DataMover: Robust Terabyte-Scale Multi-file Replication over Wide-Area Networks

Alex Sim, Junmin Gu, Arie Shoshani, Vijaya Natarajan
Lawrence Berkeley National Laboratory
(asim, jgu, shoshani, vnatarajan)@lbl.gov

Abstract

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, archiving them at the target mass storage system or disk systems, and recovering from transient system failures. We have developed a robust replication system, called DataMover, which is now in regular use in High-Energy-Physics and Climate modeling experiments. Only a single command is necessary to request multi-file replication or the replication of an entire directory. A web-based tool was developed to dynamically monitor the progress of the multi-file replication process.

1. 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 2006, are predicted to generate many petabytes of data. We describe in this paper one of the problems that arises from dealing with this large volume of data – 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 an analysis environment, which can be 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.

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, in trying to address this problem, we realized that we can take advantage of software components we developed for a Grid middleware project, 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 difficut 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 could build a component that communicates with SRMs to make multi-file transfer requests. This component, that we call the Data-Mover[1], was designed to respond to the requirements given to us by scientists in several domain areas. The main requirement was 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. A key point of this paper is to demonstrate the power of using SRMs to easily solve the difficult problem of robust file replication of thousands of files. We note that simply using an efficient file transfer service does not address dynamic space allocation or recovery from failures.

In the remainder of the paper we describe the design of the DataMover, its deployment in a couple of application domains for routine production use, and the experience in using this capability. One of the more interesting items learned was that by analyzing the logs with a visual tool it is possible to identify where the bottlenecks of the system are. We start by describing the functionality of SRMs in section 2. In section 3, we present the DataMover interaction with SRMs, and the series of actions that take place for each file being transferred. In section 4 we describe a file monitoring tool that was developed to track the progress of lengthy multi-file transfers. In section 5, we describe our experience and analysis of bottlenecks from the logs. We conclude in section 6.

2. Storage Resource Managers

Storage Resource Managers (SRMs) are Grid middleware components whose function is to provide dynamic space allocation and file management on shared storage components on the Grid. They are designed to provide effective sharing of files, by monitoring the activity of shared files, and making dynamic decisions on which files to replace when space is needed.

Managing shared storage resources on the Grid is a necessary and complex task because of the diversity of the storage resources. Storage resources can vary in complexity: a single disk under a UNIX file system, large sets of disk caches or disk RAIDs, or mass storage systems (such as HPSS) that provide access to data on robotic tape systems. Making such resources sharable through Grid middleware technology requires that these systems are exposed through a uniform interface. Thus, requesting space from any of these systems should look the same to a client. We have shown that through international collaboration of achieving agreements on such common interfaces, various storage systems can interoperate. Most notably, SRMs have been built for several mass storage systems both in the US (HPSS at multiple sites, Enstore at Fermilab, JASMine at Jefferson Lab, MSS at NCAR) and in

[1] The name DataMover was proposed by the NCAR scientists

Europe (Castor at CERN, SE at Rutherford Lab), and shown to interoperate smoothly. Furthermore, several SRMs were built for disk systems as well, and they interoperate with SRMs for mass storage systems. This approach of standardizing on the functionality and the interfaces of SRMs is the backbone to the interoperation of shared storage systems on the Grid. It allows multiple groups to implement their own SRMs and thus make the underlying storage system viewed as a Grid service.

A practical problem of managing shared resources is that files are deposited in such systems and often not removed. This tends to clog and make storage systems ineffective. The problem is that the system administrators do not know which files can be safely removed. For this reason, SRMs have been designed to associate a lifetime with files that have temporary use. Accordingly, SRMs support the "pinning" of files for the duration of a lifetime, as well as "releasing" files as explicit requests. Lifetime is a mechanism for the SRMs to reuse space that is not actively in use.

