STORM: GRID MIDDLEWARE FOR DISK RESOURCE MANAGEMENT

F. Donno, CERN, Geneva, Switzerland
A. Ghiselli, L. Magnoni, R. Zappi, CNAF, Bologna, Italy

Abstract

Within a Grid the possibility of managing storage space offering POSIX access to disk is fundamental. On the other hand, the increasing availability of high performance computing resources raises the need for fast and efficient I/O operations and drives the development of parallel, distributed file systems. Through StoRM (**Sto**rage **R**esource **M**anager) an end user client can reserve and manage space on disk-based storage systems. It can then access the space either in a Grid environment or locally in a transparent way via classic POSIX calls. The StoRM architecture is based on a pluggable model that uses file-systems such as GPFS or LUSTRE with an SRM interface. StoRM includes quota management and a space guard, and serves as policy enforcement point (PEP) for the Grid Policy Management System over disk resources. In this paper we describe the StoRM architecture and the preliminary functional results obtained.

INTRODUCTION

Today's data intensive applications demand large storage systems capable of serving hundreds of terabytes of storage space. The access to such a space needs to be fast. Even though tape servers can provide the desired capacity, they do not satisfy the performance requirements. Therefore, they are used essentially as tertiary data stores accessible through a user transparent interface.

In order to achieve fast data access to large storage resources, high performance I/O solutions based on parallel file systems on very high-speed connections (GPFS on Infiniband, etc.) are deployed for large computer farms. Such solutions provide for POSIX I/O, centralized management, load balancing, monitoring, and fail-over capabilities, among others.

In the Grid community, there is a tendency to provide disk pool managers capable of serving large amounts of disk space distributed over several servers. However, most of the time, such systems do not allow for POSIX I/O, but file access is guaranteed via Grid or specific protocols, sometimes with lower efficiency. Moreover, users do not always have control over their applications. Adopted proprietary solutions, legacy software, performance factors, etc. often do not allow for changing (re-writing) an application in order to make it Grid-aware, i.e. use the provided protocols.

Before moving jobs, that generate large output files, to Grid computing systems with available CPU capacity, a Grid scheduler should check for availability of the required space and then allocate it. Therefore, the possibility to manage disk-space via the Grid becomes essential for running data-intensive applications on the Grid. Data movement and replication are also important functions to guarantee optimised access to files.

In the Grid environment the need for a homogeneous, transparent interface to storage devices has brought Grid scientists to the definition of the Storage Resource Management (SRM) interface [1].

In the following document we present StoRM, a disk-based storage solution with an SRM interface that allows applications to access files on a parallel file-system directly via POSIX calls, whether scheduled via Grid or only through the local batch system, in a transparent way with the guarantee to find the required disk space. StoRM is aware of local and Grid application usage of the storage resources and reports the information about space and allocation contention.

The integration of existing high performing, parallel file-systems into a Grid infrastructure allows users to take advantage of such technologies.

In what follows we present the details of the proposed solution and a prototype implementation. We also report on the tests executed, which include real use-cases of applications running on the Grid today. In the next section we discuss the functional requirements imposed by the proposed system on a generic high-performance distributed file-system and in particular we focus on systems such as GPFS, GFS, LUSTRE and PVFSv2. In the architecture section we give details on the goals of our project and a technical description of the design of the StoRM system. In the following section we describe the implementation of the first functional prototype. A discussion on SRM and our implementation is given. Finally, related work and conclusions follow.

FILESYSTEM REQUIREMENTS

In the following section we discuss the requirements for a distributed file system eligible to satisfy the demand coming from data intensive Grid applications.

A file-system should provide for:

- Uniform access to data – single-system image or name space across a cluster.
- POSIX interface – no modifications for applications.
- Full administrative API for file-system management (space management, security, administrative functions, etc.).

- High capacity – large files (10-50GB), 100TB file-systems.
- High throughput – wide striping, large blocks, many GB/s throughput.
- Reliability and fault-tolerance - node and disk failures.
- Online centralised system management – dynamic configuration and monitoring.
- Parallel data and metadata access – shared disks and distributed locking.
- Space allocation at file level.
- Quota, meta-data and file lifetime management.
- Access Control Lists (ACLs).

