Experiences on File Systems: Which is the best file system for you?

J Blomer
CERN, CH-1211 Genève 23, Switzerland
E-mail: jblomer@cern.ch

Abstract. The distributed file system landscape is scattered. Besides a plethora of research file systems, there is also a large number of production grade file systems with various strengths and weaknesses. The file system, as an abstraction of permanent storage, is appealing because it provides application portability and integration with legacy and third-party applications, including UNIX utilities. On the other hand, the general and simple file system interface makes it notoriously difficult for a distributed file system to perform well under a variety of different workloads. This contribution provides a taxonomy of commonly used distributed file systems and points out areas of research and development that are particularly important for high-energy physics.

1. Introduction
In high-energy physics, we store files in a variety of distributed file systems. The first and foremost advantage of using a file system is application portability. Distributed file systems resemble the API of local file systems. A locally developed application that has been tested on a small local data set, for instance, can be readily integrated in a distributed computing workflow where it can work on a large data set provided by a distributed file system. Secondly, as a storage abstraction, file systems are a sweet spot. They provide some data organization due to the hierarchical namespace but no schema is imposed on the contents of the files.

There is a large number of publicly available, production grade distributed file systems. Nevertheless, no single file systems seems to be a universally applicable silver bullet. Most distributed file system implementations are optimized for a particular class of applications. For instance, XrootD is optimized for high-throughput access to high-energy physics data sets [1], the Hadoop File System (HDFS) is designed as a storage layer for the MapReduce framework [2,3], the CernVM File System (CVMFS) is optimized to distribute software binaries [4], and Lustre is optimized as a scratch space for cooperating applications on supercomputers [5]. These use cases differ both quantitatively and qualitatively. As an example, table 1 compares some of the characteristics of LHC application software files and LHC data files. Fundamental differences in average file size, (meta-)data access pattern, and confidentiality result in fundamentally different designs of file systems optimized for one use case or another.

There are several publications on taxonomies, case studies, and performance comparisons of distributed file systems [6–12]. This contribution classifies distributed file systems by use case and points out areas of research and development that are particularly important to high energy physics and that are not necessarily well-covered by off-the-shelf distributed file systems.
Table 1. Comparison of some of the characteristics of LHC experiment software and LHC experiment data.

| Software          | Data                      |
|-------------------|---------------------------|
| $10^8$ objects    | $10^9$ objects            |
| consistent namespace | independent files        |
| tens of terabytes | hundreds of petabytes     |
| whole file access | partial file access       |
| absolute paths    | arbitrary path            |
| open access       | confidential              |

2. File System Interface
We usually associate the POSIX file system interface with a hierarchical namespace and the basic functions for creating and deleting files, retrieving meta-data (e.g. file size), opening and closing files, sequential reading and writing, and directly accessing an arbitrary file position. Designed for local file systems, for better or worse the POSIX interface provides a much richer interface, including file locking, extended attributes, hard links, device files and much more. It is often difficult or impossible (sometimes also pointless) to resemble this functionality in a distributed context. Since the level of POSIX compliance differs for different file systems, high-energy physics applications should be tested and validated when a critical file system is exchanged. A typical pitfall is the use of SQLite databases on a file system with no proper support for file locks. But deviations from the standard can be more subtle. HDFS, for instance, writes file sizes asynchronously for performance reasons and thus it returns the real size of a file only some time after the file has been written.

On the other hand, information about physical file location, file replication status, or file temperature are not provided by the file system interface. Such information would be very useful in a distributed context. Collecting and exploiting this information is one of the main purposes of a distributed data management, a layer on top of pure file systems [13].

2.1. Connecting Applications to File Systems
File systems are traditionally belong to the core business of an operating system kernel. Indeed some distributed file systems are implemented as an extension of the operating system kernel (e.g. NFS [14], AFS, Lustre). That allows for best performance but the deployment is difficult and implementation errors typically crash the operating system kernel.

Equally good performance plus the ability to provide non-standard features can be provided by user space libraries that directly connect applications to a file system. HDFS or XRootD are examples of file systems that implement this approach. The library typically closely resembles a POSIX file system but it is in certain aspects tailored to a specific use case. For instance, HDFS extends POSIX semantics by an atomic append [15], a feature particularly useful for the merging phase of MapReduce jobs. A user-space library is also helpful to use a particular file system on resources where mounting a custom file system cannot be enforced.

