-directory basis. In order to check if we can actually get better performance from Lustre we forced Lustre to stripe over all available OSTs using the command:

```
lfs setstripe -c -1 /lustre
```

Results are quite interesting as shown in Figure 12:

In this case we clearly see that striping over all the OSTs almost double the performance of a single writer. Another very interesting observation can be done looking at figure 13, where we report a four threads experiment executed with striping option active. Surprisingly enough, increasing the number of clients does not negatively affect the performance, but it actually improves it, as clearly shown in Figure 13. This means that it is actually more convenient to stripe four files over four OSTs instead of writing them separately on each OST.

5. Conclusions

We have reported here the results obtained on installing and benchmarking three different parallel filesystems products on a small testbed. Our aim was to evaluate them and acquire some insights on how a low cost parallel file system solution can be attached to a small/medium size cluster for HPC computing. We observed that, from the point of view of the installations/configuration procedures, all the parallel filesystems are equivalent: the procedures are not particularly complicated, at least for HPC sys-admins with some experience. Results obtained and discussed above on simple benchmarks, are showing that PVFS2 is not delivering enough performance with respect to Lustre and GPFS. Moreover, we report that Lustre was flexible enough to allow relatively simple tuning that lead to a net increase of performance. However we collected just some preliminary results that are showing quite a complex landscape. This requires some further in-depth testing and careful analysis to get a full picture of what one can expect in term of perfor-

mance and configuration/installation issues. We are therefore planning some additional experiments on our testbed on both hardware and software level. We finally remark that this work was started as lab session within our recent training activity at ICTP [14] and then evolved into a more structured research activity. We still believe that the pedagogical motivation is valid and for this reason we are actually planning to organize materials (scripts, tools and ad hoc developed documentation) in a package to make it available to other groups and/or students to easily repeat similar benchmarking analysis on different testbeds.

References

[1] S. Spier. Parallel file systems, seminar paper "hot topics in operating systems", March 2006.

[2] Wikipedia. *Parallel computing*. `http://en.wikipedia.org/wiki/Parallel_computing`, April 2011.

[3] K. D. Cormen Thomas. *Integrating theory and practice in parallel file systems*. pages 64–74, 1993.

[4] A. Saify, G. Kochhar, J. Hsieh and O. Celebioglu. *Enhancing high-performance computing clusters with parallel file systems*. Technical report, DELL, 05 2005.

[5] D. Giampaolo. *Practical File System Design with the Be File System*. Morgan Kaufmann Publishers, Inc., 1999.

[6] D. A. Heger. *Parallel file system technologies in a linux cluster and grid environment*. In *Int. CMG Conference*, pages 429–442, 2007.

[7] PVFS. PVFS FAQ. `http://http://www.pvfs.org/cvs/pvfs-2-8-branch.build/doc/pvfs2-faq/pvfs2-faq.php`, April 2011.

[8] SUN MICROSYSTEMS. *Lustre Operation Manual Version 1.8*, March 2010.

[9] IBM Corporation. *Concepts, Planning, and Installation Guide Version 3.3*, September 2009.

[10] M. Baricevic. *EPICO - eLab Procedure for Installation and Configuration*, submitted to CLCAR2011 conference, May 2011.

[11] SUN MICROSYSTEMS. SPC benchmark 2 executive summary. `http://www.storageperformance.org/results/b00016_Sun-ST6140-Mirroring_SPC2_executive-summary.pdf`, April 2011.

[12] NLANR/DAST. Iperf. `http://iperf.sourceforge.net`, April 2011.

[13] *IOzone Filesystem Benchmark*, `http://www.iozone.org/docs/IOzone_msword_98.pdf`, September 2009.

[14] ICTP Advanced School on High Performance and Grid Computing. `http://cdsagenda5.ictp.it/full_display.php?ida=a10135`, April 2011.