THE DATA SERVICES ARCHIVE

Rena A. Haynes
Sandia National Laboratories, MS 0822
Albuquerque, NM 87185-5800
Tel: +1-505-844-9149
e-mail: rahayne@sandia.gov

Wilbur R. Johnson
Sandia National Laboratories, MS 1137
Albuquerque, NM 87185-5800
Tel: +1-505-845-0279
e-mail: wrjohns@sandia.gov

Abstract

As access to multi-teraflop platforms has become more available in the Department of Energy Advanced Simulation Computing (ASCI) environment, large-scale simulations are generating terabytes of data that may be located remotely to the site where the data will be archived. This paper describes the Data Service Archive (DSA), a service oriented capability for simplifying and optimizing the distributed archive activity. The DSA is a distributed application that uses Grid components to allocate, coordinate, and monitor operations required for archiving large datasets. Additional DSA components provide optimization and resource management of striped tape storage.

1. **Introduction**

In the ASCI environment, deployment of massively parallel computational platforms and distributed resource management infrastructure for remote access allows computations to execute on the platform that can best perform the calculation. Data generated by very large calculations can be hundreds of gigabytes to terabytes in size. Datasets are typically distributed across hundreds to thousands of files that range in size from megabytes to gigabytes. Extracting information from the data and transporting the reduced data to a local platform that can provide the interactivity required for the analysis is used to accomplish analysis and visualization of large datasets.

Large datasets typically remain in the site that generated the data until they are archived to a locally managed parallel tape storage system (HPSS [1]). Although high-speed wide area networks and parallel file transfer components allow high performance movement of data, the amount of time required to transmit large datasets allows opportunities for errors that abort the transfer. Recovery from aborted transfers requires retransmission of files. A disk file system is used to buffer the dataset files before they are transferred across the local area network to HPSS parallel tape storage. Both the wide-area and the local-area transfers cause sustained high-speed bursts of data. Until the file data has been placed onto tape storage, files remain on file system storage. Without resource control and coordination, multiple occurrences of this activity can overwhelm both the intermediate file system and the archival system, especially as more capability platforms, like ASCI Q, Purple, and Red Storm, are deployed.

Grid middleware [2] can address some of the issues involved in archiving large distributed datasets. Core services in Grid middleware provide uniform methods for scheduling, allocating resources, and monitoring data transfer processes on distributed systems. More recently, middleware developed for Data Grids address issues involving data location, storage resources, and cache management. The SDSC Storage Resource Broker (SRB [3]) supports location transparency and data replication. Storage Resource Managers [4], [5] implement storage access and cache management policies. Current Data Grids [6], [7], [8], [9] are built to support efficient and cost effective access to large data collections for a geographically distributed community of scientists. Because of this, the focus has been on enabling distributed access to a store of metadata and data, which may also be replicated in the data grid. Our focus is on usability, robustness, and resource issues involved in transferring large datasets to a parallel tape archival storage.

We describe the Data Service Archive (DSA), a service oriented capability for simplifying and optimizing the distributed archive activity. The DSA is a distributed application that uses Grid components to allocate, coordinate, and monitor operations required for archiving large datasets. Additional DSA components provide optimization and resource management capabilities for striped tape storage. The following sections present requirements, a functional overview, and performance of the DSA application. Planned extensions to the DSA are also discussed.

2. High Level Requirements

The requirements for the DSA include the usual requirements of a simple, easily managed user interface to make archival and retrieval requests. Ease of use includes desktop access to clear interfaces for control and status as well as to a well-defined mechanism for making archive requests. The DSA should mitigate complexities of the distributed environment by providing a common interface regardless of data location and recovery from system failures where possible; however, file and resource brokering are not required. Any desktop software should have minimal prerequisites with automated installation and update procedures. Command line invocation is also required to support requests from running processes.

Resource management requirements for the DSA include managing concurrent archival requests, scheduling requests and data transfers, optimizing data transfers, and balancing the load on the file system cache and the tape storage system. Optimizing data transfers and balancing the load on the parallel tape storage system requires obtaining knowledge of network topology, tape drive resources, and striping policies of the tape system.

Software management requirements call for a clean integration with HPSS, preserving the storage system namespace and storage system policies. Accessing the storage system must be through supported mechanisms.

