



pCFS: A Parallel Cluster File System

P.A. Lopes, P.D. Medeiros

published in

Parallel Computing:

Current & Future Issues of High-End Computing,

Proceedings of the International Conference ParCo 2005,

G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata
(Editors),

John von Neumann Institute for Computing, Jülich,

NIC Series, Vol. 33, ISBN 3-00-017352-8, pp. 515-522, 2006.

© 2006 by John von Neumann Institute for Computing

Permission to make digital or hard copies of portions of this work for personal or classroom use is granted provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise requires prior specific permission by the publisher mentioned above.

<http://www.fz-juelich.de/nic-series/volume33>

pCFS: A Parallel Cluster File System *

Paulo Afonso Lopes ^a, Pedro D. Medeiros ^a

^aCITI and Department of Informatics - Universidade Nova de Lisboa
Monte de Caparica, Portugal
email: {pal,pm}@di.fct.unl.pt

This paper proposes pCFS, a high performance shared disk cluster file system (CFS) targeted at small to medium sized clusters, which aims to support a broad spectrum of I/O intensive applications, including support for parallel I/O.

Traditional CFSs perform control operations (locking and cache coherence) over the LAN and data access over the Storage Area Network (SAN); the proposed architecture will exploit all the available interconnect infrastructures (SAN and LAN) to maximize I/O bandwidth and minimize latency. It will also offer a high level of availability without sacrificing performance by taking into account two complementary views: hardware and software. By hardware, we mean multiported disk arrays exporting RAID volumes to a SAN with multiple access paths; on the software side, pCFS will use cooperating caches with replication of modified data blocks, allowing delayed writes without the fear of data loss.

The benefits of data transfer over LAN and SAN together have already been validated by an experiment involving a modified version of the GFS cluster file system.

1. Introduction

High performance I/O for cluster architectures has been a subject for a lot of research. While several big clusters have adopted low cost-per-node distributed disk (DD) architectures where I/O nodes have inexpensive internal disks, small-to-medium sized clusters, used in scientific research and in business data infrastructures to support large data bases, have been favoring the shared disk (SD) approach.

In DD architectures compute nodes perform data movement to and from I/O nodes either across inexpensive Ethernet interconnects, or via more expensive specialized ones such as Myrinet, SCI, or Infiniband. In SD architectures, both nodes and storage devices are attached to a storage area network (SAN), an infrastructure that commonly uses Fibre Channel (FC). The cost per node for both types of infrastructures, FC and Myrinet or Infiniband, is similar.

This diversity among architectures has obviously spurred different file system approaches: distributed disk architectures are usually handled with a distributed file system (DFS), while shared disk architectures resort to cluster file systems (CFS). Some file systems are specifically designed to cater for HPC needs, and these usually have the word "parallel" standing out in their names, e.g., *Parallel Virtual File System* (PVFS) and *General Parallel File System* (GPFS).

This paper proposes pCFS, a shared disk cluster file system which aims to achieve high performance in a broad spectrum of I/O intensive applications ranging from computational access to large data sets to video streaming and databases, including efficient support of parallel I/O.

pCFS is targeted at small to medium sized clusters where data is stored in shared devices on a SAN. It merges concepts and techniques that were successful both in established CFSs and DFSs,

*CITI is a research centre funded by the Foundation of Science and Technology of the Portuguese Ministry of Science and Universities. This research was also supported by an IBM Equinox grant.

but goes beyond current practice in the following aspects:

- while current CFSs use SANs to access storage devices and LANs for the exchange of control information, pCFS performs data access using both;
- while techniques such as cooperative caching were previously used in DFSs only to increase the size of a "global cache", pCFS again goes beyond its original intent, and uses it to achieve several goals simultaneously: to decrease latency, minimize data movement to and from disk devices, and increase fault tolerance.

The rest of the paper is organized as follows. Section 2 is an overview of some well known distributed and cluster file systems, and an assessment of their shortcomings. The proposal of pCFS, a new architecture that will overcome the observed limitations, and its distinctive features against both established and state-of-the-art file systems for clusters, is described in Section 3. Section 4 reports on the proof-of-concept tests carried out, and Section 5 concludes the paper with remarks on main contributions of pCFS and future work.

