A High-Performance Cluster Storage Server

Keith Bell, Andrew Chien
Department of Computer Science and Engineering
University of California, San Diego
<achien, kbell>@cs.ucsd.edu

Mario Lauria
Department of Computer and Information Science
The Ohio State University
Columbus, OH
lauria@cis.ohio-state.edu

Abstract

An essential building block for any Data Grid infrastructure is the storage server. In this paper we describe a high-performance cluster storage server built around the SDSC Storage Resource Broker (SRB) and commodity workstations. A number of performance critical design issues and our solutions to them are described. We incorporate pipeline optimizations into SRB to enable the full overlapping of communication and disk I/O. With these optimizations we were able to deliver to the application more than 95% of the disk throughput achievable through a remote connection. Then we show how our approach to network-striped transport is effective in achieving aggregate cluster-to-cluster throughput which scales with the number of connections. Finally, we present a federated SRB service over MPI that allows fast TCP connections to stripe data across multiple server disks reaching 97% of the combined write capacity of multiple nodes.

1. Introduction

Data-intensive applications constitute an increasing share of high performance computing (HPC). An increasing number of applications in domains such as genomics/proteomics [1,2,3,4], astrophysics [5], geophysics [6], computational neuroscience [7], or volume rendering [8], need to archive, retrieve, and process increasingly large datasets. These applications are prime candidates for Grid computing [9] as they involve remote access and extensive computation to many data repositories. Several Grid middleware projects [10,11,12] specifically target the management of application data. They offer sets of basic concepts and tools for storing, cataloging, and transferring application data on the Grid. We will refer to that type of middleware as *Data Grid middleware* and recognize that it provides the fundamental building blocks for data-intensive computing.

An essential building block for any Data Grid infrastructure is the storage server. The model of Grid we refer to is a collection of clusters located in supercomputing centers or high performance computing labs with high speed connectivity to regional or national backbones such as the Internet2. Some of the clusters are used for computation, while others are dedicated to data storage following distributed data manipulation models proposed by the Data Grid community. In this paper we explore the system design of such storage servers. The specific requirements that need to be addressed are large cluster-to-cluster throughput, high I/O performance delivered to the application, scalable disk bandwidth, good matching of disk and network throughput. The concept of cluster-based server has already been proposed in connection with specific domains of applications such as video servers [13,14] and Internet data caches for large dataset acquisition [15]. In this paper we focus on clusters employed as general purpose data servers in the context of high performance data-intensive computing. In our study, a (possibly parallel) application is running on a client cluster and we want to maximize the performance of accessing the data stored on a remote storage server. We assume that the data is accessed according to a remote file access model through Unix I/O style primitives, an approach commonly adopted by Grid middleware.

We assume a cluster based architecture for our server because it uses inexpensive off-the-shelf PC components, offers an inherently scalable aggregate I/O bandwidth, and can take advantage of existing cluster installations through double-use or upgrade of older hardware. By leveraging the high-speed communication afforded by the cluster interconnect, large files can be stored in a scalable fashion by striping the data across multiple nodes. With single disk capacities of 160 GB and prices as low as $1/GB, 10 TB of disk storage can be added to a

small cluster for less than $10,000. At the current rate growth of disk size, inexpensive 50-100 TB clusters will be realistic in another year or so. By distributing the disks across a sufficient number of cluster nodes, aggregate bandwidth in excess of 1 GB/s can be easily obtained with current hardware, a two orders of magnitude improvement over single disk performance. Further, the availability of CPU and memory on each node offers the flexibility of additional data manipulations such as pre-processing and caching.

The representative middleware tool employed in our study is the Storage Resource Broker (SRB) [11]. SRB is representative of Grid remote storage access tools not only in its interface and client/server design, but also in that it is not optimized for large data transfer. In this paper we show how restructuring the SRB protocol according to a pipelining concept can enhance the throughput of large data transfers. We then expose the performance bottlenecks existing along the entire data path from the storage server disks all the way to the application. For each of these bottlenecks in turn we implement a remedial solution and we measure the performance improvement. The main contribution of this paper is to show the relevance of concepts such as end-to-end pipelining, disk and network operation overlapping, disk and network striping, in achieving an efficient design of a cluster based storage cluster. Furthermore, we expose several aspects of operating system and middleware interaction that are relevant to the careful implementation of these concepts.

The remainder of this paper is organized as follows: section 2 describes the SDSC Storage Resource Broker. Sections 3, 4, and 5 discuss the SRB performance enhancements implemented; related work is covered in section 6. Finally section 7 concludes the paper.

2. Storage Resource Broker

2.1 The Original SRB

The Storage Resource Broker (SRB) [11] was developed by the San Diego Supercomputer Center (SDSC) as part of the Data Intensive Computing (DICE) effort. SRB was designed to provide a consistent application interface to a variety of data storage systems. Applications use the SRB middleware to access heterogeneous storage resources using a client-server network model consisting of three parts: SRB clients, SRB servers, and a metadata catalog service, MCAT. SRB client applications are provided with a set of simple, Unix-like API's to interface with the remote SRB server and thereby access various systems on different servers. Each SRB server controls a distinct set of physical storage resources, so a special scheme called federated operation was added to provide interaction between servers controlling different resources. In the federated operation, one SRB server acts as a client to another SRB servers.