Additional functional requirements for the DSA include support for data integrity features, multiple storage patterns, and archive persistence management. Data integrity features should be available to verify correct wide area network transfers as well as the final image on tape storage. Storage patterns initially to be supported include requests to create an archive of tarred files as well as a directory hierarchy of files. Archive persistence introduces the notion of useful data lifetime for an archive.

3. DSA Architecture and Functional Overview

The DSA is a distributed application with components (shown in Figure 1) located on the user's desktop, a web server, the remote platform where the dataset resides, and the local data analysis cluster that is integrated with HPSS.

The DSA GUI is started from the user's desktop by accessing the Data Services DSA URL, which automatically checks for updates and optionally downloads the latest software release. The GUI allows users to define an archive request, which requires filling in a minimal amount of information required for data transfer. Features for archive formats and data integrity options may also be selected. An optional comment field is available that can be viewed in context of the archive action request. When the request is submitted for execution, DSA prompts for an identification that will be used to generate an archive identification that uniquely identifies this archive action from all others. The user may check status from the GUI or close it and check status at a later time through the information management service interface in the GUI.

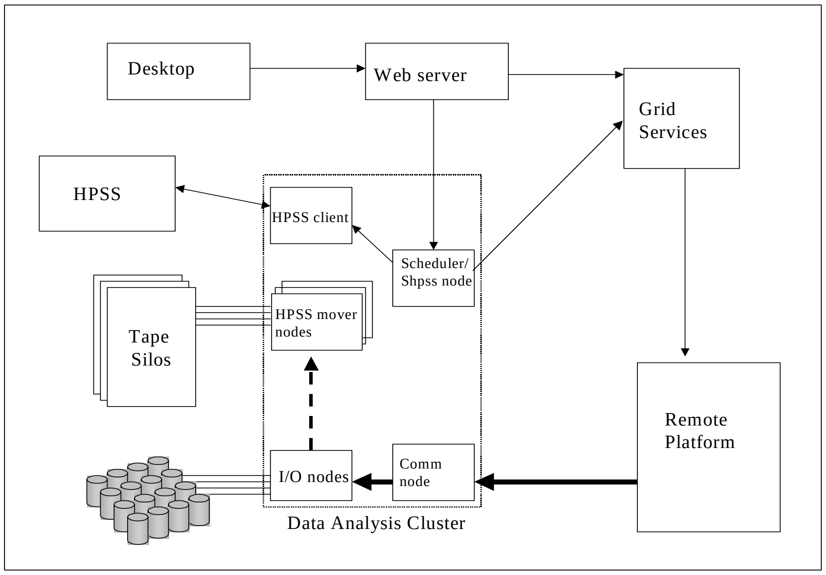

Figure 1.

Desktop software interacts with a servlet on the DSA web server to transmit the archive action request. The servlet maintains state throughout the archive process to enable recovery, restart, or cancellation. The servlet creates metadata that identifies the archive request, then starts a two-step process to archive the data to HPSS. The servlet constructs and submits a transfer request to the scheduler on the data analysis cluster. Once scheduled, the request is sent to the grid services application server, which starts an instance of the dataset transfer program (Zephyr) on the remote platform in behalf of the user.

Zephyr, which runs with the user's credentials, determines what files are to be transferred and how the transfer is to take place. If verification is requested, a checksum is generated for each file and the checksum information is placed in the archive. If a format option is selected, the data is processed on the remote platform. After completing the information gathering and formatting steps, Zephyr transfers the data to a Parallel Virtual File System (PVFS [10]) disk storage on the data analysis cluster using a parallel file transfer protocol. This completes the first step of the archive process.

The current DSA implementation does not preallocate space on the PVFS disk cache. If a storage ceiling threshold is reached on the file system, the DSA service becomes unavailable until space is reclaimed. Dataset transfer operations detect the error and may be recovered through normal DSA recovery mechanisms.

The transfer to HPSS is scheduled immediately after completion of the transfer from the remote platform by a component called shpss. When data files are transferred as a directory hierarchy, shpss places files in groups called partitions. Files in a partition have the same approximate file characteristics (currently file size) so they will be placed in the same storage class on HPSS and, consequently, use the same tape resources. The number of files in a partition is limited to facilitate recovery if errors occur and to improve overall transfer throughput by preventing large transfers from starving small transfers in the scheduling process.

