an object-based file system
for federated IT infrastructures.

The XtreemFS Architecture

Eugenio Cesario [1], Toni Cortes [2], Erich Focht [3], Matthias Hess [3], Felix Hupfeld [4], Björn Kolbeck [4], Jesús Malo [2], Jonathan Martí [2], Jan Stender [4]

[1] Institute of High Performance Computing and Networks of the National Research Council of Italy (ICAR-CNR), C\O DEIS-UNICAL, P. Bucci 41-C, 87036 Rende, CS, Italy

[2] Barcelona Supercomputing Center (BSC), UPC, Campus Nord, C\ Jordi Girona, 31, 08034 Barcelona, Spain

[3] NEC High Performance Computing Europe GmbH, Hessbruehlstr. 21b, 70656 Stuttgart, Germany

[4] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany

Abstract

This paper describes the architecture of XtreemFS, a federated and globally distributed file system. XtreemFS has to cope with scalability problems, a huge number of clients connected over WANs, the related high latency and security problems, as well as distributed file data and metadata. The XtreemFS architecture solves performance, scalability and grid-related problems by a novel design combining techniques used in grid file systems with state-of-the-art cluster file system methods.

This article is published as part of the LinuxTag 2007 programme.

XtreemFS is part of the XtreemOS project.

XtreemOS is an Integrated Project supported by the European Commission's IST program #FP6-033576.

Introduction

In recent years, the amount of data stored on file systems has grown by several orders of magnitude [1]. Today's systems used in science and industry have to deal with amounts of data in the range of petabytes. Managing data at such a scale requires a highly parallel and distributed storage infrastructure.

Object-based storage [2,3] has gradually gained importance as a design paradigm for distributed and parallel file systems. The object-based approach involves a separate management of file content and metadata. While the term *file content* describes the data stored in a file, the term *metadata* comprises any data other than file content that a file system has to manage. This includes information about how the file system is organized, as well as descriptive data related to single files, such as file name, file size, access rights or ownership.

While traditional file systems store the file metadata together with the file content on the same device, object-based file systems physically separate the responsibilities for these two kinds of data. Access to metadata is offered through a Metadata Service (MDS), whereas the content of files resides on Object Storage Devices (OSDs). OSDs are independent machines that store file content in the form of objects. An object is a portion of a file with a high-level read/write interface. Object-based interfaces hide the design of their underlying traditional storage devices. Clients are an essential component of the file system architecture, as they have to know how to interact with the MDS and OSDs in order to perform operations.

The separation of file content and metadata associated with object-based file systems sets the stage for preventing file I/O bottlenecks. File content can be spread across a large number of OSDs in order to share the read/write throughput, while dedicated metadata servers can independently guarantee a fast access to metadata. The inherent scalability makes object-based storage well-suited to large-scale distributed file systems.

The concept of a scalable globally-distributed file system is also supported by the notion of grid computing. A grid can be seen as a multi-organizational federation of computing resources. Grids are not under the control of a central authority, as they are usually built of clusters from different organizations. Decentralization plays an important role for a multi-cluster and organization spanning file system installation, as a centralized approach would severely limit the availability of files in the presence of network failures as well as the performance in wide-area networks.

In globally distributed systems, hardware failures as well as temporary partitions of the underlying network pose a particular problem. Despite unreachable servers, a file system has to ensure that data can be accessed fast and is highly available. A general approach to deal with this issue is replication. State-of-the-art object-based file systems, however, are mostly designed to run on local clusters rather than on a global scale, and include only rudimentary replication mechanisms if they support replication at all.

This paper describes XtreemFS, a federated replicated object-based file system. XtreemFS is designed to operate on a set of clusters that are connected by a wide area network. It runs on commodity hardware and doesn't require any specific equipment like SANs or Fibre Channel networks. Although its components are distributed, XtreemFS provides the user with a consistent view of the en-

tire file system. It supports replication of both file metadata and objects. Furthermore, metadata can be partitioned and objects can be striped over multiple servers for a high-throughput parallel access. A POSIX [4]-compliant access layer enables applications to use XtreemFS without modification.

