

SEMPLAR: High-Performance Remote Parallel I/O over SRB *

Nawab Ali and Mario Lauria
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{alin, lauria}@cse.ohio-state.edu

Abstract

One of the challenges in high-performance computing is to provide users with reliable, remote data access in a distributed, heterogeneous environment. The increasing popularity of high-speed wide area networks and centralized data repositories lead to the possibility of direct high-speed access to remote data sets from within a parallel application. In this paper we describe SEMPLAR, a library for remote, parallel I/O that combines the standard programming interface of MPI-IO with the remote storage functionality of the SDSC Storage Resource Broker (SRB). SEMPLAR relies on parallel TCP streams to maximize the remote data throughput in a design that preserves the parallelism of the access all the way from the storage to the application. We have provided I/O performance results for a high-performance computing workload on three different clusters. On the NCSA TeraGrid cluster, the ROMIO `perf` benchmark attained an aggregate read bandwidth of 291Mbps with 18 processors. The NAS `btio` benchmark achieved an aggregate write bandwidth of 74Mbps with 16 processors. The benchmark results are encouraging and show that SEMPLAR provides applications with scalable, high-bandwidth I/O across wide area networks.

1 Introduction

Recent trends in high-performance computing have seen a shift towards distributed resource management. High-performance applications are increasingly accessing data stored in remote locations. This trend is a marked deviation from the earlier norm of co-location of application and its data. This paradigm shift has been brought about partly by the availability of high-speed wide area networks and the large amounts of data generated by these applications [16].

*This work is supported in part by the National Partnership for Advanced Computational Infrastructure, and by the Ohio Supercomputer Center through grants PAS0036 and PAS0121. Mario Lauria is partially supported by NSF DBI-0317335. Support from Hewlett-Packard is also gratefully acknowledged.

Data-intensive scientific applications in experimental physics, computational genomics and other engineering fields generate large volumes of data. This data is of the order of terabytes and petabytes and is shared by the entire scientific community. The sheer magnitude of this data makes it impossible for researchers to own individual copies of datasets.

Traditionally, researchers working with remote datasets have manually transferred data to their local filesystems using protocols such as FTP. This process, known as *staging* [16] is then reversed for the resulting output data. There are a few drawbacks associated with staging. It requires that the researcher be aware of the filesystem details of the remote data location. This technique also prevents the overlapping of data transfer and computation, resulting in suboptimal application performance. Staging also goes against the requirement of seamless integration of remote I/O with the distributed computing environment [16].

This paper presents a new data access technique that mitigates the problems associated with *staging*. We have developed a scalable, high-performance, remote I/O library called SEMPLAR that performs I/O over the Internet. We have integrated our library with MPI-IO [14] thereby enabling remote data access from within a parallel application.

SEMPLAR is based on the SDSC Storage Resource Broker (SRB) [8, 4]. SRB is a middleware which provides applications with a uniform interface to distributed, heterogeneous storage resources. The storage resources consist of filesystems, databases and archival storage media. SEMPLAR uses multiple, parallel TCP streams [21, 6] across several cluster nodes to connect to SRB's data grid interface. The storage virtualization provided by SRB coupled with remote, parallel data access provides applications with high I/O bandwidth for remote data.

Given the availability of high-speed wide area networks, the main performance bottleneck in remote storage access is frequently the TCP protocol processing overhead at the endpoints [12]. In the remaining cases, the limiting factor is one of the intervening routers along the connection. Multiple,

parallel connections mitigate the performance bottleneck issue provided the connections have separate endpoints. By using a cluster on the client side and a multiprocessor machine with multiple Gigabit Ethernet interfaces on the server side, we effectively take advantage of the parallelism on the two sides to achieve scalable throughput. By adopting a parallel I/O interface such as MPI-IO, the parallelism is maintained all the way from the storage to the application.

This paper describes an I/O library that integrates parallel I/O with remote storage access. We study the performance that such I/O libraries achieve on typical clusters using benchmarks and application kernels. We also demonstrate the scalability of our approach vis-à-vis traditional data transfer techniques.

The rest of the paper is organized as follows. Section 2 summarizes the prior research that has been done in remote I/O. Section 3 presents an overview of SRB. Section 4 characterizes some of the issues related to the design and implementation of SEMPLAR. Section 5 and Section 6 present our experimental setup and benchmarks respectively. Section 7 discusses the results. We end by presenting our conclusions and future work in Section 8.

2 Related Work

High-performance remote I/O is an active area of research. Foster et al. [16] proposed a remote I/O library, RIO which is used by applications to access distributed filesystems. RIO uses parallel I/O interfaces to provide high-performance end-to-end data transfer. Another application that provides high-performance bulk data transfer is GridFTP [17, 6]. GridFTP is an extension to the FTP protocol. It uses among other things, network striping to increase the end-to-end I/O bandwidth of applications.

