Storage Resource Managers: Middleware Components for Grid Storage

Arie Shoshani, Alex Sim, Junmin Gu
Lawrence Berkeley National Laboratory
Berkeley, California 94720
{shoshani, asim, jgu}@lbl.gov
tel: +1-510-486-5171
fax: +1-510-486-4004

Abstract

The amount of scientific data generated by simulations or collected from large scale experiments have reached levels that cannot be stored in the researcher's workstation or even in his/her local computer center. Such data are vital to large scientific collaborations dispersed over wide-area networks. In the past, the concept of a Grid infrastructure [1] mainly emphasized the *computational* aspect of supporting large distributed computational tasks, and managing the sharing of the *network* bandwidth by using bandwidth reservation techniques. In this paper we discuss the concept of Storage Resource Managers (SRMs) as components that complement this with the support for the *storage* management of large distributed datasets. The access to data is becoming the main bottleneck in such "data intensive" applications because the data cannot be replicated in all sites. SRMs are designed to dynamically optimize the use of storage resources to help unclog this bottleneck.

1. Introduction

The term "storage resource" refers to any storage system that can be shared by multiple clients. We use the term "client" here to refer to a user or a software program that runs on behalf of a user. Storage Resource Managers (SRMs) are middleware software modules whose purpose is to manage in a dynamic fashion what resides on the storage resource at any one time. SRMs do not perform file movement operations, but rather interact with operating systems, mass storage systems (MSSs) to perform file archiving and file staging, and invoke middleware components (such as GridFTP) to perform file transfer operations. There are several types of SRMs: Disk Resource Managers (DRMs), Tape Resource Managers (TRMs), and Hierarchical Resource Managers (HRMs). We explain each next. Unlike a storage system that allocates space to users in a static fashion (i.e. an administrator's interference is necessary to change the allocation), SRMs are designed to allocate and reuse space dynamically. This is essential for the dynamic nature of shared resources on a grid.

A Disk Resource Manager (DRM) manages dynamically a single shared disk cache. This disk cache can be a single disk, a collection of disks, or a RAID system. The disk cache is available to the client through the operating system that provides a file system view of

the disk cache, with the usual capability to create and delete directories/files, and to open, read, write, and close files. However, space is not pre-allocated to clients. Rather, the amount of space allocated to each client is managed dynamically by the DRM. The function of a DRM is to manage the disk cache using some client resource management policy that can be set by the administrator of the disk cache. The policy may restrict the number of simultaneous requests by each client, or may give preferential access to clients based on their assigned priority. In addition, a DRM may perform operations to get files from other SRMs on the grid. This capability will become clear later when we describe how DRMs are used in a data grid. Using a DRM by multiple clients can provide an added advantage of file sharing among the clients and repeated use of files. This is especially useful for scientific communities that are likely to have an overlapping file access patterns. One can use cache management policies that minimize repeated file transfers to the disk cache for remote grid sites. The cache management policies can be based on use history or anticipated requests.

A Tape Resource Manager (TRM) is a middleware layer that interfaces to systems that manage robotic tapes. The tapes are accessible to a client through fairly sophisticated Mass Storage Systems (MSSs) such as HPSS, Unitree, Enstore, etc. Such systems usually have a disk cache that is used to stage files temporarily before transferring them to clients. MSSs typically provide a client with a file system view and a directory structure, but do not allow dynamic open, read, write, and close of files. Instead they provide some way to transfer files to the client's space, using transfer protocols such as FTP, and various variants of FTP (e.g. Parallel FTP, called PFTP, in HPSS). The TRM's function is to accept requests for file transfers from clients, queue such requests in case the MSS is busy or temporarily down, and apply a policy on the use of the MSS resources. As in the case of a DRM, the policy may restrict the number of simultaneous transfer requests by each client, or may give preferential access to clients based on their assigned priority.

A Hierarchical Storage Manager (HRM) is a TRM that has a staging disk cache for its use. Thus, it can be viewed as a combination of a DRM and a TRM. It can use the disk cache for pre-staging files for clients, and for sharing files between clients. This functionality can be very useful in a data grid, since a request from a client may be for many files. Even if the client can only process one file at a time, the HRM can use its cache to pre-stage the next files. Furthermore, the transfer of large files on a shared wide area network may be sufficiently slow, that while a file is being transferred, another can be staged from tape. Because robotic tape systems are mechanical in nature, they have a latency of mounting a tape and seeking to the location of a file. Pre-staging can help mask this latency. Similar to the file sharing on a DRM, the staging disk in an HRM can be used for file sharing. The goal is to minimize staging files from the robotic tape system. The HRM design is based on experience in a previous project reported in [2].