2.2 The Pipelined SRB

Analysis of the SRB protocol showed that a pipelined transfer would increase throughput [16]. The crucial advantage of the pipeline is that it enables the overlapping of the different stages of the file transfer - protocol processing, transport, and disk access. In a previous project we restructured the SRB protocol to implement a pipelined model of transport. We demonstrated a performance improvement of 43%/52% for remote reads/writes larger than 1MB among nodes connected to the same LAN. More details of the pipelined SRB are discussed in [17]. The emphasis of our previous project was on the analytical modeling of the pipeline, and on the application of the model to solve design and runtime configuration issues such as selecting the optimal chunk size. In this paper, we focus on the interplay between all the elements of the data path, and how they affect the pipelining. For the analysis described in this paper, we ported the pipelined version of SRB to the Windows NT environment used on our clusters. The pipelined version was derived from an earlier release of SRB (1.1.2); to the best of our knowledge, there have been no substantial changes to the base transport protocol in the more recent releases.

3. SRB Performance Enhancements

3.1 Experimental Setup

Our setup consisted of a client and a server cluster. The first was a Myrinet-interconnected cluster of 32 dual Pentium II/450MHz HP Netserver systems running Windows NT 4.0. The other was a Myrinet-interconnected cluster of 32 dual Pentium II/300MHz HP Kayak systems, also running Windows NT 4.0; each node was equipped with a 3Ware DiskSwitch IDE RAID controller with four 20GB IDE disks configured as RAID 0 (striped disk array). A subset of nodes on the client and server systems was connected by Gigabit Ethernet through a Packet Engines PowerRail 5200 switch.

3.2 Baseline Throughput

The goal of this project was to maximize the fraction of server disk I/O bandwidth presented to the remote application through SRB. The main bottleneck for remote

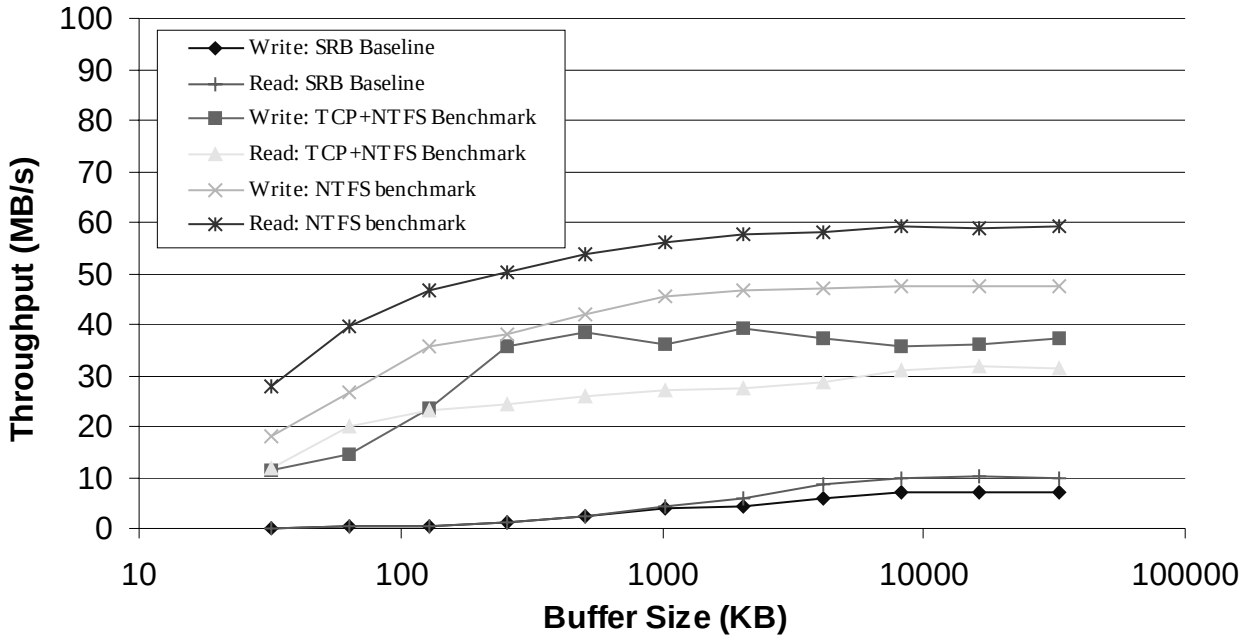

Figure 1: Original SRB baseline, NTFS, and TCP+NTFS benchmark throughput.

