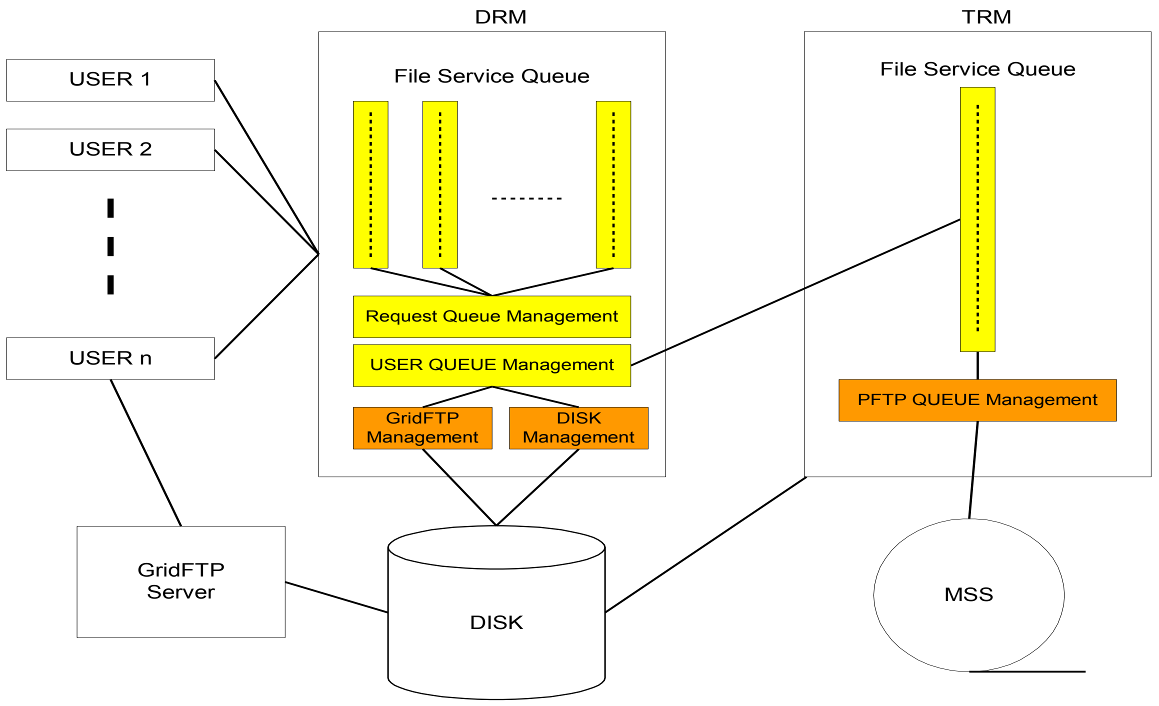
Berkeley Storage Manager (BeStMan)
LBNL implementation of SRM version 2.2

Original Dec. 1, 2003
Revised Mar. 30, 2004
Revised June 27, 2007
Revised July 17, 2009

Introduction
As SRM specification version 2.2 and its corresponding WSDL are defined, we need to develop a new version of our SRM. This new version is based on WSDL/SOAP over http with GSI (httpg). The new SRM specification contains many new features, and there is very small intersection with our current CORBA based SRM v1.1 specification. Rather than modifying and adding features in the current CORBA based implementations, we need to develop our new version of SRM from scratch. We describe the advantages and complexities in the new development in the following sections.

History
Our current implementation of HRM (SRM) consists of DRM and TRM: DRM manages users, requests, queues, file transfers and disk spaces. TRM manages MSS accesses. This design came from the old STACS in Grand Challenge that had Query Manager (QM) and Cache Manager (CM). At the time we developed HRM, we used many of ideas and some codes from STACS.