Another aspect of SRM functionality came from the desire to simplify the Grid client's interaction with storage systems. For example, it is a lot simpler for an application client to request one thousand files in a single request from an SRM regardless of their location on the Grid, rather than having to get each file from it source location. SRMs have been designed to provide a service of accepting multi-file requests, queuing each file request, getting the files from the source locations (using a file transport service such as GridFTP) based on space availability, and streaming the files to the client. If files are found locally, they are pinned for a certain lifetime. SRMs can therefore share files between clients, making storage usage more effective, and avoiding unnecessary file transfers over the Grid. We note that SRMs use GridFTP [3] from the Globus project [4] for invoking file transfers. This choice provides a way of dealing with Security, since GridFTP support GSI (Grid Security Infrastructure) authentication, and also used large windows (1 MB) and multiple streams for more efficient file transfer. However, the SRM APIs have been designed to support any transfer protocol by negotiation between the client and the server [see 2]. The interface to the SRMs can be through a web service or other programming language APIs (such as a c-API or a java-API).

It is the feature of accepting a multi-file request, and getting the files as storage permits, that makes SRMs so useful to the DataMover. SRM that are placed in front of storage systems mange their own space, where files are temporarily placed. As files are "released", the SRM can use the space to get additional files, effectively streaming the files through the disk cache to the destination. To illustrate this capability, we describe in Figure 1, an SRM which is placed in front of a mass storage system, such as HPSS [5]. This type of SRM, which we refer to as a Hierarchical Resource Manager (HRM), manages a disk cache of its own, and interacts with HPSS to stage and archive files. In Figure 1 we show a series of interactions that occur between a client and an HRM server placed in front of HPSS.

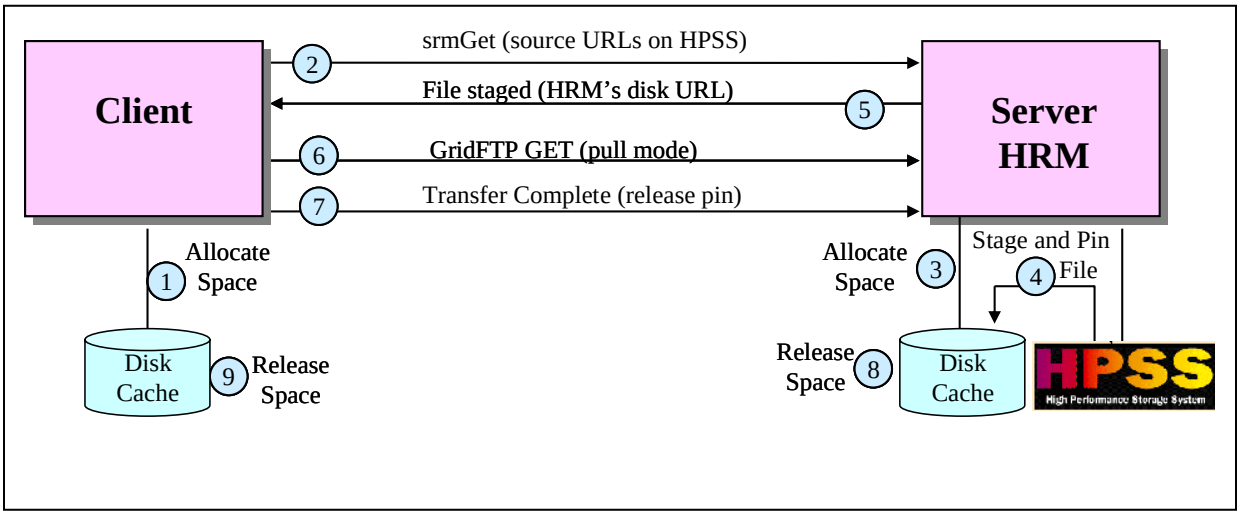

Figure 1: steps of getting a file transferred from the MSS to the client's disk using an HRM

It is important to go through the steps of this interaction, since this is what the DataMover take advantage of. Given a multi-file request (say 1000 files) the following steps take place:

1. The client allocates space on its disk. Say, enough space for 50 files.
2. The client sends a request (referred to in the figure as srmGet) to HRM for these 50 files.
3. The HRM server queues the request, and allocates space on its own disk. For example, suppose that this is limited to hold only 10 files.
4. The HRM sends 10 concurrent staging requests to HPSS for the 10 files. The level of concurrency is limited by HPSS and can be smaller or higher than 10.
5. As soon as a file that is staged the HRM pins the file and sends a notice to the client that this file is ready for transfer. Thus, multiple such notifications can be sent concurrently.
6. The client issues a file transfer call (GridFTP) to pull the file. Multiple file transfer requests may be active concurrently
7. When the transfer completes, the client send a notice to the HRM that it is finished and "releases" the file.
8. The HRM releases the pin and makes the space available for further transfers. It can reuse this space to stage additional files.
9. The client consumes the file (i.e. passes it to the client), and releases the space on its disk.