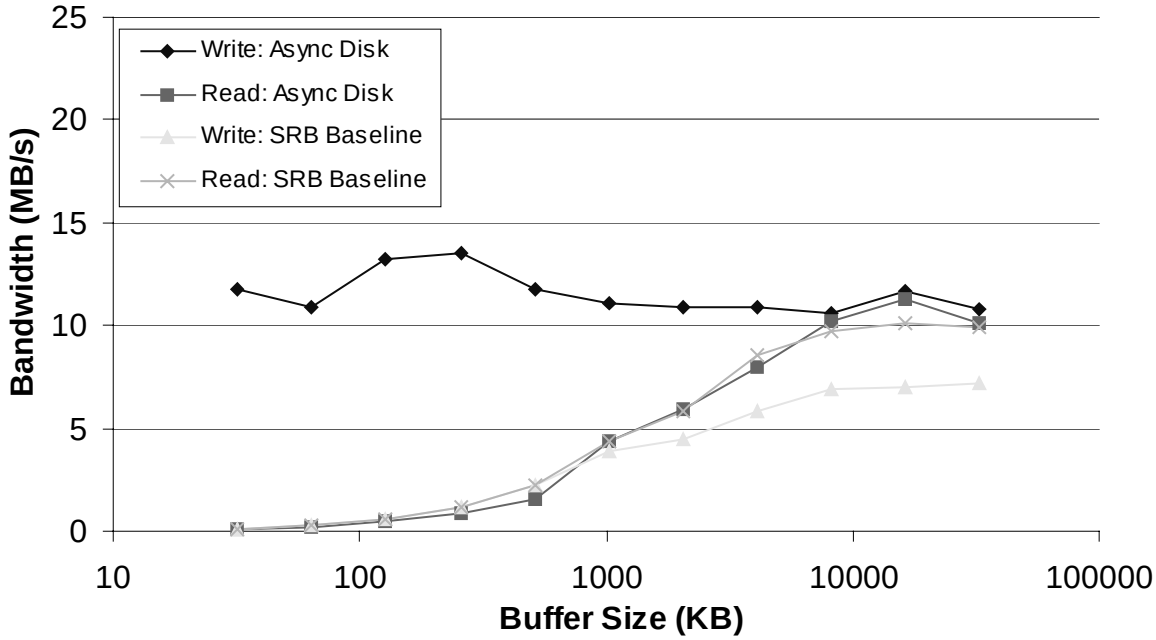

Figure 2: SRB with asynchronous disk I/O.

access is usually the network; depending on the configuration, disk throughput on the server can also become a bottleneck. Therefore we first optimized the SRB throughput in the traditional single client/server configuration using pipelining; then we used network striped transfer between the client and server cluster. Finally, we took advantage of the high-speed interconnects available in clusters to implement federated SRB service over the Message Passing Interface (MPI) [18]. This federated SRB approach allows one edge node with high TCP throughput on the server to stripe data across several storage nodes with low disk throughput.

A simple SRB client application was used to measure the baseline throughput of the original SRB for remote read and write operations as shown in Figure 1. These throughput measurements were taken from a client application running on a dual Pentium II/450 HP Netserver. The SRB server was running on a dual Pentium II/300 HP Kayak; the client and server systems were connected by Gigabit Ethernet. The other two curves shown in Figure 1 are the local disk access bandwidth measured on the server (shown as NTFS), and the throughput of accessing the disk through a TCP/IP connection (TCP+NTFS). These curves represent respectively the maximum local and remote disk access bandwidth against which to compare our modified SRB. From these curves it is apparent that the remote access performance is network limited on our experimental setup. To obtain the first curve we used the *sio* [19] benchmark. For the latter, a benchmark was created by carefully combining the *ttcp* [20] network and the *sio* disk benchmark tests. The combined benchmark used separate threads for network and disk I/O while maintaining synchronization through shared buffers. Using separate threads for network and disk operations enabled the overlapping of network and disk operations required for maximum performance. The *sio* benchmark showed that NTFS throughput was highest at larger block sizes (16MB), whereas *ttcp* showed that TCP throughput was highest for smaller block sizes (128KB). For this reason, the shared buffer code was designed to allow several small TCP blocks to be transferred for each larger disk I/O block.

3.3 Asynchronous Disk I/O

The baseline SRB throughput results were far below the TCP and NTFS bandwidth measured in benchmark tests. As shown in an earlier analysis of the SRB protocol [17], the serialization of network transfer and disk access was the major bottleneck in the system. The first performance enhancement for the pipelined SRB was to overlap network and disk operations using asynchronous disk I/O primitives. As each chunk of data is received by the SRB server, a non-blocking disk write is issued so that the next receive operation could be executed while the disk I/O proceeds in the background. We implemented non-blocking disk access using the Win32 I/O completion routine mechanism.

Using a shared buffer to implement a circular queue, blocking TCP receive operations fill the queue, while non-blocking disk writes empty the queue. Only when the TCP processing needs to use a buffer that is currently in use does the disk I/O routine get polled for completion. This approach works well for writing data, but not for reading since the blocking TCP send must wait for the non-blocking disk read to complete before continuing, thereby preventing the desired overlap of disk and network I/O. As shown in Figure 2, the asynchronous disk operations did not work as well for reads as for writes.

3.4 Aggregate Acknowledgements

For pipelining to work, the pipeline must be kept full. The next bottleneck was due to the fact that the original SRB protocol required every buffer transmission to be acknowledged before executing the next buffer transmission. A modification of the internal SRB protocol was required so that remote I/O requests can be sent repeatedly without waiting for an acknowledgement. Three new functions were added to the client library: *srbFileAioWrite*, *srbFileAioRead*, and *srbFileAioReturn*. The read and write functions ensure that the appropriate data has been sent or received via TCP, and the return function polls for completion of all outstanding read or write operations as shown in Figure 3. The aggregation of acknowledgements is achieved by using the return function to explicitly poll for the final acknowledgement in a data transfer, while the modified read and write primitives no longer wait for any acknowledgements. As shown in Figure 4, allowing the SRB client to send or receive data continuously keeps the TCP pipeline full, providing improved throughput compared to baseline SRB results.