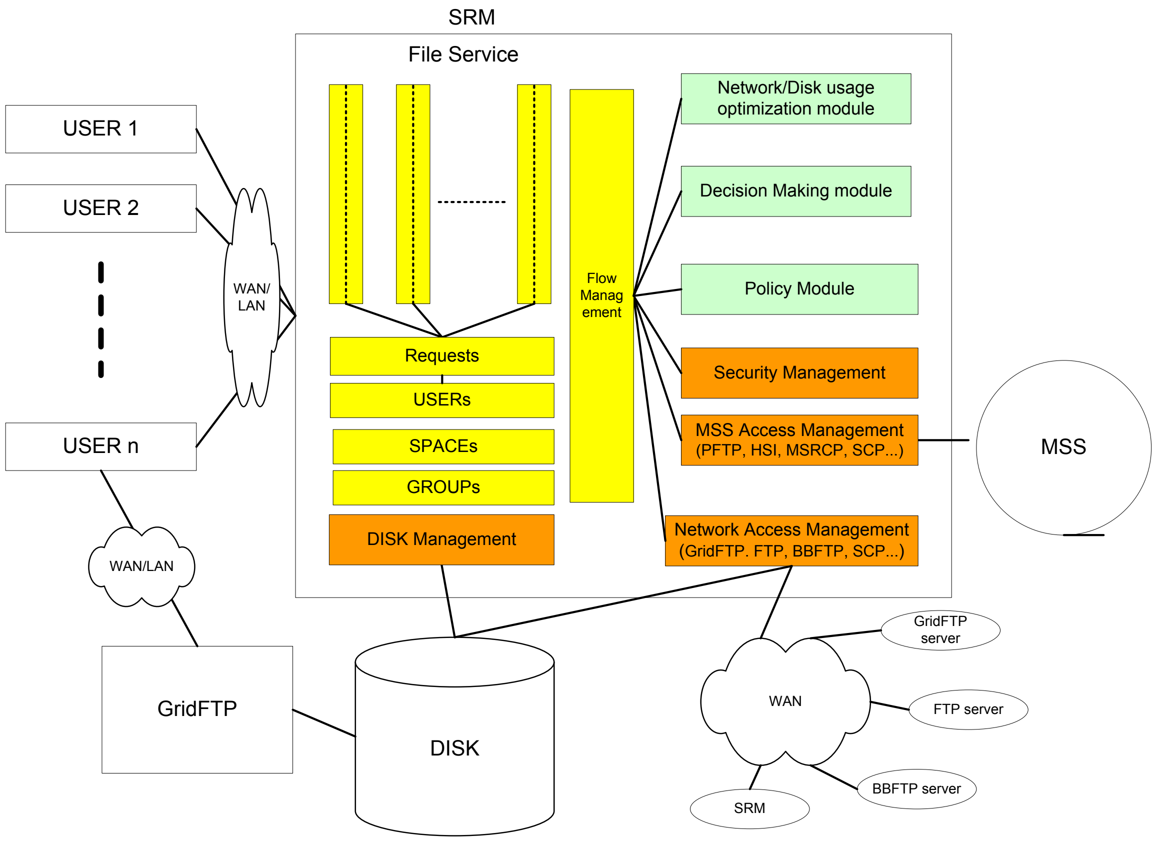


DRM manages different file services queue per user according to the user's disk quota as well as the number of pin limits and other criteria. If there are multiple users, it does round robin to serve different user. On the other hand, TRM has one simple queue to connect to MSS based on the PFTP limits (in case of HPSS). We had an assumption that there are multiple users, but access to MSS is always through one group account.