Our contribution starts with an overview of existing networked and distributed file systems in section 2. Section 3 outlines the design goals of XtreemFS and describes the architecture, including the components involved. In section 4, we discuss problems and challenges involved in our goal of developing a global-scale file system.

Related Work

Networked file systems are mainly characterized by distribution, semantics and storage mechanisms. The degree of distribution can range from single servers to globally distributed systems. The semantics of a networked file system describes the way in which clients become aware of modifications made by other clients. Most applications expect file systems to offer semantics as defined by POSIX, which might require applications to be adapted in order to run with a non-POSIX-compliant file system. Finally, the mechanisms used by a file system to store and locate files are very diverse and have a strong influence on the former two aspects.

NFS [5] is one of the oldest and probably the most popular networked file system. It ensures a POSIX semantics and relies on a single server that exports volumes (sometimes referred to as mounts) to clients. Having only a single server, however, may cause bottlenecks in terms of performance and availability which leads to enormous problems in larger installations. To deal with such problems, a parallelized version of NFS called pNFS [6] is currently under development.

AFS [7] and later Coda were developed to overcome the limitations of NFS by partitioning data among many file servers. Similar to NFS, directory structure and file metadata are stored together with file content. The main difference lies in the way changes to files are handled by the systems. AFS offers a session semantics. During a session, a client works on a local copy of a file. A server-side callback mechanism notifies clients of changes concurrently made by other clients. When the file is closed, the session ends and changes are transmitted back to the server.

Coda extends the AFS session model by a transactional semantics which allows clients to continue even when disconnected from the server. In Coda, changes are eventually reconciled when the server is available again. Both Coda and AFS have the limitation of not offering POSIX file system semantics.

Traditional block-based parallel file systems like GFS [8] and OCFS2 [9] follow a different approach. They rely on the capability of every computer to access the block devices over a SAN (storage area network) and parallelize the file system code to enable coordinated concurrent access to the storage devices. The direct access to the block devices is usually realized using Fibre Channel or iSCSI. Typically, every file system client has at the same time the functionality of a file server and is deeply involved in managing the blocks on the disk storage devices. Files are striped across multiple disk devices which are virtually concatenated through a distributed logical volume manager (LVM) (in GFS). This way storage size is scaled as well as file access performance, by using multiple disk devices in parallel accesses through multiple paths. File metadata lives on the block devices as well

and is managed in parallel by the file system clients, too. The scalability of this approach is limited by the burden of the clients to manage the block devices explicitly. GPFS [10] is a proprietary representative of the block based file systems which has offloaded a part of the block management from the clients to NSD (network storage device) servers and shows improved scaling behavior. All mentioned parallel file systems have POSIX semantics for concurrent writes.

To overcome the difficulties associated with managing storage on block level, object storage devices (OSD) were introduced. Instead of working with blocks which requires knowledge about the layout of the storage media, OSDs offer a simple access to storage and hide the complex management of the underlying media. This makes the object approach particularly suitable for networked file systems. Lustre [11], Panasas' ActiveScale [12], Ceph [13] and to some extent GoogleFS [14] use this approach. However, all these file systems are designed for being used on a single cluster and are not suitable for a distribution over wide area networks. While Lustre, Panasas and Ceph do offer POSIX semantics, Google FS is optimized for append operations and has an undefined semantics for concurrent writes.

File system design has also been affected by the advent of peer-to-peer (P2P) technologies. P2P systems deal with failures and dynamism in the underlying network infrastructure by constructing overlay networks. Examples for storage systems relying on P2P overlay networks are Farsite [15], Pangaea [16] and OceanStore [17]. Since OceanStore and Farsite are designed as archival systems rather than file systems in the traditional sense, both do not offer POSIX semantics. The goal of Farsite is to implement a distributed storage for untrusted networks which requires encryption and redundancy as well as protection against malicious participants. In contrast, Pangaea is designed to be a file system but relies on a consistency model that does not guarantee POSIX semantics.

