XtreemFS – a case for object-based storage in Grid data management

Felix Hupfeld [1], Toni Cortes[23], Björn Kolbeck[1], Jan Stender[1], Erich Focht[4], Matthias Hess[4], Jesus Malo[2], Jonathan Marti[2], Eugenio Cesario[5]

[1]Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
[2]Barcelona Supercomputing Center (BSC), C Jordi Girona, 1-3, Barcelona, Spain
[3]Universitat Politecnica de Catalunya (UPC), C Jordi Girona, 31, Barcelona, Spain
[4]NEC HPC Europe GmbH, Hessbruehlstr. 21b, 70656 Stuttgart, Germany
[5]Institute High Performance Computing and Networks of the National Research Council of Italy (ICAR-CNR), DEIS-UNICAL, P. Bucci 41-C, 87036 Rende, CS, Italy

Abstract. In today's Grids, files are usually managed by Grid data management systems that are superimposed on existing file and storage systems. In this position paper, we analyze this predominant approach and argue that object-based file systems can be an alternative when adapted to the characteristics of a Grid environment. We describe how we are solving the challenge of extending the object-based storage architecture for the Grid in XtreemFS, an object-based file system for federated infrastructures.

1 Introduction

The file abstraction is one of the success stories of system architecture, and the current computing world is unthinkable without file systems. Files are the technology of choice for any unstructured data, and provide an efficient container for abstractions with more structure.

However, conventional network file systems are ill-adapted to Grid-like environments. These file systems are usually heavily geared toward centralized installations in a single data center and lack reliable support for remote access over wide-area networks (WANs) across multiple organizations. For Grid data management an approach was needed to compensate these weaknesses of installed local, network or distributed file systems. Instead of extending file system architectures with the necessary features, Grid data management systems are imposed on the existing file system. Remote access protocols like GridFTP [3] make files and namespaces remotely accessible, and replica catalogs index the whereabouts of a file's copies.

While this approach of superimposing Grid data management on file systems has proven to be efficient and effective it is not without drawbacks. Foremost, certain characteristics of the typical Grid data management architecture prevent these systems from performing as well as other, more integrated architectures. In addition, they can not guarantee the consistency of file content across replicas and force applications and users to adapt their usage of the system accordingly.

In this paper, we claim that an object-based file system architecture [11] can be extended to be suitable for Grid environments and argue that it is a viable alternative architecture for file data management in Grids for many use cases. To illustrate this argument, we demonstrate how we are solving some of the relevant design issues in XtreemFS, a distributed object-based file system for federated wide-area infrastructures.

We continue this paper with a detailed study of Grid data management from a system architecture perspective (Section 2) with emphasis on its structural shortcomings. In Section 3, we give an overview of an object-based storage architecture for file systems, and then describe how it can be extended for federated wide-area environments in Section 4. Section 4.1 presents the architecture of XtreemFS as an example of an object-based Grid file system. Section 5 gives further details on existing file systems and Grid data management solutions.

2 Common Characteristics of Grid Data Management

Grid data management systems provide their clients with access to file data that is stored at remote storage and file systems. They *index* files from storage resources and provide clients with a *unified interface* for accessing files. Typically the system relies on a daemon on the storage resource that mediates the remote access protocol with the heterogeneous local access interfaces.

A *replica catalog* stores the access locations for a file (sometimes called its Physical File Name) and abstracts the Grid file itself from its replicas at various storage resources. It is often integrated with a *metadata catalog*, which imposes a namespace on top of this Grid file abstraction and structures the file space for later retrieval of particular files. Common structuring methods are hierarchical name spaces (with Logical File Names, LFNs), database-like extended metadata attributes, and collections of files.

In order to access a file, the application has to download or replicate the file to its local file system first (Fig. 1). When the file is on the application's local disk, the application can access the file normally. Similarly, newly created and modified files are uploaded to one of the storage resources or the new local file is registered with the system as a replica.

