Computer Sciences Department

DiskRouter: A Flexible Infrastructure for High Performance Large Scale Data Transfers

George Kola
Miron Livny

Technical Report #1518 (Changed from 1513 in September)

December 2004

UNIVERSITY OF
WISCONSIN
MADISON

DiskRouter: A Flexible Infrastructure for High Performance Large Scale Data Transfers

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

Abstract

The steady increase in data sets of scientific applications, the trend towards collaborative research and the emergence of grid computing has created a need to move large quantities of data over wide-area networks. The dynamic nature of network makes it difficult to tune data transfer protocols to use the full bandwidth. Further, data transfers are limited by the bottleneck link and different links become the bottleneck at different times resulting in under-utilization of other network hops. To address these issues, we have designed a flexible infrastructure that uses hierarchical main memory and disk buffering at intermediate points to speed up transfers. The infrastructure supports application-level multicast to reduce network load and enables easy construction of application-level overlay networks to maximize bandwidth. It can perform dynamic protocol tuning, use higher-level knowledge, and is being in real-life to transfer successfully several terabytes of astronomy images and educational research videos.

1. Introduction

The data sets of scientific applications are increasing rapidly [14]. Scientists from different organizations are collaborating to attack hard problems [5]. Computational resources across organizations are being connected together allowing remote usage of idle resources. The above three factors require moving large quantities of data over the wide area network.

Wide-area network bandwidth is increasing. Unfortunately, many of the applications are unable to use the full available bandwidth. New data transport protocols are capable of using almost the entire bandwidth, but tuning them to do so is difficult. Further, users want the ability to give different bandwidth to different applications. Currently, this is very difficult to accomplish.

Some organizations have computational resources in a private network with only a head node connected to the outside world. Remote applications that want to use this resource have to stage the data to the execute node via the head node. Some organizations have firewall policies that permit only outgoing connections. At present, it is difficult to move data between such organizations even though they may want to collaborate.

To address the above problems, we have developed a flexible infrastructure called the DiskRouter. DiskRouter in its simplest form provides the functionality of UNIX pipe over the wide area, matching the speed of sender and receiver to improve the overall throughput.

It enables easy construction of application-level overlay network to maximize the bandwidth and provides sophisticated application-level multicast capabilities over the wide-area. It can use higher-level knowledge, provide feedback to higher-level planners, and be managed by a fully automated data placement scheduler like Stork.

2. Related Work

The Internet Backplane Protocol (IBP) [4] is a middleware for managing and using remote storage. IBP provides depots where blocks of data can be stored. The difference between IBP depots and DiskRouter is analogous to the difference between UNIX file system files and pipes. Just as we can build the functionality of pipes using files, we can build the functionality of DiskRouter on top of IBP depots.

It is difficult to transfer terabytes of data via IBP depots if that much storage is not available. This is similar to the need for having disk space equal to the size of pipe-data while using files instead of pipes. Further, DiskRouter has a performance advantage because most data does not go to the disk. While kernel buffer cache may initially help the IBP depot, as large amounts of data flows, the write bandwidth of the disk becomes the bottleneck.

GridFTP [2] is a high-performance, secure, reliable data

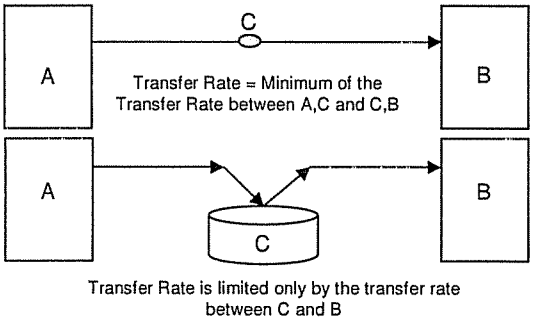

Figure 1. Advantage of buffering at an intermediate node.

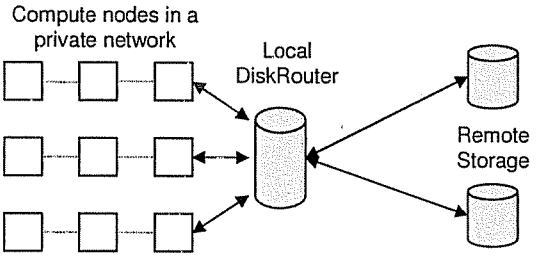

Figure 2. Streaming data via a local DiskRouter.

transfer protocol optimized for high-bandwidth wide-area networks. DiskRouter data transfer protocol is very similar to GridFTP with the addition of automatic buffer tuning for data transfers. The network buffering provided by DiskRouter is orthogonal to the underlying transport protocol and DiskRouter can use GridFTP as the transport protocol.