The concept of an SRM can be generalized to the management of multiple storage resources at a site. In such cases, the site SRM may use "site-file-names" (directory path

+ file names) which do not reflect the physical location and file names. This gives the site the flexibility to move files around from one storage device to another without the site-file-names changing. When a client accesses a file using a site-file-name, it may be given in response the physical location and file name. The client can then use the physical file name to execute a file transfer.

In general, it is best if SRMs are shared by a community of users that are likely to access the same files. They can be designed to monitor file access history and maximize sharing of files by keeping the most popular files in the disk cache longer.

2. The role of SRMs in a Data Grid

Suppose that a client runs an analysis program at some site and wishes to get data stored in files located in various sites on the grid. First, the client must have some way of determining which files it needs to access. Checking a file catalog, using some index, or using a database system containing information about the files can accomplish this step. We refer to this step as "request interpretation". The information used in this step is often referred to as a "metadata catalog". The result of this step is a set of logical file names that need to be accessed. The second step is to find out for each logical file where it physically resides or replicated. Note that a single logical file can be replicated in multiple sites. Files can be either pre-replicated in multiple sites based on expected use by a system administrator or replicated dynamically because they were accessed by clients at these sites. In a grid environment, the information on the locations of replicated files exists in a "replica catalog", a catalog that maps a single logical file name to multiple site-specific files. The site-specific file name includes the name a machine and possibly port at the site, the directory path on that system, and the file name.

In many grid environments today, the burden for the above work is being thrust on the clients. Therefore, it is now recognized that such tasks can be delegated to middleware components to provide these services. A "request manager" is the term used to refer to such services. The request manager performs "request planning" based on some strategy, and then a "request execution" of the plan. This terminology is used by several grid projects, notably PPDG [3], GriPhyN [4], and ESG [5]. There are three options to consider for request planning: either move the client's program to the site that has the file, move the file to the client's site, or move both the program and the data to another site for processing. All three possibilities are valid, and much of the middleware development addresses this issue. In all these cases, SRMs play an important role. In the case that the program moves to the site where the file exists, it is necessary to "pin" the file in that site; that is, to request that the file remains in that site, so that when the program is executed the file is found in the cache. When the program completes, the file can be "released". In the case that the file needs to be transferred from a source site to target site (either to the client's site, or to another site), it is necessary to "pin" the file in the source site, to reserve the space in the target site, and maintain this state till the transfer to the target site is complete. Then the "pin" can be released. Here, the SRM at

the source site has the role of managing the "pinning", and the SRM at the target site has the role of allocating space (i.e. making space by removing other files if necessary), and reserving the space till the transfer completes. SRMs need to deal also with various failures, so that space reservations do not persist forever, and "pins" do not persist in case that a "release" is not performed. The concept of "pinning a file" is central to SRMs and will be discussed further later in this document.

In a recent paper [6], the authors describe 5 layers needed to support grid applications: fabric, connectivity, resource, collective, and application layers. The purpose of this layered approach is that services in each layer can rely on services in layers below it. The fabric layer consists of computational resources, storage resources, network resources, catalogs, code repositories, etc. The connectivity layer consists of communication, authentication, delegation, etc. The resource layer consists of components (and protocols) for managing various resources: computing, storage, network, catalog, inquiry, etc. We see SRMs as belonging to the "resource layer". The collective layer consists of services such as replica catalog, replica selection, request planning, and request execution. Request management is a generic term that uses any of the services in that layer, as well as services below it. The application layer consists of application specific services. The "request interpretation" we mentioned above belongs to this layer, since finding which logical files are needed by an application is specific to that application.

3. A practical use case: an analysis scenario

We describe below an analysis scenario where the computation is performed at the client's site, and the needed files are in other sites on the grid. This is a common special case of grid resource usage in many scientific communities. The schematic diagram of this analysis scenario is shown in Figure 1.

As shown in Figure 1, at the client's site there may be multiple clients sharing a local disk cache. Each of the clients issues a logical request, typically consisting of a logical predicate condition for what they wish to analyze. A typical example of such a request in the high-energy physics domain (where atomic particles are accelerated and made to collide at high speeds) might be: "find all the collisions (called "events") that have an energy more that 50 GEV, and produced at least 1000 particles". A similar request for climate model analysis may be "get all temperatures and wind velocity for summer months in the Pacific Ocean region for the last ten years". These requests may be produced by a graphical user interface or composed by the client using some query language. The Request Interpreter is a component that accepts the logical query and produces a set of logical file names that contain the desired data. A Request Planner may check with a Replica Catalog and other network services such as the "network weather service" (which provides an estimate of current network availability) to determine the replica site from which to get each file. The Request Executer then executes this plan. An example of a request executer, called DAGMAN (for Directed-Acyclic-Graph Manager) was recently developed by the Condor project [7].

The request executer could communicate with various SRMs on the grid, requesting space allocation and file pinning, and making requests for file transfers. However, we have decided to delegate the task of making requests for file transfers to the SRMs.

