Chapter 20

STORAGE RESOURCE MANAGERS

Essential Components for the Grid

Arie Shoshani, Alexander Sim, and Junmin Gu
Lawrence Berkeley National Laboratory

Abstract

Storage Resource Managers (SRMs) are middleware components whose function is to provide dynamic space allocation and file management of shared storage components on the Grid. They complement Compute Resource Managers and Network Resource Managers in providing storage reservation and dynamic information on storage availability for the planning and execution of a Grid job. SRMs manage two types of resources: space and files. When managing space, SRMs negotiate space allocation with the requesting client, and/or assign default space quotas. When managing files, SRMs allocate space for files, invoke file transfer services to move files into the space, pin files for a certain lifetime, release files upon the clients request, and use file replacement policies to optimize the use of the shared space. SRMs can be designed to provide effective sharing of files, by monitoring the activity of shared files, and make dynamic decisions on which files to replace when space is needed. In addition, SRMs perform automatic garbage collection of unused files by removing selected files whose lifetime has expired when space is needed. In this chapter we discuss the design considerations for SRMs, their functionality, and their interfaces. We demonstrate the use of SRMs with several examples of real implementations that are in use today in a routine fashion or in a prototype form.

Keywords: storage resources, storage elements, storage management

1. INTRODUCTION

The grand vision of the Grid is to provide middleware services that give a client of the Grid the illusion that all the compute and storage resources needed for their jobs are running on their local system. This implies that the client only logs in and gets authenticated once, and that the middleware software figures out what is the most efficient way to run the job, reserves compute, network, and storage resources, and executes the request. Initially, the Grid was envisioned as a way to share large compute facilities, sending jobs to be executed at remote computational sites. For this reason, the Grid was referred to as a *Computational Grid*. However, very large jobs are often data intensive, and in such cases it may be necessary to move the job to where the data sites are in order to achieve better efficiency. Thus, the term *Data Grid* was used to emphasize applications that produce and consume large volumes of data. In some applications, the volume of data is so large (in the order of hundreds of gigabytes to terabytes) that partial replication of the data is performed ahead of time to sites where the computation is expected to take place.

In reality, most large jobs that require Grid services, especially in the scientific domain, involve the generation of large datasets, the consumption of large datasets, or both. Whether one refers to the Grid as a Computational Grid or a Data Grid, one needs to deal with the reservation and scheduling of storage resources when large volumes of data are involved, similar to the reservation and scheduling of compute and network resources.

In addition to storage resources, Storage Resource Managers (SRMs) also need to be concerned with the *data resource* (or files that hold the data). A data resource is a chunk of data that can be shared by more than one client. For the sake of simplifying the discussion, we assume that the granularity of the data resource is a file. In some applications there may be hundreds of clients interested in the same subset of files when they perform data analysis. Thus, the management of shared files on a shared storage resource is also an important aspect of SRMs. In particular, when a file has to be stored in the storage resource that an SRM manages, the SRM needs to allocate space for the file, and if necessary remove another file (or files) to make space for it. Thus, the SRM manages both the space and the content of that space. The decision of which files to keep in the storage resource is dependent on the cost of bringing the file from some remote system, the size of the file, and the usage level of that file. The role of the SRM is to manage the space under its control in a way that is most cost beneficial to the community of clients it serves.

In general, an SRM can be defined as a middleware component that manages the dynamic use of a storage resource on the Grid. This means that space can be allocated dynamically to a client, and that the decision of which files to keep in the storage space is controlled dynamically by the SRM.

The initial concepts of SRMs were introduced in [SSG02]. In this chapter we expand on the concepts and functionality of SRMs. As described in [SSG02], the concept of a storage resource is flexible; an SRM could be managing a disk cache (we refer to this as a Disk Resource Manager - DRM), or managing a tape archiving system (Tape Resource Manager - TRM), or a combination of both, called a Hierarchical Resource Manager (HRM). Further, an SRM at a certain site can manage multiple storage resources, thus have the flexibility to be able to store each file at any of several physical storage systems it manages or even to replicate the files in several storage systems at that site. The SRMs do not perform file transfers, but can invoke middleware components that perform file transfers, such as GridFTP [ABB+02b].

All SRMs have the same uniform interface, thus allowing any current or future hardware configurations to be addressed in the same manner. For example, an archive does not have to be a robotic tape system; a disk system could serve as archival storage as well. As far as the client is concerned, getting a file into an SRM-managed space may incur a delay because the file is being retrieved from a tape system or from a remote site over the network, or both.

This chapter is structured as follows: Section 2 overviews the basic functionality of SRMS by walking through several related scenarios. The concepts of permanent, volatile and durable files are defined in Section 3. Section 4 describes how to manage space reservations, and Section 5 details the concepts of negotiations, site and transfer URLs, and the semantics of file pinning. Section 6 describes several projects currently using SRMs in practice. We conclude in Section 7.

2. THE FUNCTIONALITY OF STORAGE RESOURCE MANAGERS

2.1 Example Scenarios

To illustrate the functionality that SRM services expose to a client, let us consider a series of related scenarios. Each will illustrate increasingly more complex functionality required of the SRMs.

2.1.1 Local Data Consumption by a Single Client

Consider a simple scenario where a set of requested files are to be moved to the clients system. Let’s assume that the client needs files from several locations on the Grid, that the client knows exactly the site of each file (the machine name and port), and the physical location of the file (i.e. directory and file name). In this case, the client can go directly to each location and get each file. In this case, the client needs to monitor the file transfers to make sure that each is completed successfully, and to recover from failures.

