EUROPEAN MIDDLEWARE INITIATIVE

EUROPEAN MIDDLEWARE INITIATIVE

STORM: FUNCTIONAL DESCRIPTION

Document version:	**Xx.Yy.Zz**
EMI Component Version:	**Aa.Bb**
Date:	**June 21, 2011**

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.

CONTENTS

1 SERVICE OVERVIEW

The StoRM service is a storage resource manager for generic disk based storage systems separating the data management layer from the underlying storage systems. It implements the SRM interface version 2.2. StoRM provides a flexible, configurable, scalable and high performance SRM solution. It supports standard Grid access protocols as well as direct access (native POSIX I/O call) on data, fostering the integration of non Grid aware application providing local access on shared storage. Another important characteristic of StoRM is the capability to identify the physical location of a requested data without querying any database service but evaluating a configuration file, an XML schema that describes the storage namespace and input parameters as the logical identifier and SRM attributes. StoRM relies on the underlying file system structure to identify the physical data position.

1.1 STORM COMPONENT

StoRM has a multi-layer architecture(Fig. 1) made by two main stateless components, named *Front-End* (FE) and *Back-End* (BE). A database is used to store SRM requests and the StoRM metadata. A modular architecture decouples StoRM logic from the different file system supported, and plug-in mechanisms allow an easy integration of new file systems. With this approach data centres is able to choose the preferred underlying storage system maintaining the same SRM service.

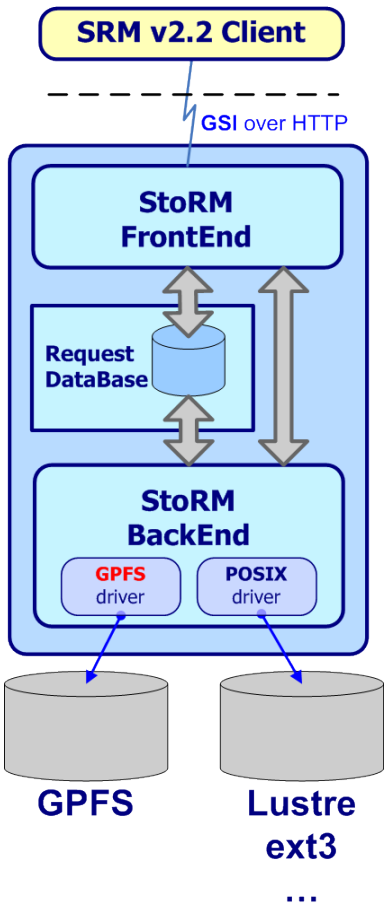

Figure 1: StoRM Architecture.

2 StoRM Frontend

The Front-End component exposes the SRM web service interface, manages user authentication and stores the data of the SRM requests into the database. It's mainly written in C/C++. It relies on the GSOAP framework to expose the SRM interface and it uses the CGSI-GSOAP plugin to manage secure connection with clients.

2.1 GSI Authentication

The GSI authentication is managed by the FE. To properly establish a secure connection between SRM client and StoRM server, the Certificate Authorities (CA) RPMs and the VOMS server RPMs have to be up to date both on the client and server machine. Once the secure connection has been established, the FE parse the user proxy, getting DN and VOMS FQANs. The FE then perform a first check on user identity verifying if the requestor identity is mapped in a local user on the FE machine. This is done querying the LCMAPS service on the FE machine with user credential, passing both DN (and FQANS) . If a mapping for the user exists (whatever it is) the SRM request go ahead, otherwise it got an error at GSOAP level. (This check is optional from StoRM v.1.4 and it can be disabled, see the FE configuration section)

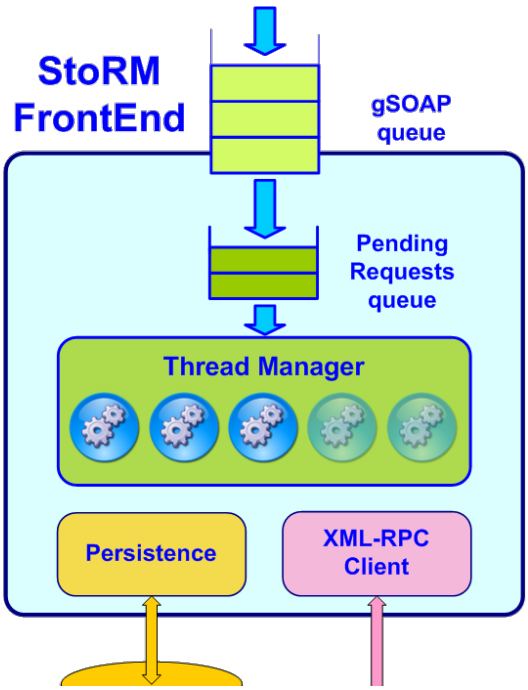

Figure 2: StoRM Frontend Component.