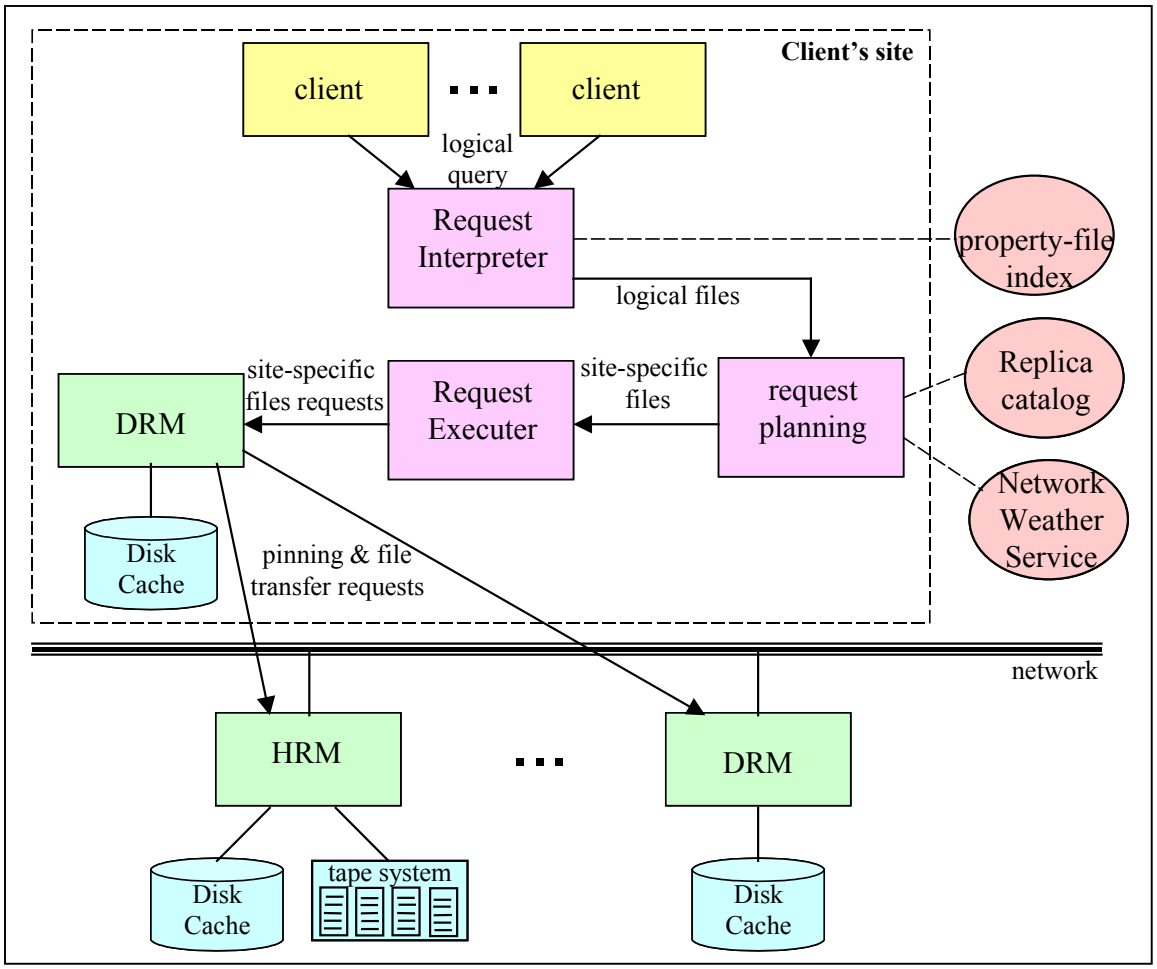

Figure 1. A schematic diagram of an analysis scenario

Specifically, if a request for a set of files is made to an SRM, it is its responsibility to dynamically allocate space for the files, to negotiate with remote SRMs the pinning of files at the remote site, to invoke file transfer services to get the files from other sites and to release the files after they are used. By making this fundamental design choice, we not only simplify the request executer's task, but also permit clients to communicate *directly* with SRMs making multi-file requests. The ability for clients to request files directly from an SRM was a basic requirement that guided our design since, in general, one cannot assume the existence of request managers. Furthermore, clients should be able to make direct requests to SRMs if they so choose. A secondary advantage of this design

choice is that it facilitates file sharing by the SRMs. Since clients can make multi-file requests to the SRM, it can choose to serve files to clients in the order that maximizes file sharing, thus minimizing repeated file transfers over the network.

For the analysis scenario shown in Figure 1, where all the files have to be brought to the local disk cache, the request executer makes its file requests to the local DRM. The local DRM checks if the file is already in its cache. If it is in the cache, it pins the file. If it is not, it communicates with other SRMs to get the files.