While this architecture has considerably simplified access to data that is kept at heterogeneous storage resources and integrates well with existing infrastructures, its architectural properties restrict the evolution of the basic approach.

Typical Grid data management systems do not exercise control over data access beyond what is necessary for security purposes. The daemon that is running on the storage resources only mediates remote access to its files and has no notion about the system's state. It does not know where other replicas for its files exist, nor does it know in which state its files are. Also, Grid data management systems do not control client access to downloaded file copies.

Due to this lack of control and information, Grid data management systems cannot make guarantees about the consistency of a file's replicas and thus applications are generally restricted to write-once usage patterns of files. Applications

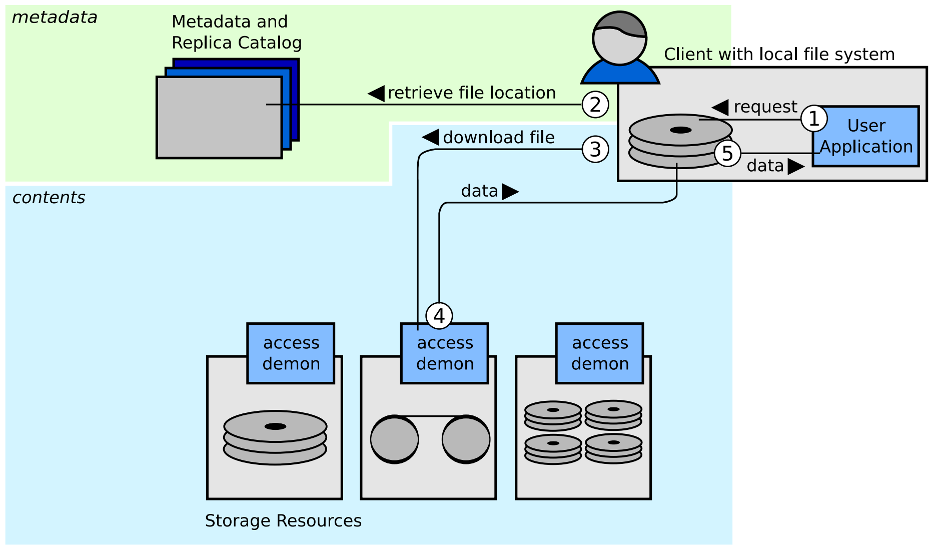

Fig. 1. Components and their relationships of a typical Grid data management system. Typically, files are fetched from a storage resource before they can be accessed.

download read-only input files, process them to generate output files and upload the latter to the data management system. In addition, storage resources are unaware of the state of their files and can not oversee the replica creation process themselves. Thus, an extra service is usually needed for the reliable creation of replicas.

The architecture also has implications for performance. Typical Grid data management systems only operate on complete files, which increases the latency to first access of the file, because the client's access to the data has to be deferred until the file is fully recreated in the local file system. The data management system can not automatically prioritize parts of the data that would be accessed first, nor can it make partial replicas that skip downloading parts which are not required.

It can also be preferable to avoid downloading any data at all. Today, access to a remote file server can be much faster than the access to the application's local hard disk. With the fast network connection commonly found in today's computing systems, the application can exploit the aggregate bandwidth of many remote disks or profit from a large file cache in the server.

3 Object-Based Storage

Recently, distributed file systems have advanced from predominantly block-based architectures to so-called object-based storage. Block-based file systems distribute file data as blocks over the network, with blocks decoupled from their association to a file. The mapping between files and blocks and the management of free storage is centralized in the file system server. With more processing power becoming available per gigabyte, file systems architects decided to raise the level of abstraction and shifted the responsibility for block management to the storage devices and address file content directly over the network as so called objects.

These object-based architectures [11][5] store the pure file content (the *objects*) on one or more object storage devices (*OSDs*, Fig. 2) that are distributed on the network. The file namespace and other POSIX [9] metadata are kept in a metadata server. After a file system client is authorized by the metadata server to access the file's objects, all file IO is done directly at one or more OSDs, thereby avoiding the potential bottleneck of the metadata server.