2. File Systems for Cluster Architectures

The implementation of a DFS or a CFS is a careful decision placement along four dimensions: a) richness and adequacy of the file model to its target architectures and applications; b) performance c) resilience, fault-tolerance and recoverability; and d) security.

2.1. Representative File Systems

PVFS [3] is well known to HPC users; it ² uses a client/server approach, with computational nodes accessing both I/O server as well as metadata nodes through a TCP/IP network; inexpensive implementations range from Fast through Gigabit Ethernet, whereas more expensive ones use low latency, high bandwidth networks such as Myrinet or Infiniband. A PVFS filesystem is created on top of a logical group of Linux ext2 filesystems, each one stored in a local disk of a distinct I/O server; files are then round-robin striped across these *PVFS disk units*. PVFS is integrated with the VFS interface, so existing binary applications using the standard file I/O API (memory mapped files are not supported) can be executed; it also has its own API, which implements the PVFS "native" I/O model, and includes operations such as collective I/O. A point worth mentioning is that PVFS does not use client caching at all, at the expense of a decrease in performance; but, then, it is able to "almost" ³ offer POSIX single node semantics without requiring the complexity of dealing with cache coherency.

On the other hand, file systems such as GFS [12] or GPFS [10] used in shared disk clusters use the SAN to transfer data, and the host interconnection network to transfer control information (e.g., locking). They are usually symmetric, ⁴ and are referred to as Cluster File Systems (CFS). GPFS is an IBM-proprietary shared disk file system that runs on AIX Power and on IBM supplied Linux clusters connected to an FC SAN. In a GPFS cluster, non SAN-attached nodes can still access shared data through a software layer, called Virtual Shared Disks (VSD), running on top of a general purpose network infrastructure. GPFS supports two forms of caching: client-side (CS), the standard operation mode where POSIX single-node equivalent semantics is supported, and server-side (also called data-shipping, DS). DS mode is used to boost performance in "heavy sharing" situations, but

²For the purpose of this discussion the new version, PVFS2 [4], is not different from the previous one, PVFS.

³For a brief introduction, please read the PVFS2 User's Guide, available on <http://www.pvfs.org/pvfs2>

⁴Every node has direct access to every storage device.

has several restrictions: it requires program modification, where processes desiring to use it issue a "DS start" call and block other processes from accessing the file; it does not preserve POSIX single-node equivalent semantics; and it cannot be used together with some other file system calls.

GFS has been recently made into an open source CFS; a very short description of its features, without repeating ourselves, would be to say that "it's GPFS minus the VSD and the DS mode".

2.2. Shortcomings of current HPC-targeted File Systems

How may these distributed file systems we have just presented, be positioned along the four dimensions listed in 2.1, thus showing their strengths as well as limitations? PVFS main functional shortcoming is the lack of support for file locking, whereas its main operational drawback is fault-tolerance: if an I/O server fails, its local disks will be unavailable, and all the PVFS file systems depending on it will fail. Possible solutions are: to use a node to store a replica of data stored in another node (e.g., using RAID "over-the-LAN"); or to dispense with local disks altogether and instead use a SAN, with disk arrays whose LUNs are then privately mounted by the hosts. Both solutions, however, do incur in performance penalties: the software solution, because there is some amount (twice if mirroring is adopted) of overhead data being transferred to the "mirroring host" over the network; the hardware RAID solution because, while it introduces another network, the SAN (which is expensive), it does not take full advantage of it (client nodes cannot access the disks directly using the SAN) and, worst of all, performance can be degraded due to the bottleneck that may arise if several I/O servers simultaneously access their disks to get hold of data striped on them.

GPFS, on the other hand, scores highly on all dimensions. The only aspects that we will point out are: it is not an open source product; it does not support POSIX equivalent single-node semantics in data shipment mode; and, more importantly, it does not make the "best" use of all the available infrastructures.

