

Benchmarking I/O Solutions for Clusters

Stefano Cozzini and Moshe Bar

Democritos INFM National Simulation Center, Trieste (Italy)
{cozzini,moshe}@democritos.it

Abstract. This paper details our benchmarking results of some solutions for I/O in High Performance Throughput clusters. Our goal is to present I/O performance data in order to identify and implement the best solutions for our specific I/O requirements within our research center. This benchmark is not based upon abstract workloads but rather on real scientific production workloads used on a daily basis here at INFM National Simulation Center.

1 Introduction

Today, Linux clusters offer viable solutions for low-cost, high performance parallel computing. The continuous increase of performance in commodity hardware (both CPU and networks) together with the availability of many ready-made software packages for key services (like management tools /networking, etc.) make this kind of parallel platforms an appropriate solution for many scientific computational problems. Clustered systems offer, indeed, many advantages for scientific “demanding” applications: they can deal with huge CPU-bound requirements and allow to distribute the RAM needed among many nodes. However, many scientific applications process massive amounts of data, and therefore require high performance distributed storage next to parallel I/O. Supporting High Performance I/O (HPIO) in cluster environments has proven to be a difficult task. Certain software packages provide various elements in this arena. However, due to huge complexity of these packages, in term of configuration and usage parameters a clear general purpose solution of the I/O problem is not today available.

In this article we will discuss today’s I/O solutions for clustering by presenting our benchmarking work. This work was performed at Democritos, the INFM ¹ National Simulation Center at Trieste (Italy), where different Linux clusters are integrated within the computational environment. In a recent paper [1] we discussed in details different cluster technologies adopted within our center. Now, we turn our attention to the I/O area by analyzing performance data of available solutions for High Throughput Computing (HTC), as opposed to High Performance Computing (HPC). We focus here mainly on serial applications with intensive I/O requirements. We test therefore the openMosix File System

¹ INFM (Istituto Nazionale Fisica della Materia) is the Italian National Institute for condensed matter Physics

(oMFS), being the openMosix cluster technology [2] adopted in our center. Furthermore we present some performance data on UCFS a proprietary cluster File System packaged along with Qclusters QOS². One is also particularly interested in knowing if the parallel file systems' ability to stripe I/O operations across several disk devices provides any substantial performance improvement in case of serial I/O. We report further down performance numbers obtained using the GPFS [3] and PVFS [4] parallel file systems.

This paper is organized as follows: in Section 2, we present our computational environment giving an overview of our clusters configurations together with some technical details of the cluster file systems installed on our facilities. In Section 3 results obtained are presented and discussed. Some conclusions will be drawn in Section 4.

2 Computational Environment

2.1 Cluster Facilities

The Linux cluster facilities tested in this study consist of two production clusters: one installed at ICTP³ and dedicated to parallel applications, the other installed at Democritos and devoted to serial computation with the adoption of the openMosix paradigm. A third cluster is reserved for testing and at this moment is running 1.1 beta version of Qclusters OS. The ICTP cluster is devoted to parallel computing and it is equipped with Myrinet 2000 cards. On this cluster, as detailed below, IBM GPFS and PVFS parallel file systems are installed. The openMosix cluster, named "Hokule", is devoted to serial computations and uses a standard Fast Ethernet network: the file system investigated in this case is of course the oMFS one. Finally the testing facility, named "Moleta", allows to benchmark the UCFS file systems. The technical specifications of all the clusters mentioned in this paper are reported in the table 1.

2.2 File Systems Technologies

We offer here a brief overview discussing the technical specifications for each file system which we use for the purpose of this paper. We report specific configuration parameters as well.