What do we need to see in the next version of our SRM, not in the current implementation:

1. We need some way for users to plug-in their codes; e.g. filtering program, proprietary MSS connection.
2. MSS access (TRM) needs to have a file service queue per request and per user, as well as the whole queue management, to serve multiple users with different logins,
3. When DRM and TRM are running on the same machine, we need some coordination between DRM and TRM for the number of outgoing connections and incoming connections It is because we have the network connections to MSS from TRM, GridFTP connections from clients, and GridFTP connections by DRM. If there is no synchronization among these network connections, we will have a saturated network problem that degrades the whole machine performance as well as the network performance that affects HRM performance.
4. One related problem to item 3 is in the management of disk i/o. Currently user's gridftp performance (disk reading) on the DRM disk cache is significantly degraded by the TRM's disk access for MSS archiving (disk reading) or staging (disk writing). For example, when a user performs a gridftp to read out a file from DRM's cache, and little later when TRM starts to stage a file from MSS to DRM's cache, the user's GridFTP performance before and after the TRM's staging process has very significant differences (by 50% or more in most cases).
5. Number of incoming file transfers (typically gsiftp) needs to be coordinated dynamically with the archiving data rates, specially when network transfer rate is low. We need to collect network speed for each site as progresses and adjust the number of file transfers automatically depending on the transfer rates. (Policy)
6. This is a team effort.

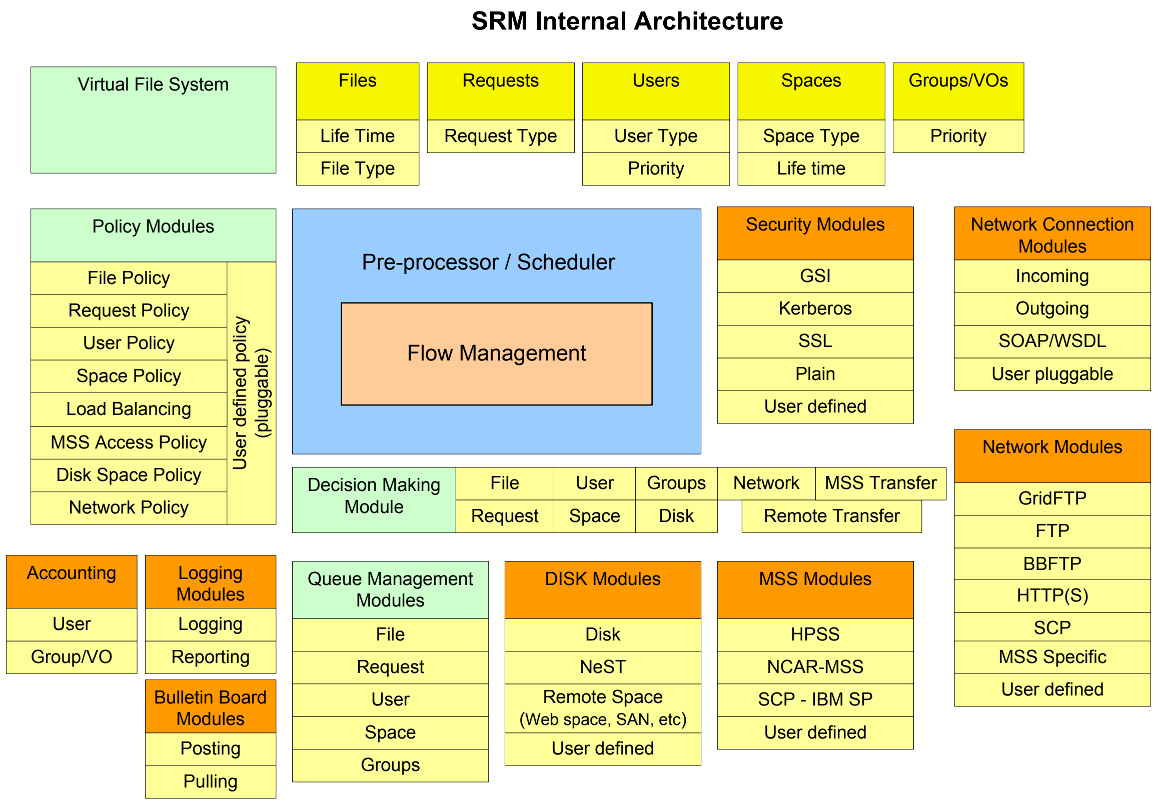
Proposed design:



Advantages of new design

1. Simpler design
2. Easy to maintain by modularization, and easier deployment
    a. Configurable by the options during the installation
    b. Enable us to provide user defined modules
    c. Less dependency on one developer
3. The coordination of network connections and disk i/o between different components can be removed as a separate component.
4. One service queue management with local policy
5. Easy to add local policy with pluggable end
    a. Enable us to support user defined policy
6. Overall amount of work can be reduced by modularity
7. More user appealing features via abstraction of interface
    a. pluggable user defined policies
    b. pluggable site specific software or hardware
8. More flexibility in the user community by giving options in the SRM configuration
9. Still “Hierarchical Resource Manager” because SRM manages disks and MSS in a hierarchical fashion, when MSS access is configured
10. Will use Java 1.4.x with JNI for C/C++ only libraries if necessary
    a. alternative to globus container would be possible
    b. can be more graphical on the reporting

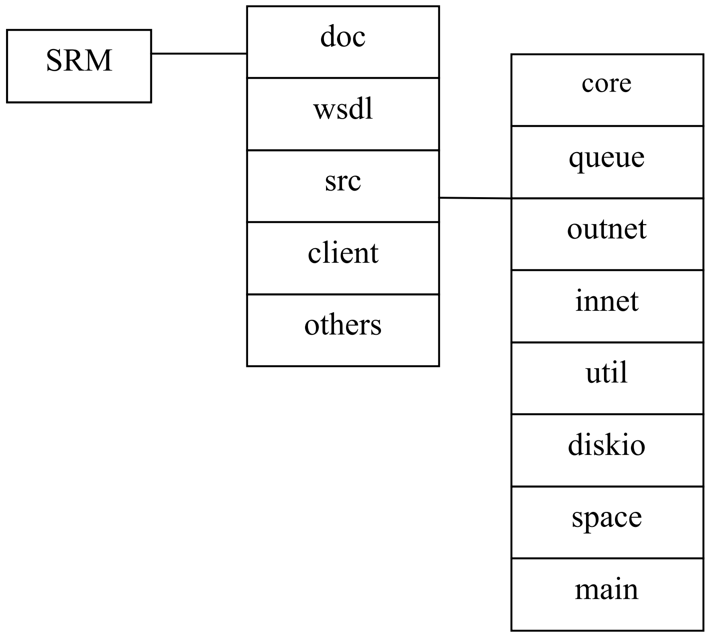
General SRM Internal Design



SRM

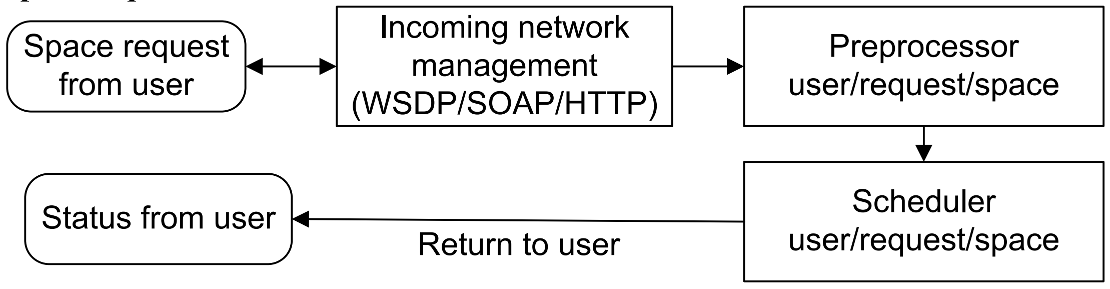
- Core: files, space, request, user
- Queue management: policy, outgoing network, user, request, space, file
- Outgoing Network Management (protocol): ftp, http, gsiftp, srm, pftp, mss, hsi, scp, ssh, nest, (udp)
- Disk I/O: disk path, I/O
- Utility: logging
- Incoming network management: http
- Interface management: WSDL/SOAP
- Space management: reservation, compact, …
- Main: workflow