There is, however, a very important issue that has been somehow neglected when championing the "pure" DFS approach (as used by PVFS, NFS, or Lustre [1]): CPU power available for the application. It has been reported [2] that keeping a gigabit Ethernet interface running close to its full bandwidth consumes quite a lot of CPU: we have measured close to 40% of a 2.6 GHz Xeon with regular-sized, and about 30% with Jumbo frames. Thus, in small to mid-sized HPC clusters with DFSs such as PVFS or Lustre, this may be an important issue. Sure, an "expensive" CPU offloading interface (e.g., Myrinet) can be added, but then the price advantage over SAN-based cluster file systems is lost. This is quite different from what happens in configurations with CFSs, say, GPFS or GFS, where all nodes may be compute nodes and file system clients simultaneously, and where the FC boards do offload the CPU.

3. A new, multi-purpose, Cluster File System

We will now present the driving ideas behind pCFS, including targeted environments, architecture, major building blocks and distinctive features.

3.1. Targeted hardware architectures

pCFS is targeted to shared-disk clusters; it is being designed to extract very good performance and scalability on small to medium-sized (less than a hundred nodes) clusters, where nodes are SAN-connected to several storage disk arrays. As pointed out in subsection 2.2, we have three strong arguments in favor of using a CFS on mid-sized SAN-based cluster solutions:

1. all nodes are available for running applications;

2. I/O tasks running on the nodes do not cause excessive CPU load; and,
3. the architecture is inherently fault tolerant.

3.2. Software architecture

The overall software architecture of pCFS and its relationship to other relevant kernel modules is depicted in Fig. 1 below.

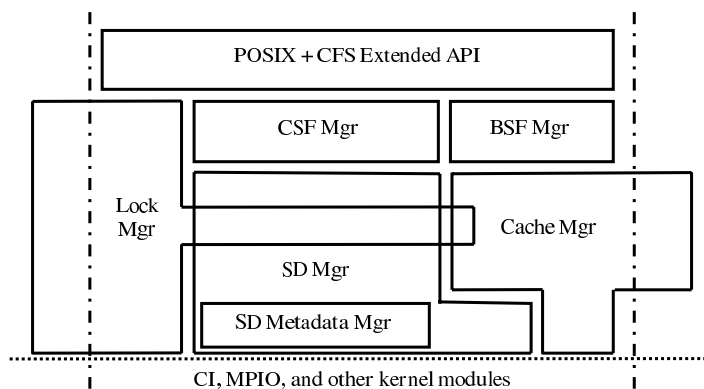


Figure 1. pCFS software architecture. pCFS modules are depicted above the dotted line and inside the "dash-dot" lines; modules, such as Lock and Cache Manager, that extend to the outside of "dash-dot" boundary lines represent interactions with other OS modules and subsystems; those drawn below the dotted line represent subsystems that provide services to pCFS.

The *pCFS API* can be thought as the union of three sets of primitives: standard POSIX I/O primitives, including advisory locking support; lock primitives to implement other locking semantics, such as "Linux-style" mandatory locking; and finally a set of primitives to support parallel, strided, and collective I/O operations.

The *Shared Data (SD) module* is the one that ultimately performs block read/write operations to the "physical" disks, while the SD Metadata Manager sub-module deals with the "on disk" data structures: bitmaps, superblocks, inodes, etc., or their equivalents in the pCFS world.

The *Cache module* keeps recently accessed data blocks (or pages, depending on the grain size being used) in memory; distinct cache coherency policies may be used across distinct files according to user-defined parameters; cache coherency is implemented with the help of services provided by a lock manager. Cooperative caching [5] is a technique that has been used to extend the cache of a node with the help of memory from other nodes; if a node knows that some other, "buddy" node, has the desired data block on its cache, it can fetch the data from that node's memory, without the need of a disk access. Cooperative caching has been quite used in DFSs, but not in CFSs; in pCFS it has a double function:

- On reads, it may be used as a regular cooperative cache; but while in a DFS reading from another node memory is the only way to read from its local disk, in our shared disk architecture it is simply another path available to be used on reads, one that can be regarded as a powerful performance enhancing mechanism, as it enables both infrastructures, SAN and LAN, to be used for data block movement.

