Computer Sciences Department

Building Data-Pipelines for High Performance Bulk Data Transfers in a Heterogeneous Grid Environment

Tevfik Kosar
George Kola
Miron Livny

Technical Report #1487

August 2003

UNIVERSITY OF
WISCONSIN
MADISON

Building Data-pipelines for High Performance Bulk Data Transfers in a Heterogeneous Grid Environment

Tevfik Kosar, George Kola and Miron Livny
Computer Sciences Department
Uversity of Wisconsin-Madison
1210 West Dayton Street
Madison, WI 53706
{kosart,kola,miron}@cs.wisc.edu

Abstract

The drastic increase in the data requirements of scientific applications combined with an increasing trend towards collaborative research has resulted in the need to transfer large amounts of data among the participating sites. The heterogeneous nature of the storage systems employed by the different sites makes transfer of data among them a difficult problem. The general tendency has been to either use simple scripts which require human intervention to deal with failures, or dump data to tapes and mail them. We introduce a method to build and operate data-pipelines between mass-storage systems lacking a common interface. This method can be applied easily and efficiently to transfer data between various mass storage systems. It does not need any human intervention during transfers, and it can recover automatically from various kinds of storage system, network, and software failures, guaranteeing completion of the transfers.

1. Introduction

With the increase in collaborative research, the amount of data that has to be transferred between participating sites is increasing. In many cases, due to the lack of a common interface and the know-how to perform high performance bulk data transfers, researchers have resorted to dumping of data to tapes and shipping them via Federal Express [6]. Owing to the various data grid initiatives [11] [13] [21], the underlying network capacity has increased enough to be able to support such bulk data transfers. The lack of a common interface between mass-storage systems necessitates the building of a data-pipeline via intermediate nodes. Appropriately designed data-pipelines can provide additional functionality, improve performance and ensure fault isolation.

The issues involved in designing data-pipelines are not well understood. Furthermore, automatic failure recovery has not been attempted in data-pipelines. In this work, we present a method to build data-pipelines and automate their operation. We also discuss the functionality of the data-pipelines we have built and report their performance. We show the different failures that occurred during the operation of our pipelines and the extent of our success in handling failures in an automated manner. To the best of our knowledge, this is the first work of its kind in transferring data between heterogeneous systems in a fully automated manner.

2. Motivation

Heterogeneity is inherent in a grid [7] computing environment. Multiple protocols have been developed for accessing data with each having some advantage over the other. The only way to transfer data between heterogeneous systems lacking a common interface is to build a data-pipeline via intermediate nodes. Unfortunately there is no documented work on designing such a pipeline or comparison of different pipeline designs. Furthermore we have not come across any effort to automate the operation of such pipelines or report the issues involved in operating such a pipeline.

Building a data pipeline with a single intermediate node is generally sufficient to transfer data between heterogeneous systems. On the other hand, building a pipeline with two intermediate nodes, one close to the source and another close to the destination provides additional functionality and potential performance improvement.

Mass-storage system protocols are designed for local-area access and may not work well in the wide-area. Building a pipeline with two cache nodes lowers the load on the

mass-storage servers as they have to transfer data only in the local-area network to/from the cache nodes and do not have to handle wide-area failures. It improves wide-area performance as we can employ wide-area optimized protocols for data transfer between the cache nodes. It gives us fine granularity of control over wide-area network resource usage. We can employ data transfer applications which have bandwidth-regulation capabilities [22]. This allows us to enforce a policy that best suits our needs. For example, we can transfer data between the cache-nodes at 200 Mbps during night and at 50 Mbps during the day if we desire to use the idle bandwidth during the night and not load the network during the day.

Because we are transferring large amounts of data, TCP checksum may not be sufficient [20]. With two cache nodes, we can increase the reliability of the transfers by computing a md5 or similar checksum at the source cache node and verifying it at the destination cache node. Similarly we can devise ways to ensure that there is no data-corruption in the data flow between the mass-storage and cache nodes.