On the other hand, a library interface comes at the cost of transparency; applications are developed and compiled for a particular distributed file system. It often makes application integration clumsy. For instance, we cannot seamlessly use the well-known UNIX tools (grep, sort, awk, ...) in our scientific workflows but files have to be staged in and out.

To overcome these limitations, so called interposition systems introduce a layer of indirection that transparently redirects file system calls of an application to a library or an external process. The Parrot system creates a sandbox around a user-space process and intercepts its system calls [16]. The Fuse kernel-level file system redirects file system calls to a special user-land process.
### Table 2. File Systems by Use Case. Entries marked † are widely deployed in HEP.

| Field of Use     | File Systems                  | Important Characteristics                                                                 |
|------------------|-------------------------------|-------------------------------------------------------------------------------------------|
| Big Data         | HDFS †[3] QFS †[18] MapR FS  | Integration with MapReduce workflows, fault-tolerance on commodity hardware               |
| Supercomputers   | Lustre †[5] GPFS †[19] OrangeFS [20] BeeGFS Panasas [21] | Parallel write performance, support for high-performance interconnects such as Infiniband |
| Shared Disk      | GFS2 †[22] OCFS2 †[23]       | High level of POSIX compliance                                                            |
| General Purpose  | (p)NFS †† Ceph †[24] Gluster-FS †[25] XtreemFS [26] MooseFS | Incremental scalability, ease of administration                                           |
| Personal Files   | AFS †[27] Tahoe-LAFS [28] owncloud/dropbox | Privacy, sharing, synchronization                                                         |
| HEP              | dCache †[29] XRootD †[1] CernVM-FS †[4] EOS †[30] | Tape access, name space federation, software distribution, fault-tolerance on commodity hardware |

(“upcall”) [17]. Ideally, interposition systems would give the application full control over the mounted file system tree while introducing only negligible performance overhead. Our community would benefit from more research and development in this area, given our world-wide distributed, heterogenous resources (grid, cloud, supercomputers) that are often not under control of the application owners.

### 3. File System Taxonomy

In a first attempt to organize the space of file systems, we classify them by the field of use for which they are optimized. Table 2 provides a classification for commonly used file systems. We routinely use many of these file systems in our data centers. In some areas, however, we need customized file system developments. HEP has challenging demands in handling a complex storage hierarchy including tapes, we need to provide access to complex software stacks on globally distributed resources, and we manage large data volume of hundreds of petabytes in a geographically distributed manner.

Often we can derive file system characteristics from its architecture. Modern file systems implement an object-based architecture, which separates data management and meta-data management. Files are spread over a number of servers that handle read and write operations. A meta-data server maintains the directory tree and takes care of data placement. As long as meta-data load is much smaller than data operations (i.e. files are large), this architecture allows for incremental scaling. As the load increases, data servers can be added one by one with minimal administrative overhead. This is a typical architecture for MapReduce environments. Due to
its global view on the name space, the meta-data server can help in an optimized scheduling
of jobs. The commonly used Hadoop implementation of MapReduce unfortunately does not
perfectly match high-energy physics workflows, for instance it lacks integration with the ROOT
I/O. A lesson to learn, however, is that it is beneficial to co-design a file system with a particular
computing workflow.

For supercomputer applications, the object-based architecture is typically refined to a parallel
file system (e.g. Lustre) that cuts every file in small blocks and distributes the blocks over many
nodes. Thus read and write operations are executed in parallel on multiple servers for better
maximum throughput. This is important when it comes to application checkpointing and many
nodes write out state information at the same time.

As the number of nodes that use a distributed file system quickly increased to tens of thousands
and more, file system designers discovered that a central meta-data handling becomes a bottleneck.
To improve the situation, a file system with distributed meta-data can be used. Distributed
meta-data handling is more complex than distributed data handling because the meta-data servers
are closely coupled in order to maintain a consistent state of the name space. The Ceph file system
implements such an architecture. After many years of development Ceph is becoming a production
grade option that most likely will work well for a large number of use cases. Multi-Petabyte
installation of Ceph’s underlying object storage layer are deployed today in multiple HEP data
centers.

