WG Internal
OGSA-DMI

SRM Copy Use-case

Alex Sim, Arie SHoshani
Version 1: 21 November 2006

Status of This Document

This document provides information to the Grid community regarding use cases for srmRemoteCopy functionality in the Storage Resource Manager. Distribution is unlimited.

Contents

1. Abstract

This document provides information about the srmRemoteCopy use-cases, and their implementation with only high level details. Typically, large scientific datasets (order of terabytes) are generated at large computational centers, and stored on mass storage systems. However, large subsets of the data need to be moved to facilities available to application scientists for analysis. File replication of thousands of files is a tedious, error prone, but extremely important task in scientific applications. The automation of the file replication task requires automatic space acquisition and reuse, and monitoring the progress of staging thousands of files from the source mass storage system, transferring them over the network, and storing them on target disk systems or archiving them at the target mass storage system. Storage Resource Manager (SRM) technology has been used to achieve robust file replication for several scientific domains. SRMs are now in regular use in High-Energy-Physics and Climate modeling experiments. The SRMs allocate space, monitor the staging, transfer, and archiving of files, and are able to recover from transient failures. Only a single command is necessary to request multi-file replication or the replication of an entire directory.

2. Introduction

Modern supercomputer systems have ushered a new era of scientific exploration. High granularity simulations of scientific phenomena are now possible, exposing details never possible to observe before. The increase in precision has brought about a tremendous increase in the amount of data generated. For example, currently a single Community Climate System Model (CCSM) simulation at a resolution of 280 km for each side of a simulation cell over 100 years generates about 0.75 TBs. The increase of resolution to 70 km along with 3 times higher resolution in time points, and better chemistry in the model is predicted to increase the amount of

data by a factor of 100-1000. Measurements, using high-precision, more sensitive devices, such as devices mounted on satellites, are now producing terabytes of data, and are expected to grow. Experiments, such as high energy physics (HEP) experiments, are already producing hundreds of terabytes of data. Future HEP experiments, such as ATLAS, scheduled to be launched in 2007, are predicted to generate many petabytes of data. Such applications point to the need of supporting robust massive data movement over wide-area networks.

The scientific exploration process typically consists of two phases: data generation and data analysis. In the data generation phase, large volumes of data are generated at supercomputer centers or collected by experiments, and stored on large mass storage systems (MSSs) that archive data on robotic tape systems. Future MSSs may not have robotic tape storage, but they will still exist. It is predicted that the next generation of storage devices will consist of thousands of disks (disk farms) each holding a few terabytes of data. The MSSs will continue to be the primary storage facilities of these huge datasets, and will require routine maintenance. In the data analysis phase, large subsets of the data need to be moved to one or more analysis sites, which can consist of a large regional center or a small cluster at some university. This process of moving hundreds of gigabyes to a few terabytes to the scientist for analysis turns out to be one of the more tedious, time consuming tasks that wastes the scientist's time. Why is this seemingly simple, boring task so difficult? Why aren't there simple solutions available?

One of the most common practices for moving large data volumes consisting of thousands of files is to write a simple script that reads each file in turn from the source storage system, issues an FTP (File Transfer Protocol) request to transfer the file, and repeats till all the files are moved. The problem with this approach is that the script has to run for hours, and invariably something goes wrong: the mass storage system may have a transient failure or a scheduled maintenance, the network may have a transient failure, power failures may disrupt operations, etc. Thus, the script has to be monitored, the failures discovered, the files already moved need to be checked for their integrity, and the process resumed from the point where it failed.

Another problem is the efficiency of the process. Using simple FTP for large volume of data is very inefficient, because FTP servers are set to break each transfer into small blocks (called windows) of about 2-10 Kbytes. This introduces too much overhead, and therefore larger window sizes (in the order of 1-10 Mbytes) need to be used. Also, one can set an FTP session to support multiple parallel streams to increase throughput. But, most users do not know the details of dealing with such efficiency issues. In addition, getting more than one file concurrently from a mass storage system requires writing a multi-threaded program – again too complex for most scientists. Thus, the transfer process takes longer than necessary even if the network capacity is high.

To complicate matters for the scientists, they have to deal with security issues, as well as firewalls set in front of the various sites. Here again, they need to get help from administrators before they can even proceed to transfer files.