Structurally, the object-based architecture bears many similarities to the typical architecture of Grid data management systems at first sight. Both typically separate file metadata from file data and expose the notion of a file to their storage resources. In addition, however, object-based file systems mediate any operation of their clients, and can exercise full control over the operations where necessary. Also these file systems treat their storage resources as pure storage devices for file content, and do not offer additional functionality that could restrict the architecture.

The changes in technology have not only made more processing power available per hard disk, but also enlarged the performance gap between disk and network IO. It can be much faster to access data in a computer's memory that is located across the globe than accessing one's local hard disk. When a file's objects are striped over multiple OSDs, a client can leverage the aggregate bandwidth of all hard disks of these OSDs, and access the file at a higher speed than would be possible with a local hard disk.

Existing object-based file systems are designed as parallel file systems for clusters or enterprise file systems with centralized IT infrastructures. A typical installation is a rack of metadata servers and many OSDs with a lot of hard disks. In this homogeneous environment, all OSDs are equal from a latency and bandwidth perspective, and objects can be assigned to disks in a deterministic way. The predominant source of failure are hard disks, which is handled by introducing redundancy via RAID. Users of these file systems are part of the locally controlled administration domain.

4 Extending Object-Based Storage for the Grid

While object based-storage has all the amenities of a file system and can exploit the resources of today's hardware in an economic manner, it assumes a local homogeneous and controlled environment and does not readily fit the dynamic and heterogeneous environment of Grids.

Grid installations usually encompass multiple sites, organizations, and administration domains. It is therefore essential, that the file system is prepared for the case that parts of its installation join, leave, or fail at any time. It needs

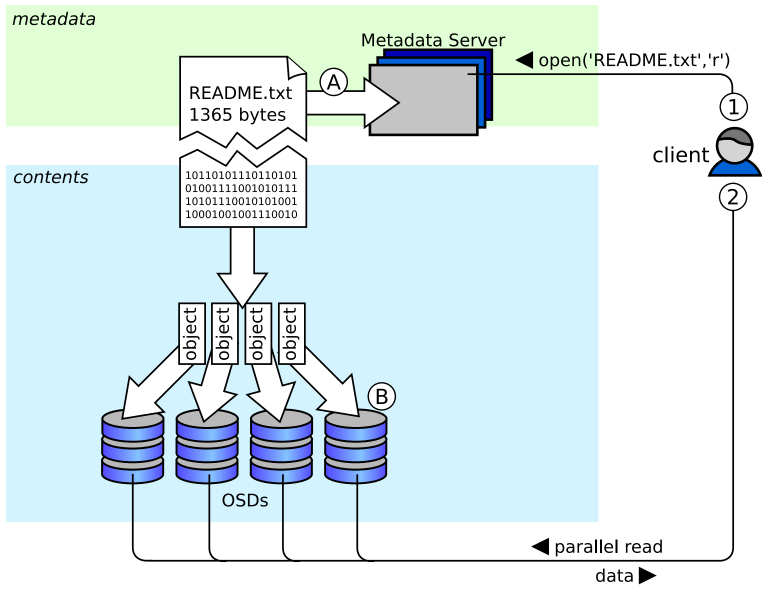

Fig. 2. Components and their relationships in an object-based file system. The metadata server authenticates and authorizes the client on open() and issues a ticket. Subsequently, all IO operations are directly performed at the object storage devices.

a *federated* structure, where no part is preferred to the others and partial absence due to failure or downtimes can be tolerated. The file system must also support authentication and authorization mechanisms of *Virtual Organization (VO) infrastructures*.

In order to ensure high availability and access performance, a file system for the Grid must also support replication of files and file metadata. Because the environment of the Grid is inherently unreliable, data must be replicated at several locations so that it is available in case of failure. Also, it is often preferable to replicate data closer to the consumers so that the file system clients can profit from shorter network latencies and higher bandwidth. The system should not place restrictions other than for security on the placement of these replicas, so that replicas can be created where and when they are most needed.