- *NFS* is the poor man's cluster file system. Note that NFS is declaredly not a clustering file system but rather a network file system (hence its acronym). As such, NFS does not guarantee cache consistency. However, a significant number of cluster installations chose to use it as a replacement for a clustering file system and that is why we include it in our benchmarks. In particular

² Qclusters OS (QOS) is a clustering operating system created by Tel-Aviv based Qclusters, Inc. See www.Qclusters.com

³ ICTP (International Center for Theoretical Physics) is a UNESCO organization located in Trieste(Italy). The condensed matter group of ICTP is part of Democritos

Table 1. Technical characteristics of the Linux clusters used

Cluster	ICTP	Hokule	Moleta
HARDWARE			
master node	IBM x345	IBM x342	IBM x330
computing nodes	IBM x335	IBM x330	IBM x330
processor	dual PIV 2.4GHz	dual PIII 1.4GHz	dual PIII 1.4GHz
CPUS (total)	80	18	8
Cache (L2)	512 KB	512 KB	512 KB
DRAM (x node)	2 GB	1 GB	1 GB
PCI bus	66 MHz 64bit	33 MHz 64 bit	33 MHz 64 bit
network	Myricom 2000	Intel EPro 100	Intel EPro 100
SOFTWARE			
Linux kernel	2.4.18-10smp	2.4.18	2.4.18QOS
Kernel Patch	GPFS/PVFS	openMosix 2.4.18-2	QOS 1.1b
File systems	GPFS/NFS	oMFS/NFS	UCFS/NFS

on all our installations we export the “home” partition defined on the master node via NFS protocol in order to make it available to all the computing nodes. This is quite standard solution that simplify job submission for many task not I/O bounded.

- *oMFS* openMosix File System (oMFS in short) is openMosix’ virtual file system for clustering. As such it does not itself store data but relies on an underlying physical file system and provides a view to a unified name-space by giving access (privileges permitting) from all cluster nodes to all other nodes’ file systems. Because it unifies the name-spaces, a user or program on node A can read/write or execute a file residing on node N while maintaining full UNIX access semantics. Because of openMosix’ inherent process migrating capabilities, oMFS also maintains full cache coherency among nodes, ie the local caches of all nodes are maintained up to date in case of changes onto a file from any one node. In oMFS additional consistencies are guaranteed, such as time stamp consistency (making the file’s time stamps consistent with the user’s or process’ home node time of the day as opposed to the file server’s time of the day) and link consistency (making symbolic links resolve properly in the unified name-space). oMFS also constantly monitors usage and in case of a smaller program address space accessing a bigger, remote file the address space migrates to the file’s location instead of moving the file over the network to the process. oMFS is not a Remote Procedure Call (RPC) based file system and therefore is much less dependent on buffer size and totally avoids the remote name lookup (subdirectory path verification) overhead problem inherent of all RPC network file systems such as NFS.
- *UCFS* Qlusters’ Unified Cache-consistent File System (UCFS in short) is a re-write and significant extension of the previously mentioned oMFS file system. Whereas in oMFS most development effort went into proof-of-concept and cache-consistency, UCFS expanded into aggressive use of cache wherever

possible and HPC-oriented smart read-ahead read patterns. Cache usage is optimized significantly by identifying write-once, never-read again patterns often seen in HPC applications and therefore forcing drop-behind caching. Drop-behind is a proprietary Qlusters algorithm for bypassing Linux' inherent caching [5] of all reads and writes into its buffer and page cache when not necessary. A good example is the reading-in of an MP3 file; a listener is not very likely to play again the same song from an MP3 and therefore there is not need to keep its contents in the page cache (in this case, it is still advisable to keep the buffer cache, ie inode meta-information). Same applies to HPC applications writing huge data sets; UCFS predicts intelligently when data is not required to pollute the page cache and drops-behind immediate use. Furthermore, due to Qlusters implementation of distributed shared memory (DSM) important system calls like mmap are now also available for it's UCFS file system (mmap() shared is not supported on oMFS due to lack of DSM there).