This protocol is necessary in order to insure that files are not removed prematurely, and that space can be released as soon as the file occupying it is unpinned.

As soon as the client's space is released, the client can send additional srmGet requests to the HRM. This has the effect of streaming multiple files concurrently from the source HPSS to the target disk. This process repeats until all 1000 files in the original request have been transferred. We note that if the client pre-allocates space for all 1000 files, it can issue a single request for all 1000 files to the HRM. The HRM would have queued the 1000 files, and proceed to stage them as fast as the HPSS system will permit.

The SRMs use URLs (Uniform Record Locators) to refer to files they request or manage. 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.

There are several robustness features that our implementation of an HRM provides:

1. After a file is staged, the HRM checks for a successful completion code, but in addition it compares the size of the file on its disk cache to the number of bytes staged. If they don't match, it removes the file and re-issues the stage request.
2. If the transfer from HPSS is interrupted for any reason (such as the system brought down for maintenance), the HRM removes the partial file and re-issues the transfer request repeatedly until the HPSS system recovers.
3. If for some reason the client stops getting the files (for example, because of a network failure), the HRM disk space will eventually fill, and the HRM will wait until files are read and released.

These monitoring and recovery features are essential for the successful completion of a multi-file request. In the next section, we describe how the DataMover uses two SRM to achieve robust multi-file replication between two or more storage systems.

3. The DataMover

The DataMover is an asynchronous client program that is designed to interface to two SRMs, a source SRM and a target SRM. In the example discussed next, we use two HRMs: one developed to work with HPSS, called HRM-HPSS, and one that adapts the HRM-HPSS to work with a legacy mass storage

system at NCAR (National Center for Atmospheric Research), called HRM-MSS, making it accessible from the Grid. In Figure 3, we show how the DataMover interacts with these two HRMs. In the figure we describe a request to replicate an entire directory that could contain thousands of files.