- On writes, the traditional cooperative cache can be extended with a form of replication to increase file system availability: instead of writing through to a disk, we can replicate the block to some other nodes and, for practical purposes, guarantee that we can flush the updated block on a node failure; or, we can take the performance enhancing route: when a node wants a modified block cached in another, we may simply forward it, thus avoiding the need to flush it to disk and re-read it from the disk in the other node.

We will need to keep track of cached blocks, and a directory-based protocol [6] or other technique usable for software coherence protocols [8] is the way we plan to do it. In brief, what we are proposing is a global unified cache across the cluster nodes.

The *Lock module* is a building block for the implementation of both user-level locking primitives (POSIX and others, as needed) and file sharing semantics on cluster-wide open files. Thus, the Lock module will be a client of some Locking Subsystem, and will have to interact with the Linux cache and VFS. Each Locking subsystem is a distinct implementation of a "lock server" with a minimum set of operations capable to support the needs of the Locking Module; this concept, taken from the GFS "Lock Harness" module, allows different implementations, corresponding to different levels of complexity to be used: from a simple, centralized, locking server, through a high available one, up to a Distributed Lock Manager [7] implementation.

Finally, the *Character (CSF) and Block Special-File (BFS) Managers* are the modules that provide standard character-device and block-device access to applications.

3.3. Services provided to the pCFS subsystem

The pCFS subsystem needs some services provided by other subsystems; among them are the Cluster Infrastructure (CI), Multi-path I/O (MPIO), and the I/O drivers.

Cluster Infrastructure provides Cluster Membership, a Services Database, and Reliable Communications across cluster nodes. This subsystem closely follows the ideas laid out on [11].

The Multi-path I/O module has a sub-module for each class of I/O protocol available on an interconnect; for example, every TCP/IP running device, be it an Ethernet, Myrinet, or other device is kept under a sub-module that allows that device to be used together with other devices to provide for a sort of "link aggregation" feature that offers a wider bandwidth and higher availability path.

3.4. Distinctive features of pCFS

When comparing a full pCFS implementation with parallel distributed file systems such as PVFS, we believe the following advantages will show up:

- a) existing applications that need locking will run unmodified on pCFS, while they must be modified to run on PVFS;
- b) application performance in pCFS will be less dependent on data partitioning and placement decisions (although achieving the highest level of performance still requires "optimal" striping)
- c) provisions will exist on pCFS to establish QoS contracts for sustained I/O bandwidths;
- d) due to its cooperative cache replication/update strategy both write and read performance can be significantly larger in pCFS;
- e) maximum I/O bandwidth can be achieved in pCFS by using the whole I/O infrastructure, including both the SAN and the LAN; and,

- f) there will be no single point of failure in pCFS, whereas in PVFS, if a node is down there is no way to access its locally stored data ⁵, and a whole file system instance will be unavailable.

Comparing the pCFS with the recently proposed Lustre file system is not an easy task; Lustre [1] is a very ambitious file system: everything is apparently covered, from security to high availability, from scalability to integration of "legacy" file systems (such as Linux ext2, ext3, ReiserFS, etc.) with new Object Based Storage proposals, to ease of recovering after crash, etc. Unfortunately, we could find only one published report [9] and it shows Lustre performance to be similar to PVFS, and somewhat lower than GPFS. CFS Inc., is publicly releasing old versions only; the one currently available, although covering a lot of the expected functionality, seems ⁶ to have several annoying bugs that were fixed on releases not publicly available, and to be very difficult to install [9].

When comparing the pCFS with a cluster file system such as GPFS, we believe that the advantage expressed in e) above also applies here, while the one in d) is not quite the same: in this case, the cooperative cache will enable pCFS to achieve good performance on applications that heavily write-share the same file block across several nodes (in GPFS this is done by modifying the application to use "data-shipping"), while still preserving POSIX single-node equivalent semantics and allowing any process to access the file using the standard API (which GPFS does not).