Resilient Overlay Network (RON) [3] builds an application level overlay network to route around failures and minimize latency. DiskRouter builds an overlay network to maximize bandwidth and does not try to reduce latency. In fact, DiskRouter trades latency for additional bandwidth. Similar to RON, DiskRouter can route around failures by using one or more alternate paths.

Yang-hua Chu et al have used application-level multicast [9] to enable audio/video conferencing and reduce the load on the network. DiskRouter uses application-level multicast to distribute large datasets among collaborating organizations. Because the DiskRouters can use the disk to buffer, DiskRouter multicast does not require end-points to be of same bandwidth and is more flexible.

3. Functionality

In this section, we present the functions that DiskRouter currently performs.

3.1. Store and Forward Device/Wide-area Unix Pipe

DiskRouter in its simplest form is a store and forward device. It uses buffering to match the speed of sender and receiver. It is smart, uses main memory first, and then disk to perform the buffering. It is slightly different from the normal UNIX pipe in that it provides a tagged block abstraction instead of a continuous stream abstraction. The tagged blocks may arrive out-of-order and the DiskRouter clients at the end-points handle the re-assembly.

Figure 1 shows a case where such a store and forward device improves throughput. A source A is transferring large amounts of data to destination B, and C is an intermediate node between A and C. Placing a DiskRouter at C improves throughput if the bandwidth fluctuation between A and C is independent of the bandwidth fluctuation between C and B. When the bandwidth in the path between A and C is higher than the bandwidth between C and B, data gets buffered at C and when the bandwidth between C and B is higher than the bandwidth between A and C, the buffer drains. Such scenarios occur quite often in real world where A and B are in different time zones and C is an intermediate point.

3.2. Data Mover

DiskRouter functions as a data mover. Typically, compute nodes want to get rid of the generated data as quickly as possible and get back to computation. They do not want to spend time waiting for the wide-area transfers to complete and this time can be non-deterministic in the presence of failures. In such cases, the computation nodes can write to a local DiskRouter and expect it to take care of pushing the data to the destination. In this function, DiskRouter behaves similar to Kangaroo [15]. It is more efficient, because the data does not have to traverse the disk.

The data mover is very useful when the compute nodes are in a private network and only the head node is accessible outside. In this case, we can deploy DiskRouter on the head-node and use it to stream data to/from the compute nodes. Figure 2 shows this process.

DiskRouter has a significant performance advantage over simply writing the data to disk on the head node and then transferring it because for large amounts of data, disk becomes the bottleneck. Further, the head node may not have enough storage to accommodate all the data. DiskRouter has dynamic flow control whereby it can slow or stop the sender if it runs out of buffer space and make the sender resume sending data when the buffer space becomes available.

3.3. Application-level Overlay Network

DiskRouter enables easy construction of application-level overlay network to maximize the throughput of the transfers. While other application-level overlay networks like Resilient Overlay Network (RON) help in reducing latency, DiskRouter overlay-network helps in maximizing bandwidth. Below, we give a concrete example of where this is useful.

In the UW-Madison wide-area network, we have two physical paths to go to Chicago. The direct path has a lower latency but the bandwidth is limited to 100 Mbps. There is an another path to Chicago via Milwaukee which has a bandwidth of 400 Mbps. Unfortunately, because of limitations of current networking(we cannot use two paths and dynamically split data between them), we can use only one path and the current networking based on reducing latency chooses the lower latency (and lower bandwidth) path.

We have been able to deploy a DiskRouter at Milwaukee and exploit the combined bandwidth of 500 Mbps for the data transfer. DiskRouter is able to split the data and dynamically determine the fraction that has to be sent directly and the fraction that has to be sent via Milwaukee. The DiskRouter client reassembles the data and passes the complete data to the application. We find similar cases in other environments as well.

DiskRouter overlay network can also be used to route around failures. Users can build more complex overlay networks and may even dynamically build an overlay network and re-configure it.

3.4. Application-level Multicast

Large collaborative projects have a number of participating sites. The source data needs to be moved to the different participating sites. Some of the participating sites are physically located close by. For example, scientist in NCSA Urbana-Champagne, Illinois and Yale, New England needs the data from Palomar telescope in California.

Unfortunately, IP multicast is not available over this wide-area. The only viable alternative is to build an overlay network with application-level multicast capability. DiskRouter helps accomplish that. Since the DiskRouter has buffering capabilities, not all the end-points need have the same bandwidth.

In this scheme, a DiskRouter at Chicago would provide the necessary buffering, make copies of the data, and send one copy to NCSA and the other copy to Yale. If terabytes of data are being moved, the network bandwidth saving is quite significant.