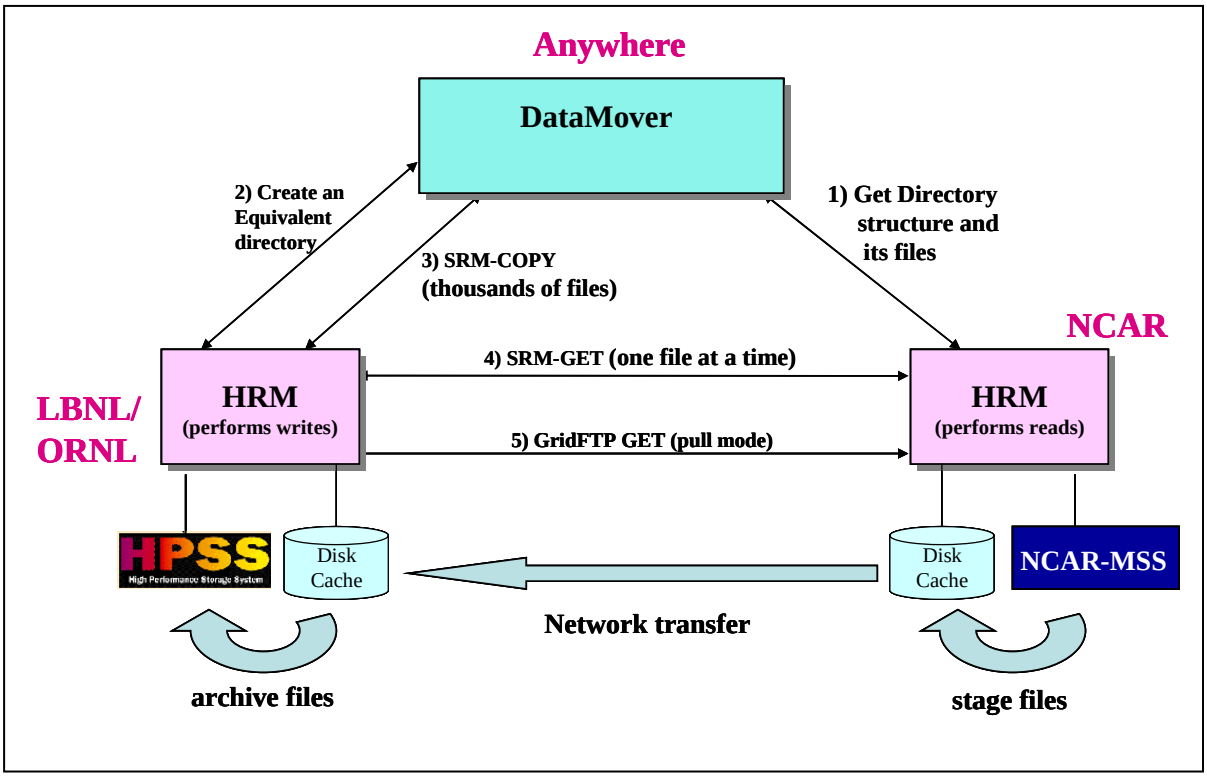

Figure 2: The use of two HRMs to achieve data replication between two incompatible mass storage systems.

Before space allocation and file transfer can take place, the DataMover interacts with the HRMs to generate equivalent directories. First it gets the entire directory structure from the source HRM for NCAR-MSS (step 1 in the figure). If the source MSS cannot provide this information in a single command, the HRM interacts with it recursively to obtain this information. It then sends a request to the target HRM-HPSS to create such a directory (step 2 in the figure). Here again the SRM usually needs to interact with the MSS repeatedly to build a multi-level directory. Then the DataMover constructs a multi-file request to move files from matching source to target locations in the corresponding directory structures (step 3 in the figure). At this point the DataMover can quit since the HRMs take over. They negotiate space and files transfer requests according to the detailed protocol described in section 2 (we show this in the figure as steps 4 and 5 without going into the detailed protocol).

By this design we achieved all the requirements we set up to achieve:

1. The DataMover is asynchronous – as soon as the multi-file request is constructed it quits.
2. Robustness is achieved by the inherent robustness of the HRMs when they stage files at the source and archive files at the target.
3. An additional robustness feature is provided by the target HRM when requesting file transfers over the network. The HRM monitors such pull transfers and reissues the file transfer request if the transfer fails.
4. Concurrency is achieved at all stages: when staging files from the source MSS, when transferring files over the network,

and when archiving files at the target MSS.

5. File transfer efficiency is achieved by using GridFTP or any other file transfer protocol that uses large windows and multiple streams.

In addition, since the HRMs are adapted specifically to the MSS, we were able to overcome security concerns. The files are read securely to the HRM disk internally to the site, and moved between the HRMs' disk caches using secure GridFTP. We were also able to overcome firewall issues by having the HRM behind the firewall initiate the file transfers (push mode) when necessary.

We have described above a DataMover between two HRMs. However, the DataMover have been designed to work in other situations as well. First, they can work with any SRM, including SRMs designed for disk storage only, called DRMs (Disk Resource Managers). Thus, for example, if a client DRM exists at the scientist's site, the scientist can request the DataMover to move files from one or more remote SRMs. Furthermore, we also implemented a DataMover that will move files directly to the user's disk space, but in this case the responsibility of space allocation and file removal fall on the user. In some cases that may be desirable, since it is not necessary to install as DRM at the client's site, and the client prefers to manage the space and file usage. Finally, we emphasize that a request to get multiple files to a target site can be made from many source sites, since the URLs contain information about the source sites.

4. The File Monitoring Tool

Monitoring the state of complex transfers is requires dynamic updating of the file transfer progress. Since the target SRM is the agent that issues the transfer request, and is monitoring the transfers for failures, we extended the SRM functionality so it communicates to a file monitoring tool (FMT) server daemon this information dynamically. We have also developed an FMT web-based client that requests updated information from the FMT server. The information comes in two forms. First, a summary that shows how many files transferred so far, total bytes, and any failures that have occurred including requests to files that do not exist, or sites that do not respond. Second, we have developed a graphical tool to show the progress visually. This is shown in Figure 3. It can be invoked from any web-tool at any time to check the progress of the multi-file replication request.