Directory structure

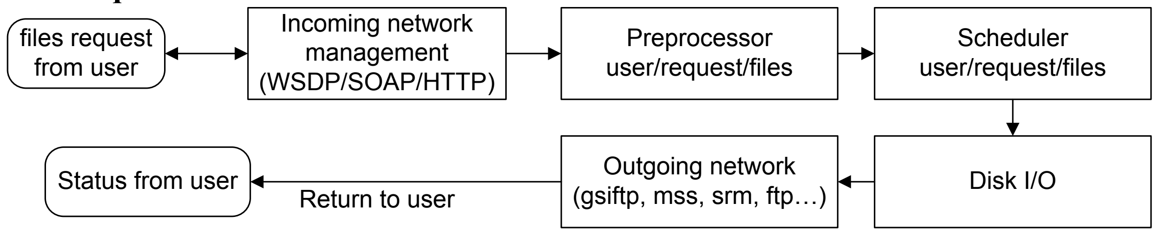


Typical request flow

Space request


Space request from user
Incoming network management (WSDP/SOAP/HTTP)
Preprocessor user/request/space
Scheduler user/request/space
Return to user
Status from user

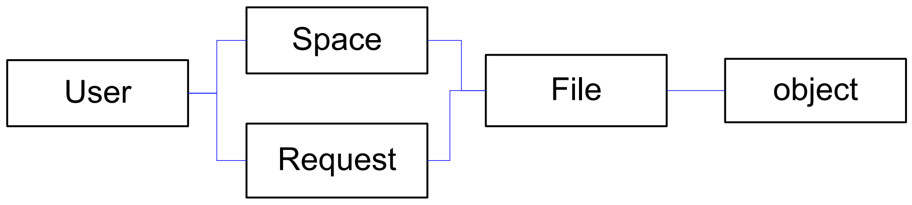
Files request


files request from user
Incoming network management (WSDP/SOAP/HTTP)
Preprocessor user/request/files
Scheduler user/request/files
Disk I/O
Outgoing network (gsiftp, mss, srm, ftp…)
Return to user
Status from user

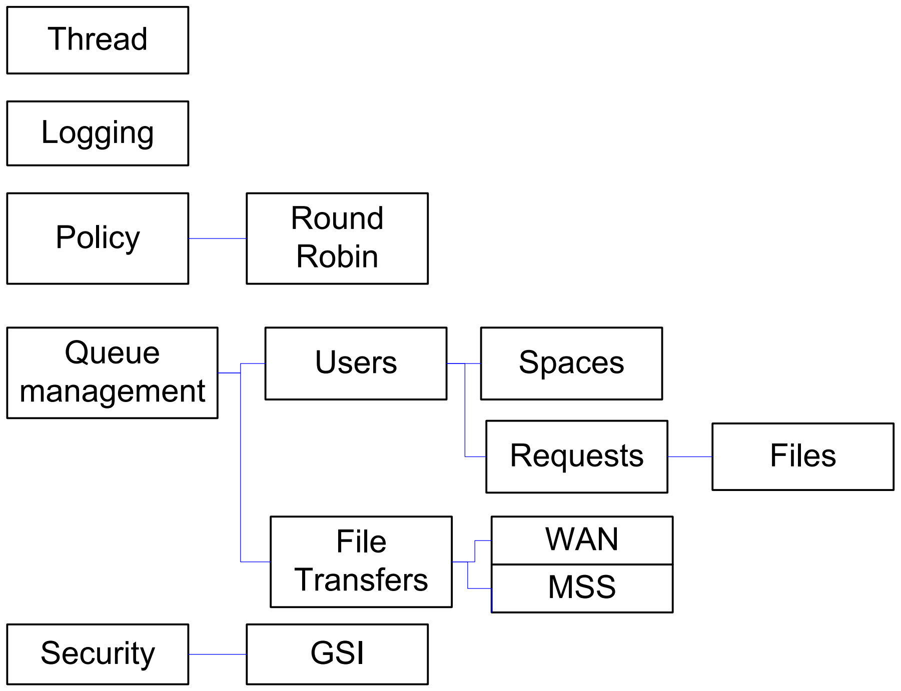
Developmental plan

- Each box (class) would have a test program.
- Dates are tentative schedules.
- All the developmental documents are done as progresses, including documents for each functional interface.
- WSDL development needs to some to be agreed among collaborators, so has later dates.