Figure 3 shows an application-level multicast overlay network created using DiskRouters R1-R5 for transferring data from source S to destinations D1-D3. The routing

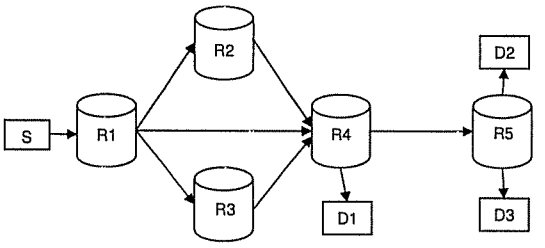

Figure 3. Source S uses a DiskRouter overlay network with DiskRouters R1-R5 to multicast data to destinations D1-D3.

here is a little complex. First, the source sends data to DiskRouter R1. R1 dynamically splits the data and sends fractions of the data to R2, R3 and R4. R2 and R3 send all the data received from R1 to R4. R4 sends the data to destination D1 and to the next DiskRouter R5. R5 sends a copy of data to destinations D2 and D3.

3.5. Running Computation on Data Streams

DiskRouter allows uploading filters to choose incoming data. Recipients can run choose to run arbitrary computation on the specified amount of data before deciding whether to accept it. Users can use DiskRouters ability to make on-the-fly copies to perform computation on the nodes close to each DiskRouter and then distribute the result through the DiskRouter overlay network. It is also possible to combine data movement and data processing using DiskRouters.

3.6. Network Monitor/Dynamic TCP tuning

By using *pathrate* [7] to estimate the network capacity and observing actual transfers, DiskRouter dynamically tunes the TCP buffer size and the number of sockets needed for the transfer to utilize the full bandwidth. Pathrate uses packet dispersion techniques to estimate the network capacity. DiskRouter tunes the buffer size to be equal to the bandwidth delay product. For every 10 ms of latency, DiskRouter adds an extra stream. This is an empirical value that is known to work well [10]. If multiple streams are being used, the buffer size is split equally among the streams.

It is also possible to regulate the bandwidth used by DiskRouter and the bandwidth used by each DiskRouter stream. At this point, this works by limiting the TCP buffer size.

Since DiskRouter performs this periodic latency and network capacity estimation, it can function as a network monitor.

Table 1. Routing Table

Source	Stream Name	Destination	Next DiskRouter	fraction
128.105.165.34/24	*	198.202.74.20/32	206.220.241.13	dynamic
128.105.165.34/24	*	198.202.74.20/32	129.89.57.112	dynamic

3.7. Integration with Higher Level Planners

DiskRouter has features that enable easy integration with higher-level planners. We believe that this ability is the key to addressing failure and fault tolerance issues.

For instance, it is possible when using DiskRouter overlay network that a wide-area network outage disconnects one of the nodes. The DiskRouter client has a tunable timeout and if some pieces are missing after the timeout, it can directly fetch them from the source.

While this handling works, to make better decisions and dynamically reconfigure the overlay network, higher-level planners need this information. DiskRouters and DiskRouter clients pass this information and a summary of the different link status (bandwidth, failures encountered) to higher-level planners, which can then use this information to plan data movement strategies.

In real-world experiments, we have integrated DiskRouter into a data placement framework managed by Stork [11], a fully automated data placement scheduler. Data placement schedulers can make better scheduling decision using the information provided by DiskRouters.

4. Architecture

DiskRouter behaves like a router with a 1 MB logical packet size. Table 2 shows the packet header fields.

Table 2. Packet Header

Field	Size (bytes)
Stream Name	64
Checksum	16
Source Host	4
Destination Hosts 1-16	64
Stream Offset	8
BlockDataSize	4
Flags	8

DiskRouter uses TCP/IP as the underlying data transfer protocol. We decided to use TCP/IP due to the need to traverse firewalls. In our real-world experience, we found that site administrators were more willing to let a TCP stream rather than a UDP stream pass through a firewall. Further, some NAT/Firewall combinations had difficulty with UDP forwarding.

Setting up a data path with a single DiskRouter is simple. The sending client connects to the DiskRouter and specifies the destination/destinations and a set of flags. One of the flags 'Try to Connect' specifies whether the DiskRouter should try to connect to the destination. After authentication and authorization, the DiskRouter checks if the destination is connected. If it is not, and if 'Try to Connect' flag is set, the DiskRouter tries to connect to the destination. Allowing clients to connect to the DiskRouter allows client behind firewalls that allow only outgoing connections to transfer data via the DiskRouter.

In a more complex setup involving a number of DiskRouters, a connection setup phase is required. Each DiskRouter maintains a routing table and this connection setup phase adds entries to the routing table. During this phase, the routing to be used for a data stream is specified to each DiskRouter in the path. The routing can be specified based on stream name, source host and destination host. Wild carding is allowed.

If multiple paths are specified, the DiskRouter automatically sends some of the packets along each path. DiskRouter dynamically determines the fraction to be sent along each path. There is a replicate flag which when set replicates the data along the specified multiple paths. It is also possible to specify the fraction of data to be sent along each path if the dynamic determination is not preferred.