The complexity of mass-storage systems makes problem identification difficult. If we were to perform a direct transfer between two mass-storage systems and encounter a problem, we would have difficulty identifying whether the problem is at the source mass-storage, destination mass-storage, the wide-area network or a combination of them. Building a pipeline with the two cache nodes allows us to locate the fault easily.

With the two cache-nodes, we can insulate the mass-storage servers from wide-area failures providing fault isolation and fault tolerance. Typically mass-storage servers have periodic scheduled maintenance. If we know the time window of scheduled source mass-storage maintenance, we can aggressively cache data at the source cache node before the maintenance and keep data flowing during the maintenance window. Similarly we can buffer data at the destination cache node during the destination mass-storage maintenance.

3. Related Work

Allcock et al [1] introduce the GridFTP protocol and Replica Catalog and discuss how they can be used for secure and efficient data transfer and data replication. Reliable File Transfer Service(RFT) [19] allows byte streams to be transferred in a reliable manner. It is able to handle a wide variety of problems like dropped connections, machine reboots, and temporary network outages automatically via retrying. Kangaroo [23] provides high throughput wide-area data movement for remotely executing jobs by overlapping CPU and I/O. Kangaroo also has a certain degree of fault tolerance to cope with failures that occur in the wide-area. GridFTP, RFT and Kangaroo are tools that can be used to move data between systems supporting their interface, but they cannot move data between heterogeneous storage systems lacking a common interface.

Feng [6] mentions a case where visualization scientists at Los Alamos National Lab dump data to tapes and send them to Sandia National Laboratory via Federal Express as it is faster than electronically transmitting them via TCP over the 155 Mbps(OC-3) WAN backbone.

Lightweight Data Replicator (LDR) [15] can replicate data sets to the member sites of a Virtual Organization or DataGrid. It was developed for replicating LIGO [17] data, and it makes use of Globus tools to replicate data. In its present form, LDR expects the use of a single data transport protocol (GridFTP). Our work deals with transferring data between systems which do not support a common data transport protocol. Further we feel that our work is more general in nature and can also be used for replication. Our work also exposes the different types of failures that can occur in such heterogeneous transfers.

4. Methodology

In our approach, we regard data transfers as real jobs which need to be queued, scheduled and managed just like computational jobs. Due to the storage limitations of intermediate nodes in a data-pipeline, we need to remove the files from intermediate nodes after they have reached the next stage of the pipeline. A Directed Acyclic Graph(DAG) with nodes representing jobs allows us to easily represent dependencies between the jobs. We represent the file-removal, checksum and data transfer as nodes in a DAG and insert appropriate dependencies between them. We use a manager to parse the DAG and submit the jobs to the appropriate scheduler to execute on the appropriate machine.

We believe that data transfer jobs should be treated differently from computational jobs such as checksums, since they may have different semantics and different characteristics. Existing computational job schedulers do not understand the semantics of data transfers well. For example, if the transfer of a large file fails, we may not want to simply restart the job and re-transfer the whole file. Rather, we may prefer to transfer only the remaining part of the file. A computational job scheduler may not be able to handle this case. For this purpose, data transfer nodes and computational nodes in the DAG should be differentiated. Data transfer jobs should be submitted to a scheduler capable of scheduling and managing data transfers, and computational jobs should be submitted to a scheduler capable of scheduling and managing computational jobs.

Each submitted job should be monitored carefully. When a job completes successfully, the next job should be submitted to be queued and executed by the corresponding scheduler. If a job fails, it should be resubmitted until it succeeds.