Shpss acts as tape resource manager by placing partition transfers in queues based on the partition file characteristics. The shpss queues use a node allocation mechanism to model the scheduling activity so that it tracks HPSS allocation of tape units. Use of this node allocation mechanism allows management of queues and queuing policies through standard system queue manager commands. If enough tape units are available, the transfer starts immediately. The partition transfer uses the HPSS parallel file transfer protocol interface with local file system commands to move data from PVFS on the data analysis cluster to tape over the cluster interconnect. Once files in a partition have been transferred to HPSS, they are removed from the PVFS.

DSA retrieval requests are initiated through the DSA user interface, as are archival requests. Users can request retrieval of individual files or directories of files based on the HPSS namespace to a target system and directory. Retrieval requests, like the archival requests, are processed by web service middleware that maintains state and starts a two-step process to retrieve the data from the tape storage system and place the data onto the target directory. Shpss is launched on the data analysis system to stage the data from tape storage to the PVFS disk cache. As in archival requests, shpss groups file retrievals into partitions and places each partition into a scheduling queue based on the tape resource requirements. If tape resources are available, transfers start immediately. When all data for a retrieval request has been staged to the disk cache, the web middleware starts a Zephyr process to move the data to the target directory.

For shpss to set up partition groups for a retrieval operation, it must query HPSS to obtain information about each file in the retrieval request. Currently, only tape striping information is obtained. File media identification information is not queried. Delays can occur if files are not co-located on media or they are retrieved in a different order than stored.

The DSA gives users complete control over the portion of the HPSS namespace where their data is written. Therefore it is possible for more than one archive request to reside in a common part of the namespace (e.g. subdirectory). As HPSS performance can be affected by read requests not matching the tape file patterns created during the write process, it is possible for users to present read requests through the DSA that are less than efficient. In the ASCI environment, retrieval requests from the tape archive are made much less frequently than storage requests. Considerable thought was given to tracking tape storage patterns within the HPSS namespace. It was determined that at the present time the cost/benefit of ordering read requests to tape file images was not economical in light of the amount of work required to add the capability and the complexity introduced into the overall system. If a read request points to the 'top' of a single archive in the HPSS namespace, the file read ordering will be the same as the original write ordering

4. DSA and other Technologies

Sandia National Laboratories has had a grid in place since 2001 [11] based upon a workflow processor layered on top of the globus toolkit. Being a distributed application, the DSA exploits grid technologies by using the grid workflow processor to sequence the Zephyr and shpss components. The Globus toolkit is used for submitting partitions when scheduling transfers within shpss. Some modifications were made to the globus toolkit to enhance the operation of the DSA. Run time stdout/stderr capability was added to get immediate feedback from components running in PBS, and the ability to request DSA specific resources within the cluster where added.

As mentioned above the DSA is manipulated using servlets on a web server. This presents a well-defined API for building additional features into the system. The DSA, through the web server, can be managed either through traditional POST and GET operations through HTTPS, or by passing java objects to requisite servlets themselves. The SimTracker [12] application, being developed at the three primary weapons laboratories, is in the process of integrating DSA into its functionality through this interface.

The DSA does not use data grid technologies in the classic sense, although it is possible that gridftp [13] could be used for remote transfers. This is primarily due to a differing set of requirements between the programs. The file-tracking requirement employed in the data grid is not necessary since there is only one HPSS system where data is stored to tape and the user controls where files go in the HPSS namespace. An evaluation of current requirements is underway that may lead to rudimentary replica management to enhance the robustness of the archive.

The ability to integrate DSA with other archiving technologies or intermediate data stores is also being examined. The infrastructure where DSA resides includes a file migration capability to assist analysts with moving data between computational resources. Each resource has its own file system(s), which are dedicated to work performed on the resource. The DSA has access to these resources and can read or write data to and from any system supporting parallel ftp.

5. Performance

Moving large data sets within the ASCI environment creates contention for infrastructure resources. The impact and severity of this contention is dependent upon the location of the data and the performance capability of the resources affected. Significant effort has been put forth to implement performance enhanced resources. High-speed networks, parallel file systems and HPSS are examples of resources where significant investment has produced high performance point solutions within the distributed architecture. However, relying on point solutions as general mechanisms for moving large amounts of data is often inefficient, impractical or requires a large investment in user effort and education.