What is needed is a utility that has the following features: (1) a simple way to invoke the file transfer, similar to a "remote copy in recursive mode" in unix (rcp –r) from a directory in one site to a directory in another site; (2) because the transfer may take many hours, this utility needs to be asynchronous; that is, after the call is made and accepted, the user can quit; (3) there needs to be a guarantee that the transfer will succeed even when transient failures of the systems and the network are involved; and (4) there needs to be a dynamic update on the state of the transfer available to the user in order to monitor the progress. These are the basic requirements for an effective srmRemoteCopy function.

Achieving a solution to this problem is a difficult task, especially having to deal with a variety of file systems and mass storage systems. However, to achieve this goal, one can take advantage of software components called Storage Resource Managers (SRMs) [1, 2]. These components were developed for the purpose of supporting access to storage systems on the Grid, but could readily be applied to this difficult problem. An SRM is a software module placed in the vicinity of a storage system; that is, on a machine that is on the same local area network. Since these modules are designed to access mass storage systems as well as disk systems, we can extend

SRMs to make multi-file transfer requests. The main requirement is that entire directories or subdirectories can be moved in a single command, using a simple command-line interface. The advantage of using SRMs is that they all use the same interface (protocol) to communicate with each other regardless of the type of storage system they access. We note that simply using an efficient file transfer service does not address dynamic space allocation or recovery from failures when accessing storage systems.

The srmRemoteCopy function is designed as a high level request to copy multiple files (or a directory). It uses URLs for the source and destination sites and directories. The SRMs use URLs (Uniform Record Locators) to refer to the source and target files (or directories). For example, the URL "srm://hpss.lbl.gov:3003/tmp/fileX" represents a file fileX in the directory tmp of the machine hpss.lbl.gov and managed by the SRM. The SRM is on port 3003. Note that the protocol "srm" is used to specify that file access is managed by an SRM. After the file is staged, the SRM returns a "transfer URL" to the client. The transfer URL contains the location of the file, and the protocol to be used for transfer. For example, if the SRM stages the file to its disk space into location /home/xyz/ on its machine cs.lbl.gov, and chooses to use GridFTP as the transfer protocol, then the transfer URL is: "gridftp://cs.lbl.gov:4004/home/xyz/fileX". This provides the client all the necessary information to issue the GridFTP transfer request.

The implementation of the srmRemoteCopy function can take many forms. It is the choice of an implementation which techniques to use. However, at a high level, all SRM implementations have to allocate space at the target site, manage multiple file transfers concurrently to gain efficiency, monitor file staging, transfer, and archiving for completion, recover from transient failures, and proceed iteratively until the entire request is satisfied. Since the request can be long lasting, it is made asynchronously, and therefore SRMs provide the function srmRemoteCopyRequestStatus.

3. Storage Resource Managers and srmRemoteCopy

Storage Resource Managers (SRMs) are middleware components whose function is to provide dynamic storage space allocation and file management of shared storage components on the Grid. Introductory information about SRM concepts and the design of their functionality can be found in http://sdm.lbl.gov/srm-wg/papers/SRM.book.chapter.pdf. The functionality of srmRemoteCopy is being standardized through the OGF Grid Storage Management Working Group (GSM-WG).

srmRemoteCopy function is used to copy files from source storage sites to target storage sites. Source and targets site URLs (SURLs) pairs can be provided for multi-file copies. For copying entire directories, an SURL pair can be specified from a source directory to a target directory. This specifies a recursive directory replication. If a specified target space token is provided, the files will be placed in the targeted space associated with the space token. Space may be allocated dynamically or reserved ahead of time. This function is asynchronous, and thus a request token is returned. The status may be checked through *srmRemoteCopyRequestStatus* with the returned request token.

3.1 Assumptions

- The fundamental unit of transfer is always the file.
- The VO role is assumed to be outside of the scope of SRM. The client's role is pre-assigned before the client makes a request to the SRM, and SRM will honour the role supported by the VO are known to the SRM.
- The source storage and/or the target storage need to be managed by an SRM.