4. pCFS proof-of-concept tests

To validate the fundamental assumptions, we decided to make small modifications to GFS, a well established, production-level CFS, that closely implements the same architectural ideas that stemmed from the VAX Clusters research and products. After carefully evaluating Oracle's OCFS [13] and openGFS [14] (seemingly phased out when GFS moved to "open source" status), both documented, we ended up studying thousands of lines of GFS code (as documentation is not available) and we have opted for carrying out the tests as a simulation inside GFS kernel code.

We have modified GFS to follow one out of two different code paths when reading a file:

SAN path: When a process in one node is reading a file that is opened across the cluster, the regular GFS code path is followed: shared read locks are placed on the disk blocks, and, if another node has modified one or more file blocks, the node has to flush them out to disk before granting the lock.

LAN path: When directed to do so by the simulation test, the kernel code on the reader node follows another code path, where a) lock requests and grants are simulated by message exchange between the nodes, and b) the writer node supplies a copy of the modified page(s).

To implement the LAN path we have built two kernel modules: the client module is called by the modified GFS code when a decision has been made to get data directly from another node; it forwards the request to the other node, where the server module handles it by shipping back the data. The performance data was collected when running a single writer/multiple readers application which access the same data blocks; after producing new data, the writer signals the readers to consume it. Tests were carried out using IBM x335 dual Xeon nodes attached at 1 Gbps to a Brocade SilkWorm 2800 switch and IBM FAStT 200 storage array.

The single writer/multiple readers scenario was chosen to expose the known limitations of cluster file systems in heavy sharing situations. Figures 2 to 4 summarize the experiment results and validate the benefits of using cooperative caching as proposed by pCFS.

⁵Or we must resort to one of the solutions proposed in subsection 2.2

⁶We accessed the Lustre discussion forum on <https://lists.clusterfs.com>

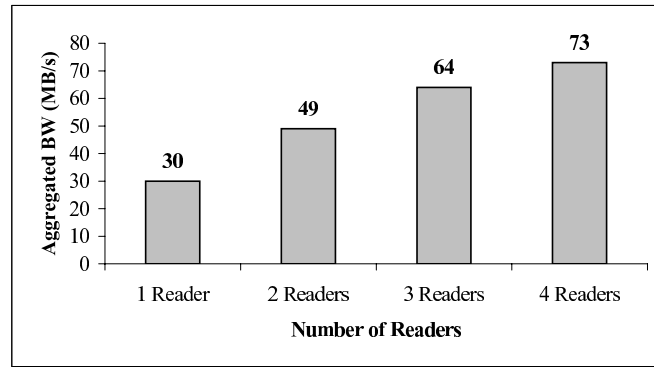


Figure 2. Aggregated application disk bandwidth in non-modified GFS. Bandwidth increases with the number of readers (one per node). Block size is 4 Kbytes.

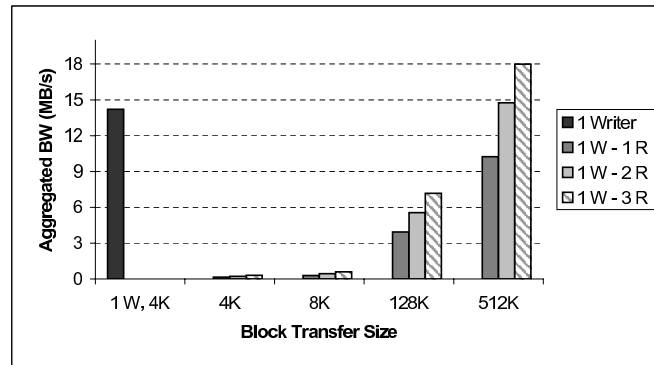


Figure 3. Single writer/multiple readers aggregated bandwidth in non-modified GFS. Block size ranges from 4 K to 512 Kbytes. Note the bandwidth drop when adding the 1st concurrent reader.