We have implemented several versions of DRMs as well as an HRM that interfaces to the HPSS mass storage system. The HRM is implemented as a combination of a TRM that deals with reading/writing files from/to HPSS, and a DRM for managing its disk cache. Both the DRM and the TRM are capable of queuing requests when the storage systems they interface to are busy. For example, a TRM interfacing with HPSS may be limited to perform only a few staging request concurrently, but it may be asked to stage hundreds of files. These requests are then queued, and performed as fast as HPSS will perform. The SRMs use grid-enabled secure file transfer services provided by the Globus project [8], called GridFTP. These DRM and HRM components are in the process of being used by one of the experiments of the Particle Physics Data Grid (PPDG) [3], and the Earth Science Grid (ESG) [5] to perform grid file replication functions. The HRM was also used in a demo for SuperComputing 2000 as part of an infrastructure to get files from multiple locations for an Earth Science Grid application (ESG). This was described in a recent paper [9]. We are now evaluating several "cache replacement policies" to be used by DRMs, by both conducting simulations and setting up real testbeds.

4. The implementation of the analysis scenario

The analysis scenario described in Figure 1 was implemented as part of a demo during the Supercomputing 2001 conference. The application used in the demo was high-energy physics (HEP). Figure 2 shows the actual setup of the demo. From a client's point of view the system accepts a logical query request, and takes care of all the details of figuring out what files should be transferred, and where to get them from. The client can observe in a graphical display the progress of file transfers over time. Figure 3 shows the progress of transfer of each file managed by the client's DRM. Partially filled bars represent transfer in progress. When a file that arrives is processed and released by the client, it may be removed automatically by the DRM if it needs to make space for additional files.

In order to illustrate the usefulness of SRMs, we describe next in some detail the steps of processing a logical query in a grid environment. In figure 2, the Bit-Map index is a specialized index used as the "request interpreter", which was developed as part of another project [10]. It gets as input a logical request made of logical conditions over range predicates. An example of such a request in this HEP application is to find all files that contain collisions (or "events") for which the following condition holds:

((0.1 < AVpT < 0.2) ^ (10 < Np < 20)) v (N > 6000),

where AvpT is the "average momentum", Np is "the number of pions" produced in this collision, and N is the "total number of particles produced in this collision". The result of the Bit-Map index is a set of logical file names, such as:

{star.simul.00.11.16.tracks.156,…, star.simul.00.11.16.tracks.978},

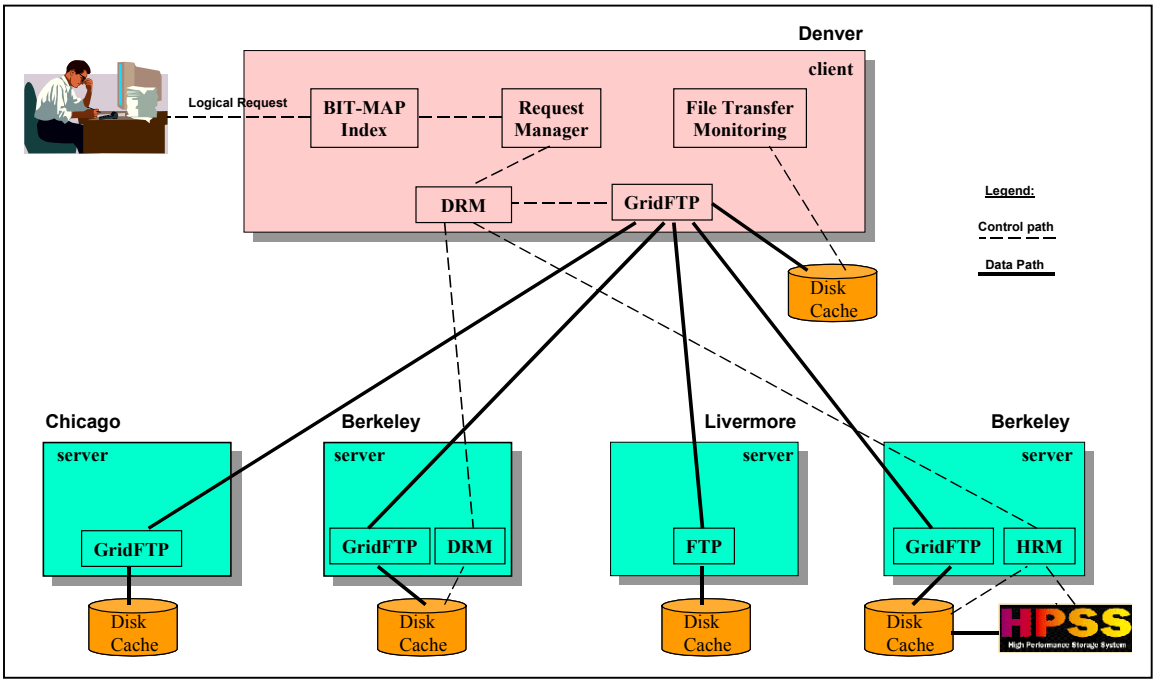

Figure 2. A setup for processing logical analysis requests over the grid

where "star" is the name of the experiment at Brookhaven National Laboratory, "simul" means simulation data, "00.11.16" is the date the data was generated, "tracks" refers to the type of data in the file, and the number is the file ID. This set of logical file names is given to the next component, the Request Manager.