The DSA examines three key areas that affect performance when users manually store data in the HPSS system—resource contention, storage patterns, and scheduling algorithms. HPSS integration in the computational environment looks at issues involved in resource contention in normal and failure conditions. Storage patterns affect how efficiently an archive can be placed into HPSS as well as how efficiently the tape storage is used. Scheduling algorithms impact system overhead and tape resource. HPSS schedules tape mounts within the system where it has knowledge only of the current file request and not the entire archive request. Integrating advanced scheduling with HPSS can eliminate system overhead by holding transfer sessions that cannot immediately be serviced and increase tape utilization by the algorithms used to select transfers to run.

5.1 HPSS Integration in the Computational Environment

Upon completion of a computation, data is not 'local' to the HPSS system. By local it is meant that there are systems and networks extraneous to HPSS that are involved in the movement of the computational data. These systems and networks are not dedicated to the archival process, thus are shared with other processes causing potential bottlenecks and points of failure. Contention for resource can affect the performance of data movement causing poor performance for the activity at hand as well as for other work that is contending for that resource. While poor performance can be unacceptable in the computational environment, in the worse case, resource failure can occur.

Intermittent resource failure is considered performance degradation, and must be managed by the DSA. In a manual process this is often handled by restarting the entire archival operation, as the user does not have the capability or time to manage intermediate restarts. Whether the simulation data resides on the same network as HPSS or across the wide area, it may not be practical or possible for the user to manage the involved systems and networks. Thus the most practical method for eliminating the probability of resource failure is to reduce the number of failure points in any given process.

In order to reduce resource contention, the DSA moves data to a file system that is directly accessible by HPSS tape movers. At first glance this seems counterproductive since the data is actually moved twice. However experience has shown that increased performance in network throughput and the localizing process of data to HPSS outweighs the duplicated movement. This is especially true when the data resides at a remote location over the wide area where the network is unable to reach even modest levels of performance.

5.2 Storage Patterns

The HPSS system achieves much of its performance by striping files across multiple tape drives. This allows data to be written in parallel increasing writes speeds by a factor equivalent to the number of tape drives involved in the write process. The number of tape drives used during a write is determined by two factors:

1. The size of the file will determine a stripe width through an internal mapping defined in HPSS.
2. The ftp client may force a stripe width through a class of service request.

Both of the above features have tradeoffs in either HPSS overhead or in the ability to efficiently read data back from HPSS. The user does not want to be concerned with the details of how to effectively write data to HPSS. Furthermore, the ftp client does not lend itself to storing multiple directories of varying file sizes, some very large and some relatively small. This often leads to bulk transfers on a directory basis regardless of file size and the number of striping changes that can occur.

Using the internal mapping mechanism can cause tapes to be mounted and dismounted many times during a transfer. Long delays may be experienced due to waiting for one or more tape units to become free. In an oversubscribed HPSS system, the results can be drastic as data transfers contend for tape units. Client connections time out, small data transfers are starved by large transfers and the user must be vigilant in monitoring the entire activity to assure all their data is successfully stored.

By using the client specified mechanism, all data is stored in the same striping scheme. If the client chooses a stripe width that is too narrow, large files are written inefficiently which also affects HPSS overall throughput. If the client chooses a striping width that is too wide, small files are split across many tapes and the transfer must wait for drives to become available.

Again, in an oversubscribed HPSS system, these effects are magnified.

By using either of these methods, large data movements starve smaller ones. This has two affects on practical data storage. First, by design, the ftp client-server model can time out if asked to wait too long for an activity. Second, in the ASCI environment, security credentials can time out causing any automation of transfers to fail.

The DSA manages these issues by partitioning an entire transfer, including files in multiple directories, into data movements that share a common stripe width. Each partition is transferred one after the other at a time determined by the DSA scheduler (see below). Partitions also have finite length thus allowing the scheduling of smaller transfers between larger ones.

5.3 Scheduling Techniques

Integrating with a sophisticated system such as HPSS can be performed in two ways. The DSA can both tightly couple with internal information and state through some well-defined interface, or it can couple in a loose fashion by modeling the HPSS information and state through an external mechanism. The DSA chooses the latter in order to simplify its architecture and reduce development time. DSA transfers data using the common HPSS ftp client and models tape drive allocation using a batch queuing system scheduler.