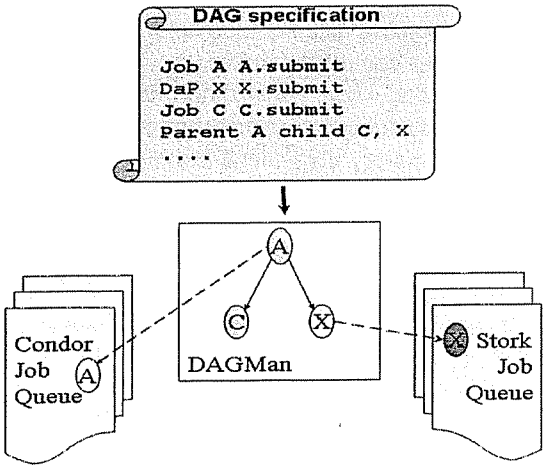

Figure 1. Prototype Model. *A DAG specification file consisting of both computational and data placement jobs is submitted to DAGMan. DAGMan then submits computational jobs to Condor, and data placement jobs to Stork.*

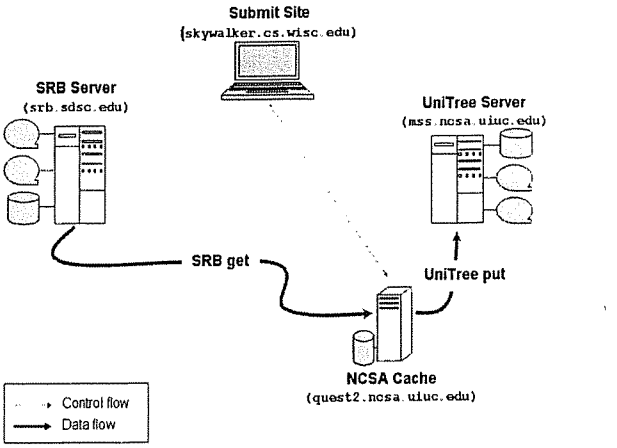

Figure 2. Data-pipeline with One Intermediate Node. *Building a data-pipeline with one intermediate node may be sufficient for data transfer between two heterogeneous storage systems.*

Some transfers may just hang for a long time due to problems in the system, or bugs in the underlying protocol implementation. There should be timeouts, so that if a transfer takes longer than a certain amount of time, it should be terminated and restarted. These "retry" and "kill-and-restart" features can be incorporated either in the DAG management level, or in the scheduling level.

In the prototype of our model, we used Stork [16] as the scheduler for data transfer jobs. Stork is a specialized scheduler for data placement activities in heterogeneous environments. Data placement encompasses all data movement related activities such as transfer, staging, replication, space allocation and deallocation. Stork can queue, schedule, monitor, and manage data placement jobs and it ensures that the jobs complete.

The Condor [18] workload management system was selected as the scheduler for computational jobs in our model. Condor provides a job queuing mechanism and resource monitoring capabilities. It allows the users to specify scheduling policies and enforce priorities. Condor has an extension called Condor-G [9], which allows users to submit their jobs to inter-domain resources by using Globus Toolkit [8] functionality. In this way, user jobs can get scheduled and run not only on Condor resources but also on PBS [10], LSF [25], LoadLeveler [12], and other grid resources.

To perform the management of the DAGs, we employed the Directed Acyclic Graph Manager (DAGMan) [4] [24] which is a service for executing multiple jobs with dependencies between them. DAGMan accepts a declaration that specifies the jobs to be executed and the order of their execution. It logs the execution of the DAG to persistent storage, allowing it to resume a DAG where it left off, even in the face of crashes and other failures.

We have introduced the concept of data placement jobs to DAGMan. Previously data placement jobs in DAGMan were performed through pre-scripts and post-scripts. Whenever a job failed, all of its pre-scripts should have to be redone again from scratch, which might require retransfer of large files which may have been transfered successfully before. Current DAGMan can treat data placement jobs as real jobs, which is actually a requirement of our method. It can differentiate between computational jobs and data placement jobs, and then submit computational jobs to Condor/Condor-G and the data placement jobs to Stork. The progress of both computational and data placement jobs can be monitored through user log files of Condor and Stork. Figure 1 shows how all of these pieces com together in our prototype model.