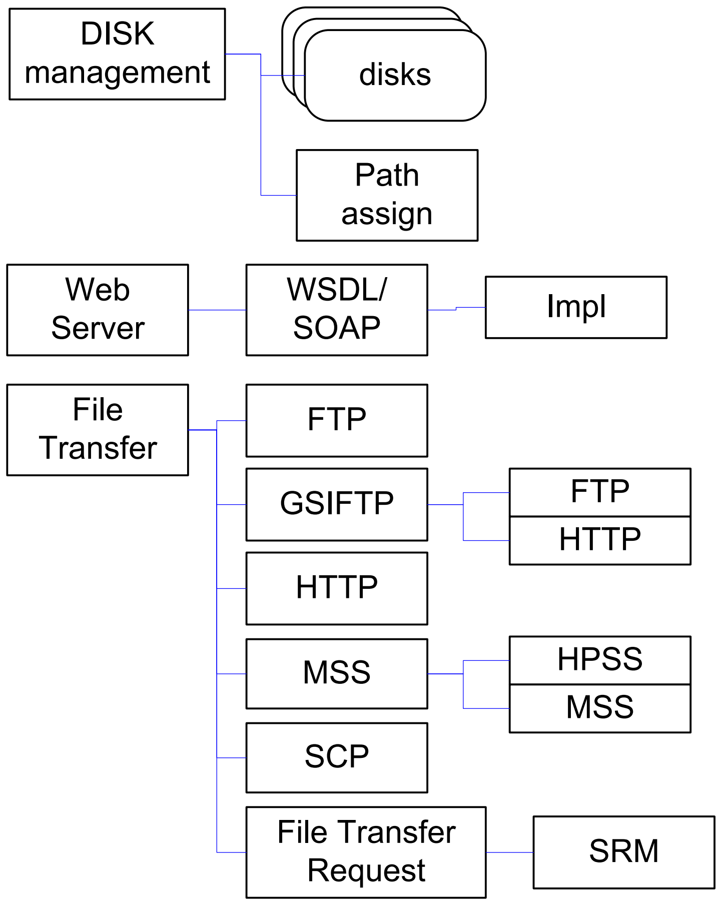
1. Core



2. Utility
   Policy
   Queue
   Security



3. Disk Management
   Network
   File Transfers



Implementation details

Policy Related

Space:

On Reservation:

- In first implementation, a fixed size & lifetime is given to all users.
- Space is granted based on needs. (best effort)

Types of space to support:

- Volatile (Yes)
- Permanent (Yes, on either MSS or disk)
- Durable (Yes)
  - When space is needed, expired files will be archived and admin will be notified. If MSS exists, will store archived files there.
  - Recovering is left between admin and user. Later implementation will deal with smart recovering.

For reusing Volatile and Durable Space:

- A threshold is set to each space, when it is reached, SRM will start cleaning up the expired files. For example, Volatile space can set the threshold to 100% and Durable to be 80% full, because archiving expired files takes time to finish.
- When space is needed for Volatile space, a lazy removing policy is used.
- When space is needed for Durable space, all files in an expired space token are removed.

What if:

- For a Get request to a Volatile space, the lifetime of the space expires before the request is finished?
  - We can reduce the space size and extend the lifetime to allow the request to finish
  - In later versions, we can treat users differently depends on how active they are with the finished files, and how busy the system is.
- The space in the above scenario is Durable?
  - Request fails.
- A Get request comes with an invalid token?
  - SRM refuses it. SRM will not do automatic space reservation
- A Get request comes with some file sizes exceed the size of space?
  - File sizes in a request are just advisory.
- User asks for a file that is in his space(not expired) but file is expired?
  - New lifetime will be assigned to the file.
- Space is empty after compact()?