2.2 POOL OF WORKER THREADS

The FE uses a pool of worker threads to manage SRM requests. Once a request has been authorized, the FE assigns it as a new task for a worker thread. In case there are no free threads in the system, the request is maintained in an internal queue. The size of the pool and the size of the queue are important parameters, their value have to be defined depending on hardware resources and performance required. Depending on the type of SRM request, each thread should have two main task to do, as explained in the next paragraph.

2.3 XML-RPC COMMUNICATION

Synchronous SRM requests are a category of SRM calls that return the control to the client only when the request has been executed by the system. Most of the SRM call belongs to this category: Namespace operations (srmLs, srmMkdir,etc.), Discovery operation (srmPing), Space operations (srmReserveSpace, srmGetSpaceMetadata, etc.). For this type of request, the FE perform a direct communication to the Backend using a RPC approach, based on the XML-RPC protocol. XML-RPC is a simple protocol to exchange XML structured data over HTTP. The Back-End provides an XML-RPC server and the FE(s) acts as client. A worker threads in case of synchronous requests performs this steps:

- structure the SRM data in XML
- send a request to the BE XML-RPC server
- wait until the execution
- get result from XML and unmarshall it in SOAP
- return the control to the client

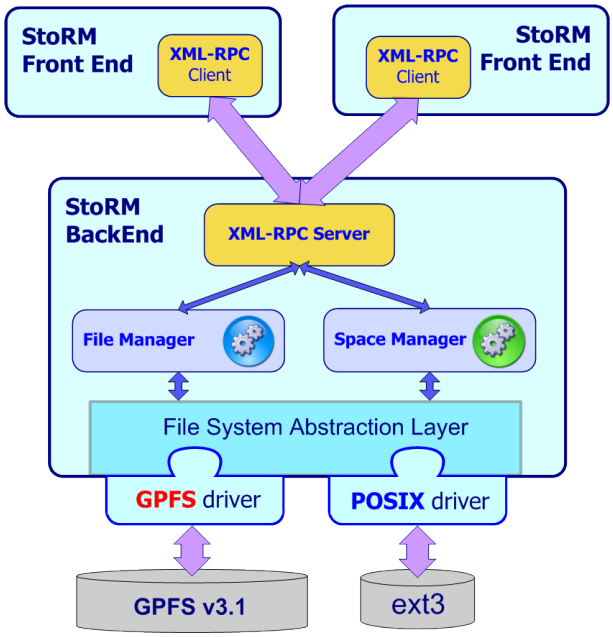

Figure 3: Management of SRM synchronous request.

2.4 ASYNCHRONOUS REQUESTS

Asynchronous SRM requests are a category of SRM calls that return the control to the client as soon as the request has taken in charge by the system. Clients get a request token that can be use to retrieve the status of the request in a second time. srmPrepareToPut, srmPrepareToGet, srmBringOnLine, srmCopy belong to this category. For this type of requests, the FE insert the SRM data into the Database and the BackEnd retrieve the new request to execute with a polling mechanism. The BE process the SRM request and updates the information into the DB. The FE manages also the srmStatusOf[PtG-PtP-etc] request simply querying the status of the request into the database.

3 BACKEND COMPONENT

Section **in progress**.

The Back-End is the core of the StoRM service, it executes all SRM functionalities. It takes care of file and space metadata management, enforces authorization permissions on files and interacts with external Grid services. It is mainly written in Java.

3.1 INTERNAL MACRO COMPONENT

List of Backend internal macro component:

- Asynchronous request manager
- Synch request manager
- XML-RPC server

- Persistence manager
- Namespace component
- Authorization component
- Filesystem manager

3.2 XML-RPC SERVER

The Backend hosts an XML-RPC server to manage direct communications with the FE services. A proper network configuration is needed to make accessible the XML-RPC port (8080 by default) from all the FE hosts. For security reasons, the XML-RPC server is configured to allow connections only from specified IPs. The IPs enabled by default are localhost and the resolved hostname of the FE machines. An high view on this architecture is shown in fig. 3. All this information are configurable, see the BE Configuration section for more details.