There are two primary reasons to schedule data transfers into HPSS. First is to put on hold any ftp sessions that cannot be immediately serviced by HPSS, thus eliminating unnecessary system overhead. This situation is made worse when an active ftp connection that is transferring data alters its required HPSS resource (number of tape drives) and cannot immediately be serviced. Such situations can result in protocol timeouts and a general level of confusion on the part of the user. The second reason for scheduling data transfers is to increase HPSS utilization.

HPSS schedules tape resources in a FIFO manner when a file is opened for reading or writing. Such FIFO queuing is known to be less than optimal when there are free resources (tape drives) and a pending request that could use those resources but is not at the head of the queue.

The DSA models HPSS storage resources in a similar fashion as nodes in a cluster. The number of tape drives is managed by the queuing system and the DSA requests a specific number of tape drives when submitting a transfer. This requires the DSA to examine the size of the data to be transferred and to map that size to a particular stripe width. This also requires that data be grouped into partitions that do not violate HPSS stripe width policy for any given transfer request.

In order to optimize transfer requests, the batch queuing system scheduler is configured to use a technique called backfill. When scheduling jobs (transfers) using backfill, a standard FIFO queue is employed to determine when a job should start. However, when the job at the head of the queue cannot be started because the amount of free resources is not sufficient, the scheduler looks 'back' into the queue to see if any other pending job can be satisfied by the amount of free resources. If such a job exists and the amount of time required by that job is not longer than the wait time for the request at the head of the queue, the backfill job is started. Such a technique requires that DSA calculate the amount of time required for a transfer to take place.

Determining the exact amount of time required to transfer a group of files into HPSS is not necessarily possible. While transfer rates onto tape are fairly well behaved, it is not possible to determine with high certainty, the number of tapes that will be required to service a particular transfer which effects the calculation of total transfer time. Hardware compression, the amount of existing data on a tape and the number of tape loads that must be serviced prior to any given load operation all play a roll in the level of uncertainty. The DSA uses the following equation to estimate the amount of time required to perform a given transfer:

$$T_{\mathrm{job}} = T_{\mathrm{login}} + \sum_{\mathrm{i=1}}^{\mathrm{i=N}} \left[T_{\mathrm{startup}} + \left\lceil \left(X_{\mathrm{i}} / R_{\mathrm{rate}} \right) \right\rceil + \left(S_{\mathrm{width}} \times T_{\mathrm{load}} \right) \right]$$

The total time required to perform a transfer (T_{job}) equals the amount of time required to perform an ftp client login (T_{login}) plus the sum of the amount of time for each individual file to transfer. The time to transfer an individual file is a factor of some startup time for the transfer (T_{startup}), the file size (X_{i}) divided by transfer rate (R_{rate}), and the time required to perform a new load of all required tapes ($S_{\mathrm{width}}\mathrm{x}T_{\mathrm{load}}$). It is known that this estimation is less than optimal since files can span tape volumes, which would require additional tape loads. It could be possible to obtain a fair estimate of how many tape loads a particular transfer will require, but at this point we leave such optimization to future work.

Lastly, knowing that external influences can affect a transfer and cause the time needed to move the data onto tape to exceed the requested job time, the DSA has the ability to track what files have been successfully moved. This permits ‘retrying’ files that did not move by creating a new partition containing the files that did not transfer and repeating this process.

5.4 Transfer Comparisons

A complete treatment of DSA performance is a paper unto itself. We try here to give a heads up of how well the unique features of the DSA compare to conventional manual processes for moving data. There are three questions we want to answer:

1. Does the partitioning of data files into common storage class transfers increase performance over the practice of 'mput *'?
2. Can we schedule data for transfer that will make more efficient use of HPSS resources (tape drives) and increase throughput?
3. What performance is gained when minimizing network resource utilization by staging data to a file system that is directly accessed by the HPSS movers?

To help answer these questions we ran simple experiments that transferred data into HPSS in a controlled environment. No other activity was permitted when the experiments where run. Care was taken to keep comparisons as equal as possible taking into account that seek times would increase and deferred mounts could skew results when compared to actual mounts.