3.5 Asynchronous TCP Operation

Empirical tests of TCP throughput on our Windows NT cluster showed that there were several tuning parameters that affected performance. First, we set the system registry key for tcpRecvWindow to the maximum value of 64KB. Changing this registry setting from the default value had a large impact on TCP throughput, and all of the measurements presented in this paper use the modified setting. Next, we changed the TCP socket send and receive buffer size to zero, which eliminates a copy from the IP stack into the user buffer. The measurements indicated that non-blocking TCP provided better

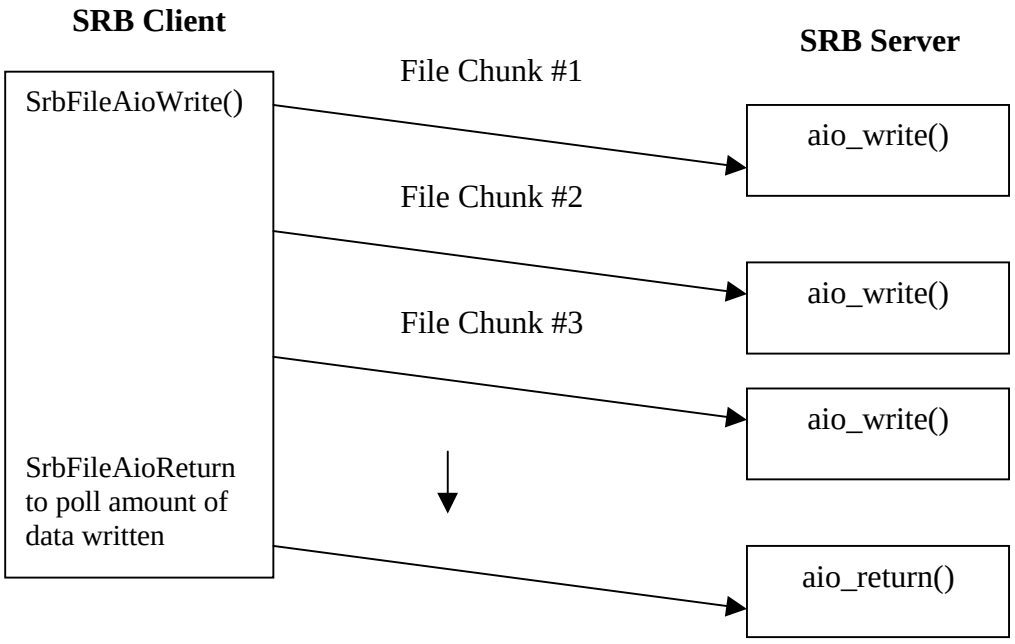

Figure 3: SRB asynchronous write operation.

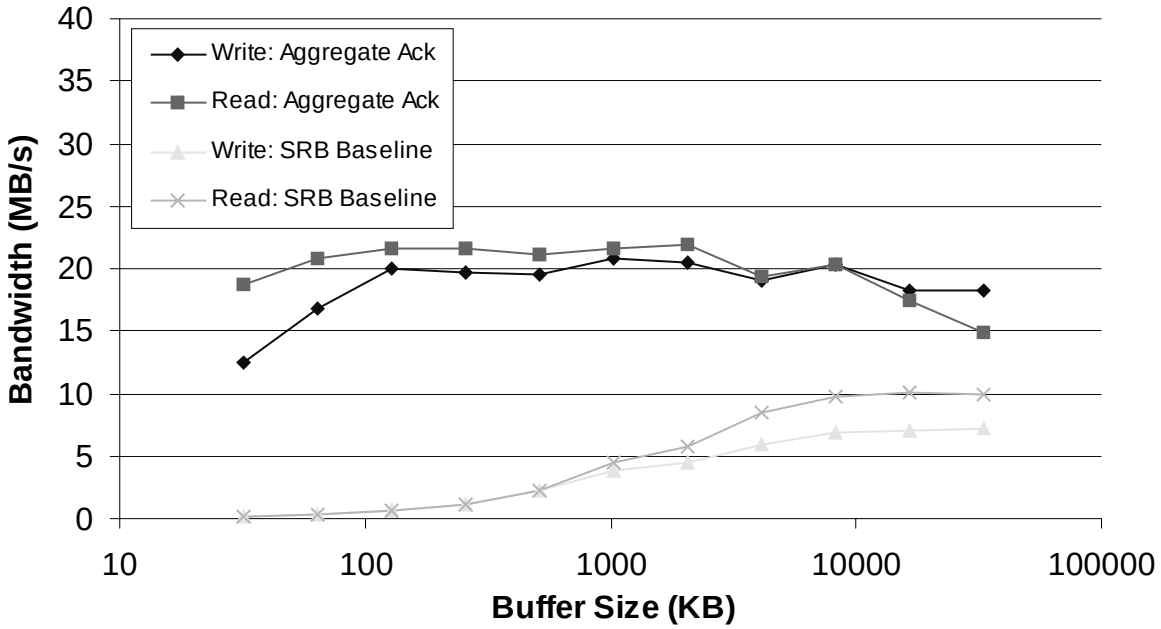

Figure 4: SRB with aggregate acknowledgements.