The Request Manager (which consists of both a Request Planning and Request Execution components) is a component that chooses the site where to get each file, and then oversees the execution of the request. Given that a file may be replicated in multiple locations, it chooses the most appropriate location. Each file is assigned a "site file name" in the form of a URL, such as:

gsiftp://dg0n1.mcs.anl.gov/homes/sim/gsiftp/star.simul.00.11.16.tracks.156,

where "gsiftp" is the protocol for transferring the file, "dg0n1.mcs.anl.gov" is the machine name, "homes/sim/gsiftp" is the directory path, and

"star.simul.00.11.16.tracks.156" is the file name.

Similarly, if the site that has the file is managed by an SRM, the protocol used will say "hrm" or "drm". For example, for accessing the same file out of an HPSS tape system, the URL used is:

hrm://dm.lbl.gov:4000/home/dm/srm/data1/star.simul.00.11.16.tracks.156,

where "dm.lbl.gov:4000" is the name of the machine that has HRM running on it, and the port used by HRM, "home/dm/srm/data1" is the directory on the HPSS system where the file resides, and "star.simul.00.11.16.tracks.156" is the file name.

Note that files can reside on systems that may or may not have an SRM managing the storage system. We set up the demo to illustrate that an SRM can work with systems managed by other SRMs, or systems that have some grid middleware (such as GridFTP), or even systems that have no middleware software at all (using only FTP to transfer files). In the demo, we set up four types of nodes: one with a DRM managing the storage system (at LBNL), one with an HRM managing access to an HPSS system (at NERSC-LBNL), one that has no SRM but has GridFTP available on it (at ANL), and one that has only FTP available on it (at LLNL).

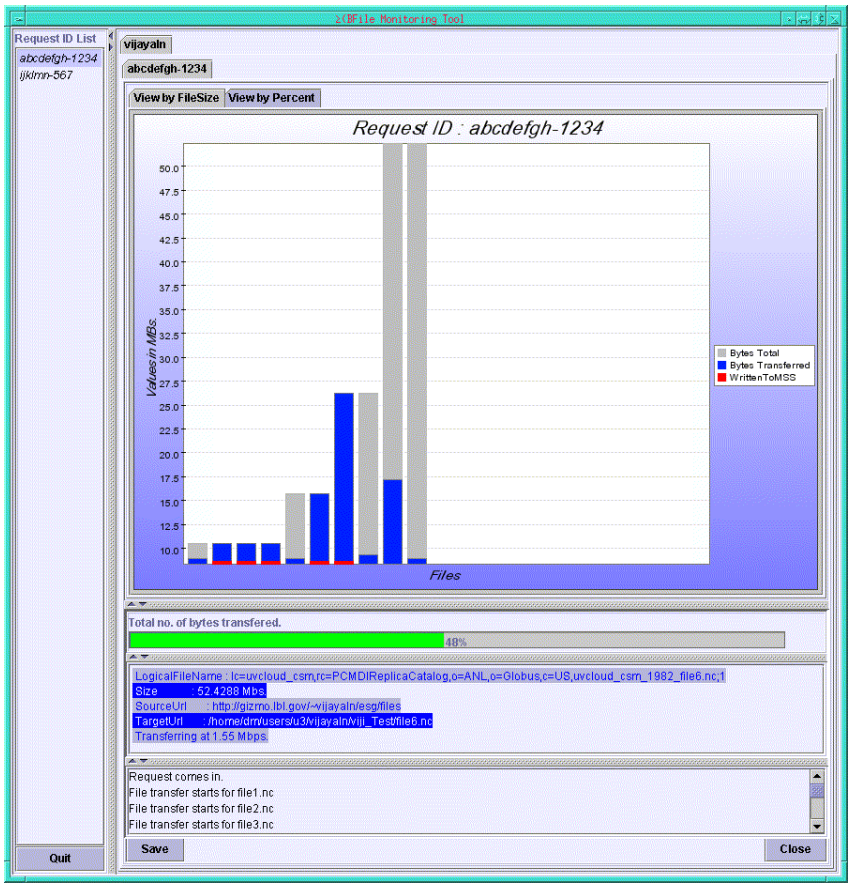

Figure 3. Display of the dynamic progress of file transfers

Once the Request Manager has assembled the set of URLs for the files needed, it invokes the local DRM (at the Supercomputing Conference floor at Denver). The local DRM then checks for each file if it is already in cache, and if the file is not found it contacts the site that has it, requesting pinning of files, and invoking the appropriate file transfer service (GridFTP or FTP in this demo). Once a file is transferred, it sends a "release of file" notice to the source site.