Another system for globally distributed storage is Grid Datafarm (Gfarm)[18] which, however, does not use P2P technologies. This system was developed to support scientific applications on the grid. It is specialized for workloads in which applications create large amount of data which is consumed by other applications later on. Gfarm allows files to be written only once and is not a file system. Gfarm v2 [19] aims to offer full file system functionality by allowing file modifications. A consistency semantics similar to AFS is implemented.

Architecture

The design of the XtreemFS architecture was mainly driven by the following design goals.

- global distribution
  XtreemFS must allow components to be distributed globally across multiple clusters. It must be able to cope with problems arising from distribution on wide area networks like latency, message loss and network outages.
- global replication
  A file system which is globally distributed must be able to coordinate on-line replication of files. To reduce the latencies when accessing files, XtreemFS must be able to move replicas to consumers
- federated design
  Individual clusters must stay operational even when disconnected from other replicas (e.g. in case of a network outage). A volume should have a home cluster in which it is always available.
- enhanced file organization
  Traditional file hierarchies and file names are not sufficient to organize large file systems. Retrieval of files must be possible in a manner similar to database queries.
- POSIX compliance
  XtreemFS must offer a POSIX compliant API and semantics to ensure that legacy applications will work without modification. However, this must not restrict advanced features like concurrent append [GoogleFS].
- commodity hardware
  Off-the-shelve PCs must be sufficient for XtreemFS, no special hardware ought to be required.

We decided in favor of an object-based architecture for XtreemFS, because of its inherent scalability and suitability for wide-area network file systems. The main components involved in our architecture are the Metadata and Replica Catalog (MRC), the Object Storage Device (OSD) the client/Access Layer and the Replica Management Service (RMS). In the following, we describe the different components and how they work together, which is illustrated in figure 1.

Client/Access Layer. The XtreemFS access layer represents the interface between user processes and the file system infrastructure. It manages the access to files and directories in XtreemFS for user processes as well as the access to grid specific file system features for users and administrators. It is the client side part of the distributed file system and as such has to interact with all components of XtreemFS: MRC, OSD and RMS. The access layer provides a POSIX interface to the users by using the FUSE [20] framework for file systems in user space. It translates calls to the POSIX API into corresponding interactions with OSDs and MRCs. A XtreemFS volume will be simply mounted like a normal file system and users will not need to modify or recompile their application for accessing it. In addition to the POSIX interface the access layer provides tools for creating, deleting XtreemFS volumes, create and move file replicas, check file integrity, query and change the file striping policies, and other grid specific features.

OSD. OSDs are responsible for storing file content. The content of a single file is represented by one or more objects. With the aim of increasing the read/write throughput, OSDs and Clients support striping. Striping of a file can be performed by distributing the corresponding objects across several OSDs, in order to enable a client to read or write the file content in parallel. Likewise, OSDs support replication of files, for the purpose of improving availability and fault tolerance or reducing latency. Files are replicated by holding replicas of the corresponding objects on different OSDs.

MRC. MRCs constitute our metadata service. For availability and performance reasons, there may be multiple MRCs running on different hosts. Each MRC has a local database in which it stores the file system metadata it accounts for. Similar to NFS files are organized in volumes. The MRC offers an interface for metadata access. It provides functionality such as creating a new file, retrieving information about a file or renaming a file.

RMS. The RMS is the responsible for deciding when new replicas should be created or removed. According to the application needs, the topology of the network and the legal limitation on file location, the service will initiate the creation or deletion of replicas. In addition, it will also decide what striping policy is best and which OSDs should be used. Finally, the RMS will keep track of replica usage and will remove superfluous replicas.