Now, suppose that the client wants to get 500 files of 1 GB each, but the amount of available space on the local disk is only 50 GBs. Suppose further that the client can process each file independently of the other files, and therefore it is possible to bring in files incrementally. The client has the burden of bringing in the files, monitoring the file transfers, recovering from failures, keeping track of which files were brought in, removing each file after it is processed to make space for additional files, and keeping track of available space.

An SRM can be used to alleviate this burden of dealing with a large number of files by providing a service where the client issues a single request for all 500 files. The SRM deals with dynamic space allocation, removal of files, and recovering from transient failures. This is illustrated schematically in Figure 1. Note that the local storage system is a disk, and that the two remote systems shown are a disk, and a mass storage system that includes a robotic tape. The solid line from the client to DRM represents a multi-file request. The dashed line from the client to the disk represents file access (open, read, write, close, copy, etc.).

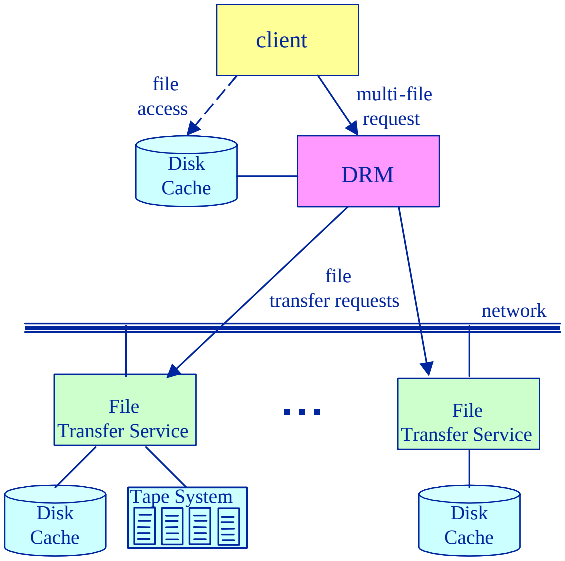

Figure 20.1. A multi-file request service by an SRM.

Each file in the multi-file request can be represented as a Uniform Resource Identifier (URI) given that the protocol to get the file is known (such as GridFTP, ftp [PR85], http [FGM+99], etc.). For example, `gridftp://` `\verb`+cs.berkeley.edu:4004/tmp/fileX+ represents a file `fileX` in the directory `tmp` of the machine `cs.Berkeley.edu` and the `gridftp` transfer protocol that uses port 4004. In this manner, the multi-file request can be composed of files on different systems.

The functionality provided by the SRM to support such a multi-file request includes: (a) queuing the set of files requested, (b) keeping track of the files in the disk cache, (c) allocating space for each file to be brought in, (d) if the requested file is already in cache because of a previous use, mark it as needed, so it is not removed by the SRM, (e) if the file is not in cache, invoke a file transfer service to get the file.

2.1.2 Shared Local Data Consumption by Multiple Clients

Figure 2 shows a similar situation to Figure 1, except that multiple clients share the local disk. This situation is not unusual, especially for applications that need to process a large amount of data and it is prohibitive to provide each client with a dedicated large disk. Another advantage of sharing a disk cache is that files accessed by one client can be shared by other clients. In this case, the file does not have to be brought in from the remote site again, saving time to the clients and unnecessary traffic on the network. Of course, this can be done for read-only files or when the master file has not been updated since the last read.

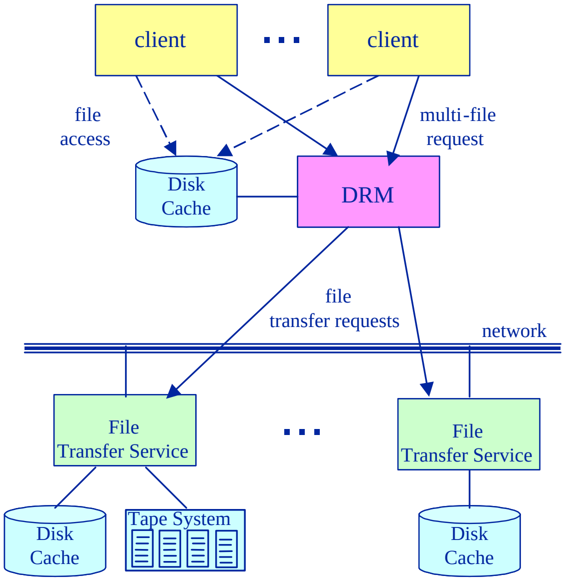

Figure 20.2. Sharing a DRM-managed disk cache.

Even with this simple scenario, one can already notice an important concept emerging: The concept of *file pinning*. When a file is brought into the disk cache, there is no guarantee that it will stay there, because the space may be needed by other clients. Thus, in order to keep the file in cache, it is necessary for the SRM to *pin* the file. It is also necessary to provide the client with a release pin call so that the SRM can unpin the file. Another related concept that needs to be supported is the concept of a *lifetime of a pin*. This is necessary in case that a client neglects to *release* (or *unpin*) the file. This can happen if the clients program crashes, goes into a loop, or simply terminates without releasing the file. The lifetime of a file is a necessary feature for automatic garbage collection of unused files.