5. Case Study

National Center for Supercomputing Applications (NCSA) scientists wanted to perform certain processing on the Digital Palomar Sky Survey (DPOSS) [5] image data residing on SRB [2] mass storage system at San Diego Supercomputing Center (SDSC) in California. The total data size was around 3 TB (2611 files of 1.1 GB each). NCSA located in Illinois has its own mass-storage system called UniTree [3]. Since there was no direct interface between

SRB and UniTree at the time of the experiment, the only way to perform the data transfer between these two storage systems was to build a data pipeline. For this purpose, we designed three different data pipelines using our model to transfer the data.

5.1. First Data Pipeline Configuration

The first approach we employed was to set up a cache node (quest2.ncsa.uiuc.edu) at the NCSA site very close to the UniTree server. This approach allowed us to transfer the DPOSS data first from the SRB server to the NCSA cache node using the underlying protocol of SRB, and then from the NCSA cache node to UniTree server using the underlying protocol of UniTree. This pipeline configuration is shown in Figure 2.

The NCSA cache node had only 12 GB of local disk space for our use and we could store only 10 image files in that space. This implied that whenever we were done with a file at the cache node, we had to remove it from there to create space for the transfer of another file. Including the removal step of the file, the end-to-end transfer of each file consisted of three basic steps, all of which we considered as real jobs to be submitted either to the Condor or Stork scheduling systems. Then all of these three node DAGs were joined together to form a giant DAG with dependencies between transfers for the whole process as shown in Figure 3, and the whole process was managed by DAGMan.

The SRB and UniTree servers had gigabit ethernet(1000 Mb/s) interface cards installed on them and the NCSA cache node had a fast ethernet(100 Mb/s) interface card installed on it. We found the bottleneck link to be the fast ethernet interface card on the NCSA cache node. Figure 4 shows the topology of the network, bottleneck bandwidth and latencies. We got an end-to-end transfer rate of 40Mb/s from the SRB server to the UniTree server. We observed that the bottleneck was the transfers between the SRB server and the NCSA cache node. So we decided to add another cache node at the SDSC site to regulate the wide area transfers.

5.2. Second Data Pipeline Configuration

In the second pipeline configuration, we introduced another cache node (slic04.sdsc.edu) to the system. This cache node was placed at the SDSC site, very close to the SRB server. In this case, the data is first transfered from the SRB server to the SDSC cache node using the underlying protocol of SRB, then from the SDSC cache node to the NCSA cache node using third-party GridFTP transfers, and finally from the NCSA cache node to the UniTree server using the underlying protocol of UniTree. This pipeline configuration is shown in Figure 5. The space limitations of the NCSA

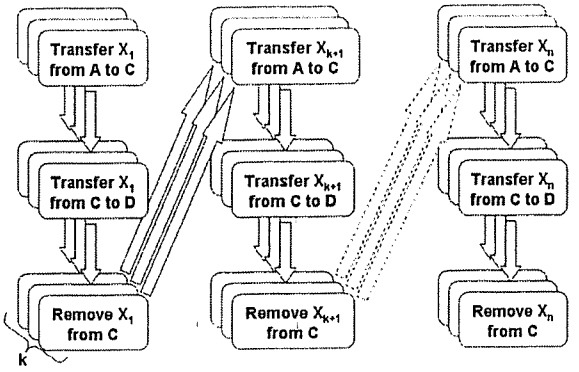

Figure 3. Three Node DAGs. *Three Node DAGs are combined into a giant DAG to perform transfers in the first data-pipeline, with concurrency level = k.*

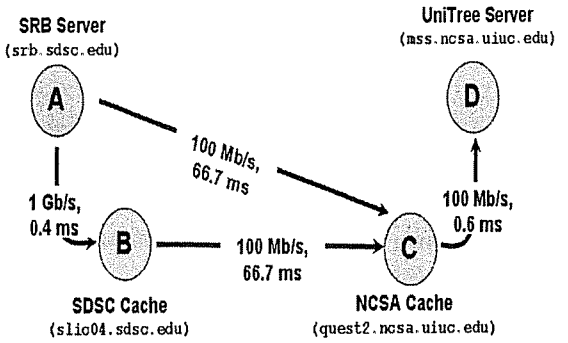

Figure 4. Network Topology. *The topology of the network used in the transfers, with the bottleneck bandwidth and latency between each node.*

cache node applied to the SDSC cache node as well, which required careful cleanup of transferred files at both nodes. Including the cleanup steps, the end-to-end transfer of each file consisted of five basic steps as shown in Figure 6. Then all of these five node DAGs were joined together to form a giant DAG as in the previous pipeline. And then all of these jobs were executed by Condor and Stork systems.