Space will be automatically released.
Clarification:

- ChangeFileType(), upon success, will not do deallocation to the file's original space.
- ChangeFileType() will not allocate space automatically. User needs to provide a valid spaceToken to ensure the function runs successfully.
- Mkdir() and reserveSpace() are not related. In other words, a user is allowed to call mkDir() without having a space token in SRM

LBNL SRM:

- If spaces(Virtual/Durable/Permanent) involved in changeFileType() are using the same file system, then SRM will just need to do flag switch.
- If the spaces are not using the same file system, then changeFileType() implies copy the file to the designated space type, and removing the file from the original space type
- A Virtual File System will be used to mantain user's directory structure.
- We will derive a uid from proxy's DN by keeping the chars and digits in it. User's top directory is assumed to be ~uid
- SrmMkDir() will assume user's top directory. I.e. srmMkdir("tmp") is equivalent to srmMkdir("uid/tmp") where uid is the user's uid in SRM from his DN.
- Will support file:/ protocol in addition to srm:// protocol in get(), put() and copy()
- User priority will not be supported in first implementation. It will be considered in future implementations. For example, support Top/Normal priority types.

- We will use round robin (RR) between users and within each user's requests. If a user wants his request being served one by one, he has to submit a new one after the existing one is done.
- MSS policy:
  - Will issue calls to check tape-id periodically for a segment of MSS requests.

Enforcements:

- General:
  - MaxFileRequestsPerSRM
  - MaxNumberOfUsers
  - <MaxFileRequestsPerUser = MaxFilesRequestsPerSRM/MaxNumberOfUsers>
  - MaxConcurrentFTP(including MSS transfers)
  - ConcurrencyLevel(max num of threads)
- MSS:
  - Total MSS connection per SRM
  - Total MSS connection per user
- File replacement policy:
  - Latest Recently Used (LRU)
  - May use other ones in later versions
- Load Balancing policy (not sure whether it will be in first implementation)
  - E.g. keep as many MSS transfers going as possible since it is slow and would likely be the bottleneck.
  - Assigns priorities to transfer protocols.
- Network Policy:
  - MaxNumberOfHTTPConnections
- Disk IO policy: (not clearly understood yet)
  - MaxConcurrentReads
  - MaxConcurrentWrites
  - MaxConcurrentReadsAndWrites

Not discussed yet:

- What functions are we NOT going to support?
- What about limit of pin extensions

Network/Disk optimization

Definitions:

Incoming: writing into the disk
Outgoing: reading out of the disk

- Entities needed: dynamic per disk or per partition
  - Number of gridftp granted to be read from the disk (outgoing)
    - Buffer size, number of parallel
  - Number of gridftp writing into the disk (incoming)
    - Buffer size, number of parallel
  - Previous gridftp transfer rates (for all incoming) : moving averaged
  - Number of MSS retrieving processes (incoming, writing into the disk)

        - Previous MSS transfer rates for all incoming : moving averaged
    - Number of MSS archiving processes (outgoing, reading from the disk)
        - Previous MSS transfer rates for all outgoing : moving averaged
    - Number of total users
    - Number of total requests
    - Number of total files
- Entities needed: static or near static
    - Max number of HTTP connection for requests
    - Max number of MSS transfers
    - Max number of incoming network transfers initiated by SRM
        - (all gridftp, ftp, http, bbftp, etc…)
    - Max number of outgoing network transfers initiated by the clients
    - Number of disks or partitions
        - Number of directories does not count
    - Max bandwidth of the hosted network
    - Buffer size of the gridftp server on the host
    - Number of CPUs
    - Physical memory size

Queue Management and Memory Usage

- When the request comes in, we do NOT want to create File objects because of the memory usage. Instead, File information should be kept around until round-robin with quota takes the turn on the request for additional files to be processed.
- There should be a limit to the size of all queues.