The major challenge with replicated data is to keep it consistent. When multiple replicas are changed concurrently, the system must ensure that the file replicas are consistent and that the clients see the expected semantics of a POSIX interface. These guarantees must not be weakened by any failures in the environment.

When these challenges are addressed, the users can benefit from all the advantages of a complete file system. Because all operations of an application go through the controlled file system interface, the application is decoupled from any internal aspects of the system. The file system can see and influence any operation of the application and act accordingly to provide it with the best possible performance. In turn, the application can simply mount and access the file system's data transparently.

4.1 The Architecture of XtreemFS

XtreemFS is an object-based file system that has been specifically designed for Grid environments as a part of the XtreemOS operating system. As an object-based design, it is composed of clients, OSDs and metadata servers that are also responsible for keeping replica locations (the Metadata and Replica Catalog, MRC, see Fig. 3). In addition, a directory service acts as a registry to locate servers and volumes in the system.

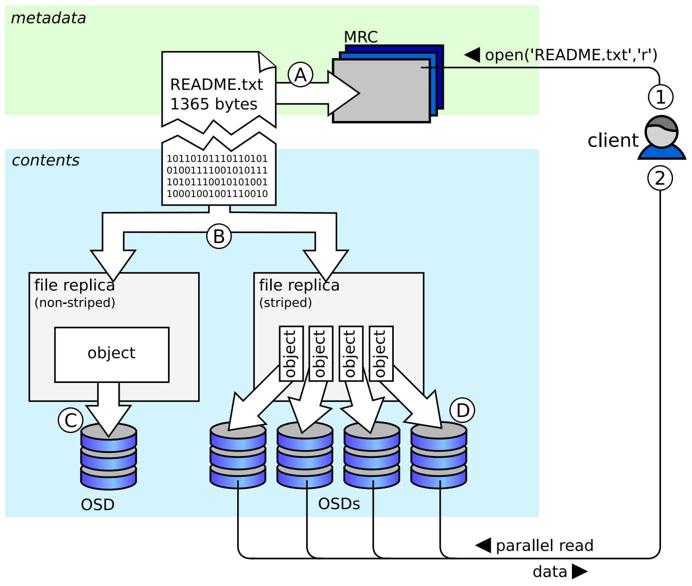

Fig. 3. Components and their relationships in XtreemFS. While supporting striping over a group of OSDs, XtreemFS allows files to be replicated to different locations.

XtreemFS manages file system *volumes* that represent mountable file systems. A volume's files and directories share certain default policies for replication and access. To ensure availability, volumes can be replicated to multiple MRCs. In order to be able to accommodate larger file systems on commodity hardware, volumes can also be partitioned across multiple MRCs.

Given proper access rights, clients can mount XtreemFS volumes anywhere in the Grid. Volumes are registered in the directory service, where a client can look up one or more MRCs hosting the volume's metadata. XtreemFS integrates with common VO authentication methods to check a user's credentials. The user's operations are subject to access policies. These access policies can implement normal file system policies like Unix user/group rights or full POSIX ACLs. In a federated environment, policies also restrict the range of OSDs to which an MRC will replicate files, or the set of MRCs from which an OSD will accept replicas.

Apart from these policy restrictions, our design allows files to be replicated to any OSD. In addition, a file's replica can be striped across a group of OSDs, which allows us to leverage the aggregate bandwidth of these storage devices by accessing them in parallel.

As a fully integrated part of XtreemFS, OSDs are aware of the existing replicas of a particular file. This knowledge allows them to coordinate their operations with OSDs that are hosting other replicas of the file. Through this coordination XtreemFS can guarantee POSIX semantics even in the presence of concurrent accesses to the replicas. In order to coordinate operations on file data, OSDs negotiate leases [10] that allow their holder to define the latest version of the particular data without further communication efforts. OSDs also keep version numbers in order to be able to identify which OSDs have the latest version of the file data.