The SRMs are multi-threaded components that can support simultaneous file transfer requests from multiple clients. Thus, given a request for multiple files, the client's DRM will initiate the coordination of space reservation, pinning of files, and multiple file transfer requests to multiple sites. The number of such concurrent processing of file transfer requests is a policy decision. Since multiple clients may share a local DRM, the

DRM may have a policy to restrict the amount of space and the number of files that a client can hold simultaneously.

The display of file transfers in Figure 3 was designed to show dynamic progress. The local disk cache is checked every 10 seconds (a parameterized choice) for the size of files being transferred, and the display is updated. The horizontal bar below file progress display shows the total bytes transferred as a fraction of the total bytes requested. Moving the curser over any of the file bars provides information of the source location, size, and transfer rate. This is shown in the lower section of the display. Finally, there is a "message section" at the bottom to inform the client of events as they occur, including failures to access files and the reasons for that, such as "system down".

The above scenario was limited to cases where all the files are moved to the client's location. The generalization of this scenario is that the request planner generates a plan where the execution of the analysis can be partitioned to run on multiple sites (perhaps the sites where the data reside to minimize file transfer traffic). In this general scenario, both data and programs can move to the locations best suited to execute a request in the most efficient manner possible. The general scenario also includes moving the results of computations to the client, as well as storing results in storage systems and archives on the grid. Thus, in general, SRMs can be invoked at multiple locations by a single client to satisfy the request plan.

5. Advantages of using SRMs

As can be deduced from the discussion above, the main advantage of an SRM is that it provides smooth synchronization between shared resources by pinning files, releasing files, and allocating space dynamically on an "as-needed" basis. A reasonable question is why use SRMs if it is possible to use GridFTP and FTP directly as was done in the above demo. We recall that SRMs perform two main functions: dynamic space allocation and dynamic file pinning. Indeed, if space is pre-allocated, and the files are "permanently" locked in the source site there is no need for SRMs. However, in a grid environment where resources need to be reused dynamically, SRMs are essential. SRMs perform the management of quotas, the queuing of requests when resources are tight or if the clients exceed their quota, the freeing of space of files allocated but not released by clients (similar to "garbage collection"), and providing the management of buffers for pre-staging from mass storage systems. Pre-staging and buffering are important because the network bandwidth available to a client may vary in an unpredictable fashion.

A second advantage of using SRMs is that they can eliminate unnecessary burden from the client. First, if the storage system is busy, SRMs can queue requests, rather than refuse a request. Instead of the client trying over and over again, till the request is accepted, an SRM can queue the request, and provide the client with a time estimate based on the length of the queue. This is especially useful when the latency is large such

as for reading a file from tape. If the wait is too long, the client can choose to access the file from another site, or wait for its turn. Similarly, a shared disk resource can be temporarily full, waiting for clients to finish processing files, and therefore queuing requests is a better alternative than simply refusing the request.

A third advantage to the clients is that SRMs can insulate them from storage systems failures. This is an important capability that is especially useful for HRMs since MSSs are complex systems that fail from time to time, and may become temporarily unavailable. For long lasting jobs accessing many files, which are typical of scientific applications, it is prohibitive to abort and restart a job. Typically, the burden of dealing with an MSS's temporary failure falls on the client. Instead, an HRM can insulate clients from such failures, by monitoring the transfer to the HRM's disk, and if a failure occurs, the HRM can wait for the MSS to recover, and re-stage the file. All that the client perceives is a slower response. Experience with this capability was shown to be quite useful in real situations [2].

A fourth advantage is that SRMs can transparently deal with network failures. SRMs can monitor file transfers, and if failures occur, re-try the request. They can provide clients the information of such failures, so that clients can find other alternatives, such as getting the file from its original archive if a transfer from a replication site failed. Recently, there is an interest of managing the inherent unreliability of the network as part of an extended middleware file transfer service, called "Reliable File Transfer" (RFT). It is intended as a service layer on top of GridFTP that will try to re-transfer files in case of temporary failures of the network, will queue such requests, and will provide status of the requests. When such services are available, SRMs can take advantage of them. Otherwise, as is the case for systems that have no grid middleware software (e.g. only FTP), SRMs need to protect the clients from unreliable network behavior.