Another approach to solve the meta-data bottleneck is decentralized meta-data handling.
Instead of asking a meta-data server, decentralized or peer-to-peer file systems (e.g. GlusterFS [25])
let clients compute the location of data and meta-data by means of a distributed hash table.
Zero-hop routing in a distributed hash table is restricted to a local network, however, in which
every node is aware of every other node. On a global scale, tree based routing as being done by
XRootD is simpler to implement and it shows better lookup performance than globally distributed
hash tables. Furthermore, high peer churn (servers that frequently join and leave the network)
pose a hard challenge on distributed hash tables.

4. Technology Trends and Future Challenges
Most of our future challenges are on improving the infrastructure, data management software,
and computing workflows in order to process data sets at the exascale. Capacity-wise, current
projections of growth rates for hard disks and tapes are promising. The recently introduced
Shingled Magnetic Recording (SMR) technology, for instance, allows for 8 TB hard drives today,
and 20 TB hard drives seem to be in reach in a few years from now.

At the same time, however, the storage throughput does not nearly scale at the same pace.
Table 3 compares the development of throughput and capacity of several storage technologies
in the last 20 years. For comparison the table also shows the development of the bandwidth
of Ethernet networks, which scaled at a similar pace than the capacity of hard drives. This
development suggests a storage architecture that avoids a segregation of compute nodes and
storage nodes. For a maximum number of concurrently accessible storage devices, every compute
node could be part of the storage network, exploiting the high bisection bandwidth among
compute nodes.

To some extent, the limited throughput of hard disks can be mitigated by flash based SSDs.
Due to the high price of SSD storage of the order of one dollar per gigabyte (hard disks: <5
cent per gigabyte), their use should be well-considered. File system meta-data are a particular
appealing use case because not only the higher throughput but also the good random access speed
of SSDs can be exploited. Otherwise, SSDs are likely to become a part of the storage hierarchy
between DRAM and hard disks.

At the software level of storage space management, log-structured storage systems provide
an interesting approach that can help to utilize the available bandwidth in an optimal way.
Table 3. Development of capacity and bandwidth in the last 20 years. Method and entries marked † from Patterson [31]. Other numbers refer to DDR2-400 DRAM (2004) and DDR4-3200 DRAM (2014) and the 100 GbitE IEEE 802.3bj standard (2014).

| Year | Capacity | Throughput (r/w) | Capacity | Throughput | Bandwidth |
|------|----------|------------------|----------|------------|-----------|
| 1993 | 16 Mibit/chip† | 267 MiB/s† | | | |
| 1994 | 4.3 GB† | 9 MB/s† | | | |
| 1995 | 100 Mbit/s† | | | | |
| 2003 | 73.4 GB† | 86 MB/s† | | | |
| 2004 | 512 Mibit/chip | 3.2 GiB/s | | | |
| 2008 | 160 GB | 250/100 MB/s³ | | | |
| 2012 | 6 TB | 220 MB/s³ | 800 GB | 500/460 MB/s⁴ | |
| 2014 | 2 TB | 2.8/1.9 GB/s² | | | |
| 2015 | 8 TB | 190 MB/s² | 8 Gbit/chip | 25.6 GiB/s | 100 Gbit/s |
| Improvement | ×1860 | ×24 | ×12.5 | ×11.2 | ×512 | ×98 | ×1000 |

¹Seagate ST6000NM0034, http://www.storagereview.com/seagate_enterprise_capacity_6tb_35_sas_hdd_review_v4
²Seagate ST8000AS0012 (SMR), http://www.storagereview.com/seagate_archive_hdd_review_8tb
³Intel X25-M, http://www.storagereview.com/intel_x25-m_ssd_review
⁴Intel SSD DC S3700, http://www.storagereview.com/intel_ssd_dc_s3700_series_enterprise_ssd_review
⁵Intel SSD DC P3700, http://www.tomshardware.com/reviews/intel-ssd-dc-p3700-nvme,3858.html

Log-structured systems provide near-optimal write performance for small and large files because all changes, new data and meta-data are appended to the physical medium [32, 33]. While log-structured data organization is popular in distributed key-value stores and object stores, it is only occasionally used in distributed file systems, possibly because most of them assume large sequential writes. In high-energy physics, this assumption is often valid but not always. Counter examples are the many small files of software and source code in CernVM-FS, or the merging phase of a parallelized physics analysis or simulation with many small distributed parts of the final result. Moreover, log-structured data organization is an efficient means to manage storage space throughout the memory hierarchy from DRAM to flash memory to hard disks [34].