The SDSC cache node had a gigabit ethernet interface card installed on it, but the link between the SDSC cache node and the NCSA cache node still had a bandwidth of 100Mb/s due to the fast ethernet interface of the NCSA cache node. Using this configuration, we got an end-to-end transfer rate of 25.6 Mb/s, and the link between the SDSC cache node and the NCSA cache node turned out to be the bottleneck.

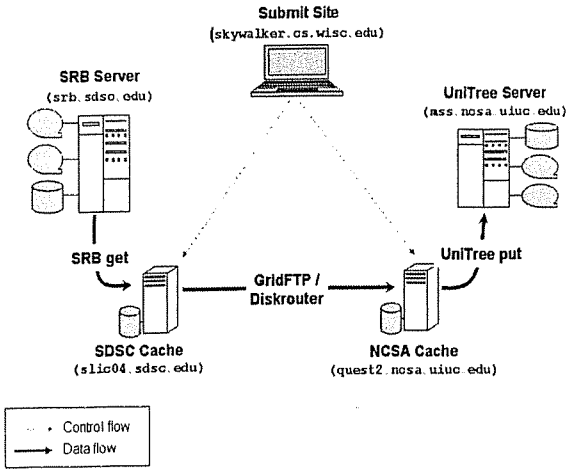

Figure 5. Data-pipeline with Two Intermediate Nodes. *Building a data-pipeline with two intermediate nodes, one close to the source and one close to the destination, may provide additional functionality and increase performance.*

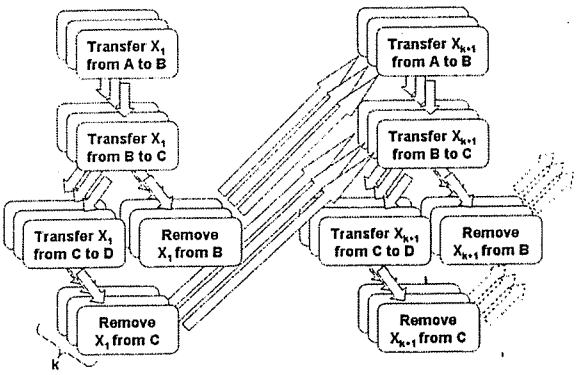

Figure 6. Five Node DAGs. *Five Node DAGs are combined into a giant DAG to perform the transfers in the second and the third data-pipelines. k is the concurrency level.*

5.3. Third Data Pipeline Configuration

The third data pipeline configuration was almost the same as the second one, except that we replaced third-party GridFTP transfers between the SDSC cache node and the NCSA cache node, which were the bottleneck, with third-party DiskRouter [14] transfers.

This time we got an end-to-end throughput of 47.6 Mb/s, which was better than either of the previous data-pipeline configurations.

6. Results and Highlighted Points

This study has three main contributions. First, we show that data pipelines can be built between heterogeneous resources using heterogeneous middleware if necessary, to transfer data between otherwise non-compatible systems. Second, we show that all possible network, server, and software failures can be recovered automatically and completion of transfers can be guaranteed without any human intervention. And finally, we compare the performance of different possible pipeline alternatives, and observe the effect of introducing new nodes to the system.

6.1. Data Transfer Between Heterogeneous Systems

We have successfully built a data-pipeline between two heterogeneous mass-storage systems, SRB and UniTree. We have also employed different data transport protocols like GridFTP and DiskRouter for the wide-area data transfer. Finally, we fully automated the operation of the pipeline and successfully transferred around 3 Terabytes of DPOSS data from the SRB server to the UniTree server.