A fifth advantage of SRMs is that they can enhance the efficiency of the grid, eliminating unnecessary file transfers by sharing files. As mentioned above, it is typical of scientific investigations that multiple clients at the same site use overlapping sets of files. This presents an opportunity for the SRM at that site to choose to keep the most popular files in its disk cache longer, and providing clients with files that are already in the disk cache first. Managing this behavior is referred to as a "replacement policy", that is deciding dynamically which file to replace when space is needed. This problem is akin to "caching algorithms", which have been studied extensively in computer systems and web caching. However, unlike caching from disk to main memory, the replacement cost in the grid can be quite high, as files have to be replaced from remote locations and/or from tertiary storage. Deploying efficient replacement policies by the SRMs can lead to significant reductions in repeated file transfers over the grid.

Finally, one of the most important advantages of using SRMs is that they can provide a "streaming model" to the client. That is, they provide a stream of files to the client programs, rather than all the files at once. This is especially important for large

computing tasks, such as processing hundreds, or even thousands of files. Typically, the client does not have the space for the hundreds of files to be brought in at once. When making such a request from an SRM, the SRM can provide the client with a few files at a time, streaming of files as they are used and released. This is managed by the SRM enforcing a quota per client, either by the amount of space allocated and/or by the number of files allocated. As soon as files are used by the client and released, the SRM brings in the next files for processing in a streaming fashion. The advantage to this "streaming model" is that clients can set up a long running task, and have the SRM manage the streaming of files, the pre-staging of files, the dynamic allocation of space, and the transferring of files in the most efficient way possible.

6. "Pinning" and "two-phase pinning"

The concept of *pinning* is similar to locking. However, while locking is associated with the *content* of a file to coordinate reading and writing, pinning is associated with the *location* of the file to insure that a file stays in that location. Unlike a lock, which has to be released, a "pin" is temporary, in that it has a time-out period associated with it, and the "pin" is automatically released at the end of that time-out period. The action of "pinning a file" results in a "soft guarantee" that the file will stay in a disk cache for a pre-specified length of time. The length of the "pinning time" is a policy determined by the disk cache manager. Pinning provides a way to share files that are not permanently assigned to a location, such as replicated files. This permits the dynamic management and coordination of shared disk caches on the grid. Since we cannot count on pins to be released, we use the pinning time-out as a way to avoid pinning of files forever.

Two-phase pinning is akin to the well known "two-phase locking" technique used extensively in database systems. While two-phase locking is used very successfully to synchronize writing of files and to avoid deadlocks, two-phase pinning is especially useful to synchronize requests for multiple files *concurrently*; that is, if the client needs several files at the same time, it can first attempt to incrementally pin these files, and only then execute the transfers for all files, then releasing them as soon as each is transferred. We note, that even if file replicas are read-only, a deadlock (or pin-lock) as a result of pinned files can occur if we allow requests for multiple files concurrently. However, if we assume that file requests are asynchronous and that time-outs to release files are enforced, pin-locks are eventually resolved because pinned files are released after they time-out. Nevertheless, two-phase pinning is a useful technique to avoid system thrashing by repeatedly pinning and pre-emptying pins. It requires coordination between the SRMs.

7. The design of "Read" and "Write" functionality of SRMs

When a request to read a file is made to an SRM, the SRM may already have the file in its cache. In this case it pins the file and returns the location of the file in its cache. The client can then read the file directly from the disk cache (if it has access permission), or

can copy or transfer the file into its local disk. In either case, the SRM will be expected to pin the file in cache for the client for a period of time. A well-behaved client will be expected to "release" the file when it is done with it. This case applies to both DRMs and HRMs.

If the file is not in the disk cache, the SRM will be expected to get the file from its source location. For a DRM this means getting the file from some remote location. For an HRM, this means getting the file from the MSS. This capability simplifies the tasks that the client has to perform. Rather than return to the client with "file not found", the SRM provides the service of getting the file from its source location. Since getting a file from a remote location or a tape system may take a relatively long time, it should be possible for the client to make a non-blocking request. To accommodate this possibility the SRMs provide a callback function that notifies the client when a requested file arrives in its disk cache and the location of that file. In case that the client cannot be called back, SRMs also provide a "status" function call that the client can use to find out when the file arrives. The status function can return estimates on the file arrival time if the file has not arrived yet.

HRMs can also maintain a queue for scheduling the file staging from tape to disk by the MSS. This is especially needed if the MSS is temporarily busy. When a request to stage a file is made, the HRM checks its queue. If the HRM's queue is empty, it schedules its staging immediately. The HRM can take advantage of its queue to stage files in an order optimized for the MSS. In particular, it can schedule the order of file staging according to the tape ID to minimize tape mounts and dismounts, as described in [2]. Like a DRM, the HRM needs to notify the client that the file was staged by issuing a callback, or the client can find that out by using "status".