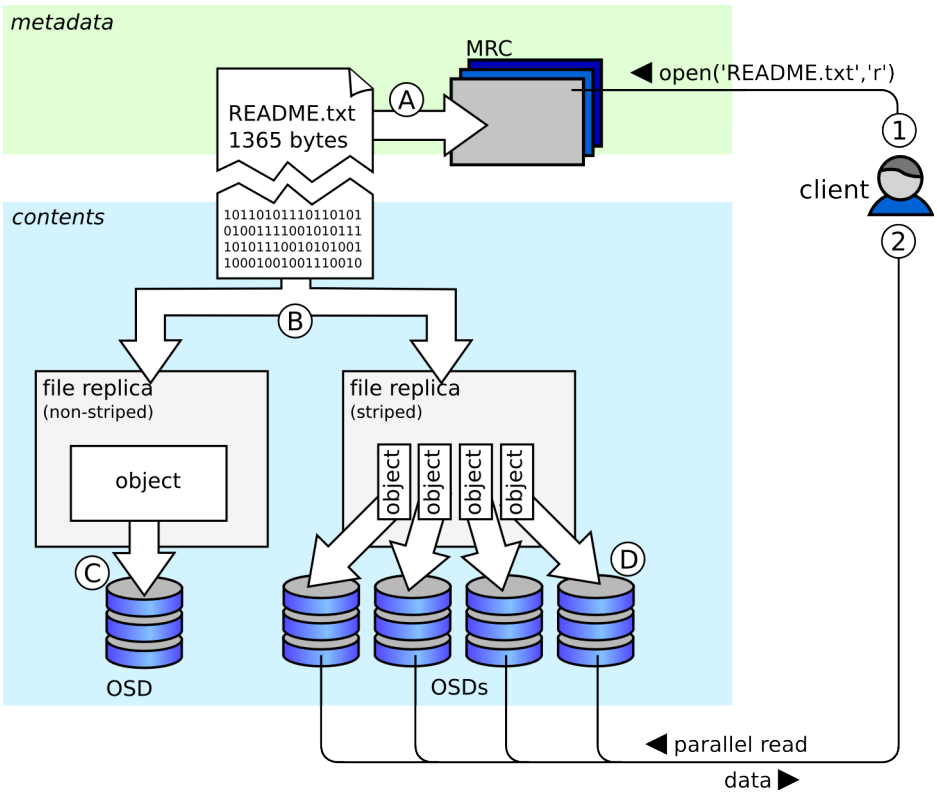

Figure 1: Separation of metadata and file contents. In step 1 the client requests permission to open a file from the MRC. In step 2 it reads the file from all OSDs holding a stripe of the replica. (A) shows that the metadata is stored in the MRC. The file content is stored in two replicas (B). The non-striped replica is stored in a single OSD (C), the striped replica is plit in four objects stored on different OSDs (D).

Challenges

In the course of the development of XtreemFS, we have identified various key issues associated with the different components. This section gives an overview of what makes up the complexity of a scalable object-based file system for wide-area networks.

MRC. The overall availability of XtreemFS volumes does not only depend on the replication of file content. Replication of volume metadata stored on the MRCs is also necessary. This offers better availability and higher data safety, but also allows users to choose an MRC close to their site which is important to reduce latencies. In the Master/Slave replication scheme, one MRC holds the master copy of a volume, changes can only be made at the master. The slaves hold up-to-date copies of the volume which are read-only. If the master fails a slave is elected and replaces the master. This scheme is easy to implement and suitable for most workloads. However, if modification operations are frequent, a symmetric replication scheme is necessary to allow clients to execute all operations on all MRCs holding a volume replica. Mechanisms for symmetric replication are far more complex than master/slave and not all known approaches are suitable for systems with high latency. The evaluation and implementation of suitable mechanisms will be a major research topic for the MRC.

Another aspect that is closely related to replication is metadata partitioning. Since metadata operations are fairly frequent in file systems [21], access times to metadata should be minimized. If certain parts of a volume are accessed frequently, server-side bottlenecks might occur. Moreover, volumes may become too large to be stored in a single database. To tackle this problem, it is reasonable to split and distribute volumes across multiple MRCs. This, however, poses the problem that multiple MRCs might have to be contacted for a single operation. If parts of a file path are managed by different MRCs, the latency of operations on that path might be increased, as MRCs have to be contacted one-by-one. This, for instance, can happen in connection with a recursive evaluation of access rights, as required by POSIX. Developing suitable strategies for metadata partitioning is a challenging problem, to which only few solutions have been suggested, such as dynamic subtree partitioning [13].