Table 1 shows a sample routing table entry. Source and Destination are specified in Classless Inter-Domain Routing notation as IP address/significant bits pair. The routing table specifies that data from source hosts 128.105.165.0 - 128.105.165.255 (UW-Madison machines) sent to 198.202.74.20 (a San Diego Supercomputing Center machine) is to be dynamically split among DiskRouters at 206.220.241.13 (Starlight) and 129.89.57.112 (Milwaukee). The dynamic flag specifies that the fraction to be sent to each DiskRouter is to be determined dynamically. If replicate flag is set instead of dynamic, a copy of the data is sent to each DiskRouter. Any fraction can also be set in which case DiskRouter will send that fraction of data along that path. Setting dynamic is preferred because if one of the DiskRouters in the path is not accessible, then all data are sent via the other DiskRouters and this has the effect of routing around failures.

There is flow control between DiskRouters whereby the receiver can slow down the sender. This flow control is used to determine the fraction of data to be sent along each path.

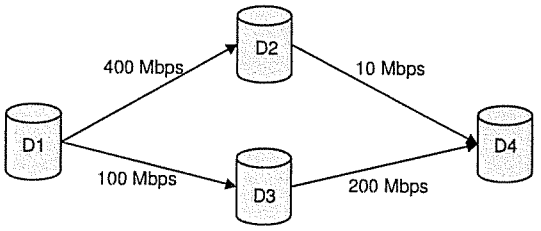

Figure 5. Case where tweaking flow control helps.

This flow control can also be tweaked at connection setup time. By default, a DiskRouter starts applying flow control only when the amount of space used reaches a configurable threshold.

At times, the default flow control may not be optimal. This may happen in certain rare multi-diskrouter overlay networks and an instance is shown in figure 5. Here DiskRouter D2 has high inflow bandwidth from D1 but low outflow bandwidth to D4. It may be advantageous to send data only at 10 Mbps to D2 instead of sending using the full 400 Mbps and filling up its buffer. Note that this buffer limit can be configured for each stream. To handle such cases DiskRouter gives periodic feedback on the buffer usage to previous DiskRouter on the path and higher-level planners. The previous DiskRouter can make a local decision and reduce the flow to this DiskRouter and the higher-level planner can dynamically tweak the buffer threshold to effect this. Who does the tweaking is a choice of local versus global decision-making. At times only a higher-level planner can make the decision while at other times a local decision by the upstream DiskRouter may be sufficient.

We provide a library for applications to link against to use the DiskRouter. We also provide a simple client. This client uses the DiskRouter libraries, behaves like a file transfer server, and supports third-party transfers. The client also supports direct transfer to the destination client and can split the data and send a part directly and a part to a DiskRouter.

The DiskRouter servers and client libraries are highly optimized for data transfers. In real world, we get very close to the maximum transfer rate attainable through a network interface card. When a client connects to a DiskRouter server, there is a negotiation of TCP buffer sizes and the number of streams. The DiskRouter server performs periodic network bandwidth estimations using *pathrate* tools and uses that to tune the TCP buffer sizes.

There is an option to specify the buffer-sizes on the client side. This feature is useful when DiskRouter is used in a framework that has network monitoring and tuning capabilities. Here the higher-level planner in the framework may determine the optimal buffer size and number of streams and specify it to the client, which then passes it on the DiskRouter during connection time. If the client specifies the buffer size, DiskRouter does not perform dynamic tuning.

5. Insights from Real Life DiskRouters

Figure 4 shows the real life DiskRouters that have been deployed since May 2002. The bandwidth and latency keep fluctuating and what is shown in figure 4 is just a snapshot. The overlay network can be changed dynamically according to the conditions.

The real life deployment gave us a lot of insight. The addition of dynamic buffer tuning was the result of gained experience.

The buffer size is set exactly to the bandwidth delay product, using bandwidth and latency determined by *pathrate*. We found that if the buffer size were set larger, some transfers would never complete. This was puzzling at first because this buffer size was the maximum allocated TCP buffer size and TCP should apply its congestion avoidance and control mechanism. A careful investigation using packet traces revealed that the Linux kernel used in the DiskRouter machines would send out packets at the interface speed(1000 Mbit) and during slow-start there would suddenly be a considerable loss as the routers did not have enough buffer to buffer that burst with the net result that the TCP stream would reduce the window to zero and almost never recover. Note that TCP does not pace each burst of packets, but only the interval between bursts. What happened was that the instantaneous burst was overflowing the buffers resulting in packet loss. It appeared that TCP pacing [1] would help.