Policies dictate how many replicas an OSD forwards changes to before acknowledging the write operation of the client. The user can for example choose a strict policy, which always keeps at least three replicas up-to-date at different sites, or select a looser policy which updates other replicas lazily or on demand.

The awareness of OSDs about replicas also allows us to logically create new replicas very quickly and reliably. From an external perspective a new replica is created as soon as the OSD is aware of being the home for the data. Its versions of replica file objects are marked obsolete. Subsequently, the replica is physically created, either on demand by a client's accesses or automatically when a policy instructs the replica to do so. Replicas are therefore always created reliably as a decentralized interaction between the OSDs. There is no need for extra services that initiate, control or monitor the transfer of the data.

In order to be able to create replicas in the presence of failures of some of the OSDs, and to be able to remove unreachable replicas, we have designed a replica set coordination protocol that integrates with the lease coordination protocol. The replica set protocols ensures that even in the worst failure case, the replicated data can never become inconsistent, while still allowing replicas to be added or removed in many failure scenarios.

This design allows us to make new replicas available very quickly, even if file data has not been completetly copied by the system. When an OSD's client only accesses a certain part of the replica, the replica only needs to keep that particular slice. The remaining data is automatically marked as being obsolete and falls behind other replicas.

Because it involves a distributed consensus process that is inherently expensive, the replica lease coordination process does not scale well. When too many OSDs per file are involved, the necessary communication increases excessively.

Fortunately, a moderate number of replicas is sufficient for most purposes. If a large number of replicas is required, XtreemFS can switch the file to a read-only mode and allow an unlimited number of read-only file replicas, which fits many common Grid data management scenarios.

4.2 Common Grid Use Cases and File Systems

The architecture of XtreemFS as an object-based file system enables many features that current Grid applications can take advantage of. The objective of this section is to present some use cases that would clearly benefit from the possibilities of accessing their data through a file system.

We first consider scientific applications that access large files routinely. For example, the Large Hadron Collider at CERN generates large files that are used in a read-only way from many nodes in a Grid. The current way of using these huge files is to copy them to the node where the file will be processed and to remove the file eventually when the processing has finished (traditional stage-in).

Being a file system, XtreemFS is able to control where and how its clients access a file's replicas. If there is a replica close enough to the client, applications can access this replica directly in a transparent way. Also, if only parts of a large file are accessed, the file system can replicate these parts and avoid transfering the whole file. A file system can also reduce the latency to first access considerably by creating the replica in the background and redirecting to local data as it becomes available.

Database Management Systems (DBMS) would also benefit from automatic and partial replica creation that can be supported by a file system. Given that databases are normally huge files that are frequently accessed only in parts, a file system could replicate only the parts of the database that are being used, reducing the amount of replicated data and the time and resources consumed for creating the replica.

As XtreemFS allows replicas to be physically desynchronized, we can allow MPI applications to work on different replicas of the same file when the different processes of the application are very far apart. XtreemFS assigns these processes a nearby replica for writing. Depending on the replication policy, the written data may be lazily synchronized to other replicas over time or later on demand. In either case, XtreemFS guarantees that replicas appear consistent for any subsequent read operation. Grid data management systems are not able to support this kind of access pattern and thus applications have to be adapted.

5 Related Work

While many Grid projects have developed custom solutions for their data management problems, a couple of software products for Grid data management have emerged and are widely adopted. Each of these system is following an architectural approach similar to the one presented in Section 2, but have different focuses for their application domains.

Among the most prominent systems are the data management services of the Globus toolkit [6][7]. Globus users install GridFTP [3] daemons on their storage resources that export the host file system and enable it for remote access. A replica catalog (RLS) indexes these storage locations and implements additional naming facilities on top of it. Globus integrates well with the local structure and access rights management of the local system and is able to preserve file naming across replicas. Its GridFTP framework [3] allows high-performance parallel transfer of files between the resources.