6.2. Automated Failure Recovery

The most difficult part in operating data-pipelines is handling failures in an automated manner. During the course of the 3 Terabytes data movement, we had a wide variety of failures. At times either the source or destination mass-storage systems stopped accepting new transfers. Such outages lasted about an hour on the average. In addition we had windows of scheduled maintenance activity. We also had wide-area network outages, some lasting a couple of minutes and others lasting longer. While the pipeline was in operation, we had software upgrades. We also found a need to insert a timeout on the data transfers. Occasionally we found that a data transfer command would hang. Most of the time, the problem occurred with third-party wide-area transfers. Once in a while, a third-party GridFTP transfer would hang. In the case of DiskRouter we found that the actual transfer completed but we were not notified of the completion. Because of these problems, we set a timeout for the transfers. If any transfer does not complete within the timeout, it is terminated, cleaned up and restarted.

Figure 7 shows two outages. The first outage happened because UniTree refused new transfers. It lasted for 40 minutes. At that point, two transfers to UniTree were in progress. The transfers completed before the timeout expired. The second outage lasted slightly more than one and a half hours. It was caused by a reconfiguration of the DiskRouter system. We would like to mention that in both of the cases, the data transfers resumed without human in-

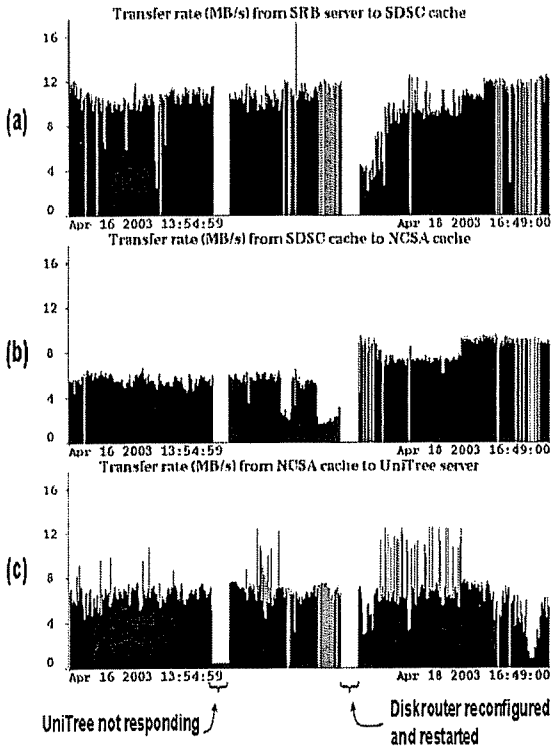

Figure 7. Automated Failure Recovery in case of Server Problem and Software Upgrade. *The transfers recovered automatically although first the UniTree server experiences some problems and then the DiskRouter servers running on the cache nodes get reconfigured and restarted.*