File-systems such as NFS and AFS do not satisfy all requirements above. Even though they are widely used, they present quite a few performance and scalability problems.

One interesting file-system that meets most of the requirements stated above is the General Parallel File System (GPFS) from IBM.

GPFS for Linux is a high-performance shared-disk file-system that can provide data access from all nodes in a Linux cluster environment. Parallel and serial applications can access shared files using standard UNIX file-system interfaces, and the same file can be accessed concurrently from multiple nodes. GPFS provides high availability through logging and replication, and can be configured for fail-over from both disk and server malfunctions.

To support its performance objectives, GPFS is implemented using data striping across multiple disks and multiple nodes, and it employs client-side data caching. GPFS provides large block size options for highly efficient I/O and has the ability to perform read-ahead and write-behind file functions.

GPFS uses block level locking based on a sophisticated token management system designed to provide data consistency while allowing multiple application nodes concurrent access to a file.

When hardware resource demands are high, GPFS can find an available path to the data by using multiple, independent paths to the same file data from anywhere in the cluster.

Increasing the number of nodes and disks assigned to support the I/O as the overall cluster configuration grows provides scalability. A scalable interconnect, such as the Myricom Myrinet™-2000, is generally used to support this overall scaling.

Other systems that satisfy the requirements above are PVFSv2, GFS, LUSTRE. However, we decided to consider GPFS for our first prototype implementation of StoRM as underlying file-system, given the availability of installations at our sites.

THE STORM ARCHITECTURE

In this section we describe the motivations that have brought us to the current design of StoRM together with the goals and objectives that we intend to achieve. Then, a general architecture overview of the system is given.

Motivations and Goals

The StoRM project has started with the attempt to provide an answer to a set of requests coming from various applications to have transparent access to storage systems from both a Grid and a local environment while providing access to high-performance, modern parallel file systems. In current Grid infrastructures, such as LCG-2, native POSIX access over the LAN to storage resources is not enabled because of the missing storage management capabilities offered by current file-systems such as NFS or AFS. Only Grid-enabled file access libraries are therefore in use, such as the Grid File Access Library (GFAL). Users require not to be forced to modify their code in order to be Grid-aware.

Among others, StoRM allows applications to perform the following operations:

- Reserve disk space
- Query space status
- Monitor free space
- Enforce space policies
- Allow for disk quota
- Manage object attributes such as file types, ownership, lifetime, etc.
- Advance reservation

Overview

StoRM is an “OGSA-oriented” web service that interacts with resource reservation and storage space management services provided by the underlying file-systems, as pointed out in the previous sections. It offers a virtualisation over several file-system implementations. The present design of the storage resource system provides for a layered model to increase flexibility and maintainability. This approach has allowed us to identify three separate types of functions that a Grid-aware storage manager service has to provide:

- **Data transfer**: In order to support data exchange between local and remote Grid sites, a Grid storage manager system has to support file transfer protocols such as the existing GridFTP. StoRM can easily be extended to support new protocols that might appear.
- **Core functionalities**: StoRM provides users with multiple functionalities such as a translation service for different namespaces, file staging into tertiary storage if available, space reservation, fine-grained access control, pinning, etc.
- **Metadata and Information**: the system has built-in *information providers’* capabilities that allow for publishing of a variety of meta-data information such as service URL, status, etc.

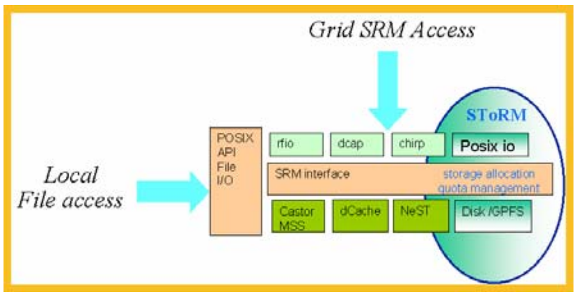

Figure 1: Access to data files via local calls or via Grid SRM with StoRM

Here, we give more details about the core functionalities provided by StoRM.