4.1. Fault-Tolerance

Fault-tolerance is an important property for a file system in high-energy physics because of the large scale of our storage systems and because mostly commodity hardware is used. Thus, hardware failures are not only the norm but they also tend to occur in a correlated manner. Power cuts are an example, or a failing controller with many connected drives.

Replication and erasure codes are the techniques used to avoid data loss and to continue operation in case of hardware failures. A critical engineering challenge when building fault-tolerant storage is the placement of redundant data in such a way that the redundancy crosses multiple failure domains. The Ceph file system, for instance, can parse the physical layout of a data center together with policies to distribute redundant data on multiple racks or disk shelves. Another option is to keep a copy of all data at a remote data center. The second engineering challenge is to decide when to start a recovery, i.e. to distinguish between short-term glitches and permanent faults. Tight monitoring can help to distinguish between the two in some cases, but in general the only available technique are heartbeats between servers and properly tuned time-out values [35].

While replication is simple and fast, it also results in a large storage overhead. Most systems use a replication factor of three. Erasure codes, on the other hand, can be seen as more sophisticated, distributed RAID systems. Every file is chunked and additional redundancy blocks are computed. Typically the storage overhead of erasure codes is much smaller than a factor of two. Erasure codes, however, come at the cost of computational complexity. Modifications to a file as well as
recovery from hardware faults is expensive because the redundancy blocks have to be recalculated and redistributed. Exploration of the trade-off between computational complexity and storage overhead in erasure codes are an active research area. Today’s systems typically deploy extra computing nodes exclusively for the (re-)computation of redundancy blocks.

Storage systems based on erasure codes are commercially available (e.g. GPFS, NEC Hydrastor, Scality RING). Proprietary storage systems based on erasure coding are also commonly used at large cloud storage providers [36,37]. The latest developments in this direction in open source file systems [18,24,30], including efforts in high-energy physics, are very promising.

5. Conclusion
Distributed file systems are at the heart of our distributed computing. As such, they should be co-designed with our distributed computing workflows. This is a lesson that can be learned from the Google file system and MapReduce.

Off the shelf distributed file systems cannot always fulfill our demands. Luckily, in the last years many very powerful open source tools appeared, most importantly in the area of object stores and key-value stores, that can serve as building blocks for file systems tailored to the needs of high-energy physics. We should put particular focus on wide area data federation, multi-level storage hierarchies from DRAM to tape, erasure coding, and coalescing of storage nodes and compute nodes.