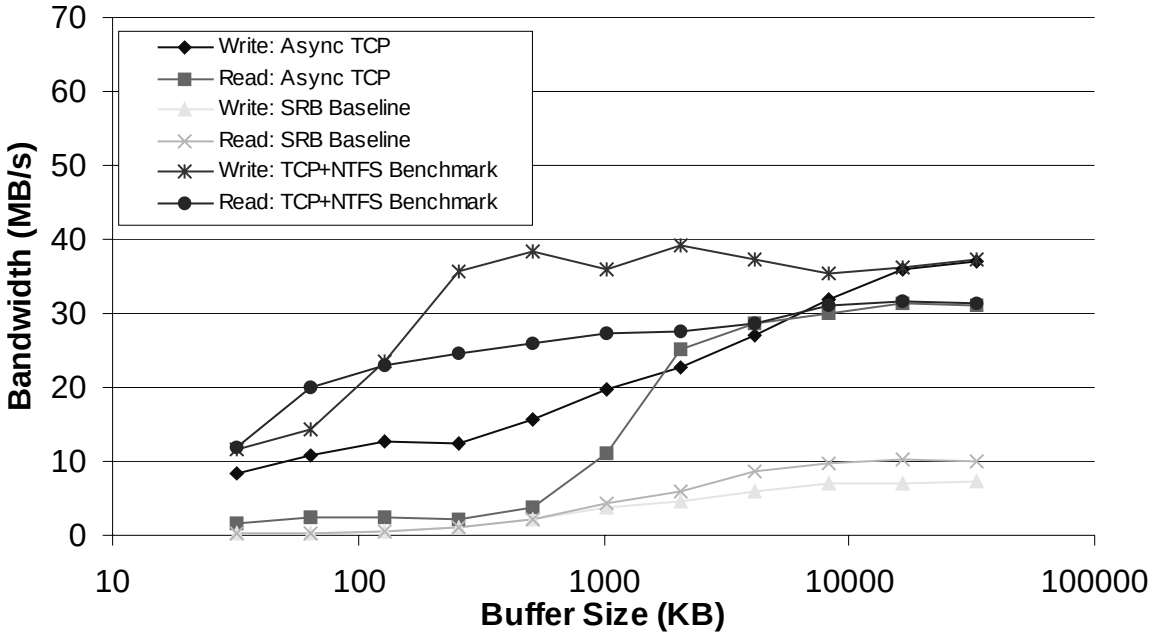

Figure 5: SRB with asynchronous TCP, SRB baseline, and benchmark throughput.

performance than the standard blocking routines, so the code was changed to support non-blocking sockets. SRB was modified to create two data sockets on separate ports to support overlapped operation. The SRB client library sends or receives consecutive chunks of the file by alternating between the two sockets. We were surprised to see a performance increase when using overlapped TCP between two hosts using two separate sockets. This may be attributed to the TCP flow control, or possibly the dual-CPU configuration of our test systems. The results shown in Figure 5 achieve over 99 percent of the available system throughput for large buffers as measured by our remote access benchmark

4. Network Striped I/O

At this point we had achieved close to the maximum possible performance of SRB on a single node server. To improve throughput further, we needed to address the two remaining bottlenecks represented by TCP and disk bandwidth. The TCP protocol processing overhead on the communication endpoints is the bandwidth limiting factor. However instead of modifying TCP, we got around the bottleneck by parallelizing the connections between SRB clients and servers as shown in Figure 6. A crucial aspect is the use of separate client/server endpoints, made possible by the parallelism of the cluster.architecture on both sides of the connection. We ran multiple SRB client processes (as an MPI application), one per client cluster node, each of which used TCP to communicate with the corresponding SRB server on anode of the storage cluster. The multiple SRB clients can be left exposed to the application (as in our tests), or hidden behind a unified programming interface (i.e. MPI-IO). Figure 7 graphs the measurements of a parallel file transfer using four clients and four servers. These results show the aggregate remote write throughput for each of the four client-server pairs used in the test. As expected, the parallel client results show that the aggregate storage bandwidth of the cluster scales with the number of cluster nodes.

5. Federated SRB Using MPI

SRB supports federation of services so that multiple SRB storage servers can coordinate operation to handle remote client requests. In this project the SRB federated service was extended to support MPI reachable hosts within a cluster. By taking advantage of the fast interconnects, data can be striped across multiple local storage nodes for each client-server TCP connection. An example federated SRB configuration is shown in Figure 8, using two storage nodes per edge node. For remote writes, each chunk of data sent from the client is received by the edge node, and then forwarded in a round-robin fashion to each of the storage nodes. For remote reads, the process is reversed. The addition of a second tier of

.