- *GPFS* General Parallel File System is a scalable cluster file system for IBM Cluster systems, including the Linux (Cluster 1350) and the AIX (Cluster 1600) systems. GPFS manages system resources to assure adequate bandwidth is available for each multimedia stream. Disk I/O is scheduled to assure smooth delivery. GPFS allows multiple streaming servers to share content on switch-attached disks. On the SP, switch-attached disks are implemented using the Virtual Shared Disk (VSD) subsystem. Tiger Shark stripes files across tens or even hundreds of shared disks for load balancing and high throughput. In addition to high-speed parallel file access, GPFS provides fault-tolerance, including automatic recovery from disk and node failures. We investigated different configurations of the GPFS file systems in this work, due to the fact the ICTP cluster is logically partitioned in two sub-clusters, serving two distinct communities of users. These configuration are the following: (i) a general storage configuration (on ICTP cluster) where one designated storage node (in this case an IBM x345 server) with several hard disks (three, in our case) are used conjunctively as one big storage area, (ii) a cluster scratch area which holds temporary job data made out of the local hard disks of all the participating cluster nodes to form one big logical file system and, (iii) a second scratch area for another cluster made of selected cluster node's (seven nodes in total) local hard-disks to form one logical file system. All the previously mentioned GPFS file systems use the low-latency, high-bandwidth Myrinet 2000 network with TCP/IP encapsulation.
- *PVFS* The Parallel Virtual File System (PVFS) Project is an effort to provide a high-performance and scalable parallel file system for PC clusters. PVFS is open source and released under the GNU General Public License. It requires no special hardware or modifications to the kernel. PVFS provides four important capabilities in one package: (i) a consistent file name space across the machine, (ii) transparent access for existing utilities (iii) physical distribution of data across multiple disks in multiple cluster nodes, (iv) high-performance user space access for applications In order to provide high-performance access to data stored on the file system by many

clients, PVFS spreads data out across multiple cluster nodes (I/O nodes). By spreading data across multiple I/O nodes, applications have multiple paths to data through the network and multiple disks on which data is stored. This eliminates single bottlenecks in the I/O path and thus increases the total potential bandwidth for multiple clients, or aggregate bandwidth. On the above-mentioned ICTP cluster, we reserve eight nodes for PVFS analysis: four nodes as I/O nodes with a locally attached hard-disk forming one PVFS logical file system available to four other cluster nodes (herein called compute nodes). The PVFS is working above a copper Gigabit Ethernet network.

Table 2 summarizes configuration characteristics for all the file systems investigated in this study.

Table 2. File System configurations used in this work. Note that for scratch2 file system the I/O nodes serve also as computing nodes

FS name	type	disks	I/O nodes	computing nodes	network
storage	GPFS	3	1 (x345)	32	Myrinet
scratch1	GPFS	7	7 (x335)	32	Myrinet
scratch2	GPFS	12	12 (x335)	12	Myrinet
home	NFS	1	1 (x342)	32	GE
pvfs	PVFS	4	4 (x335)	4	GE
oMFS	oMFS	4	-	4	FE
UCFS	UCFS	4	-	4	FE

3 Results

For general I/O benchmarking a high number of parameters must be considered and often benchmarks procedures sample only a small subset of this multidimensional space. The main criterion we use in defining our procedure is to mimic as much as possible the I/O requirements of real applications routinely used in our computational environment.

We use through this study the *Bonnie++* [7] benchmark. It tests performance using the POSIX system call level I/O functions `read()` and `write()`. For this reason *Bonnie++* is a good indicator of the performance characteristics of sequential applications like the ones our user community runs at Democritos.

We measure the throughput for all the available file systems; to avoid cache exhaustion problems the *Bonnie++* benchmark runs on files sized double the RAM of the machines.

The values reported here measure the so-called Block Sequential Output and Input tests, together with the percentage of CPU usage. In the first case the file is created using *write(2)* and the CPU overhead should be just the OS file space

allocation, while in the second test the file is read using *read(2)*. This is a test of pure sequential input performance. Tests are repeated at least three times and averaged values are presented. Error bars for all the data presented are within a 2% range.

3.1 Raw Disk and Network Performances

Raw performance data was measured for the local disks installed on the clusters. We use in this case the standard Linux ext3 file system. In table 3 we report Bonnie++ performance numbers for the four combinations of SCSI disks/hardware nodes available in our environment. IBM x345 and x342 machines are equipped with a hardware RAID controller: ext3 file system on x345 is using RAID 1 (mirroring), while on x342 there is no RAID active.

Table 3. Bonnie++ performances on different cluster nodes

File Systems	Mb/sec	% CPU	Mb/sec	% CPU
	Block Output		Block Input	
ICTP:master node:x345	13	12	46	21
ICTP:computing node:x335	15	12	55	21
Hokule:master node:x342	30	12	43	21
Hokule:computing node:x330	28	13	42	8

Master node x342 and computing node x330 are practically equivalent in performance with good performance in reading and acceptable writing speed.