The experiment with multiple streams showed a noticeable performance gain with latency greater than 10 ms. We identified the reasons for that as follows. The longer latency required larger buffer size and the TCP window scale(16-bit window size that is scaled by a 14-bit window scale field) meant that the granularity of window size change would be coarser. If we used multiple streams and split the window size among the streams, we get finer granularity of window size changes. If Linux notices congestion event on a stream to a particular path, it does not increase the window-size of other streams to that path for 10 minutes. The net result is that with multiple streams, the amount of packet loss seen is lesser and this improves TCP throughput.

Another reason is that TCP does not work well for multi-hop long latency networks where each hop can experience an independent congestion event. In this case, a single stream may never be able to utilize the full bandwidth. With multiple streams, only a single stream would be affected by the loss and the multiplicative decrease would only halve that stream and not the others with the net result being that

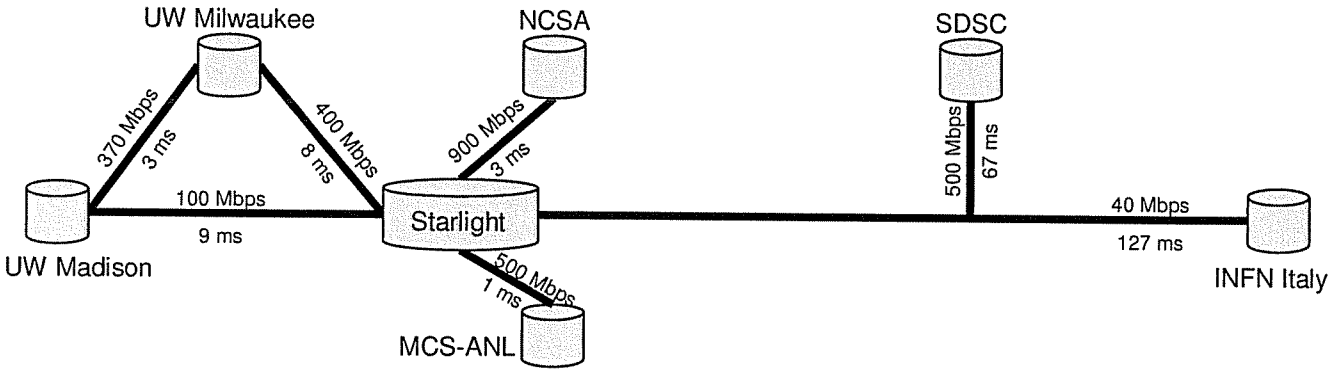

Figure 4. Real Life DiskRouters deployed since May 2002. The bandwidth and round-trip time are measured from each node to the Starlight node.

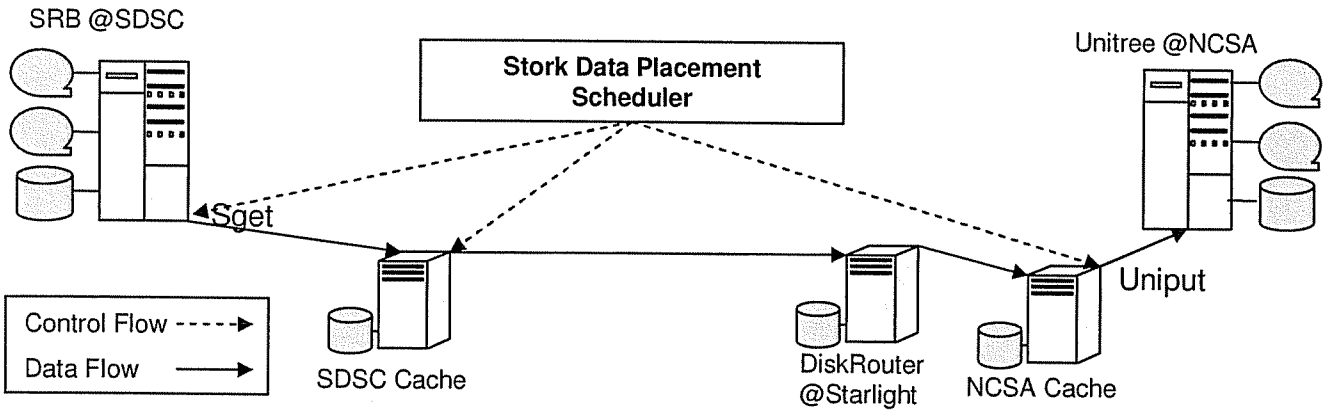

Figure 6. DPOSS data pipeline used to transfer 3 TB of DPOSS images from SRB @SDSC to Unitree @NCSA.

multiple streams would be able to use more of the bandwidth. Deploying DiskRouters between hops that experience independent congestion event would help to work around this problem.

Another minor reason that multiple streams helped in a specific case was the presence of UW CS firewall. We found that the firewall was adding latency by delaying the acks and more streams seemed to improve throughput. Since we have not experimented with other firewalls, we are not sure if this is normal or not.