The above concepts are meaningful when dealing with shared temporary space, which we refer to as *volatile* space. We use the term volatile rather than temporary because there is a guarantee that files will not be removed so long as they are pinned. As we'll see in the next sub-section, other types of files and spaces need to be supported as well.

There are several implications to the functionality described above. First, in order to support multi-file requests for multiple users it is necessary to introduce the concept of a *quota* per user. The management of the quota depends on the policy of the specific SRM. It could be fixed, or change dynamically based on the number of clients at the time a request is made. Second, a *service policy* must be supported. A good service policy will make sure that all requests are serviced in a fair manner, so that no one request is starved. Third, since files have to be removed when space is needed, a *replacement policy* has to be provided, whose function is to determine which file (or files) to evict in order to make space for a file that needs to be brought in. A good replacement policy would maximize the use of shared files. There are many papers discussing good replacement policies in the domain of web caching [AY97, CI97, DFJ$^+$96, Gue99] but these policies do not take into account the cost of bringing in the file if the SRM needs to get it again. Recent work has shown that different policies need to be used when considering large files over a Grid [OOS02].

2.1.3 Pinning Remote Files

When a file is needed from a remote site, there is no guarantee that the file will be there when the file transfer starts, or whether the file is deleted or modified in the middle of the file transfer. Normally, archived files are not removed and this issue does not come up. However, on the Grid there may be replicas that are stored temporarily in shared disk caches that may be removed at any time. For example, one model of the Grid is to have a tier architecture where the long term archive is a Tier 0 site, a shared disk cache may exist in a Tier 1 regional site (such as Southern Europe), a shared disk may exist at a Tier 2 local site (such as the Lyon area), and a Tier 3 disk cache may exist at some university. The tier architecture is designed so that files are always accessed from the closest tier if possible. However, over time the content of the intermediary shared disk caches change depending on the access patterns. Therefore, it is possible to have a file removed just before transfer is requested or while the transfer is in progress.

To avoid this possibility, and also to make sure that files are not removed prematurely, one can use the same pinning concept for remote files as well. Thus, remote sites that have SRMs managing their storage resources can be requested to pin a file. This is shown in Figure 3, where the remote sites have SRMs associated with them.

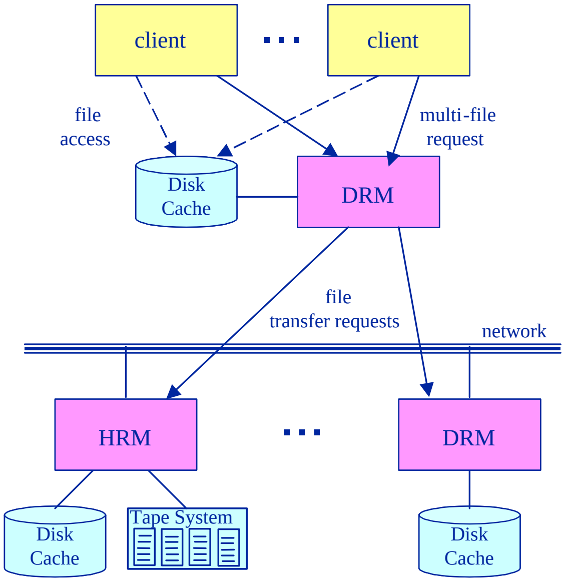

Figure 20.3. Accessing remote SRMs.

For every file that needs to be brought into a local site, the following actions take place: 1) the local SRM allocates space, 2) the local SRM requests that the remote SRM pins the file, 3) the remote SRM acknowledges that the file was pinned, and the physical location of the file is returned; this is referred to as the *transfer URL*, 4) the local SRM invokes the file transfer service, and 5) upon successful completion of the transfer, the local SRM notifies the remote SRM to release the pin.

2.2 Where do SRMs Fit in a Grid Architecture?

Running jobs on the Grid requires the coordination of multiple middleware components. It is convenient to think of these components as layered. One popular view of layering of the Grid services is described in [FKT01]. There are five layers, labeled: fabric, connectivity, resource, collective, and application. Typically, an application at the top layer makes a request for running a job to the *request manager*, a component at the *collective* layer. The request manager may include a component called a *request planner* that figures out the best way to run the job by consulting metadata catalogs, file replica catalogs, monitoring information (such as the Network Weather Service [WSH99a]), etc. The plan, which can be represented as a workflow graph (referred to as *execution DAG* [DAG] by the Condor project [LL90] described Chapter 9) can then be handed to a *request executer* that will then contact compute resource managers and storage resource managers to allocate resources for executing the job. Figure 4 shows the view where compute and storage resources can be anywhere on the Grid, and the results returned to the clients site.

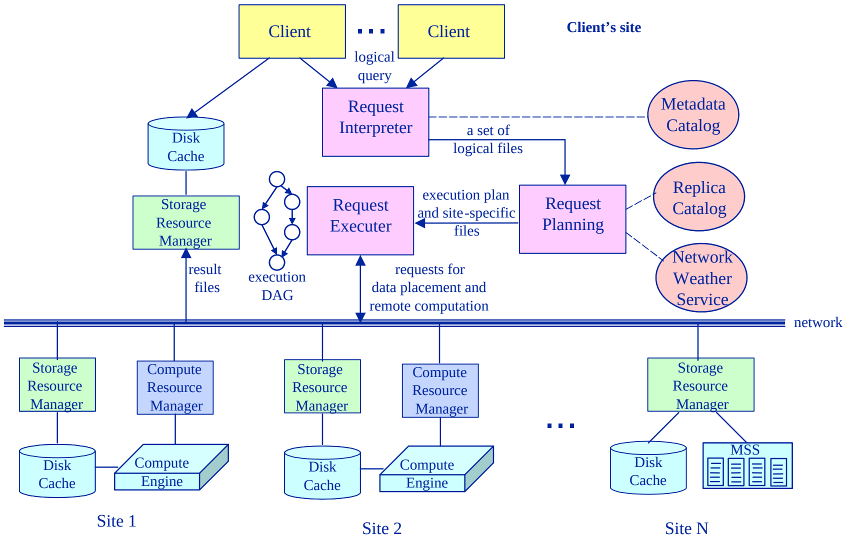

Figure 20.4. Interaction of SRMs with the request executer.