There is however a large difference between writing and reading speed on master node (x345) and computing node (x335) on the ICTP clusters. This disappointing feature has to be considered taken later when comparing performance data obtained on the cluster file systems.

We also measured network performance in term of latency and bandwidth of the TCP/IP protocol between pairs of nodes using the different NICs available. The overall performance of clustering file systems is ultimately mostly defined by the latency / bandwidth of the underlying network technology and topology. We use the standard TCP sstream test provided by netperf version 2.2pl3 benchmark [6] to measure the speed of the TCP/IP protocol both on Gigabit card (Broadcom card) and the Myrinet ones. We also included as reference standard Ethernet cards. Figure 1 reports performance number normalized to standard FE values, as function of the size of the packets. Despite the small packet size range, the difference in performance is what one can expect between GE and FE (a factor of almost ten). The gap between Myrinet and Gigabit cards is however not so pronounced as one can find when using both card in a MPI environment.

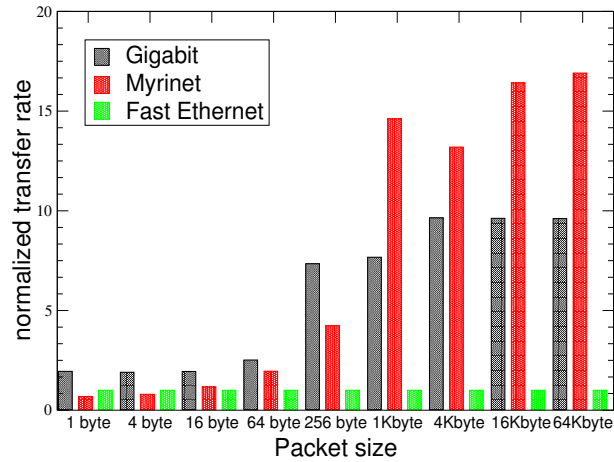


Fig. 1. Netperf performance: data are normalized versus Fast Ethernet values for all sizes

3.2 Serial Performances with Bonnie++

A global overview of collected data is presented in table 4 for all the file systems considered in this work.

Table 4. Bonnie++ Performances on the various file systems considered. Error bar is within 2%

File systems	Block Output		Block Input	
	Mb/sec	% CPU	Mb/sec	% CPU
storage(GPFS)	19	5	40	7
scratch1(GPFS)	31	7	101	18
scratch2(GPFS)	49	7	85	14
home(NFS)	10.7	5	9.5	4
PVFS	4.3	1	21	5
oMFS	8.3	9	29	14
UCFS	26	23	24	10

Data presented require some comments:

- GPFS file system “storage” at first glance seems quite disappointing; writing speed on the file system is just above the standard (local master node x345) Linux FS while reading performance are actually below the speed of the local disks. GPFS’s adjoint value here is that this performance numbers are true for all the nodes of the cluster, meaning that writing and reading sequentially from any node of the cluster can be done at the same speed: this is indeed

the real advantage of this GPFS configuration. We note that this is mainly due to the high speed network: Myrinet is in this case the right solution. Performance obtained on the two scratch area are much better and require some further analysis we present later.

- PVFS performs poorly in our case. This bad result is well documented in the PVFS FAQ: “ Bonnie uses a 16Kbyte buffer for accessing the file which it is writing to, which is a particularly small access size for the PVFS system .. TCP overhead is very apparent at requests this small.” This explanation discourages us in using PVFS as a good candidate for our serial I/O applications and we therefore do not present any other number for this FS in this study.
- UCFS performance within the same node is almost equivalent to the performance obtained on the standard ext3 file system, indicating the software overhead introduced is practically negligible.
- oMFS within the node show the same behavior as UCFS only in the reading operations. As a matter of fact there is a large performance penalty in writing locally through the oMFS file system. This is due to the decoupling of oMFS from the standard Linux file system caching, which has evolved considerably in the last few years. oMFS for kernels 2.6 will return to using the standard Linux kernel caching, according to openMosix developers.