Linux 2.4 has dynamic right sizing [8] whereby the receiver estimates the sender's congestion window-size and uses that to tune window size advertisement and the sender window size is limited only by the receiver window size advertisement. Because of 16-bit TCP window size and the fact that the 14-bit window scale should be setup during connection establishment, the TCP window may not grow above 64 KB if window scale was not set . Since setting a higher default window scale affects the granularity of window size changes for all streams, this is not recommended. We explicitly set the window size only for streams larger than 64 KB and use the Linux dynamic right sizing for smaller window sizes. The only exception is that we may explicitly set the window size to a smaller value to regulate the bandwidth used by a stream.

6. Real Life Data Movement using DiskRouters

We used the deployed DiskRouters for a number of real life data movement and detail some of them in this section.

Replication of Digital Palomar Sky Survey Data

Around 3 terabytes(2800 x 1.1GB files) of Digital Palomar Sky Survey(DPOSS) data stored in SRB mass-storage system at San Diego Supercomputing Center(SDSC), San

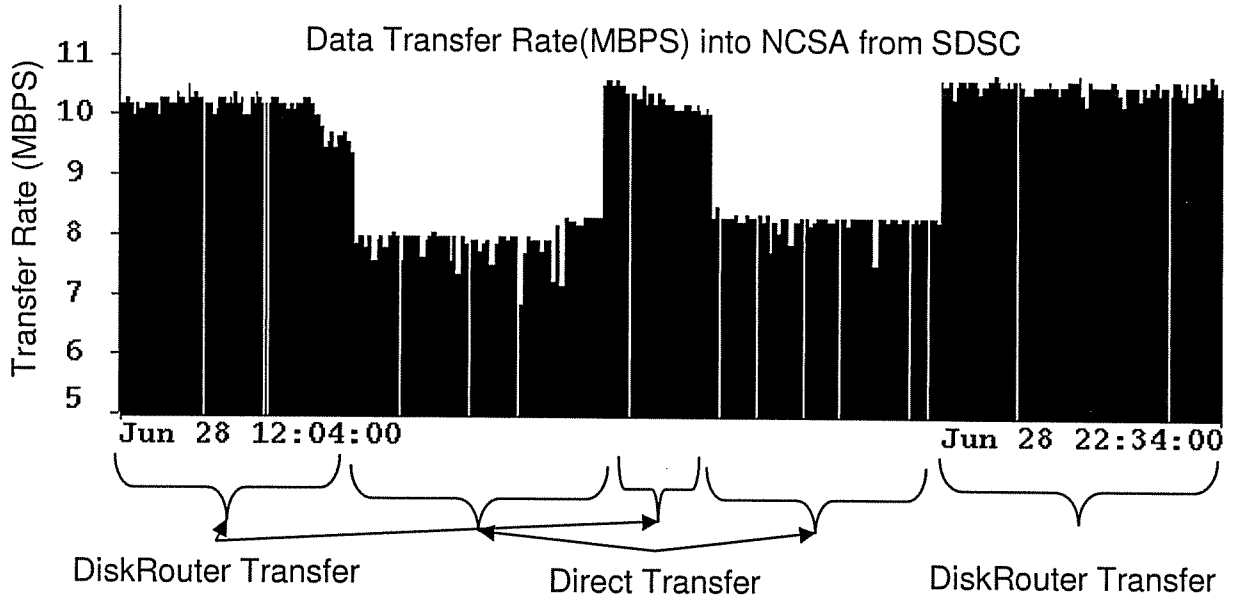

Figure 7. Data rate seen by the receiver at NCSA.