According to this view, SRMs belong in the *resource* layer, in that a component, such as a *request executer* would rely on the SRMs to perform space allocation and file management when requested to do so.

Figure 4 introduces another requirement that SRMs need to support. They can be invoked not only by clients or client applications, but also by other middleware software. This implies that SRMs should be able to report information on how busy they are (in case that they have many files to serve on their queue), how much free space they have, and what is the quota policy they support. Similarly, they should be able to provide information on whether they still have specific files in the managed storage resource.

Considering SRMs as belonging to the *storage layer* is not accurate, since they do provide a limited service of *brokering*, a function usually considered at the *collective* layer. Recall that the SRM can be handed a set of files to be accessed, each with a source URL. If the file is found locally, either because it was placed there permanently or because it was brought in by a previous request, the SRM simply returns the *location* of the file. If it is not available locally, the SRM will contact the source location to pin the file, and then invoke a transfer service to get the file. This last action is a brokering action, which is quite useful to clients. Thus, an SRM as we described it here, can be thought of as being a *local SRM* that provides a *brokering* capability of getting files from remote sites.

The general function of determining the site or sites where to reserve spaces and to move data into the spaces, is referred to as *storage scheduling* and *data placement* services, which are indeed at the collective layer. These should not be confused with SRMs. Rather, their task is to negotiate space reservation, and schedule data movement by requesting such services from multiple SRMs.

3. TYPES OF FILES

The concepts of *permanent* and *temporary* spaces are supported by most shared file systems. A permanent space is a space that a user controls, and only that user can put in and remove files from that space. A temporary space is a shared space that is allocated to a user, but can be reclaimed by the file system. If space is reclaimed, all the files in that space are removed by the file system. The implication is that files in these spaces are also permanent and temporary.

3.1 Permanent and Volatile Files

On the Grid, the same concepts of permanent and temporary can be applied to file types, but temporary files require additional functionality. Temporary files on shared Grid spaces cannot be removed arbitrarily. Some minimal amount of time must be guaranteed by the SRM for the client to rely on. This is the reason for a lifetime of a file. This feature of a lifetime for a file is associated with each user accessing the file. That is, a lifetime is associated with a file for a user when the file is made available to the user. If another user requests the same file later, the file gets a fresh lifetime associated with that user. We refer to a file that is temporary in nature, but has a lifetime guarantee as a *volatile* file. Since volatile files can be shared by multiple users at the discretion of the SRM, volatile files can be thought of as owned by the SRM, and permission to use them by the users is granted by the SRM on a temporary basis enforced by the lifetime. Alternatively, an SRM can choose to replicate files for each user, although this approach is less space efficient. The concept of volatile files is very useful for sharing space, automatic garbage collection, and sharing of files on a temporary basis. Most shared disk caches on the Grid are likely to provide support only for volatile files.

In contrast, a *permanent file* is a file associated with long-term archival storage that may or may not be shared by clients. Similar to file systems, permanent files can only be removed by the owner.

3.2 Durable Files

For Grid applications, one needs another type of a file that has properties of both permanent and volatile files. A *durable* file has the behavior of a volatile file in that it has a lifetime associated with it, but also the behavior of a permanent file in that when the lifetime expires the file is not automatically eligible for removal. Instead, the SRM may advise the client or an administrator that the lifetime expired or take some other action. For example, the SRM may move the file to a permanent space, release the durable space, and notify the client. A durable file type is necessary in order to provide temporary space for files generated by large simulations, which need to be eventually archived. Since the archiving process (for example, into a mass storage system) is usually slower than the data generation process, attempting to move files directly to the archive will slow down and waste computational resources. By storing durable files into available shared disk caches, the simulation can continue efficiently, yet it is guaranteed that the files will not be removed before they are moved to the archive as a secondary task. This is a more reasonable approach for sharing temporary space, while still protecting important files. Similar to volatile files, durable files can be released by the client as soon as the files have been moved to a permanent location.

4. MANAGING SPACE RESERVATIONS

Space reservation is a necessary feature of SRMs since reservations are needed in order to support the request planning and request execution steps of running jobs on the Grid. Space reservation is also needed in order to replicate data to locations where computations take place, or to have space to store the results of computations. However, space reservation brings up many difficult issues. What policies to use when space is requested? Is the space guaranteed? Should the SRM set aside the space requested? What if the space is not used for a long time?

The answer to these questions depends on the cost model used. To support space reservations, there needs to be a way to provide a *ticket* or a *capability* to the user in order to claim the space. There needs to be an authority that manages this capability. There needs to be a mechanism for reporting the space-time consumed. And there needs to be a way of taking away the space if it exceeds the reservation period.