GPFS performance allows some additional interesting comments that can be done with the following figure.

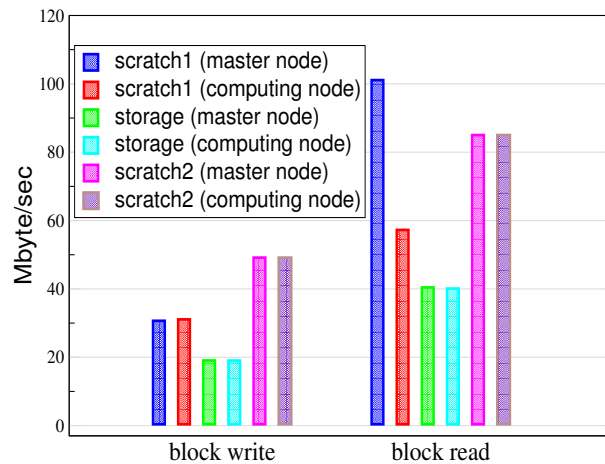


Fig. 2. GPFS performance on different configurations

We report here writing and reading speeds measured respectively on the master node and on a computing node for all the three file system considered in this study. The figures easily point out the main advantage of the GPFS

FS: performance obtained on the master node is the same one can obtain on the computing node. Some other observations can also be done: (i) on all the GPFS configurations reading performance is much better than the writing one, (ii) scratch2 FS is much more performing in writing with respect to scratch1: this is clearly due to the fact we are using here 12 disks instead of just 7 disk and therefore the striping mechanism works better. This behavior has not been observed for reading operations: reading speed is lower on scratch2 FS while on scratch1 FS reading operation on a computing node is almost a 40% slower with respect to the master node. Due to the fact that a computing node on cluster2 is also an I/O node (i.e. a GPFS disks is attached to the node) our initial guess was that scratch2 FS can deliver better serial I/O performance on the master node with respect to the computing one. However, this assumption was false: performance is in the same range with no significant differences. On the contrary the expected behavior of the scratch2 FS is observed for the /scratch1 FS. This is indeed a puzzling observation which deserves some more detailed investigations.

The capacity to deliver excellent reading and writing performance is the jointly action of the striping mechanism together with the myrinet network in the GPFS file system. It could be interesting to see if even NFS can deliver better performance when running on high speed network. Unfortunately, for production reason, we have at disposal only x335 nodes to perform such a test. NFS performance obtained using either Gigabit or Myrinet network are practically the same and are equivalent to the reading/writing speed of the local disks. We are therefore not able to spot out any difference due to the limited capability of the available hardware.

3.3 High Throughput Bonnie++ Performance

We then use Bonnie++ in a similar fashion to our high throughput cluster by launching multiple copies of it on our HTC clusters; in this test we launch 4 different copies of the benchmark on the same node using the different cluster file systems.

Data for the Hokule cluster (running openMosix) are collected for oMFS: a small script runs each of the four instances on a local file system which is accessed through oMFS. The DFSA technology should make the Bonnie processes access the local file system instead of using the network. The same is done on “Moleta”, the cluster running QOS v1.1b using a similar script in order to use the UCFS file system.

We note that repeating the experiments measured data maintain almost constant (the spread in the I/O rates is less than 2%). Figure 3 therefore shows performance obtained in reading and writing for UCFS and oMFS file systems for each node of the cluster. Node 0 is the node where processes are started while nodes 1/2/3 are cluster members. We can observe that (i) the oMFS file system seems to deliver NFS performance on different nodes: this is confirmed also by the fact that processes tend to not migrate staying to the starting nodes; (ii) in the UCFS file system the DFSA mechanism works perfectly and scales