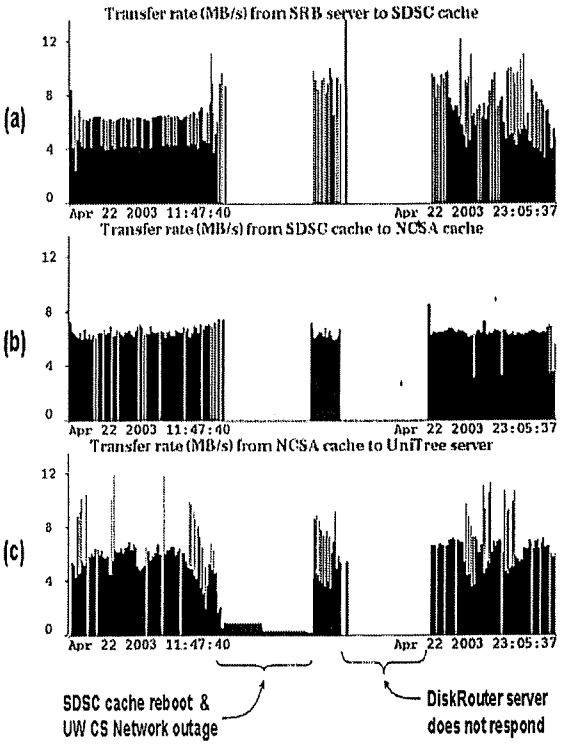

Figure 8. Automated Failure Recovery in case of Network, Cache Node and Software Problems. *The transfers recovered automatically again despite almost all possible failures occurring one after the other: UW CS network goes down, SDSC cache node goes down, and finally DiskRouter stops responding.*

tervention and we noticed them by looking at the logs.

Figure 8 shows a case where multiple failures occurred. First the SDSC cache machine was rebooted and then there was a UW CS network outage lasting a couple of hours (Note: the data transfers are initiated and managed from skywalker.cs.wisc.edu). The pipeline automatically recovered from these two failures. Finally the DiskRouter server stopped responding for a couple of hours. The DiskRouter problem was partially caused by a network reconfiguration at StarLight hosting the DiskRouter server. Here again, our automatic failure recovery worked fine.

6.3. Comparing Performance of Different Data Pipeline Configurations

We wanted to look at how the mass-storage system protocols performed in wide-area. We also wanted to look at the penalty of introducing an additional node. We wanted to see how much performance improvement can be obtained by running wide-area optimized protocols for data transfers between the cache nodes. For these experiments, we performed only rudimentary tuning of the system. For the first pipeline, we found that we got the best performance when we performed 10 parallel file transfers. Similarly, with GridFTP we got the best performance with 10 parallel streams. In the case of DiskRouter, we did not attempt any tuning as the DiskRouter auto-tune mechanism worked fine for us. Table 1 shows the end-to-end performance of data transfers from SRB server to UniTree server with our rudimentary tuning.

Comparison of performance of pipeline 1 and pipeline 2 shows the penalty associated with adding a node to the pipeline. Pipeline 3, which is similar to pipeline 2 with GridFTP replaced by DiskRouter, performs better than pipeline 1 and pipeline 2. The reason is that DiskRouter is optimized for wide-area transfers and has auto-tune capability which automatically tunes the socket-buffer size and the number of sockets according to the network conditions. We were informed that carefully tuning the I/O and socket

buffer sizes in GridFTP would substantially improve its performance. This shows that running wide-area optimized protocols between cache nodes can improve performance enough to offset the penalty of an additional node.

Configuration	End to end rate
Pipeline 1	40.0 Mb/s
Pipeline 2	25.6 Mb/s
Pipeline 3	47.6 Mb/s

Table 1. End-to-end performance of different pipeline configurations.

Another advantage of having cache nodes at both sides was that we had control over those cache nodes, whereas we had no control over the SRB and the UniTree mass-storage servers. To transfer data to/from the mass-storage servers, we need to use whatever protocol they provide. These protocols may not work well under certain conditions or may have certain limitations. Further these protocols may not allow us to tune their performance. This is especially a drawback for wide-area transfers as they benefit considerably from tuning. With a source and a destination cache node, we made the mass-storage system work in the environment (local-area) where they are known to work well and chose an appropriate protocol for the wide-area transfer and got the ability to perform the necessary tune-up. Further we found that we were able to run a checksum or other computation on the data at the intermediate nodes.

7. Future Work

We are planning to build automatic tuning capability into the system. Specifically we would like to add a feature to dynamically determine the optimal concurrency level for the different protocols. We are also considering building functionality into the system to dynamically choose the optimal pipeline configuration. We are planning on a network monitoring infrastructure which would allow us to choose where to place the DiskRouter nodes and how many to place so that we get the best performance. With respect to fault-tolerance, we are building features so that if there are multiple possible protocols between two stages of a pipeline and if one of the protocol has some temporary problem, we would be able to fall back to another protocol. The main motivation for this is that we may have a fast protocol with implementation bugs and a robust protocol with lower performance. We would like to use the fast protocol when it works and fall back to the more stable protocol when it fails.

