Sistemas de Archivos para Arquitecturas de Gran Escala

Yves Denneulin
LIG Laboratory[*]
Grenoble, Francia

Abstract

This article presents various aspects of the management of data on large scale and high performances architectures. Its focus is mainly on file systems.

1. Introduction

The management of data for high performance applications has always been a challenge. The difficulties arise from the following constraints:

- the amount of data can be very large,
- the value of the data, when they are the results of costly experiments that can't be reproduced for example,
- the speed of access, in order not to slow down the high performance applications.

This is the reasons why storage has always been a major concern when high performance computing is concerned and the super computing centers have always been on the cutting edge of the IT world in this aspect. The typical structure of a data center has three layers:

1. long term storage, typically composed of silos of tapes,
2. short term storage with lots of winchester hard drives,
3. local cache where the treatments on the data takes place.

When a client nodes wants to access a data, the data has to be identified then moved from the long term storage to its place of use. Identifying data associated to a request is typically a high level operation while the transfer implies low level mechanisms. The most common abstraction to organize the data is the file, situated in a directory, the directories building a hierarchy. The data are stored in a file while metadata associated with the data, and often with the file, are used to find them. Typical metadata for a file system are name of the file, date of creation, location of the data, access rights...

If this is not enough additional indexing techniques can also be set up, usually using a relational database to manage the high level metadata while the data are stored on a filesystem for scalability and performances.

Performances being an important, if not the most important, criterion, the file systems have to be designed in order to fully exploit the underlying hardware.

In the following we will begin by presenting the main characteristics of distributed file systems together and continue by briefly addressing data management on grids.

2. Distributed file systems

Any HPC architecture is distributed since it necessarily contains a lot of nodes. Hence any file systems for such architecture has to be distributed as well.

Two main aspects are to be considered for distributed filesystems:

- how they interact with the clients,
- how they manage the storage space.

2.1. Interactions with the clients

Every widely used file system uses the client/server model. In order to access the files, a client has to send requests to the server that will answer it either with the data it wants, in case of a read, or with an acknowledgment, in case of a write. A typical approach is the exportation of a local file system on the server to the clients. This kind of operation is called an *export*.

The clients have to be able to communicate with the server using a protocol. A good example of such a protocol, outside the scope of file systems, is the FTP protocol. Regarding file systems on HPC architectures two main protocols are used:

[*]The LIG Laboratory is funded by CNRS, INPGrenoble, INRIA and the UJF university

[†]This work is funded by the project ECOS C07M02.

NFS is the standard in the UNIX environments and, since this is the case of most HPC architectures, in the HPC world for exporting file systems. Its commonly used versions, V2 [6] and V3 [1], use a stateless protocol, each request is self sufficient. Since the server can then be stateless too, it is a protocol easy to implement, fault tolerant and that provides good performances. Here lie the reasons for its wide use.

CIFS [3] is a closed protocol used by Windows systems to export file systems. An open implementation exists, Samba. Contrary to NFS it needs a state-full server and is connection oriented.

2.2. Management of the storage space

The storage space can be managed in two ways: either in a centralized manner that can replicated for fault tolerance reasons, or distributed. The border between the two can be thin; in the case of a RAID array for example, while the storage is physically distributed hardware mechanisms hide this distribution. For the server and the clients the storage space seems to be fully centralized.

From a software point of view two categories of file systems exist: the ones that handle the distribution only from a hardware point of view, NFS, and the ones that are able to handle physical distribution and provide to the clients a centralized view, GPFS [5], Lustre [2].

3. Data on Grids

Data management on grids has to address several problems: the high level of heterogeneity of the data that makes their integration into an unique system difficult, this the problem of data analysis. A second problem lies in the large amount of data stored and on the large scale of the distributed systems on which this storage takes place.

For legacy reasons, the most used abstraction used on grids remains the file and, hence, organized in file systems. The only difference between file systems for clusters and for grids lies in the latency and bandwidth available between the sites, making caching, and so replication, mandatory for performances reasons.

3.1. The file systems

Two approaches are widely used in grids regarding file systems:

- the use of explicit localization techniques, typically URL, coupled with high performance data transfer mechanisms with the replication and movement of data handled by the programmer of the application or by a dedicated service, this is the option followed by GRAM and GridFTP [8] for example,
- the definition of a real grid file system, with caching, data movement and replication management handled transparently by the file system itself, examples of such systems are GFarm [7] or NFSg [4].

3.2. Data type management

In the previous section we saw a management of data by the use of metadata belonging to the file systems category: file name, path, length, ... This is often not enough for the large amount of data stored on such architecture. Various solutions have been proposed, the most commonly used is using a database, mostly relational, to manage the metadata, one of the metadata being the address of the data, often under the form of an URL. We find this approach in the EGEE middleware with the components or in the Globus project by the use of OGSA-DAI and of the corresponding replication manager, RLS.

References

[1] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification. RFC1813, June 1995.

[2] I. Cluster File Systems. LUSTRE: A High-Performance, Scalable, Open Distributed File System for Clusters and Shared-Data Environments, Nov. 2002. Disponible http://www.lustre.org/docs/whitepaper.pdf (2003-08-21).

[3] C. R. Hertel. Implementing CIFS The Common Internet File System. http://ubiqx.org/cifs/, 1999-2001.

[4] P. Lombard, A. Lebre, C. Guinet, O. Valentin, and Y. Denneulin. Distributed Filesystem for Clusters and Grids. In *Workshop at the Fifth International Conference on Parallel Processing and Applied Mathematics (PPAM 2003), Special session: Large Scale Scientific Computations*, Sept. 2003.

[5] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of the First Conference on File and Storage Technologies (FAST)*, pages 231–244, Jan. 2002. Disponible http://www.almaden.ibm.com/cs/gpfs.html (2003-08-14).

[6] Sun Microsystems, Inc. NFS: Network File System Specification. RFC1094, Mar. 1989.

[7] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi. Grid datafarm architecture for petascale data intensive computing. In *CCGRID*, pages 102–110. IEEE Computer Society, 2002.

[8] D. van Heesch. *GridFTP: FTP Extensions for the Grid*, 1997-2000.