Figure 3 shows the graphical interface that indicates files already transferred (in green to the left), and files in the process of being transferred (in blue to the right). When touching a file on the screen, the bottom part of the display shows the source and target sites (as URLs), the file size, and the average file transfer rate. There is also a bar that shows the percent of total size transfer so far. After the entire transfer completes, a summary is displayed on successful transfers, as well as average transfer rate for the entire request. This tool is used routinely to monitor progress of massive file replications.

5. File tracking and bottlenecks

Our experience with large transfer was that end-to-end transfer rates varied in ways that we could not explain. We embarked on analyzing logs to understand where the bottlenecks were. We observed that we need to track what happens to each file over time. There are several events that need to be tracked: when the request for the file was made, when did staging started and ended, when did transfer over the network started and ended, and when did archiving started and ended. We found that a particular technique, used in Netlogger [6] offers an effective methodology for visualizing the sources of bottlenecks. This visualization method is shown in Figure 4. The horizontal access represents time, and the vertical access has points on it representing events of file tracking. A connected line represents the events for a particular file. Figure 4 shows the tracking of a real run of archiving files from a disk cache into a mass storage system using HRM. Thus, the events are: Space Reserved (at the HRM's disk cache), GridFTP start, GridFTP end, Archive start, Archive end. We chose to display only every 10^{th} file so as not to clog the display.