8. Conclusion

In this paper, we have shown a method to build data-pipelines and operate them in a fully automated manner. Our method allows data transfers between heterogeneous systems lacking a common interface. Through a real-life data transfer involving thousands of large files, we have shown that our method works and is resilient to storage system, network, and software failures. We present our method as a viable alternative to dumping data to tapes and fedexing them or writing scripts and baby-sitting the scripts to deal with failures. We have also performed a study on different pipeline configurations and benchmarked their performance. We have shown that adding additional nodes does not necessarily decrease the end-to-end performance of the system and may in fact increase flexibility and improve performance if done properly.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *IEEE Mass Storage Conference*, San Diego, California, April 2001.

[2] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Proceedings of CASCON*, Toronto, Canada, 1998.

[3] M. Butler, R. Pennington, and J. A. Terstriep. Mass Storage at NCSA: SGI DMF and HP UniTree. In *Proceedings of 40th Cray User Group Conference*, 1998.

[4] Condor. The Directed Acyclic Graph Manager. http://www.cs.wisc.edu/condor/dagman, 2003.

[5] S. G. Djorgovski, R. R. Gal, S. C. Odewahn, R. R. de Carvalho, R. Brunner, G. Longo, and R. Scaramella. The Palomar Digital Sky Survey (DPOSS). *Wide Field Surveys in Cosmology*, 1988.

[6] W. Feng. High Performance Transport Protocols. Los Alamos National Laboratory, 2003.

[7] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 2001.

[8] I. Foster and C. Kesselmann. Globus: A Toolkit-Based Grid Architecture. In *The Grid: Blueprints for a New Computing Infrastructure*, pages 259–278, Morgan Kaufmann, 1999.

[9] J. Frey, T. Tannenbaum, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing*, San Francisco, California, August 2001.

[10] R. Henderson and D. Tweten. Portable Batch System: External Reference Specification, 1996.

[11] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International Data-Grid Project. In *First IEEE/ACM Int'l Workshop on Grid Computing*, Bangalore, India, December 2000.

[12] IBM. Using and Administering IBM LoadLeveler. IBM Corporation SC23-3989, 1996.

[13] D. Koester. Demonstrating the TeraGrid - A Distributed Supercomputer Machine Room. *The Edge, The MITRE Advanced Technology Newsletter*, 6(2), 2002.

[14] G. Kola and M. Livny. Diskrouter: A Flexible Infrastructure for High Performance Large Scale Data Transfers. Technical Report 1484, University of Wisconsin, Computer Sciences Department, 2003.

[15] S. Koranda and B. Moe. Lightweight Data Replicator. http://www.lsc-group.phys.uwm.edu/lscdatagrid/LDR/overview.html, 2003.

[16] T. Kosar and M. Livny. Scheduling Data Placement Activities in the Grid. Technical Report 1483, University of Wisconsin, Computer Sciences Department, 2003.

[17] LIGO. Laser Interferometer Gravitational Wave Observatory. http://www.ligo.caltech.edu/, 2003.

[18] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[19] R. Maddurri and B. Allcock. Reliable File Transfer Service. http://www-unix.mcs.anl.gov/ madduri/main.html, 2003.

[20] V. Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[21] B. Sagal. Grid Computing: The European DataGrid Project. In *IEEE Nuclear Science Symposium and Medical Imaging Conference*, Lyon, France, October 2000.

[22] S. Son. Network Bandwidth Regulation In Cedar, 2003.

[23] D. Thain, J. Basney, S. Son, and M. Livny. The kangaroo approach to data movement on the grid. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing*, San Francisco, California, August 2001.

[24] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In *Grid Computing: Making the Global Infrastructure a Reality.*, Fran Berman and Geoffrey Fox and Tony Hey, editors. John Wiley and Sons Inc., 2002.

[25] S. Zhou. LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems. In *Proc. of Workshop on Cluster Computing*, 1992.