Dynamic space reservation is not a concept generally supported by file systems, but is essential for supporting shared storage resources on the Grid. For example, in the Unix system, space is allocated by the administrator (the root owner) on a long-term basis. A user cannot request additional space dynamically, or release space dynamically. SRMs are designed to provide dynamic space allocation and release.

4.1 Types of Spaces

In order to support space reservations, we found it useful to apply the concepts of permanent, durable, and volatile to spaces as well. The semantics are similar. Permanent space is owned by the user and has an indefinite lifetime. Durable space is also owned by the user, but when a lifetime expires there are two cases to consider. If the durable space does not contain any files, the space is released. If it contains durable files, the system claims all the unused space, and notifies the file owner that the lifetime of the durable files has expired. Volatile space also has a lifetime, but when it expires, all the space and the files in it can be removed by the SRM.

Reserving permanent space is obviously needed for dynamic archival storage reservation. Similarly, reserving volatile space is necessary for shared disk resources that are intended to be used on a temporary basis. However, support for reservation of durable space requires an explanation as to its value.

Recall that durable files are files that have a lifetime associated with them, but the SRM cannot remove them without some explicit action. Durable files are especially useful when a client generates a large number of files that need to be stored temporarily in some shared disk cache before being archived. The purpose of durable space is similar: reserving a guaranteed space on a temporary basis that cannot be removed without an explicit action by the client. Similar to volatile space, durable space can be released by the client. Durable space is particularly useful for request planning.

In contrast to durable space, which is a space that cannot be removed without an explicit action, volatile space can be reclaimed by the SRM if space is needed. For example, suppose that a client asks for 200 GB volatile space when there are very few clients using the SRM. The SRM can initially allocate this space, but when many new clients request space the SRM can reduce the space used by the first client to say, 50 GBs, provided that there are no pinned files in that space.

4.2 Best Effort Space

The concept of *best effort* is well known in association with reservations [BS98]. It stems from the wish to use the resources efficiently, even if there is a way to charge for the space. For example, suppose that an SRM controls some space, and that a client asks and is granted a space reservation of 200 GBs for 5 hours. If this space is reserved and not used by the client, and if as a result some other clients were denied, then the space was wasted, and the client was charged for space that was not used. This can be thought of as fair, but it may not be the clients fault. It could be that another resource that was scheduled (such as a compute resource) was not available. A good request planner would have released the space, but this cannot be depended on. The result is that the allocation was lost, and the space was wasted.

Best effort is an alternative that requires some flexibility on the part of the client, but makes more efficient use of resources. The reservation is considered *advisory*. The SRM will try to honor the request within the reservation time window. Space is only allocated as it is claimed, i.e. when files move into the space. If competition is high, the client may not get all the space requested, and may need to look for additional space elsewhere dynamically. This is difficult for human beings to manage, but may be perfectly reasonable to expect from a request planner since in general it needs to deal with various failure modes on the Grid (such as a temporary network partition, archive temporarily down, unscheduled maintenance, etc.)

Best effort can be flexible in that some minimum is guaranteed. This minimum can vary with the type of space. For example, a durable space should have priority in honoring the reservation over volatile space. Here again, the policy of best effort reservation is a local policy for each SRM.

The above discussion implies that in order for a request planner to optimize resource usage, it needs to find out the policies provided by various SRMs it might want to use. The general policies can be advertised, but the SRMs need to provide dynamic information as well, such as the quota currently available in the case that a quota policy is variable. These issues are quite complex, and so far there are no standard methods for advertising and requesting policy and quota information.

4.3 Assignment of Files to Spaces

Can a volatile file reside in permanent space? At first glance, one would consider a one-to-one mapping between file types and the space types they are assigned to. However, it is actually useful to support volatile files in permanent space, for example. If a client has the right to get permanent space, it is reasonable for the client to use it for volatile files, so that space can be automatically reclaimed if it is needed. If we rank volatile, durable, and permanent files types from low to high rank, then a file of a particular type should be able to be stored in a higher ranking space type. We note, however, that such a policy of assigning files to spaces is a choice of the local SRM policies. In general, such policies may be used for more efficient space utilization, but it is harder to manage. For this reason, most SRMs will most likely assign file types to spaces of the same type.

There is a condition that needs to be enforced when assigning files to spaces: no file can be pinned for a lifetime longer than the lifetime of the space it is put into. This condition guarantees that when the space lifetime expires, the lifetime of all the files in that space expired too.

5. OTHER IMPORTANT SRM DESIGN CONCEPTS

In this section we discuss additional items that were added to the design of SRMs as a result of experience of their development by several groups.

5.1 Transfer Protocol Negotiation

When making a request to an SRM, the client needs to end up with a protocol that the storage system supports for the transfer of files. In general, systems may be able to support multiple protocols and clients may be able to use multiple protocols depending on the system they are running on. While it is advantageous to select a single standard transfer protocol that each SRM should support (such as GridFTP), this approach is too restrictive. There could be some university researcher without access to GridFTP who wishes to access files through regular FTP, or there could be another FTP service that some community prefers to use. There needs to be a way to match the transfer protocol that a client wishes to use with protocols supported by the SRMs storage system. This is referred to as *protocol negotiation*.