3.3 SRM REQUESTS FROM DATABASE

The Picker component retrieves the specified amount of new SRM requests from the Database at each time interval, and forward them to a Scheduler. The Scheduler takes care of forward the request to the rigth worker pool manager depending on the request type. Each single SRM request is then assigned to a worker thread as a new task to be executed. The request status is updated into the Database with all the information concerning request results, error and other data. This data are accessible from the FE to answer to a srmStatusOf* requests. This pattern is shown in fig 6. Most of the parameters characterizing this architecture are configurable, see the BE Configuration section for more details.

3.4 FILE SYSTEM DRIVER

StoRM interacts with the different file systems supported through a driver mechanism, as shown in Fig. 3. This driver are native library written mainly in C/C++, since most of the file system provides C library for the advanced API. StoRM BE uses JNI to connect with drivers. The functionalities provided by this driver are:

- ACL management
- Space management

The driver available with StoRM v1.4. are:

- posixfs Generic driver for posix file system. It relies on the standard `setfacl()`, `getfacl()` syscall for ACL management, and it does not provides any advanced space management capabilities.
- gpfs GPFS specific driver that relies on GPFS advanced api, such as `gpfs_prealloc` for space management and `gpfs_set_acl()` for ACL management.
- xfs XFS specific driver that uses prealloc capability for space management.

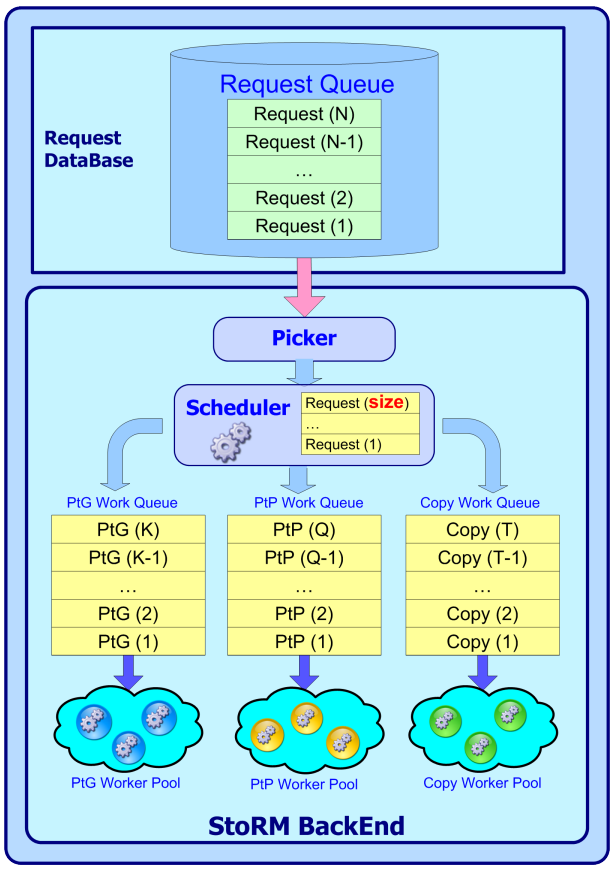

Figure 4: Retrieving new SRM requests from database

This driver mechanism implement a common interface and decouple StoRM internal logic from the different functionalities provided by the underlying storage system. The driver are loaded at run time following the storage namespace configuration (see the BE Configuration section for more details). A single StoRM sever is able to work on different file system at the same time, and with this flexible approach it can be easliy adapted to support new kind of file systems or other storage resources (see the work on Amazon S3 (cite needed))

4 SECURITY

StoRM provides a strong and flexible security mechanism that is able to fulfill requirement from several different scenario.

4.1 CREDENTIAL MANAGEMENT

StoRM rely on user credential for whot concern user authentication and authorization. StoRM is able to support VOMS extension, and to use that to define access policy (complete VOMS-awareness)

4.2 USER ACCESS MANAGEMENT

There are several steps StoRM does to manage access to file:

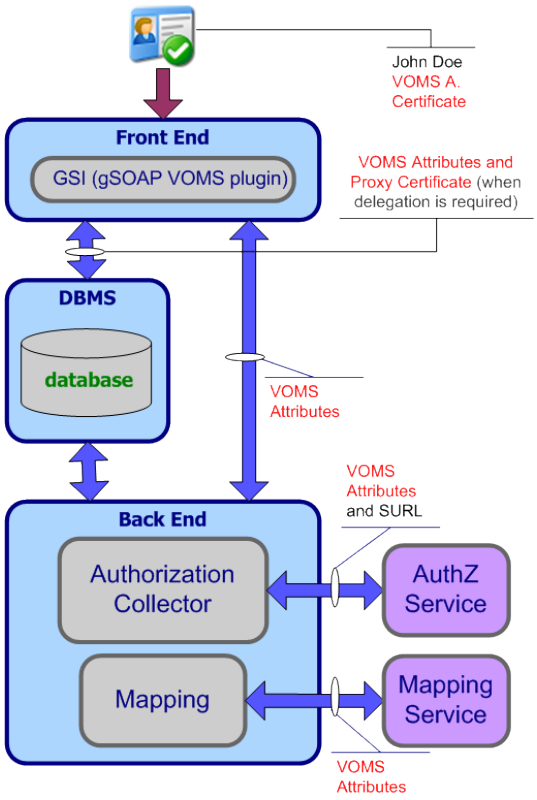

Figure 5: Credential management in StoRM

1. User makes a request with his proxy (hopefully with VOMS extensions)
2. StoRM checks if the user can perform the requested operation on the required resource
3. StoRM ask user mapping to the LCMAPS service
4. StoRM enforce a real ACL on the file and directories requested
5. Jobs running on behalf of the user can perform a direct access on the data

4.3 PERMISSION ENFORCEMENT: JIT OR AOT

StoRM queries the LCMAPs service passing user credential to get information on the local `user id` and `group id` a certain Grid user have to be mapped in according to his identiy. This mapping have to be coherent with the one that take places on the Computing Element. Once StoRM has the mapping, the enforcement on file and directerories take place in two way, in according with teh configuration of Storage Area expressed in the namespace.xml file.

- **JiT (Just In Time)**. With this approach ACL entry is set up for the **user (uid)**, even if it's a pool account, and it will remain in place only for the duration of the SRM operation. StoRM takes care of removing the ACLe entry when the operation is finished (at PutDone, or ReleaseFile time), or when the pin lifetime expires.
- **AoT (Ahead of Time)**. With this approach ACL entry is set up for the **group (gid)** of the user and it will remain in place for all the file lifetime. This is the standard approach in WLCG community.

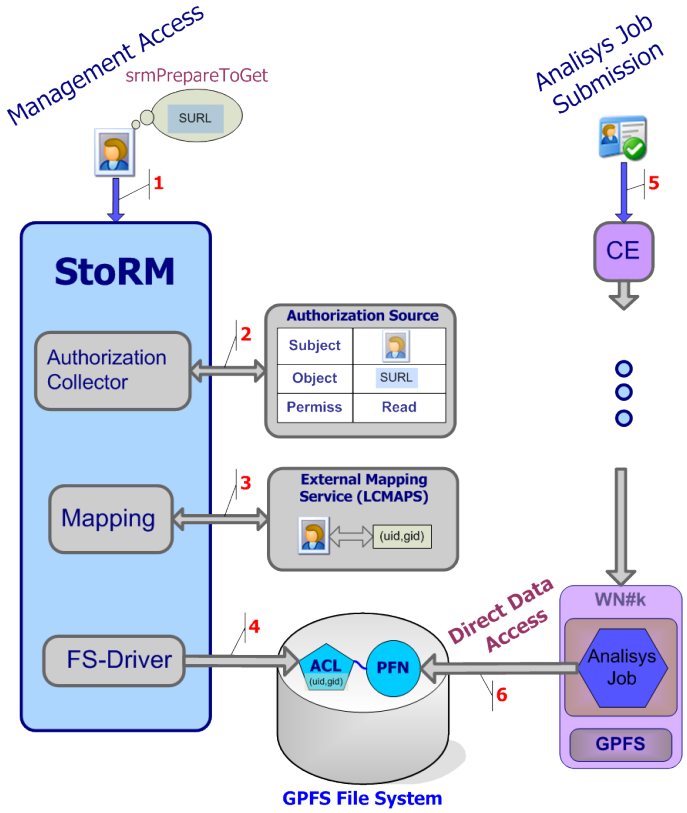

Figure 6: User access management

4.4 StoRM Default ACL

StoRM allows also to define default ACLs, a list of ACL entry that will be applied automatically on each read (srmPrepareToGet) and write (srmPrepareToPut) operation. This is useful in case of experiment use cases, such as the CMS one, that want to allow local access to file on group different than the one that made the SRM request operation. This deafult ACLs have to be set up on the desired storage area in the namespace.xml file, please look at the Configuration chapter for more details.