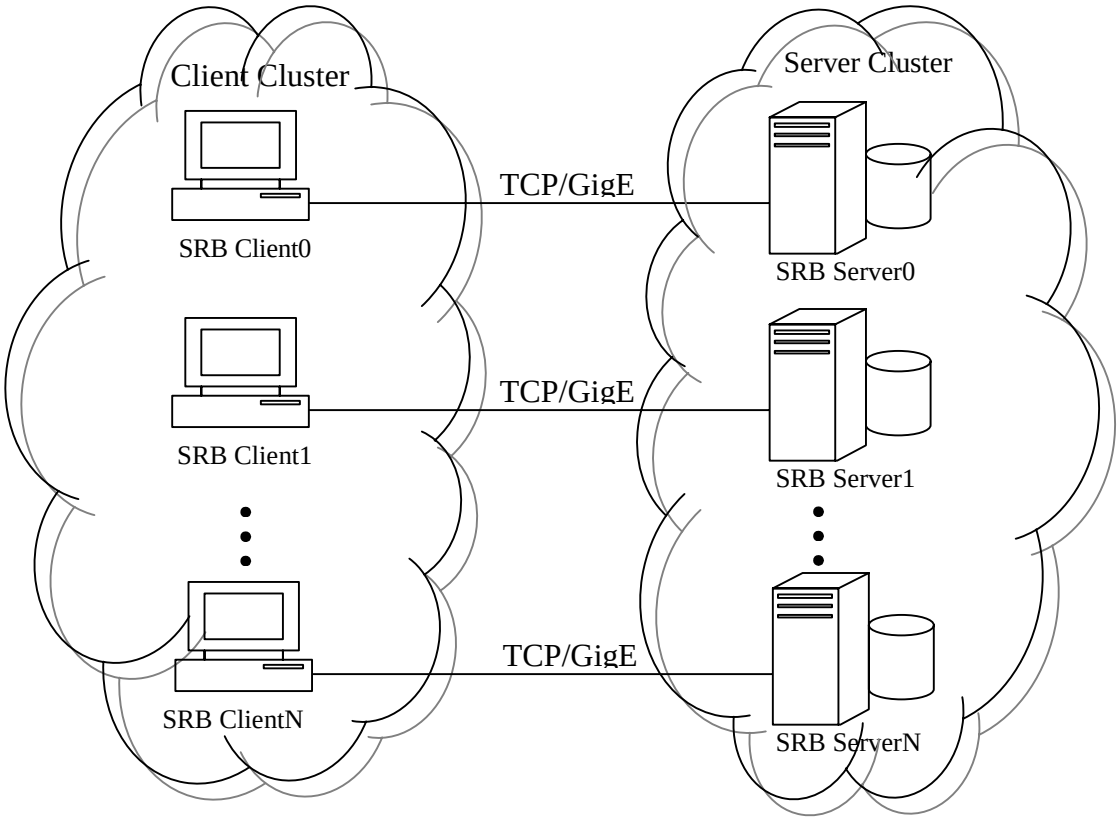

Figure 6: Network Striped Topology.

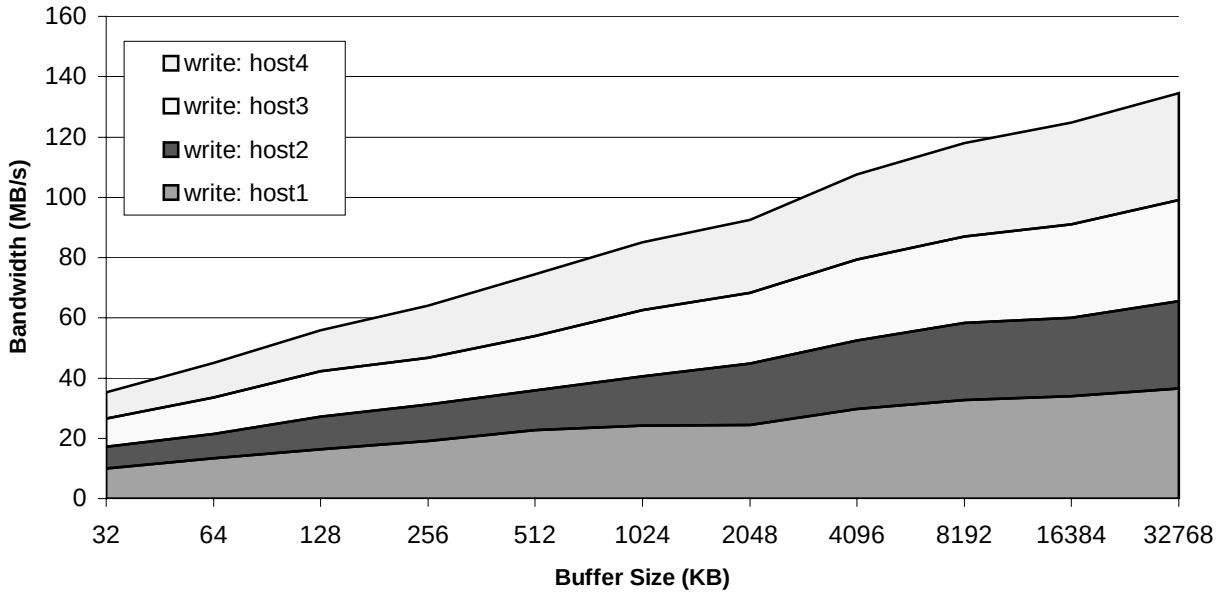

Figure 7: Aggregate network striped remote write throughput with per-node contribution.