The mechanism to support protocol negotiation is to allow clients to specify an ordered list of protocols in their requests, and let the SRM respond with the protocol that matches the highest possible ranking. For example, suppose that the client wants to use a new FTP service called FastFTP. The client can submit in the request the ordered list: FastFTP, GridFTP, FTP. An SRM whose storage system has a FastFTP service will respond with Fast FTP, otherwise it will respond with GridFTP, or with FTP if it does not have a GridFTP service. In this way all SRM sites that support FastFTP will be able to use it. This mechanism will allow a community to adopt their preferred transfer protocol, and to introduce new protocol services over time without modifying the SRMs.

5.2 Other Negotiations and Behavior Advertising

In general, it should be possible to negotiate any meaningful quantities that may affect the behavior of SRMs. In particular, it should be possible to negotiate the lifetime of spaces and files, and the space-time quantities of reservations. It may also be possible to negotiate how many simultaneous requests the same user may have. In all such cases, it is up to each SRM what policies to use, and how to respond to negotiable requests. An SRM may simply choose to always respond with the same fixed lifetime period, for example.

SRMs can also choose what type of spaces they support. For example, SRMs that manage shared disk caches may not be designed to support permanent spaces. Other SRMs may not support durable space, only volatile space. SRMs may also choose to give one type of service to certain user groups, but not to others. Furthermore, SRMs should be able to change these choices dynamically. To accommodate this dynamic behavior, it is necessary to have a way of advertising the SRMs capabilities, policies, and dynamics loads. Another approach is the passive approach, where SRMs respond to inquiries about their capabilities, dynamic resource capacity available, and dynamic load of the SRM (i.e. how much work they still have in their queue). The passive approach is preferable since the SRMs do not need to find out and coordinate where to advertise except when they are first added to the Grid.

5.3 Source URLs, Transfer URLs, and Site URLs

In previous sections we represented to the location of a file on the Grid as a URL. The example used for such a URL was `gridftp://cs.berkeley.edu:4004/tmp/fileX`. In this example, we refer to `gridftp` as the protocol, and `cs.berkeley.edu:4004/tmp/fileX` as the *Physical File Name* (PFN). Since on the Grid there may be many replicas of a file, each will have a different PFN. This necessitates a unique file name that is location and machine independent. This is referred to as a *Logical File Name*(LFN). A middleware component, such as the Globus Toolkit *Replica Catalog* [SSA+02] or the *Replication Location Service* (RLS) [CDF+02] can provide a mapping of the LFNs to one or more PFNs. SRMs are not concerned with LFNs, since it is the task of the client or the Request Planner to consult with the RLS ahead of time and to make the choice of the desired PFN.

Because SRMs support protocol negotiation, it is not necessary to specify a specific protocol as part of the source URL. Thus, a file that is controlled by an SRM would have `srm` instead of a specific protocol. For the example above, the source URL will be `srm://cs.berkeley.edu:4004/tmp/fileX`. Assume that the preferred transfer protocol in the request is GridFTP, and that it is supported at `cs.berkeley.edu`, then the transfer URL the SRM will return will be: `gridftp://cs.berkeley.edu:4004/tmp/fileX`.

The transfer URL returned to the client does not have to have the same PFN that was provided to the SRM. This is the case when the client wants the SRM to get a file from a remote site. For example, suppose that the source URL `srm://cs.berkeley.edu:4004/tmp/fileX` was provided to an SRM with the host address `dm.fnal.gov:4001`. That SRM will act as a broker, requesting the file from the SRM at `cs.berkeley.edu:4004`. It may choose to place the file in a local directory `/home/cs/newfiles/`, keep the same file name `fileX`, and select the protocol `gridftp`. In this case, after the file is transferred to the SRM at `dm.fnal.gov:4001`, the transfer URL that will be returned will be `gridftp://dm.fnal.gov:4001/home/cs/newfiles/fileX`.

Another important case where the PFN in the transfer URL may be different from the PFN in the source URL is that an SRM at some site may be controlling multiple physical resources, and may want to move the files dynamically between these resources. Such an SRM maintains a mapping between the external PFN exposed to the outside world and the internal PFN of the file depending on which storage component it is stored on. Furthermore, the SRM may choose to keep the same file in multiple storage components, each having its own internal PFN. Note, that the external PFN is not really a physical file name, but a file name maintained by the site SRM. We refer to the URL that contains this external PFN as the *site URL*.

5.4 On the Semantics of the Pinning Concept

In database systems the concept of *locking* is commonly used to coordinate the content of objects in the database (records, disk blocks, etc.) when they are subject to updates. Furthermore, the concept of locking is tightly coupled with the concept of a transaction in that locking is used as a means of ensuring that the entire transaction that may involve multiple objects completes correctly. This led to the well-known theory of serializability and concurrency control [BHG87], and to the widely used two-phase locking algorithm [GR94]. How does the concept of pinning differ from locking? Below, we will limit our discussion to files, without loss of generality. The same concepts apply to any granularity of data, but on the Grid most applications deal with files.

The concept of pinning is orthogonal to locking. While locking is associated with the *content* of a file to coordinate reading and writing, pinning is associated with the *location* of the file. Pinning a file is a way of keeping the file in place, not locking its content. Both locks and pins have to be released. Releasing a lock implies that its content can now be read. Releasing a pin means that the file can now be removed.

The concept of a lifetime is important to both locks and pins. A lifetime on a lock is a way of ensuring that the lock is not unreasonably long, making that file unavailable to others. A lifetime on a pin is a way of ensuring that the file does not occupy space beyond the time necessary for it to be accessed. Pinning is used mainly to manage space allocated to the pinned files. It is necessary for garbage collection of un-released files, cleaning up, or releasing space for reuse.