References
[1] Dorigo A, Elmer P, Furano F and Hanushevsky A 2005 WSEAS Transactions on Computers 4 348–353
[2] Dean J and Ghemawat S 2008 Communications of the ACM 51 107–114
[3] Shvachko K, Kuang H, Radia S and Chansler R 2010 Proc. of the 26th IEEE Symposium on Mass Storage and Technologies (MSST’10) pp 1–10
[4] Blomer J, Aguado-Sanchez C, Buncic P and Harutyunyan A 2011 Journal of Physics: Conference Series 331
[5] Schwan P 2003 Proc. of the 2003 Linux Symposium pp 380–386
[6] Satyanarayanan M 1990 Annual Review of Computer Science 4 73–104
[7] Guan P, Kuhl M, Li Z and Liu X 2000 A survey of distributed file systems University of California, San Diego
[8] Agarwal P and Li H C 2003 A survey of secure, fault-tolerant distributed file systems http://www.cs.utexas.edu/users/browne/cs395f2003/projects/LiAgarwalReport.pdf
[9] Thanh T D, Mohan S, Choi E, Kim S and Kim P 2008 Proc. int. conf. on Networked Computing and Advanced Information Management (NCM’08) pp 144 – 149
[10] Peddamors A, Kuun C, Spoor R, Dekkers P and den Besten C 2010 Survey of technologies for wide area distributed storage Tech. rep. SURFnet
[11] Depardon B, Séguin C and Mahec G L 2013 Analysis of six distributed file systems Tech. Rep. hal-00789086 Université de Picardie Jules Verne
[12] Donvito G, Marzulli G and Diacono D 2014 Journal of Physics: Conference Series 513
[13] Girone M Distributed data management and distributed file systems Talk at this conference
[14] Sandberg R, Goldberg D, Kleinman S, Walsh D and Lyon B 1985 Proc. of the Summer USENIX conference pp 119–130
[15] Gheesling S, Gobioff H and Leung S T 2003 ACM SIGOPS Operating Systems Review 37 29–43
[16] Thain D and Livny M 2005 Scalable Computing: Practice and Experience 6 9
[17] Henk C and Szeredi M Filesystem in Userspace (FUSE) http://fuse.sourceforge.net URL http://fuse.sourceforge.net/
[18] Osviannikov M, Rus S, Reeves D, Sutter P, Rao S and Kelly J 2013 Proc. of the VLDB Endowment vol 6 pp 1092 – 1101
[19] Schmuck F and Haskin R 2002 Proc. 1st USENIX conf. on File Storage and Technologies (FAST’02) pp 231–244
[20] Carns P H, III W B L, Ross R B and Thakur R 2000 Proc. 4th Annual Linux Showcase and Conference (AL$’00) pp 317–328
[21] Nagle D, Serenyi D and Matthews A 2004 Proc. of the 2004 ACM/IEEE conf. on SuperComputing (SC’04)
[22] Whitehouse S 2007 Proc. of the Ottawa Linux Symposium pp 253–261
[23] Fasheh M 2006 Proc. of the Ottawa Linux Symposium pp 289–303
[24] Weil S A 2007 Ceph: reliable, scalable, and high-performance distributed storage Ph.D. thesis University of California Santa Cruz
[25] Davies A and Orsaria A 2013 Linux Journal
[26] Hupfeld F, Cortes T, Kolbeck B, Stender J, Focht E, Hess M, Malo J, Marti J and Cesario E 2008 Concurrency and Computation: Practice and Experience 20 2049–2060
[27] Morris J H, Satyanarayanan M, Conner M H, Howard J H, Rosenthal D S H and Smith F D 1986 Communications of the ACM 29 184–201
[28] Wilcox-O’Hearn Z and Warner B 2008 Proc. 4th ACM int. workshop on Storage Security and Survivability (StorageSS’08)
[29] Fuhrmann P and Gülzow V 2006 dCache, Storage System for the Future (Springer) pp 1106–1113 (Lecture Notes in Computer Science no 4128)
[30] Peters A J and Janyst L 2011 Journal of Physics: Conference Series 331
[31] Patterson D A 2004 Communications of the ACM 47
[32] Rosenblum M and Osterhout J K 1991 ACM SIGOPS Operating Systems Review 25
[33] Hartman J H and Osterhout J K 1995 ACM Transactions on Computer Systems 13 274–310
[34] Rumble S M, Kejriwal A and Osterhout J 2014 Proc. 12th USENIX Conference on File and Storage Technologies (FAST’14)
[35] Ford D, Labell F, Popovici F I, Stockly M, Truong V A, Barroso L, Grimes C and Quinlan S 2010 Proc. 9th symposium on Operating Systems Design and Implementation (OSDI’10)
[36] Calder B, Wang J, Ongiu A, Nilakantan N, Skjolsvold A, McKelvie S, Xu Y, Srivastav S, Wu J, Simitci H, Haridas J, Uddaratu C, Khatri H, Edwards A, Bedekar V, Mainali S, Abbasi R, Agarwal A, ul Haq M F, ul Haq M I, Bhardwaj D, Dayanand S, Adusumilli A, McNett M, Sankaran S, Manivannan K and Rigas L 2011 Proc. 23rd ACM Symposium on Operating Systems Principles (SOSP’11) pp 143–157
[37] Muralidhar S, Lloyd W, Roy S, Hill C, Lin E, Liu W, Pan S, Shankar S, Sivakumar V, Tang L and Kumar S 2014 Proc. 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI’14)