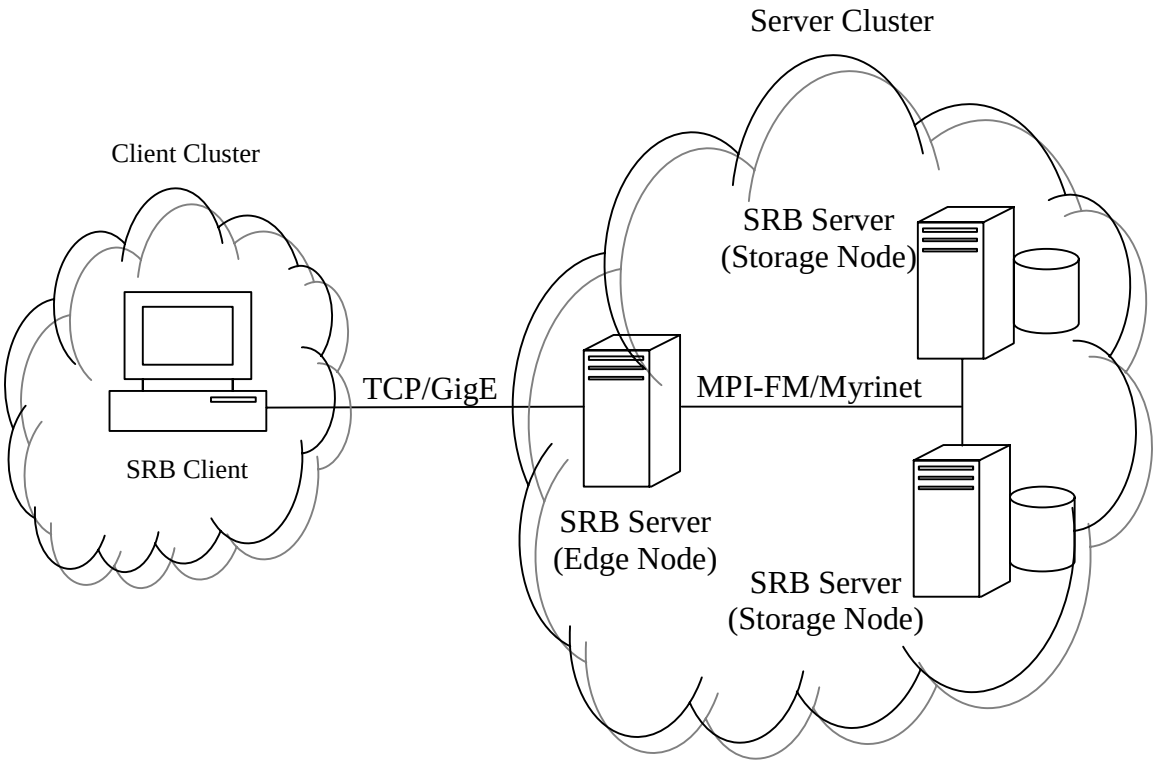

Figure 8: Federated SRB-MPI Topology.

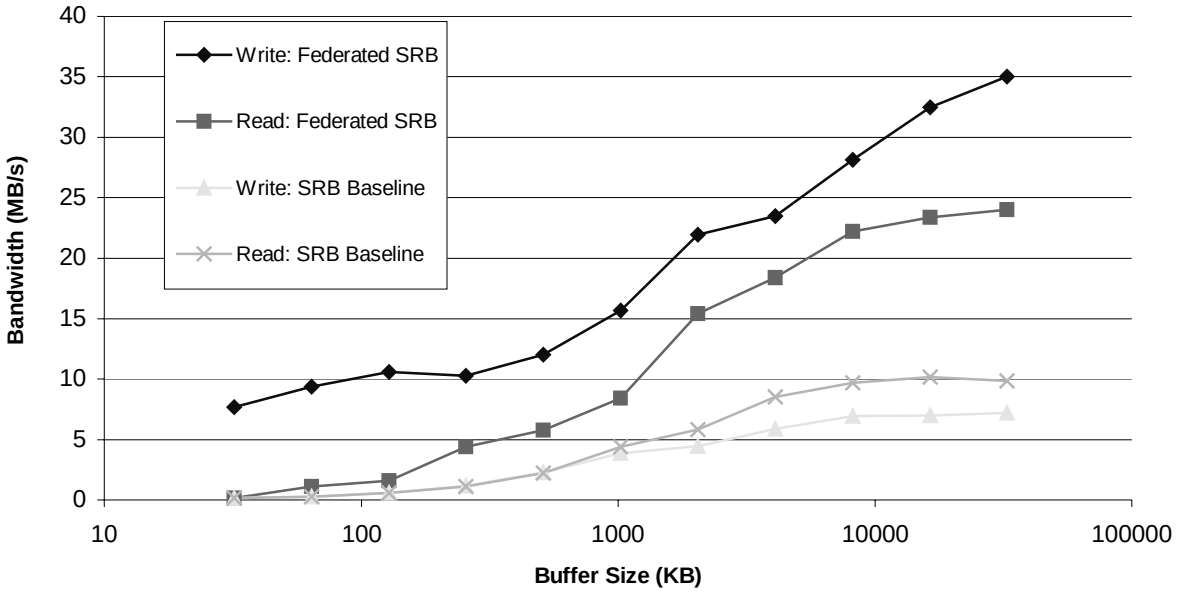

Figure 9: Federated SRB-MPI throughput using two Netserver storage nodes.

servers and of communication links increased the length of the data pipeline; a careful implementation preserving overlapped operation between all stages was crucial for performance.

For this experiment we reversed our usual setup and used the Kayak cluster as the client and the Netserver cluster as the server. With the Netservers on the server side the bottleneck is the disk throughput, which is limited to 18MB/s on each node. Figure 9 shows the results obtained using a ratio of two storage nodes per edge node. In this case, SRB-MPI throughput is much higher than a single storage node, reaching 97% of the combined write capacity of two nodes.

6. Related Work

One system that shares many of SRB design goals is the Global Access to Secondary Storage (GASS) package, which is a part of the GLOBUS [21] toolkit for distributed computing. GASS tries to optimize remote file access using a number of client-side caching schemes instead of overlapping communication and disk transfer [22]. Lee et al. [23] describe models for predicting the performance of applications that overlap computation and communication. The models consider several options for configuring application run-time environments including dedicated versus shared I/O processors and threads, the ratio of compute nodes to I/O nodes, variability of Internet throughput, and the computation/communication characteristics of the target application. These models would need to be extended to include the effects of overlapped communication and disk I/O we studied in our project. The DPSS project [15] uses a high-speed distributed data storage cache as a source and sink for data intensive applications. Real-time network and system monitoring information is used to load balance data across multiple servers using a minimum cost flow algorithm. DPSS does support parallel access from a single client to multiple servers, but there is no pipelining of network and disk operations. GridFTP [24] has a network striping functionality to improve performance, which differs from our scheme in that the multiple connections originate from a same client. The single client scheme is not effective in those cases in which the bottleneck is the protocol processing at the endpoints, and not the network throughput.

7. Conclusion

This goal of our project was to study optimization techniques for large remote file transfers, and their implementation issues on current commodity systems. By implementing our throughput optimizations in popular remote storage access tools such as SRB, our high-performance enhancements are immediately available to application programmers. By implementing performance enhancements incrementally, the contribution of each could be measured individually and compared against the unmodified SRB throughput and the maximum remote throughput.