The SRM protocol seems to gain acceptance in the Grid community and become a standard for storage access. Given the need for an application to efficiently access data and possibly not to change its I/O interface, we decided to provide users with an extended SRM compliant interface that also offers a POSIX interface. Figure 1 shows space reservation and file access locally and via Grid. A local application can create files locally via POSIX calls. The space used by such an application is monitored by the StoRM service. A Grid application that is running locally can as well create and access files using multiple protocols such as rfio, dcap, and chirp. The SRM interface can hide the details of the underlying storage system (Castor, dCache, Nest, GPFS, etc.). Other native file-system calls can be used as well by the application.

The StoRM prototype includes space reservation functionalities that complement SRM space reservation to allow applications to directly access/use the managed space through POSIX calls. Once a space reservation request arrives, the server returns a URL with a *SpaceToken* that contains a *<path>/<filename>* string that can be directly used by applications requiring POSIX access.

Moreover, StoRM includes quota management and a space guard. StoRM acts as policy enforcement point (PEP) for the Grid Policy Management System over disk resources.

Furthermore, clients can choose among several file access libraries that best fit their application to access the data served by StoRM: i.e. gfal, rfio, etc.

System Architecture

The StoRM architecture is based on a pluggable model in order to easily add new functionalities. The StoRM implementation uses modern file-systems such as GPFS or LUSTRE.

The main StoRM components are shown in Figure 2 and outlined below.

Request Manager – It manages StoRM requests, verifying request permission (policy enforcement) and forwarding the request to the appropriate module. The Request Manager handles client connections for SRM calls.

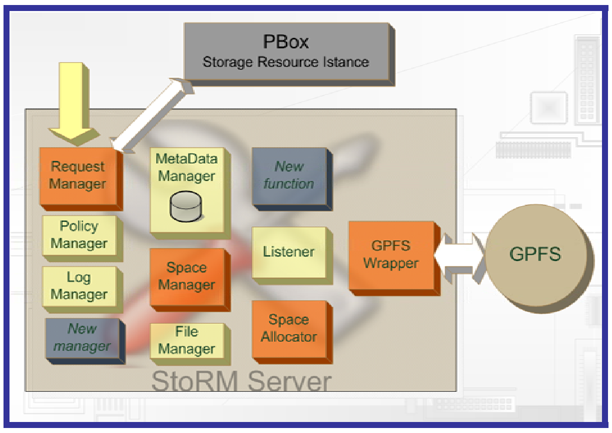

Figure 2: The StoRM architecture

Policy Manager - This module provides a GSI-based authentication and authorization mechanism for incoming requests, enforcing either local or VO specific policies.

Metadata Catalog – This catalogue module is used by StoRM to store internal status and data management information such as space availability and files. The module is a plug-in and allows for different kinds of database back-ends, from simple text based files to relational databases such as Oracle.

Space Manager - This module offers functionality for space reservation. It exposes only a storage-independent general interface. Other dedicated modules, such as the Space Allocator, implement device dependent functions. It manages data in the Metadata Catalog, and returns important information about storage status and space reserved.

File Manager - This module handles requests for garbage collection, pinning, locking and space/file lifetime.

Space Allocator - This module is the first low-level module. Its task is to satisfy requests for space allocation in the low-level storage system. Therefore, this module must be properly realized for every underlying file-system used (POSIX file-system, GPFS, etc.). In our case it includes the ***GPFS Wrapper***.

Listener - This is another low-level module. Its role is to create information about state of the physical storage system, such as free space or file size.

Here we describe how the various modules interact. A Grid application invokes the SRM v2.1 function submit_srmReserveSpace to reserve disk space for output. The request goes to the StoRM *Request Manager* that then invokes StoRM *Space Manager*. It is the responsibility of this module to analyse the physical path specified in the request and, if not explicitly defined, to define it as a function of the VO or request ID. The *File Manager* is then invoked to interact with the underlying storage system to verify the available space, the directory existence, permission management, etc. All this is done through POSIX calls. Finally, StoRM's *Space Allocator* interface *spaceAlloc(size Bytes, string Path)* is invoked. It analyses the file-system type where the space will be reserved and invokes the *write* file-system wrapper module. The client can then use the file path specified or returned to create the new file.

StoRM will constantly monitor the underlying file-system that can be used natively by client applications in order to always report space usage statistics coherently.