OSD. As we have already described, OSDs are responsible for storing objects in a distributed way. The first challenge we find in the design and implementation of OSDs is replica handling. When new replicas are created the current file content must be transferred to the OSDs holding the new replica. This operation must tolerate concurrent modifications of file content and still guarantee consistency of replicas. It is important to notice that the population of the replica does not need to be done at creation time. XtreemFS will support partial replicas that are being populated on-demand when requested parts are missing or according to a policy devised by the RMS.

The consistency of replicas is handled by the OSDs in a distributed manner. To handle consistency issues we use a leases mechanism to make sure that only a single OSD is able to modify a specified part of the data at a given time (single writer, multiple readers). If many clients want to modify the same data concurrently, they will be redirected to the OSD that has the lease to avoid unnecessary coordination of operations. In the future, we will investigate new consistency models and implement them in the OSDs. Concurrent modifications do pose another problem which requires interaction between the OSDs and the Access Layer. To guarantee cache consistency on all clients, the OSD will hand out leases to the clients and notifies them when file content has changed.

In order to improve I/O performance replicas can be striped among several OSDs. This striping is

also handled by the OSDs in cooperation with the clients. Determining the file size in such a scenario is challenging since many OSDs have to be queried to get the actual file size.

RMS. To reduce latencies experienced by clients when accessing data, the RMS will try to create new replicas close to a consumer's site. This requires the RMS to have enough knowledge on the network topology to find suitable OSDs. Moreover, the RMS must be able to predict future accesses based on previous access patterns. A pro-active approach will lead to a better overall performance and is complementary to partial replicas. The removal of unnecessary replicas is important to keep the total number of replicas as small as possible and speeds up the replication process.

Since users can specify sophisticated replication policies limiting the possible locations and number of copies, the RMS has to make sure that the policy is obeyed. If required it has to decide how many and which replicas need to be removed when new replicas are created. For instance, XtreemFS will allow users to control replicas by limiting its number, by specifying a minimum number for fault tolerance, or by stating that a given file cannot have replicas outside a specified country for security reasons.

Client/Access Layer. The complexity of the file system client goes far beyond what is usually required for the access layer of a traditional file system. The reason for this is the distributed design which needs to be able to cope with high latencies, network outages, temporary MRC or permanent OSD failures. Transaction rights are handled as leases with limited life time, this requires that the client is actively updating its leases and automatically looses them in case of disconnection. Although some of the file system functionality was moved out to specialized servers like the MRC or OSDs, the client retains sufficient tasks to make its implementation challenging. Some of these challenges are:

- Allow concurrent access from multiple clients to files: this is a strong requirement in parallel computing where programs need to be able to read but also write in parallel with high bandwidth to the same file.
- Handle file caching during concurrent file access: Using the FUSE infrastructure eases implementation because the main functionality is coded in user space and can make use of external libraries. One of the drawbacks is the loss of control over the page cache living in the kernel space. We are currently investigating mechanisms to allow the invalidation of page caches from the user space FUSE code.
- mmaped files, shared mmaps: Files which are mmaped are required to use caching through the page cache. Handling multiple read-only mappings is straight forward, but shared mmaps will be more difficult to handle. They require mechanisms for actively invalidating and dropping mapped pages on all clients which mmaped the same file, as well as mechanisms for detecting access to mmaped pages.
- Managing file striping across multiple OSDs: For high performance access to files one of the striping patterns will be RAID0, i.e. the file will be chopped in stripes of reasonable size which are stored on multiple OSDs. The access to these stripe objects will happen in parallel by using multiple storage devices and network paths at the same time. The striping logic is handled in the file system client. This approach is similar to other object based parallel file systems like Lustre, Panasas.