BeStMan-G: Simple SRM Gateway

Introduction

With a growing need of SRM in storage services, a simple but compatible and interoperable SRM interface gateway layer is necessary. BeStMan-G could be implemented as G stands for Gateway.

Idea

Leave only very necessary responses to the client calls from the BeStMan. We can also provide the skeleton version of BeStMan that people can implement their own internal processing.

Steps

We first make a separate package with independent software installations. When the client calls, most interfaces when it is compiled as is would return SRM_NOT_SUPPORTED but returns proper return structures. We need to require srmPing to retain "BeStMan-G" as part of the name. In the next steps, we need to add minimum implementations on what users/VOs need.
BeStMan-G would be in efficiency and high performance.

New design of BeStMan for the next generation

1. Introduction

Berkeley Storage Manager was born officially in March 1, 2007. There are more than 30 deployments in the last one year through Virtual Data Toolkit and Open Science Grid, and we expect to grow to more deployments in another year. Simplicity and adaptability as well as efficiency and scalability of BeStMan Gateway mode has been a success, and we see different storage and file systems in the backend of Gateway mode. We mostly see BeStMan deployments in Full mode in Earth System Grid and on a few other handfuls of sites. BeStMan SRM client tools are being recognized in many communities as the well-supported SRM clients, and being used actively. Our SRM tester is also recognized as the supported SRM functional testing tool. As our BeStMan and SRM clients and tester are being widely deployed and assume their roles in a storage and file system access gateway, we need to plan our future developmental efforts and our next generation design for other research projects.

2. BeStMan server

One of the main improvements to be released is the tomcat based bestman server, both in Full mode and Gateway mode. This will eliminate our dependency on Globus container, and it will have us move forward without dependency on Globus 3.2 libraries for httpg. It has been a security concern, and performance issues have been brought up in the past. Our dependency on Globus will remain on cog-jglobus, the java library, which does not depend on Globus 3.2. This version will also need to support https in the future as SRM collaboration agreed in May 2009. We plan our tomcat based BeStMan server and our improved SRM clients/tester by the end of 2009, as bestman2. This version would include VOMS validation support and new GUMS XACML server support.

We might have a few new research projects that would use BeStMan in full mode in the next 2 years. While we describe the new research projects, we also describe our needs of the new design of next generation of BeStMan. There are a few requirements in the new project direction:

1. Scalability in managing a few 10s of millions of files in requests and in the managed cache,
2. Robust request status and history management,
3. Consideration of scalable deployment of BeStMan servers,
4. Access Control List management on "controlled" files and directories,
5. Virtualized space management, in separation from the file system, so that managed cache could have the same directory structure as clients directs under a certain directory level,
6. Transfer optimization for channel caching and pipelining, dynamic adjustment of parallelism and concurrency,
7. Modular design for all different possibilities such as policies, protocols, i/o, security, etc. For example,
   a. Transfer server selection module for TURL,
   b. Transfer optimization module,
   c. Transfer protocol support module,
   d. Mass storage system support module,
   e. Security policy module,
   f. Custom i/o for underlying storage or file system,
   g. Call-outs for additional feature process, e.g. checksum calculation at the end of a large request,
8. Administrative interface support

9. Additional Project interface support
10. Easy adaptability to SRM v3.0 specification when available

Possibility of a few new research projects in the coming years involving BeStMan:

1. Network and Storage Provisioning
   Working with BNL's Terapath, BeStMan would have a capability to reserve network and storage bandwidth for end-to-end file transfers. BeStMan may work directly on OSCARS.
2. High-bandwidth network
   BeStMan would coordinate underlying file system and 100 Gbps network. We need scalable deployments and coordination to accommodate 100Gbps network connectivity with underlying distributed file systems.
3. BeStMan for DB
   BeStMan would be a gateway for Relational Database for database entries to be replicated to other sites selectively. Instead of files management, DB entries are manageable entities.

Details of the new design for BeStMan would be further discussed in the future.