The length of a lifetime is dependent on the application. Locking lifetimes are usually short (measured in seconds), since we do not want to make files unavailable for a long time. Pinning lifetimes can be made long (measured in minutes) since pinned read-only files can be shared. The only penalty for a long pin lifetime is that the space of the pinned file may not be available for that lifetime period. Releasing a pin as soon as possible ensures the efficient use of resources, and should be used by any middleware or applications using SRMs.

In many scientific applications, the order that the files are retrieved for processing may not be important. As an example, suppose that client A on site X needs 60 files (1 GB each) from site Y, and a client B on site Y needs 60 files (1 GB each) from site X. Suppose that each site has 100 GBs altogether. Now, for each request the 60 files have to be pinned first. If the pins happened simultaneously, each system has only 40 GBs available, no additional space can be released. Thus, the two processes will wait for each other forever. This is a *pin-lock*.

There are many elaborate schemes to deal with deadlocks, ranging from coordination between processes to avoid deadlock, to optimistic solutions that assume that most of the time deadlocks do not occur and that it is only necessary to detect a deadlock and preempt one of the processes involved. For SRMs, the lifetime on a file should be a natural mechanism for preemption. However, since we also have durable and permanent file types, it is worth incorporating some mechanism to prevent pin-locks. Assuming that pin-locks are rare, a mechanism similar to two-phase locking is sufficient, which we may call *two-phase pinning*. It simply states that all spaces and pins must be acquired first before any transfers take place. Otherwise, all pins and spaces must be relinquished, and the process can be tried again. It is unlikely, but possible that this will lead to thrashing. If this is the case, then there must be additional coordination between processes.

6. SOME EXAMPLES OF SRM IMPLEMENTATION AND USE

In this section we describe several efforts that have already shown the value of SRMs.

6.1 Using SRMs to Access Mass Storage Systems

SRMs have been developed at four laboratories as components that facilitate Grid access to several different Mass Storage Systems (MSSs). At the Thomas Jefferson National Accelerator Facility (TJNAF) an SRM was developed to provide Grid access to the JASMine MSS [BHK01]. At Fermi National Accelerator Laboratory (Fermilab) an SRM was developed as part of the Enstore MSS [BBH+99]. At Lawrence Berkeley National Laboratory (LBNL), an SRM was build to function as an independent component in front of HPSS [HPS]. At CERN, a prototype SRM was recently developed for the CASTOR system [CAS]. All these systems use the same interface using the WSDL and SOAP web service technology. The interoperability of these SRMs was demonstrated to allow files to be accessed through the same interface over the Grid.

The above experience showed the flexibility of the SRM concepts. In the case of JASMine and Enstore, it was possible to develop SRMs as part of the MSSs because the developers are either the developers of the MSSs as was the case at TJNAF, or the developers had access to the source code, as was the case at Fermilab. The reason that the SRM-HPSS was developed as an independent component in front of HPSS was that HPSS is a product supported by a commercial vendor, IBM. It would have been impossible to develop an SRM as part of HPSS without negotiating with the commercial vendor. Instead, the SRM was build as an independent component that controls its own disk.

6.2 Robust File Replication Service Using SRMs

File replication of thousands of files is an extremely important task in data intensive scientific applications. For example, large Climate Modeling simulations [CGD] may be computed in one facility, but the results need to be stored in an archive in another facility. This mundane, seemingly simple task is extremely time consuming and prone to mistakes. Moving the files by writing scripts requires the scientist to monitor for failures, to have procedures for checking that the files arrived at their destination correctly, and to recover from transient failures, such as a refusal by an MSS to stage a file.

This task is particularly problematic when the files have to be replicated from one MSS to another. For each file, three conditions have to occur properly: staging of the file in the source MSS, transferring the file over the network, and archiving the file at the target MSS. When staging files, it is not possible to ask for all files to be staged at once. If too many staging requests are made, the MSS will refuse most of the requests. The scientist's script have to monitor for refusals and any other error messages that may be issued by the MSS. Also, if the MSS is temporarily out of order, the scripts have to be restarted. Similar problems can occur on the target MSS in the process of archiving files. In addition, since we are transferring hundreds of large files (order of GBs each), it is possible that network failures occur while transfers are taking place. There is a need to recover from these as well. Replicating hundreds of large files is a process that can take many hours. Providing a service that can support massive file replication in a robust fashion is a challenge.

We realized that SRMs are perfectly suited to perform massive file replication automatically. The SRMs queue the multi-file request, allocate space, monitor the staging, transfer, and archiving of files, and recover from transient failures. Only a single command is necessary to request the multi-file transfer. Figure 5 shows the setup of having SRMs at NCAR and LBNL to achieve continuous file replication of hundreds of files in a single request. We note that in this setup the source MSS is the NCAR-MSS, and the target MSS is HPSS.