For the first question we ran ten transfers—five for an shpss run and five that performed the local file mput for pftp. All transfers used the same data set that was located on a file system mounted on the machines where the HPSS mover processes ran. Sufficient time was given between transfers to allow HPSS to settle into a common state. The data set was approximately 34 gigabytes of simulation data made up of 23 files purposefully named such that HPSS would have to request a varying number of tape drives during the pftp process.

The results were that the standard local file pftp processes averaged 15 minutes to perform the transfer while the shpss processes averaged 13 minutes. We expect this difference to grow, in favor of shpss, when the number of files and changes in class of service increases.

To examine scheduled transfers, the same considerations above were used. The HPSS system was configured with eight tape drives. Two sessions were run with each transferring nine data sets concurrently. One session started nine shpss processes in a scheduler configured to perform backfill, and one that simply ran nine different pftp sessions using mlfput. The average time taken to complete the nine pftp/mlfput transfers was 33 minutes. The average time for the nine shpss sessions to complete was 27 minutes.

To evaluate the affect of reducing network resource utilization, the first suite of tests was rerun with the standard pftp process using the normal mput command that transfers data to HPSS movers over the local area network. Results from this test showed an order of magnitude improvement in performance when the locally accessible file system cache was used.

In all, this is a brief glimpse at performance. We feel that other variables in HPSS not present during these tests could alter results. Further investigation is planned to help understand the DSA environment better and improve its performance.

6. Conclusions and Future Work

.

The focus of the DSA work to date has been on simplifying and optimizing the process of archiving large datasets. Initial testing indicates that reducing resource utilization, managing storage patterns, and scheduling transfers have accomplished performance improvements. In the near future we will be integrating the DSA service with a simulation tracking service, which maintains metadata and snapshots of results from simulation runs. We also plan to support video archiving for a video editing system. We expect these applications will require additional cache management support for retrieval operations.

Acknowledgments

The work was performed at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

References

[1] HPSS, High Performance Storage System, http://www4.clearlake.ibm.com/hpss/index.jsp .

[2] Foster, I., KesselMan, C., Tuecke, S. “The Anatomy of the Grid: Enabling Scalable Virtual Organization”, The International Journal of High Performance Computing Applications, Vol. 15, 2001, pp. 200-222.

[3] Rajasekar, A., Wan, M., Moore, R. “MySRB & SRBB – Components of a Data Grid”, 11th International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 24-26, 2002.

[4] Shoshani, A., Sim, A., Gu, J. “Storage Resource Managers: Middleware Components for Grid Storage”, 19th IEEE Symposium on Mass Storage Systems, 2002.

[5] Shoshani, A., Bernardo, L., Nordberg, H., Rotem, D., Sim, A. “Storage Management for High Energy Physics Applications”, Computing in High Energy Physics, 1998.

[6] Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S. “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets”, Journal of Network and Computer Applications, 23, pp. 187-200, 2001.

[7] Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., Stockinger, K. “Data Management in an International Data Grid Project”, IEEE ACM International Workshop on Grid Computing (Grid 2000), December, 2000, Bangalore, India.

[8] PPDG: Particle Physics Data Grid, http://www.cacr.calteck.edu/ppdg.

[9] Tierney, B., Johnston, W., Lee, J. “A Cache-Based Data Intensive Distributed Computing Architecture for “Grid” Applications”, CERN School of Computing, September 2000.

[10] Ligon, III, W.B., and Ross, R. B., "PVFS: Parallel Virtual File System," Beowulf Cluster Computing with Linux, Thomas Sterling, editor, pages 391-430, MIT Press, November, 2001.

[11] Beiriger, J., Bivens, H., Humphreys, S., Johnson, W., and Rhea, R., "Constructing the ASCI Computational Grid," Ninth IEEE International Symposium on High Performance Distributed Computing, August, 2000.

[12] SimTracker. http://www.llnl.gov/icc/sdd/img/images/SimTrack.pdf.

[13] Laszewski, G., Alunkal, B., Gawor, J., Madhuri, R., Plaszczak, P, Sun, X. “A File Transfer Component for Grids”, 2003 International Conference on Parallel and Distributed Processing Techniques and Applications, June, 2003, Las Vegas, Nevada.