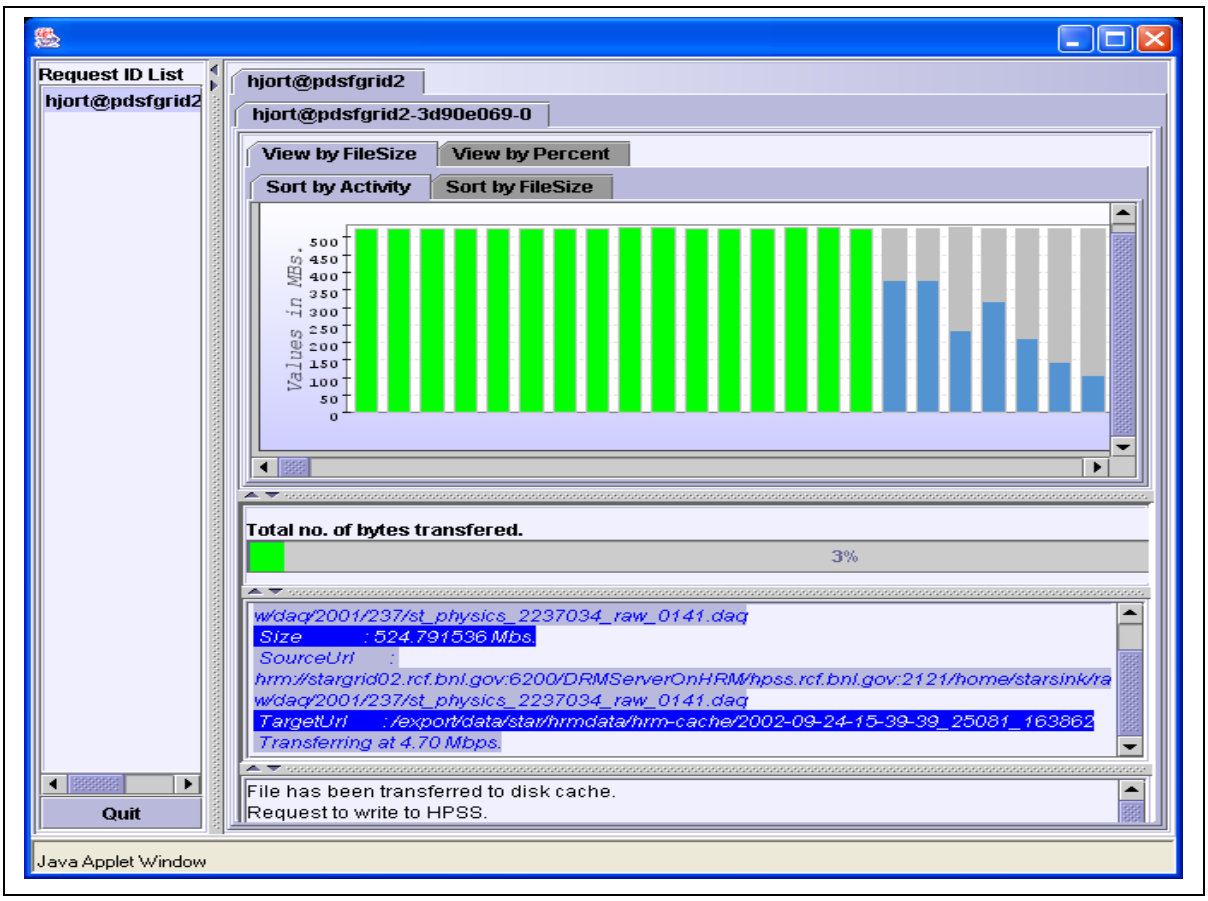

Figure 3: dynamic tracking of file movement

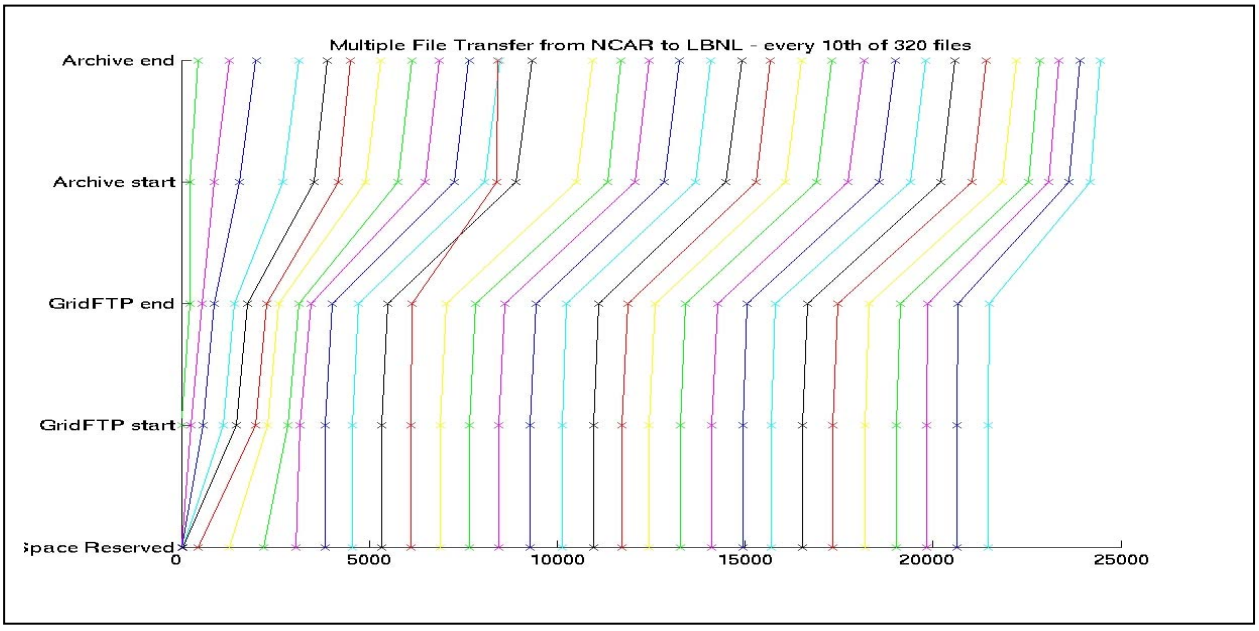

Figure 4: File tracking helps identify bottlenecks; archiving was the bottleneck

All the files moved were of the same size, except one (13th from the left) that was a very small file, and therefore was archived earlier. The interesting property to observe is that the lines between "GridFTP end" and "Archive start" are more slanted; that is this step took more time. The implication is that files were left in the HRM's cache for a long time before they were submitted for archiving. The explanation for that is that archiving at the MSS was slower than the rest of the process, including movement over the network. In general, one can observe that the event in front of an overly slanted lines causes the bottleneck.