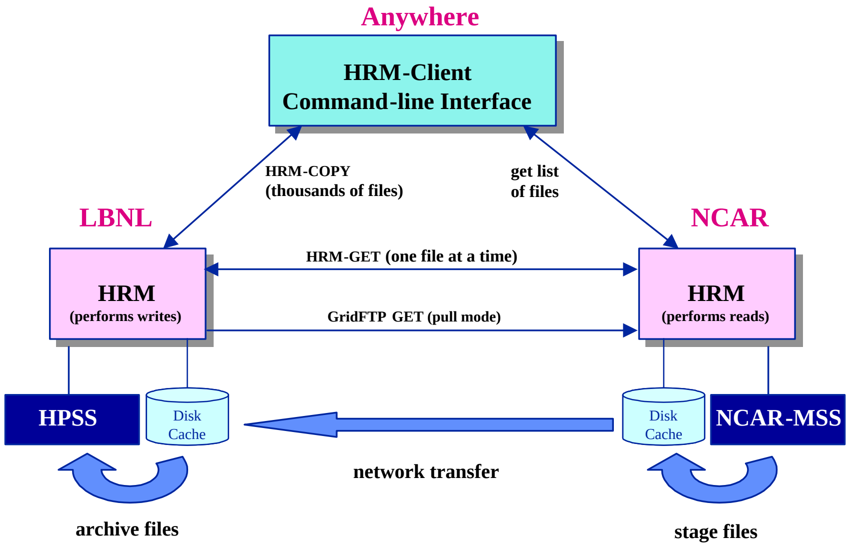

Figure 20.5. The use of SRMs for file replication between two different mass storage systems.

In Figure 5 we refer to the SRMs as Hierarchical (storage) Resource Managers (HRMs) because each HRM has its own disk and it accesses an MSS that has a robotic tape. As can be seen in the figure the request for a multi-file replication can be initiated anywhere. The request can be for an entire directory to be replicated. The SRM-Command-Line-Interface connects first to the source HRM to get the list of files to be moved, and then a single request to replicate these files is submitted to the target HRM.

The great value of using HRMs is that they perform all the operations necessary for multi-file replication as part of their functionality. This setup of file replication is in routine use between an HPSS system at Brookhaven National Laboratory (BNL) and Lawrence Berkeley National Laboratory (LBNL). Each multi-file transfer involves hundreds of files and 10-100s of gigabytes. A typical multi-file replication can take many hours. Another important feature of this setup is that multiple files can be staged at the source concurrently, multiple files can be transferred over the network concurrently, and multiple files can be archived concurrently. Consequently, we achieve a greater efficiency in the end-to-end transfer rate.

This setup can also replicate files from multiple sites. Since URLs are used for the source files, the multi-file transfer request can be from any SRM, including SRMs that manage only disk caches (DRMs). This generality of the SRM design can support a data production scenario that involves multiple sites. For example, a simulation can be run at one or more sites, and the files dumped into several disk caches. A single request to the target SRM where the files have to be archived can get the files from all the temporary disks, and release the space when the files are safely in the archival storage.

6.3 Providing GridFTP to a Storage System Through an SRM

Some storage systems may not be accessible using GridFTP. It is not uncommon to need access to a mass storage system that has not been retrofitted to support GridFTP since this is a non-trivial task and GridFTP would need to be implemented as part of the code base of that system. Using SRMs it is possible to alleviate this problem by having the GridFTP server daemon and the SRM run on a Grid-enabled machine external to the MSS hardware.

A prototype of this nature was developed for HPSS at LBNL. It works as follows. When a GridFTP request for a file is made, the GridFTP code was modified to send a request to HRM. HRM issues a request to HPSS to stage a file. When this completes, the HRM returns the location of the file to GridFTP, which proceeds to transfer the file to the requester. When this completes, GridFTP issues a release to the HRM, so that the space on the HRM disk can be reused. The opposite occurs when a file archiving is requested.

The implementation of this setup did not require any changes to the HRM. The HRM functionality was sufficient to support all the interaction with the GridFTP server daemon. We note that this can be applied with any HRM, not just HRM-HPSS without any modification to the HRM using the same modified GridFTP server code. Also, this can be applied to any system that supports a DRM, and therefore can be applied to any file system not connected to the Grid to make it accessible by GridFTP.

7. CONCLUSIONS

In this Chapter we introduced the concepts necessary to support storage resources on the Grid. Similar to compute and network resources, storage resources need to be dynamically reserved and managed in order to support Grid jobs. In addition to managing space, Storage Resource Managers (SRMs) manage the content (or files) in the spaces used. Managing the content permits file sharing between clients in order to make better use of the storage resources.

We have shown that the concepts of file pinning, file releasing (or unpinning) and lifetime of files and spaces are necessary and useful to support the reliable and efficient use of SRMs. We have also shown that the same standard SRM functionality and interfaces can be used for all types of storage resources, including disk systems, robotic tape systems, mass storage systems, and multiple storage resources managed by a single SRM at a site. SRMs have already been implemented on various systems, including specialized mass storage systems, and their interoperability demonstrated. They are also routinely used to provide massive robust multi-file replication of 100-1000s of files in a single request.

Acknowledgments

While the initial ideas of SRMs were first developed by people from the Lawrence Berkeley National Laboratory (LBNL), many of the ideas and concepts described in this chapter were developed over time and suggested by various people involved in joint meetings and discussions, including from the European Data Grid: Jean-Philippe Baud, Jens Jensen, Emil Knezo, Peter Kunszt, Erwin Laure, Stefano Occhetti, Heinz Stockinger, Kurt Stockinger, Owen Synge, from US DOE laboratories: Bryan Hess, Andy Kowalski, Chip Watson (TJNAF), Don Petravick, Rich Wellner, Timur Perelmutov (FNAL), Brian Tierney, Ekow Otoo, Alex Romosan (LBNL). We apologize for any people we might have overlooked.

This work was supported by the Office of Energy Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.