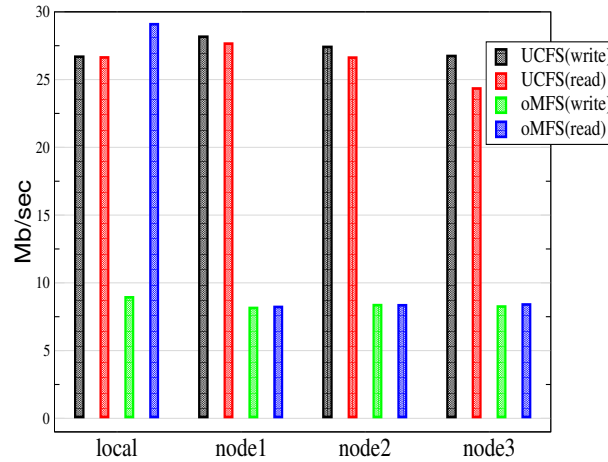


Fig. 3. UCFS vs OMFS performance: see text for explanations

up performance accordingly. We guess that the performance of oMFS could be significantly enhanced by switching to a later version of the kernel (2.4.20 is available at the time of writing of this paper). We plan to upgrade the kernel to the latest version soon.

Further down we also report here our experience with GPFS. We haven't yet tested GPFS together with openMosix for lack of available planned downtime on our production clusters. Instead we try to mimic its behavior the above experiments just using a queue system on a standard Beowulf systems. Therefore we run the four Bonnie++ processes directly on four different nodes by means of a queue system (openPBS in our case) This way we avoid the migration overhead and still have some numbers to compare with. As with previous benchmark we repeat the experiment several times on two different GPFS file system: storage and scratch1, varying also the number of jobs (i.e. the number of concurrent I/O operations). Results are presented in figure 4 where we report the I/O speed obtained for 1,2,4,8 instances executed concurrently. It is evident that GPFS scalability in this specific context is quite poor: the overhead traffic data in the striping mechanisms generated by more than 1 concurrent I/O operations reduce the performance having at disposal a limited number of disk where to put the data.

4 Conclusions and Future Works

In this paper we present some results on the I/O performance on different file systems available in the arena of cluster computing for high throughput calculations. It is clear that at the present time the two commercial file systems, GPFS and UCFS show superior performance when compared to their Open Source friends. Newly released versions of the file systems presented in this pa-

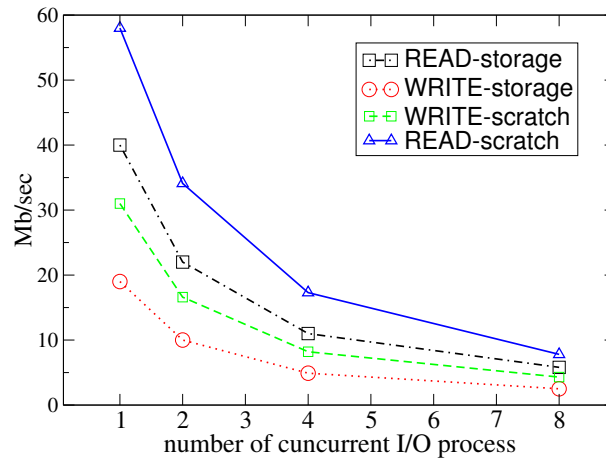


Fig. 4. UCFS vs OMFS performance: see text for explanations

per promise vastly improved performance. openMosix, for instance, is working on integrating with the new PVFS2 parallel file system, a complete re-write of the previous incarnation. We note however that GPFS File System shows no scalability at all when used in a HTC environment, despite the high speed network running underneath.

The present study is far to be complete: many different issues are still to be investigated; in particular we plan to study in details some other GPFS performance aspects in order to understand some unclear behaviors observed here. We are also planning to investigate if the same performance GPFS pattern can be spot out using a different (not so expensive) network, i.e. Gigabit network. We shall report our benchmarking findings in a future paper.

References

1. M.Bar S.Cozzini M. Davini and A. Marmodoro “openMosix vs. Beowulf: a case study” in Proceeding of the “Linux cluster the HPC revolution” October 2002.
2. Web site of the project: <http://www.openmosix.org>
3. IBM General Parallel File System fir Linux :Concepts Planning and Installation Guide, IBM Document GA22-7844-01, February 2002
4. 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 Association.
5. Linux File Systems, 2002 by Moshe Bar, McGraw-Hill New York
6. <http://www.netperf.org/netperf/NetperfPage.html>
7. R.Coker. Bonnie++, <http://www.coker.com.au/bonnie++>