These logs can also be used to track transient system failures and proper recovery by the SRMs. Figure 5 shows the entire run of moving 318 files (total of 45 GBs) from Brookhaven National Laboratory (BNL at the east coast of the US), to Lawrence Berkeley National Laboratory (LBNL at the west coast of the US). There were two HPSS systems involved, one at each site. As can be seen from the tracking graphs, shortly after the beginning of the file replication process, the source system at BNL was not available for quite a long time. Perhaps it went down for maintenance. The DataMover system recovered and continued. This was followed by a system failure at the target site at LBNL, where files were archived. Then, there was another shorter transient failure at the source system. In spite of these failures the data replication process completed successfully, which proves the robustness of this system. Another thing that can be observed from this figure is that the overly slanted lines occur just before "gridFTP start", which implies that the network was the bottleneck for this run.

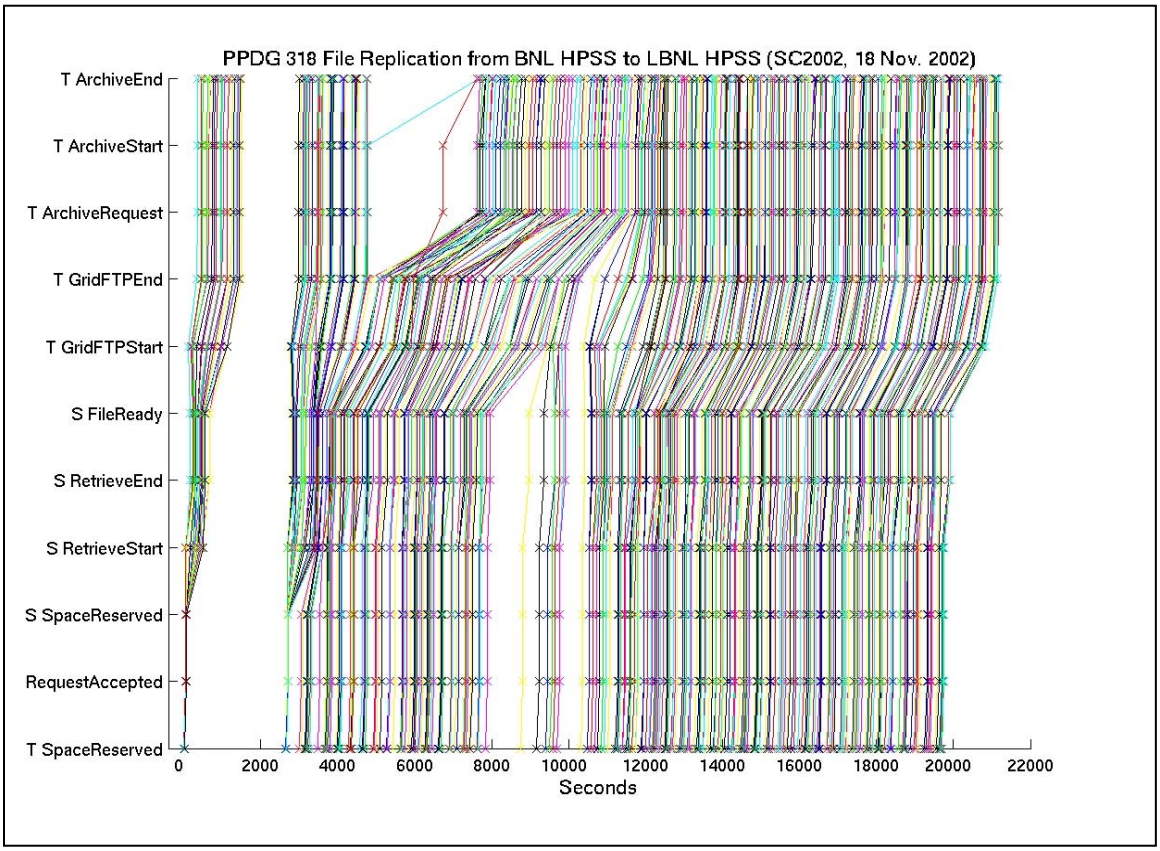

Figure 5: file tracking shows recovery from transient failures

The above shows that information from the logs can be used to identify bottlenecks and to improve the system's resource usage. Often, one of the hardest things to predict is where to provide additional resources, such as getting more disk cache, more tape drives, improve the network, etc. This methodology can help in such decisions.