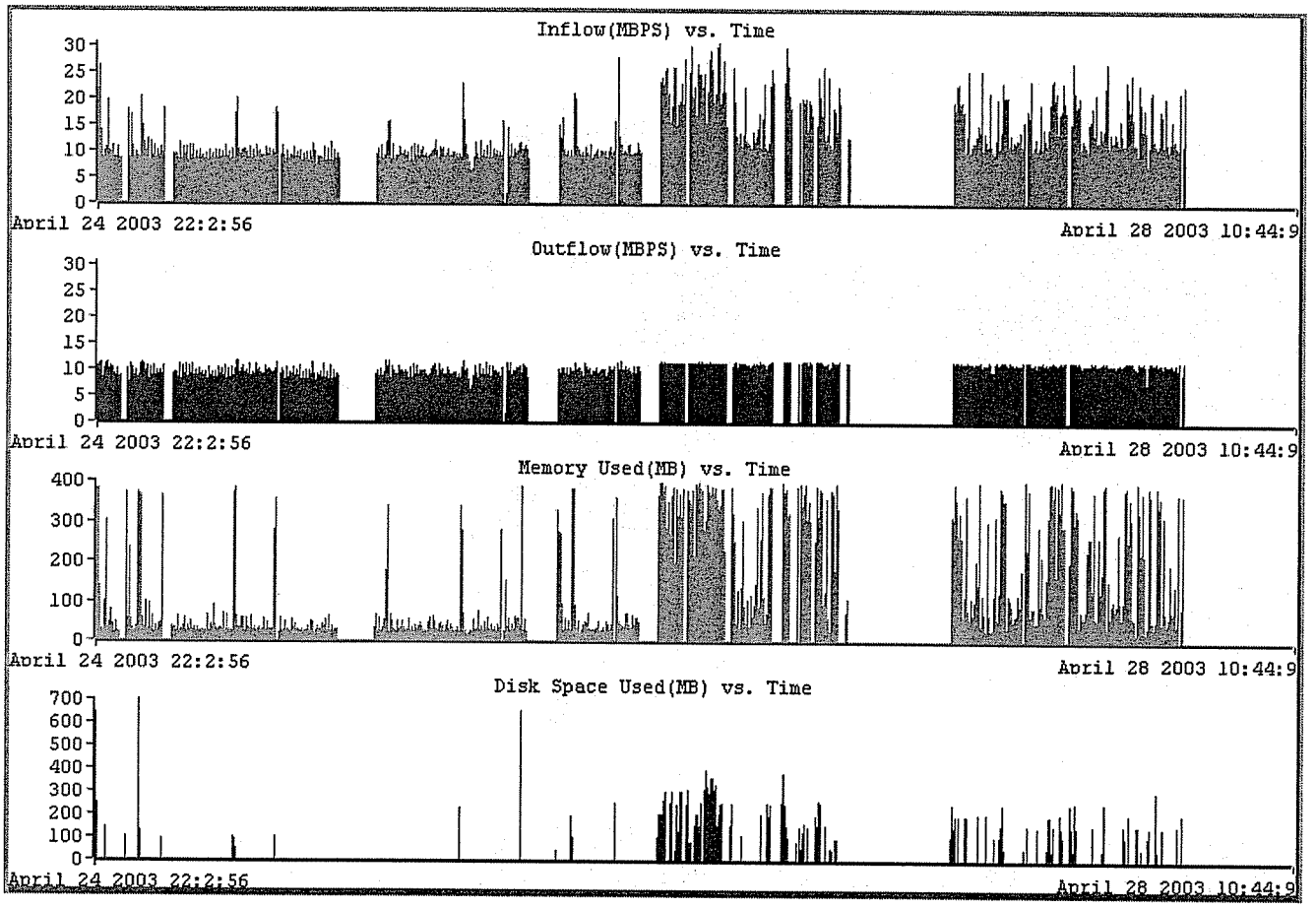

Figure 8. Statistics from the DiskRouter running at Starlight.

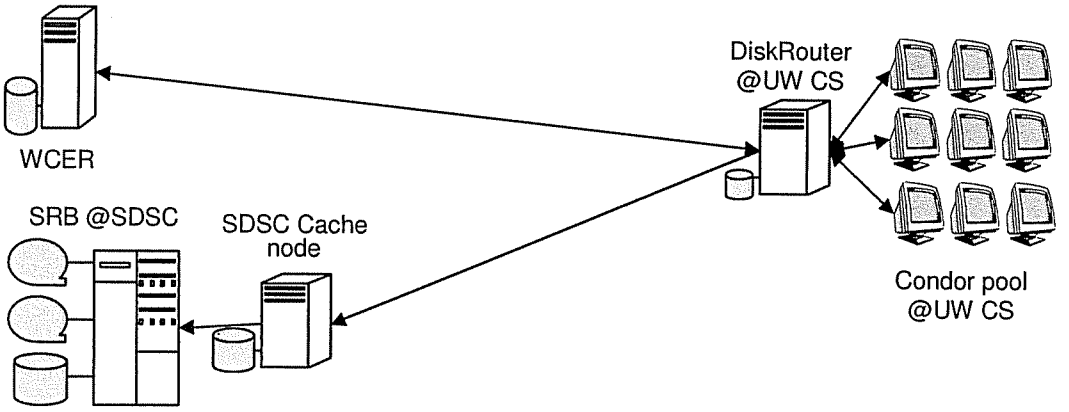

Figure 9. WCER pipeline used to encode terabytes of digital videos to MPEG-1, MPEG-2 and MPEG-2 and transfer them to SRB mass storage at SDSC and WCER.

Diego, CA had to the replicated to the Unitree mass-storage system at NCSA, Urbana-Champagne, IL. At that time as there was no common interface between SRB and Unitree, we had to use an intermediate node to perform protocol translation.

We decided to use a stage node at SDSC and another at NCSA and evaluated the different protocols and the effect of using a DiskRouter. The data pipeline is shown in figure 6. Stork, data placement scheduler, managed the whole process.