3.2 srmRemoteCopy Input and output parameters

In	Out

User ID
Authorization ID
Copy File Requests {
Source SURL,
Target SURL,
Directory Option
}
User Request Description
Source Storage System Info
Target Storage System Info
Desired Total Request Time
Desired Target SURL Lifetime
Target File Storage Type
Target Space Token
Target File Retention Policy Info
Flag for streaming
Overwrite Mode
Flag for Removing Source Files
Flag for Checksum
	Request Token
	Request Status {
	Status information
	}
	Remaining Total Request Time
	SURL Statuses {
	Source SURL,
	Target SURL,
	Status information,
	File Size,
	Estimated Wait Time,
	Remaining File Lifetime
	}

(underlined parameters are required)

Input parameters

- *User ID* (required)
  User authentication identifier.
- *Authorization ID*
  User authorization information. The *Authorization ID* may be NULL.
- *Source Storage System Info*
  Information specific to the source storage system that is associated with the *Source SURL*s. The *Source Storage System Info* may be NULL.
- *Target Storage System Info*
  Information specific to the target storage system that is associated with the *Target SURL*s or *Target File Storage Type*. The *Target Storage System Info* may be NULL.
- *User Request Description*
  Description of the request. It may be used for later retrieval of the rquest token.
- *Desired Total Request Time*
  Desired Total Request Time means: if all the file transfer for this request must be complete in this *Desired Total Request Time*. Otherwise, the request is returned as failed at the end of the *Desired Total Request Time*, and an error of SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of an error SRM_FAILURE with an appropriate explanation. All files that whose transfer completed must not be removed, but the status of the files must be returned to the client. If *Desired Total Request Time* is unspecified (specified as NULL), the request will be retried for a duration which is dependent on the SRM implementation. If *Desired Total Request Time* is 0 (zero), each file request will be tried at least once.
- *Desired Target SURL Lifetime*
  Desired life time of the Target SURL in seconds once when the file is in the target SRM. SRM may assign a default lifetime, if not provided.
- *Target File Storage Type*
  Target File Storage Type indicates which type of storage that *Target SURL*s are copied into in target SRM.
- *Target Space Token* (advanced option)
  An advanced option when Space Management feature is supported. A token associated with the space if a particular space in file storage type is to be used. The *Space Token* is acquired separately (e.g. through *srmReserveSpace*).

- *Flag for streaming*
  Boolean indication of *streaming* mode. When *streaming* mode is on, full space at the target storage does not have to be prepared to hold all files in the request.
- *Overwrite Mode*
  SRM needs to replicate the file according to the *Overwrite Mode* on the target that SRM brings the files into.
- *Flag for Removing Source Files*
  Boolean indication of file removal at the source (*Source SURL*) after the copy is performed.
- *Flag for Checksum*
  Flag for Checksum indicates that checksum calculation for all files in the request needs to be performed when files get into its designated target space.
- *Copy File Requests* (required)
  Input parameter *Copy File Requests* consists of SURL information that client wants to copy from one site to another.
  - *Source SURL* (required)
    Source SURL
  - *Target SURL*
    Target SURL
  - *Directory Option* (advanced option)
    An advanced option when Directory Management feature is supported. It includes *Flag for Source Directory, Flag for All Level Recursive* and *Number Of Recursive Levels*: *Flag for Source Directory* is a boolean indicator if *Source SURL* is a directory. *Flag for All Level Recursive* is a boolean indicator if *Source SURL* is a directory and all files under the SURL must be retrieved. *Number Of Recursive Levels* is a positive integer indicator for how many levels of the recursive directory must be browsed and all files in those directories to be retrieved.

Output parameters

- *Request Token* (required)
  Output parameter string token is associated with the request for the later asynchronous status request.
- *Request Status* (required)
  Output reporting the success or failure of the request. Textual description of explanation for what happened to the request, specially in case of any failures, would help client diagnose the case. *Status information* is required and it is recommended to return as an enumerated codes separated as status code and error code, so that when client is a non-human program, response to the status information can be pre-programmed.
- *Remaining Total Request Time*
  Output parameter indicates the remaining total request time.
- *SURL Statuses*
  Output reporting the success or failure of the each SURL requests. *SURL Statuses* may be empty and NULL. If returned to the client, *SURL* and its *Status information* are required to return.
  - *Source SURL* (required)
    SURL that client has requested to copy the file from.
  - *Target SURL* (required)
    SURL that client has requested to copy the file to. If the client did not provide SURL at the time of request, SRM server generates a reference SURL that SRM server and client can refer to the file.
  - *Remaining File Lifetime*
    It indicates the remaining file lifetime on the SURL when the file is copied into its destination target. File lifetime is assigned after file copy is completed.