A first set of improvements increases the base throughput of SRB by using a notion of pipelined transport which enables overlapping of the different pipeline stages. Different optimizations were required to fully enable overlapping: asynchronous disk I/O (write/read throughput improved by 85%/10% over original SRB), aggregate acknowledgements (214%/ 110% improvement), asynchronous TCP/IP communication (428%/210% improvement). The final value of write/read throughput corresponds to 95%/97% of the maximum achievable throughput.

Since end-to-end throughput is heavily affected by the specifics of each system, we provide two additional methods of tuning performance: network striped I/O, and federated SRB service over MPI. Network Striped I/O provides a way to match the throughput needs of the client application with the available network and disk throughput capacity; we showed a nearly linear speedup using four-way network striping. MPI over a fast cluster can be usefully employed for cluster-level disk striping. Using our federated SRB over MPI in a setup with slow disks, we were able to achieve 97% of the combined write capacity of multiple nodes and to saturate the TCP link

8. Acknowledgments

We wish to thank Reagan Moore and Arcot Rajasekar of the Data Intensive Computing (DICE) group at the San Diego Supercomputer Center for giving us access to the SRB source. This work is supported in part by the Defense Advanced Research Projects Administration through United States Air Force Rome Laboratory Contracts AFRL F30602-99-1-0534 and the National Science Foundation thru the National Computational Science Alliance, the National Partnership for Advanced Computational Infrastructure, and NSF EIA-99-75020 Grads. Support from Microsoft, Hewlett-Packard, 3Ware and Packet Engines (now Alcatel) is also gratefully acknowledged. M. L. wishes to thank Henri Bal of Vrije Universiteit in Amsterdam for the insightful discussions and for hosting him as a Visiting Scientist.

9. References

[1] D. A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, B.A. Rapp, and D.L. Wheeler, *GenBank*, Nucleic Acids Research, 28, 15-18, 2000.

[2] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, *Gapped BLAST and PSI-BLAST: A new generation of protein database search programs*, Nucleic Acids Research, 25 pp. 3899-3402, 1997.

[3] A. Krogh, M. Brown, I.S. Mian, K. Sjolander, and D. Haussler, *Hidden Markov models in computational biology: Applications to protein modeling*, JMB 235:1501-1531, 1994.

[4] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne: *The Protein Data Bank*, Nucleic Acids Research, 28 pp. 235-242, 2000.

[5] J. E. Gunn and G.R. Knapp, *The Sloan Digital Sky Survey*, in Sky Surveys, Protostarts to Protogalaxies, T. Soifer, ed., Ast. Soc. of Pacific Conference Series #43, pp. 267-79, 1992.

[6] P. Stolorz, E. Mesrobian, R. Muntz, J. Santos, E. Shek, J. Yi, C. Mechoso, and J. Farrara, *Fast Spatio-Temporal Data Mining from Large Geophysical Datasets*, in Proceedings of the First International Conference on Knowledge Discovery and Data Mining, pp. 300-305, 1995.

[7] J. Stiles, T. Bartol, E. Salpeter, and M. Salpeter, *Monte Carlo simulation of neuromuscular transmitter release using Mcell, a general simulator of cellular physiological processes*, Computational Neuroscience, pp 279-284, 1998.

[8] V. Anupam, C Baja, D. Schikore, and M. Schikore, *Distributed and Collaborative Visualization*, IEEE Computer 27 (7), pp 37-43, 1994.

[9] I. Foster and C. Kesselman (eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Fransisco, CA, 1999.

[10] The European DataGrid Project: http://eu-datagrid.web.cern.ch/eu-datagrid.

[11] The SDSC Storage Resource Broker Homepage: http://www.npaci.edu/DICE/SRB.

[12] The Internet Backplane Protocol Homepage: http://icl.cs.utk.edu/ibp.

[13] R. Muntz, J.R Santos, and S. Berson. *RIO: A real-time multimedia object server*, ACM Performance Evaluation Review, 25(2), pp. 29-35, September 1997.

[14] The General Parallel File System Homepage: http://www.almaden.ibm.com/cs/gpfs.html.

[15] B. Tierney, W. Johnston, H. Herzog, G. Hoo, G. Jin, J. Lee, L. Chen, D. Rotem, *Distributed Parallel Data Storage Systems: A Scalable Approach to High Speed Image Servers*, ACM Multimedia '94, San Francisco, October 1994.

[16] E. Nallipogu , *Increasing the Throughput of the SDSC Storage Resource Broker*, M.S. Thesis, Dept. of Electrical Engineering, Ohio State University, 2001.

[17] E. Nallipogu, F. Ozguner, M. Lauria, *Improving the Throughput of Remote Storage Access through Pipelining*, submitted for publication.

[18] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, 1997. http://www.mpi-forum.org

[19] E. Riedel, C Van Ingen, and J. Gray, *A Performance study of sequential I/O on Windows NT 4*, In Proceedings of 2nd USENIX Windows NT, pages 1-10, 1998.

[20] USNA, *TTCP: A test of TCP and UDP Performance*, December 1984.

[21] I. Foster, C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[22] J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke, *GASS: A Data Movement and Access Service for Wide Area Computing Systems*, Sixth Workshop on I/O in Parallel and Distributed Systems, May 1999.

[23] J. Lee, M. Winslett, X. Ma, and S. Yu. *Tuning High-Performance Scientific Codes: The Use of Performance Models to Control Resource Usage During Data Migration and I/O*, In Proceedings of the Fifteenth ACM International Conference on Supercomputing, June 2001.

[24] The GridFTP Homepage: http://www.globus.org/datagrid/gridftp.html