6. Conclusions

In our work with application scientists from various domains, we hear various stories on their perception of what are difficult data management problems. We were most surprised to hear that one of the most time-consuming, annoying, disruptive aspects of their work was massive file replication. After analyzing the problem, we concluded that it is indeed a difficult problem, one that requires efficiency gains as well as absolute robustness. The task could be difficult, but we have realized that technology being developed for the Grid in the area of Storage Resource Management can be used effectively to solve this problem. There are a few lessons we have learned.

Perhaps the most important lesson is that it is very economical to build complex systems by using existing robust software modules – the building block approach. This is a well known truism, but we did not expect this to help for the problem at hand. However, even though the SRM software modules were developed for Grid storage management, not for managing file replication, the functionality of supporting multi-file requests for clients by SRMs could be applied to the file replication task. Furthermore, the SRM itself uses another service, the GridFTP, to achieve secure and efficient file transfer. By using this basic file transfer service and additionally providing concurrent staging, transfer and archiving, more efficient use of the systems could be achieved.

A second lesson is that by achieving an agreement on the functionality of the SRM service, and developing a standard API, it is possible for incompatible systems to interact using compatible interfaces. This in itself is an obvious principle, but it was rewarding to see how well it works by placing SRMs in front of various file and mass storage systems.

A third lesson is that although achieving robustness in the face of multiple system failures is quite difficult, placing the right monitors and recovery mechanisms in the critical paths can achieve this goal. Since failures occur only during small percentage of the time, the recovery does not have to be efficient. In the case of file staging, transfer, and archiving failures, it is sufficient to re-issue the request. More sophisticated mechanisms of partial file recovery are not necessary.

It is worth noting that this problem is also very important to the business world. It has been addressed by industry for specific products for the purpose of protecting important data. For example, the Symmetric Remote Data Facility (SRDF) form EMC, provides remote data replication between online storage systems [7]. Such solutions are usually associated with a specific product and do not support interoperability between diverse storage systems, and especially between mass storage systems.

The system we have developed has been in daily use by one High Energy Physics project, and in frequent use by a Climate Modeling project. The modular design permitted incremental scaling of the products to support transfer requests robustly for thousands of files, and hundreds of gigabytes of data replication in a single request.

Acknowledgements

We gratefully acknowledge the benefit of working with various application scientists on identifying the problems and using the DataMover software. Most helpful were people from the STAR experiment, including Doug Olson, Eric Hjort from LBNL, and Jerome Lauret from BNL, as well as people from the Climate Modeling community, including Mike Whener from LBNL, Gary Strand and Adrianne Middleton NCAR. 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] Arie Shoshani, Alex Sim, Junmin Gu, Storage Resource Managers: Middleware Components for Grid Storage, Nineteenth IEEE Symposium on Mass Storage Systems, 2002 (MSS '02).

[2] Arie Shoshani, Alexander Sim, and Junmin Gu, Storage Resource Managers: Essential Components for the Grid, in *Grid Resource Management: State of the Art and Future Trends*, Edited by Jarek Nabrzyski, Jennifer M. Schopf, Jan weglarz, Kluwer Academic Publishers, 2003.

[3] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high performance computational Grid environments. *Parallel Computing Journal*, 28(5):749–771, 2002.

[4] The Globus Alliance, http://globus.org/.

[5] HPSS. High Performance Storage System, http://www.sdsc.edu/HPSS, San Diego Supercomputer Center, La Jolla, CA, 1997.

[6] D. Gunter, B. Tierney, K. Jackson, J. Lee, M. Stoufer, Dynamic Monitoring of High-Performance Distributed Applications, Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing, HPDC-11, July 2002

[7] http://www.emc.com/products/networking/srdf.jsp