SDSC's Storage Resource Broker (SRB) [4] provides a complete data management solution. Unlike Globus, it does not focus on preserving a storage host's file system name space and stores its files under their file identifiers. It also provides support for federated installations in multiple sites.

dCache [8] has a strong emphasis on archiving data with the help of tertiary storage systems like tape robots. It can act as a front end to these resources and allows clients to access files via a file system-like interface over NFS.

The AMGA metadata catalog [12] of the EGEE project provides support for fine-grained access control on extended metadata and features powerful replication capabilities. It can partially and fully replicate metadata on multiple sites and supports federation of metadata by a master-slave like replication semantics.

Grid Datafarm (Gfarm) is a system for managing files in Grids which follows a file system-like approach. It is specialized for workloads in which applications create a large amount of data which is consumed by other applications later on. While the first version of Gfarm allows files to be written only once [13], Gfarm v2 [14] aims to offer full file system functionality. There is also an effort underway to standardize Grid file system at the Grid File System Working Group (GFS-WG) of the Open Grid Forum (OGF).

Existing object-based file systems are designed for single-site installations and for high-performance parallel access to the storage resources. Commercial (Panasas' ActiveScale, [2]), open-source (Lustre [1]), and research systems (Ceph [2]) are available.

6 Conclusion

In this paper, we have analyzed where the typical architecture of Grid data management systems has deficiencies and argued that a Grid-aware adaptation of the object-based file system architecture is able to address them. As an example, we have shown how some of the challenges of adapting object-based storage to wide-area federated infrastructures are solved in XtreemFS.

Object-based file systems for Grids won't be able to support the full range of application domains of Grid data management systems. Object-based file systems assume storage resources with hard disks and can't interface to heterogeneous storage systems. It could also be difficult to integrate them with existing legacy installations. Nevertheless, we think that there are enough use cases, especially in new Grid installations, where applications could considerably benefit from running on a real file system that is designed for the environment.

Acknowledgements

This work has been supported by the EU IST program as part of the XtreemOS project (contract FP6-033576), and partially by the Spanish Ministry of Science and Technology under the TIN2004-07739-C02-01 grant.

References

1. Lustre: A Scalable, High-Performance File System. Whitepaper.
2. Panasas ActiveScale File System (PanFS). Whitepaper.
3. William Allcock, John Bresnahan, Rajkumar Kettimuthu, and Michael Link. The Globus Striped GridFTP Framework and Server. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA, 2005. IEEE Computer Society.
4. C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*. IBM Press, 1998.
5. Micahel Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, and Julian Satran. Object storage: The future building block for storage systems. In *2nd International IEEE Symposium on Mass Storage Systems and Technologies*, 2005.
6. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.
7. Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer Verlag, 2005.
8. Patrick Fuhrmann and Volker Gülzow. dCache, Storage System for the Future. In *Euro-Par*, pages 1106–1113, 2006.
9. The Open Group. The Single Unix Specification, Version 3.
10. B. W. Lampson. How to build a highly available system using consensus. In Babaoglu and Marzullo, editors, *10th International Workshop on Distributed Algorithms (WDAG 96)*, volume 1151, pages 1–17. Springer-Verlag, Berlin Germany, 1996.
11. M. Mesnier, G. Ganger, and E. Riedel. Object-based storage. *IEEE Communications Magazine*, 8:84–90, 2003.
12. Nuno Santos and Birger Koblitz. Distributed Metadata with the AMGA Metadata Catalog. In *Proceedings of the Workshop on Next-Generation Distributed Data Management - HPDC-15*, 2006.
13. Osamu Tatebe, Youhei Morita, Satoshi Matsuoka, Noriyuki Soda, and Satoshi Sekiguchi. Grid datafarm architecture for petascale data intensive computing. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 102, Washington, DC, USA, 2002. IEEE Computer Society.
14. Osamu Tatebe, Satoshi Sekiguchi, Noriyuki Soda, Youhei Morita, and Satoshi Matsuoka. Gfarm v2: A grid file system that supports high-performance distributed and parallel data computing. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2004)*, 2004.