A request to "write" a file requires a different functionality. In the case of a DRM, if the file size is provided, then that space is allocated, and the client can write the file to it. If the file size is not provided, a large default size is assumed, and the available space is adjusted after the file is written. In the case of an HRM, the file is first written to its disk cache in exactly the same way as the DRM description above. The HRM then notifies the client that the file has arrived to its disk using a callback, then it schedules it to be archived to tape by the MSS. After the file is archived by the MSS, the SRM notifies the client again using a callback. Thus, the HRM's disk cache is serving as a temporary buffer for files being written to tape. The advantage of this functionality by HRM is that writing a file to a remote MSS can be performed in two stages: first transferring the file to the HRMs disk cache as fast as the network permits, and then archiving the file to tape as a background task. In this way the HRM can eliminate the burden from the client to deal with a busy MSS as well as dealing with temporary failures of the MSS system.

One of the practical implementation problems that SRMs have to deal with is an incorrect or missing file size. In both cases of getting or putting a file into the SRM space, the SRM needs to allocate space before the transfer of the file into its disk cache. If the file

size provided (or assigned by default) is smaller than the actual file size, then this can cause various failures, such as writing over other files, or overflowing the total space that the SRM manages. There are various methods of dealing with this problem (such as interrupting the transfer or permitting incremental growth of the allocated space), but all require the dynamic monitoring of the file transfers, and the ability to terminate the transfer process if necessary. Since SRMs cannot terminate the transfer process initiated by the client (in the case that it puts a file into the SRM's disk cache), this problem presents a special challenge. The solution to this problem usually requires modifications to the file transfer server program.

SRMs can also be used to coordinate a third party file movement. Essentially, an SRM in site Y can be asked to "pull" a file form site X. This request can be made by a client in a third location. The SRMs in the two sites X and Y then coordinate space allocation, file pinning, and file release. The actual transfer of the file is a regular two-way file transfer from X to Y. The usefulness of this functionality is for clients that produce files, store then temporarily in some location X, and then request their movement to an archive in site Y. The inverse functionality can also be provided, where the SRM at site X is asked to "push" the file to site Y.

8. Conclusion

We discussed in this paper the concept of Storage Resource Managers (SRMs), and argued that they have an important role in streamlining grid functionality and making it possible for storage resources to be managed *dynamically*. While static management of resources is possible, it requires continuous human intervention to determine where and when file replicas should reside. SRMs make it possible to manage the grid storage resources based on the actual access patterns. In addition, SRMs can be used to impose local policies as to who can use the resources and how to allocated the resources to the grid clients. We also introduced the concept of "pinning" as the mechanism of requesting that files stay in the storage resource until a file transfer or a computation takes place. Pinning allows the operation of the coordinated transfer of multiple files to be performed as a "2-phase pinning" process: pin the files, transfer, and release pins. We have developed several versions of prototype SRMs and used them in test cases as part of the Particle Physics Data Grid (PPDG) and Earth Science Data Grid (ESG) projects. A prototype of an HRM was also developed at Fermi National Accelerator Laboratory which interfaces to their Enstore MSS. In addition, efforts are now underway to coordinate the SRM functionality across several projects, including the development of an HRM at Thomas Jefferson National Accelerator Facility to interface to their JASMine MSS, and the European Data Grid to interface to their CASTOR MSS. The emerging concepts and interfaces seem to nicely complement other grid middleware services being developed by various grid projects, such as providing efficient and reliable file transfer, replica catalogs, and allocation of compute resources.

Acknowledgements

We would like to thanks our colleagues John Wu, and Vijaya Natarajan, who provided the bit-map index and the monitoring tool display program for the SC 2001 demo. We also acknowledge the useful interactions with people involved in the PPDG and ESG projects, as well as the European Data Grid project. This work was supported by the Office of Energy Research, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

References

[1] The Grid: Blueprint for a New Computing Infrastructure, Edited by Ian Foster and Carl Kesselman, Morgan Kaufmann Publishers, July 1998.

[2] Access Coordination of Tertiary Storage for High Energy Physics Application, L. M. Bernardo, A. Shoshani, A. Sim, H. Nordberg (MSS 2000).

[3] Particle Physics Data Grid (PPDG), http://www.ppdg.net/

[4] The Grid Physics Network (GriPhyN) http://www.griphyn.org

[5] Earth Science Grid (ESG), http://www.earthsystemgrid.org

[6] Ian Foster, Carl Kesselman, Steven Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organization, *The International Journal of High Performance Computing Applications*, 15(3), (2001) 200-222.

[7] DAGMAN, part of the Condor project, http://www.cs.wisc.edu/condor/manual/v6.2/2_10Inter_job_Dependencies.html

[8] The Globus Project, http://www.globus.org

[9] B. Allcock, A. Chervenak, E. Deelman, R. Drach, I. Foster, C. Kesselman, J. Lee, V. Nefedova, A. Sim, A. Shoshani, D. Williams, High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies, *Proceedings of Supercomputing Conference* (2001).

[10] A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, and A. Sim, Multidimensional Indexing and Query Coordination for Tertiary Storage Management, Statistical and Scientific Database Management Conference (1999) 214-225.