5. Conclusions

Through detailed analysis and benchmarking, we have identified shortcomings of both distributed disk and shared disk file systems. File systems for distributed disk architectures based on "plain Ethernet" suffer from excessive CPU consumption on data movement tasks, and are usually non fault tolerant. Those for shared disk architectures are inherently fault tolerant, but do not use all the available I/O infrastructures; they also do not, as a rule, perform well under heavy sharing, and may not scale well in configurations with a large number of nodes.

We believe the proposed cluster file system architecture, by combining current "top of the breed" hardware building blocks (FC SANs and disk arrays, together with Gigabit Ethernet, Myrinet and Infiniband) with a sophisticated software architecture that includes cooperative caches with replication of modified data blocks, will be able to maximize I/O bandwidth and minimize latency and, when compared with other file systems for distributed and shared disk architectures, offer: better performance and application compatibility (via its standard POSIX I/O API); resilience, fault tol-

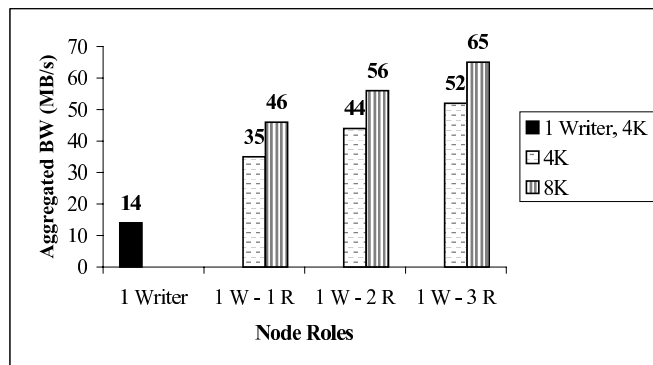


Figure 4. Same situation as figure 3, but with GFS modifications. Note that the bandwidth is much higher than in figure 3, and close to the "readers only" experiment of figure 2.

erance and recoverability; and higher node counts than currently available CFSs. We also believe this architecture is well suited to be integrated with a SSI kernel such as Kerrighed [15], and to vastly benefit from RDMA capable interconnects. The architecture and performance of pCFS will be evaluated in a future prototype implementation.

References

- [1] Braam, P. et al, The Lustre Storage Architecture, 2003.
- [2] Celebioglu, O. et al, Optimizing Linux Cluster Performance by Exploring the Correlation between Application Characteristics and Gigabit Ethernet Device Parameters, Proceedings of the 5th Linux Clusters Institute (LCI) Conference, 2004.
- [3] Carns, P. et al: PVFS: A Parallel File System for Linux Cluster, Proceedings of the 4th Annual Linux Showcase and Conference, 2000
- [4] P. Carns, Achieving Scalability in Parallel File Systems, PhD Thesis Clemson University, 2004.
- [5] Dahlin, M. et al, Cooperative Caching: Using Remote Client Memory to Improve File System Performance. Proceedings of the First Symposium on Operating System Design and Implementation, 1994
- [6] IEEE Standard No.: 1596, Scalable Coherent Interface, 2000
- [7] Kronenberg, N. et al. VAXclusters: A Closely-Coupled Distributed System. ACM Transactions on Computer Systems, Vol. 4, No. 2, 1986
- [8] Li, K. and Hudak, P., Memory Coherence in Shared Virtual Memory Systems. ACM Transactions on Computer Systems, 7(4):321-359, 1989
- [9] Margo M. et al, An Analysis of State-of-the-Art Parallel File System for Linux, 5th LCI (Linux Clusters Institute) Conference, 2004
- [10] Schmuck, F. and Haskin, R., GPFS: A Shared-Disk File System for Large Computing Clusters, Proceedings of the Conference on File and Storage Technologies (FAST'02), 2002
- [11] Cluster Infrastructure for Linux. <http://ci-linux.sourceforge.net>
- [12] GFS Project Page, <http://sources.redhat.com/cluster/gfs/>
- [13] OCFS Project Page, <http://oss.oracle.com/projects/ocfs/>
- [14] openGFS Project Page. <http://opengfs.sourceforge.net/>
- [15] Vallé, G. et al. A Case for Single System Image Cluster Operating Systems: Kerrighed Approach. Parallel Processing Letters, 13(2), 2003.