Behaviors

- Abort: When aborted, client needs to provide *Target SURLs*. When *srmRemoteCopy* is aborted, it is propagated to the remote site by aborting the *srmRrepareToGet* or the *srmPrepareToPut* request, correspondingly. Furthermore, if the abort function has the *remove* flag set, then the propagated abort should have this flag set, too. In the case of copy-push, the *srmPrepareToPut* gets aborted with the *remove* flag set, which has the effect of removing the already-copied files from the remote SRM. In the case of a copy-pull, the *srmPrepareToGet* to the remote site is aborted, but the *remove* flag affects only the local site that already-copied files from the remote SRM to the local storage will be removed.
- Third party copy is not supported, from a remote location to another remote location. Either source or target must be local to the SRM where the request is submitted.
- When *Flag for Removing Source Files* is true, SRM removes the source files on behalf of the client after the copy is done through SRM interface.
- The default value of "lifetime" for volatile or durable file types must be equal to or less than the lifetime left in the space of the corresponding *Space Token*.
- Empty directories must be copied as well.
- If a Target SURL is provided with some directory structure, the directory structure must exist, and SRM will not create the directory structure for the Target SURL. In such case, an error SRM_INVALID_PATH must be returned. *srmMkdir* may be used to create the directory structure.
- If the Source SURL and Target SURL are provided as directories (copying directories) when SRM implementation supports, then all sub directories will be copied over from the source to the target, and complete sub-directory structure will be created only if *Directory Option* indicates them.

3.3 srmRemoteCopyRequestStatus Input and output parameters

In	Out

User ID
Authorization ID
Request Token
Source SURLs
Target SURLs
offset
count
	Request Status {
	Status information
	}
	Remaining Total Request Time
	SURL Statuses {
	Source SURL,
	Target SURL,
	Status information,
	File Size,
	Estimated Wait Time,
	Remaining File Lifetime
	}

(underlined parameters are required)

Input parameters

- *User ID* (required)
  User authentication identifier.
- *Authorization ID*
  User authorization information. The *Authorization ID* may be NULL.
- *Request Token* (required)
  A handle associated with the previously submitted *srmRemoteCopy* request. The *Request Token* was returned by the function initiating the request (*srmRemoteCopy*).
- *Source SURLs*

Selective *Source SURL*s to check the status.

- *Target SURLs*
  Selective *Target SURL*s to check the status.

Output parameters

- *Request Status* (required)
  Reports the success or failure of the request. Textual description of explanation for what happened to the request, especially in case of any failures, would help client diagnose the case. *Status information* is required and it is recommended to return as an enumerated codes separated as status code and error code, so that when client is a non-human program, response to the status information can be pre-programmed.
- *Remaining Total Request Time*
  Output parameter indicates the remaining total request time.
- *SURL Statuses*
  Output reporting the success or failure of the each SURL requests. *SURL Statuses* may be empty and NULL. If returned to the client, *SURL* and its *status information* are required to return.
  - *Source SURL* (required)
    SURL that client has requested to copy the file from.
  - *Target SURL* (required)
    SURL that client has requested to copy the file into. If the client did not provide SURL at the time of request, SRM server generates a reference SURL that SRM server and client can refer to the file.
  - *Remaining File Lifetime*
    It indicates the remaining file lifetime on the Target SURL when the file is in its destination. File lifetime is assigned after file copy is completed.

3.4 Operational Modes

There are two cases for remote copies: 1. Target SRM is where client makes a *srmRemoteCopy* request (PULL mode), 2. Source SRM is where client makes a *srmRemoteCopy* request (PUSH mode).

- PULL mode

Upon the client's *srmRemoteCopy* request, the target SRM prepares a space at the target storage, and makes a request *srmPrepareToGet* to the source SRM. When TURL is ready at the source SRM, the target SRM transfers the file from the source TURL into the prepared target storage. After the file transfer completes, *srmReleaseFiles* is issued to the source SRM.

- PUSH mode

Upon the client's *srmRemoteCopy* request, the source SRM prepares a file to be transferred out to the target SRM, and makes a request *srmPrepareToPut* to the target SRM. When TURL is ready at the target SRM, the source SRM transfers the file from the prepared source into the prepared Target TURL. After the file transfer completes, *srmPutDone* is issued to the target SRM.

4. Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

5. Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

6. Full Copyright Notice