- Redundancy of files through RAID5/6-like striping: The idea of distributing file chunks across multiple devices and adding parity chunks to them is not new, it is used e.g. in ZFS[22], Panasas. The logic for redundancy is outsourced from the block based RAID devices to the file system client, i.e. the parity blocks are calculated by the client, thus offloading the storage devices. This approach to redundancy has great benefits: a broken file sector on a hard disk is affecting only one file and not the entire device, as it does on block based RAID5/6. File recovery has to be done only for files which are affected and is as simple as copying the file and re-striping it. This allows much faster and safer recovery from failure of a storage device.

Conclusion

In this paper, we presented the architecture of XtreemFS, an object-based file system designed to run in a wide-area network. XtreemFS provides fault tolerance and scalability by supporting replication and striping of files. File replica management can be automated by means of a replica management service. A replicated and partitioned metadata service ensures that metadata is highly available and metadata operations can be performed quickly. The access layer offers a POSIX-compliant interface and hides the distributed character of the file system from the user. We introduced the different components that constitute the object-based architecture of XtreemFS and outlined their specific issues and research perspectives.

Acknowledgments

This work was supported by the EU IST program as part of the XtreemOS project (contract FP6-033576).

References

[1] Jeff Bonwick's ZFS blog, http://blogs.sun.com/bonwick/date/20040925

[2] Michael 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.

[3] Michael Mesnier, Gregory R. Ganger, Erik Riedel. Object-based storage: pushing more functionality into storage. *IEEE Potentials* (Vol 24, Issue 2, pg 31-34). Apr-May 2005. IEEE.

[4] The Open Group. The Single Unix Specification, Version 3.

[5] Bill Nowicki, Sun Microsystems, Inc. RFC 1094 - NFS: Network File System Protocol specification, 1989

[6] Benny Halevy, Brent Welch, Jim Zelenka. Object-based pNFS Operations. Internet-Draft http://tools.ietf.org/id/draft-ietf-nfsv4-pnfs-obj-03.txt

[7] Mahadev Satyanarayanan. Scalable, Secure, and Highly-Available Distributed File Access. I*EEE Computer*, May 1990

[8] Global File System (GFS). RedHat, Inc. http://www.redhat.com/software/rha/gfs/

[9] Oracle Cluster File System v2 (OCFS2). Oracle, Inc. http://oss.oracle.com/projects/ocfs2/

[10] Frank Schmuck, Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, 2002.

[11] Lustre architecture whitepaper. Luster, Inc. http://www.lustre.org/docs/whitepaper.pdf

[12] David Nagle, Denis Serenyi, Abbie Metthews. The Panasas ActiveScale Storage Cluster –Deli vering Scalable High Badnwisth Storage. In *Proceedings of the ACM/IEEE SC2004 Conference*, 2004.

[13] Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06)*, 2006.

[14] Sanjay Ghemawat, Howard Gobio, Shun-Tak Leung. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003.

[15] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, Roger P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. SIGOPS Oper. Syst. Rev. 36, SI (Dec. 2002), 1-14.

[16] Yasushi Saito, Christos Karamanolis, Magnus Karlsson, Mallik Mahalingam. Taming aggressive replication in the pangaea wide-area le system. SIGOPS Oper. Syst. Rev., 36(SI):1530, 2002.

[17] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[18] Osamu Tatebe, Youhei Morita, Satoshi Matsuoka, Noriyuki Soda, Satoshi Sekiguchi. Grid Datafarm Architecture for Petascale Data Intensive Computing. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, 2002

[19] Osamu Tatebe, Noriyuki Soda, Youhei Morita, Satoshi Matsuoka, Satoshi Sekiguchi. Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing. In *Proceedings of the 2004 Computing in High Energy and Nuclear Physics (CHEP04)*, 2004

[20] FUSE. http://fuse.sourceforge.net

[21] Drew Roselli, Jacob R. Lorch, Thomas E. Anderson. A comparison of file system workloads. In *Proceedings of the 2000 USENIX Annual Technical Conference*, 2000. USENIX Association.

[22] ZFS, Sun Microsystems, Inc. http://opensolaris.org/os/community/zfs/