Figure 7 shows a snapshot of the transfer. The direct transfers used GridFTP and the DiskRouter transfer used a diskrouter at Starlight. As can be seen, DiskRouter gives a performance advantage of 20%-30%. During this transfer, the NCSA machine had 100 Mbit connectivity and was the bottleneck. There are some empty lines in the snapshot. Data received at NCSA cached node data gets pushed to Unitree and during this transfer, the acks to the DiskRouter/GridFTP server were delayed and so the cache node did not see an inflow for a certain part of the time it took to push the data to Unitree.

DiskRouter provides a rich set of statistics and allows dynamic visualization of what is happening in the system. The dynamic visualization is done using DEVise [13] where a web page is generated and the statistics is updated periodically. The period is tunable. Sample visualization is shown in figure 8. The visualization is dynamic in that the users can zoom into points of interest and can zoom out to see overall trends.

Distributed Processing of Education Research Videos and Off-site Replication

As part of the Digital Insight project [6], around 6 TB of educational research videos stored at Wisconsin Center for Education Research(WCER) are being converted to MPEG-1, MPEG-2 and MPEG-4 using distributed processing at the Condor [12] clusters at UW Madison. Each video is 13 GB in size. The MPEG-1, MPEG-2 and MPEG-4 encoding are done in parallel. The video from WCER server is transferred to UW-Madison Cómputer Science DiskRouter, which makes 3 copies and sends them to compute nodes in the Condor pool. The encoded videos are streamed back to the DiskRouter, which then makes two copies and sends one to SRB server at SDSC and another to WCER server. Figure 9 shows the process.

7. Future Work

We are collaborating with Robert Brunner and his group at NCSA and are working to use DiskRouters to help process petabytes of data from the following astronomy datasets: Quest2, CARMA, NOAO, NRAO and LSST. The interesting feature of this is that, these dataset have to be replicated to different places and processed. Since each collaborating site does not have enough processing power to process the full dataset, we need to utilize the processing power of the different sites and do this in an efficient manner. We are interested to work towards combining data movement and data processing. We feel that DiskRouter multicast would help us conserve the network bandwidth and in turn help us improve the system throughput.

8. Conclusion

We have built a flexible infrastructure called DiskRouter that uses hierarchical buffering at intermediate points to aid in large-scale data transfers. It supports easy construction

of application level overlay network and can perform routing. It performs dynamic TCP tuning to improve throughput. It supports application level multicast to help lower the network load and improve the system throughput. It has been integrated into a data placement framework managed by Stork and has been used to successfully transfer terabytes of astronomy images and educational research videos.

9. Acknowledgment

We would like to thank Robert Brunner and his group at NCSA, and Chris Torn and his group at WCER for collaborating with us, letting us use their resources and making it possible to use DiskRouter for real-life data movement and processing.

References

[1] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of TCP pacing. In *INFOCOM (3)*, pages 1157–1165, 2000.

[2] 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.

[3] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceeding of the Eighteenth ACM Symposium on Operating Systems Principles*, October 2001.

[4] A. Bassi, M. Beck, T. Moore, J. S. Plank, M. Swany, and R. Wolski. The Internet Backplane Protocol: A study in resource sharing. In *Future Generation Computing Systems*, pages 551–561, 2003.

[5] CMS. The Compact Muon Solenoid Collaboration. http://cmsinfo.cern.ch/Welcome.html/CMScollaboration/CMScollaboration.html, 2003.

[6] Digital Insight. Creating new capacity for video-based data collection, analysis and dissemination. http://www.wcer.wisc.edu/digitalinsight/, 2003.

[7] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *INFOCOMM*, 2001.

[8] M. Fisk and W. chun Feng. Dynamic right-sizing in TCP. In *Second Los Alamos Computer Science Institute Symposium (LACSI 2001)*, San Diego, CA, October 2001.

[9] Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *ACM SIGCOMM 2001*, San Diago, CA, August. ACM.

[10] G. Kola, T. Kosar, and M. Livny. Run-time adaptation of grid data-placement jobs. *Parallel and Distributed Computing Practices*, 2004.

[11] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *Proceedings of 24th IEEE Int. Conference on Distributed Computing Systems (ICDCS2004)*, Tokyo, Japan, March 2004.

[12] 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.

[13] M. Livny, R. Ramakrishnanand, K. Beyerand, G. Chenand, D. Donjerkovicand, S. Lawandeand, J. Myllymaki, and K. Wenger. DEVise: Integrated querying and visual exploration of large datasets. In *ACM SIGMOD*, May 1997.

[14] PPDG. PPDG deliverables to CMS. Deliverables to CMS, 2001.

[15] 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 (HPDC10)*, San Francisco, California, August 2001.