The **Policy Box** (**PBox**) is an external service for the definition and the management of Grid policies (see Figure 3). Policies concern authorization, priority, management, etc. PBox has a hierarchical, distributed architecture. Every instance includes a PDP (Policy Decision Point) called by the resource wrappers in charge of enforcing policies. A resource usage requests arrives to the Policy Enforcement Point (PEP) that translates it into a request for the PBox. The PBox gives an answer (for authorization, priority, obligation, etc.) that the PEP can interpret and enforce.

In our case, StoRM acts as PEP for space management requests. The policies that PBox can process for storage services concern: maximum number of files, quota management, access to the service, etc.

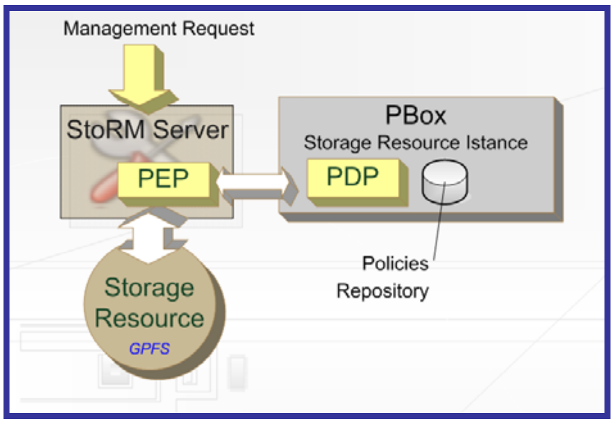

Figure 3: The PBox system

IMPLEMENTATION

The first implementation of StoRM includes functionalities of space allocation and reservation while file transfer capabilities, already existing in other SRM implementations, will follow soon. In particular, only the modules shown as light boxes in Figure 2 have been implemented in our first prototype.

The StoRM server is multithreaded. It has been implemented using AXIS SOAP. The *Request Manager* module has been written in Java while the *Space Manager*, *Space Allocator* and *GPFSWrapper* are written in C++ (because of the C++ APIs available in GPFS) and connected via JNI to the Java generic *Space Manager* module. The *GPFS Wrapper* module has been realized using mainly the *gpfs_prealloc()* GPFS API. This function is used to pre-allocate disk storage for a file that has already been opened, prior to writing data to it. The pre-allocation of disk space for a file provides an efficient method for allocating storage without having to write any data. This can result in faster I/O compared to a file that gains disk space incrementally as it grows. Existing data in the file will not be modified.

We have performed preliminary functional tests with good results using two testbeds, one located at CINECA in Bologna where GPFS is used in production, and one located at CNAF in Bologna constituted of 2 PCs with a distributed GPFS file-system of about 38GB.

SIMILARITIES AND DIFFERENCES WITH SRM

StoRM implements a subset of the functionalities specified by SRM v2.1. The compatibility with such an interface is semantic but not syntactic. In fact, we changed the binding names and we used 5 different ports. The SpaceToken defined by SRM is not just a string but it must contain <path>/<file> to allow for native POSIX access to a file.

CONCLUSIONS

We presented our proposed solution of an SRM enabled Storage Resource Manager that enables native POSIX access to files providing access to high performance parallel file-systems. Other SRM implementations are available. Among others we have to mention d-Cache and Castor SRM. However, they implement at the moment SRM v1.1 and do not allow for space allocation and POSIX access to files but force users to use specific protocols.

ACKNOWLEDGEMENTS

This work has been funded by INFNGRID Grid.it project. We acknowledge the people in CINECA Bologna and INFN-Catania for their support.

REFERENCES

[1] The SRM Interface Specification http://sdm.lbl.gov/srm-wg/doc/SRM.spec.v2.1.final.pdf

[2] F.Schmuck, R.Haskin, GPFS: A Shared-Disk File System For Large Computing Clusters, FAST'02, 28-30 January 2002, Monterey, CA .

[3] The Storage Resource Management Working Group: http://sdm.lbl.gov/srm-wg, 2004.

[4] GLUE Schema for the Storage Element, V1.1, http://www.cnaf.infn.it/~sergio/datatag/glue/v11, 2003.

[5] A. Caltroni, V. Ciaschini, A. Ferraro, A. Ghiselli, G. Rubini, R. Zappi ,G-PBox: A Policy Framework for Grid Environments, Chep 2004, 26 Sept – 1 Oct 2004, Interlaken, Switzerland.