Distributed filesystems such as NFS [22], AFS [19] and DFS provide a uniform interface for remote I/O. However, the above filesystems are geared more towards desktop workloads as opposed to high-performance computing workloads. To illustrate, the NFS bandwidth over an Ethernet LAN is around 1-3Mbps, which is significantly less than the I/O bandwidth offered by an optimized remote I/O library (~10Mbps) [16].

Parallel filesystems such as PVFS [11] and PIOFS are optimized for high aggregate I/O bandwidth on local clusters. They are not designed to deal with the issues of security, high network latency and other performance tradeoffs that are associated with wide area networks. The Condor computing system uses remote procedure calls (RPC) for I/O. Applications running on Condor are linked to a runtime library that replaces the I/O system calls with RPCs. The I/O system calls are executed on the host that submitted the job [18]. Condor however was designed for a workstation environment. It does not provide any support for caching or pipelining.

Other researchers have worked on grid middleware services such as SRB [8] and the Globus Toolkit [15]. Nallipogu et al. [20] proposed a mechanism to increase the data transfer throughput by pipelining the various data transfer stages such as disk access and network transfer. Bell et al. [9] optimized the SRB protocol by overlapping the network communication and disk access. Globus Access to Secondary Storage (GASS) [10] provides high-performance remote file access by using aggressive caching schemes. GASS however, is not a general purpose distributed filesystem. Rather, it provides support for some common grid file access patterns.

3 Storage Resource Broker

The Storage Resource Broker (SRB) was developed by the San Diego Supercomputer Center (SDSC) to provide support for data-intensive computing. SRB aims to provide high-performance I/O to scientific applications which access large volumes of data. It was also designed to provide efficient search, storage and retrieval of data stored in digital libraries [8, 20, 4].

SRB is a middleware that provides a uniform interface to distributed and heterogeneous storage resources. It provides applications with a *Logical Storage Resource (LSR)* [8]. The LSR abstracts the location and filesystem information associated with each physical resource, thus presenting a simple, uniform interface to data stored in a distributed environment. The SRB API is semantically similar to the POSIX file I/O API.

The SRB system consists of SRB clients, SRB servers and the Metadata Catalog Service (MCAT). Each SRB server controls a fixed set of physical storage resources. Multiple SRB servers interact with each other to provide a federated operation. During a federated operation, one SRB server acts as a client to other servers. The SRB server consists of a Master daemon process and SRB agents. The Master daemon listens on a well-known port for connection requests from clients. Clients and servers communicate using TCP/IP. On receiving a client request, the SRB Master spawns a SRB Agent to service the client request.

The Metadata Catalog Service manages the attributes associated with the SRB system resources. The SRB object metadata consists of information used to locate and control access to data. Applications can query the MCAT to search and locate the required data. They can then access the data by using the SRB API.

4 High-Performance Remote I/O

High-performance computing applications in fields such as high energy physics, genomics and astronomy generate large datasets. These datasets are commonly held at specialized facilities, typically regional supercomputer centers.

At the same time, due to recent technological trends and investments in grid infrastructure, production clusters are frequently connected to high-speed wide area networks and national or regional high capacity backbones. In this context it is interesting to study I/O techniques that can provide high speed access to remote datasets. SEMPLAR was designed to provide applications with high-bandwidth, remote data access using a parallel I/O programming interface.

4.1 Concurrent Parallel TCP Flows

TCP/IP connections when used for remote data access represent a well-known performance bottleneck. Depending on the specific configuration, the origin of the bottleneck can either be traced to the routing along the path of the connection or the protocol processing overhead at the endpoints. An analysis of concurrent TCP flows has shown that using multiple, parallel TCP streams across wide area networks can lead to a significant increase in the aggregate bandwidth when compared to a single TCP stream [21, 6, 17].

However, using concurrent TCP streams for data transfer does not help in situations where the protocol processing overhead is likely to be the main performance bottleneck. This is true for example in situations where the compute cluster and the storage cluster are co-located in the same computing facility or have privileged access to a common backbone. This scenario is bound to become more frequent as more supercomputer centers assume the responsibility of hosting large, scientific datasets and get connected to one of the growing number of high-speed regional or national backbones. The most general solution to high-speed remote data access thus requires addressing both sources of overhead simultaneously.

SEMPLAR uses multiple, concurrent TCP streams for I/O with separate endpoints for each TCP stream. It establishes a separate connection between each I/O node on the source and destination clusters. Since I/O data is already partitioned across cluster nodes during parallel I/O, the data scattering across multiple endpoints does not represent a problem on the source side. On the server side, we exploit the growing trend to use dedicated, parallel machines as high-performance storage servers [20, 9, 7]. In our experiments the server is a 36x SMP machine with 6 Gigabit Ethernet interfaces. Other centers use dedicated clusters as storage servers. These machines typically have state-of-the-art local file systems dimensioned for high-throughput data access.

4.2 SEMPLAR Design Issues

SEMPLAR was designed to provide storage virtualization and high I/O bandwidth to data-intensive applications performing I/O over wide area networks. One of its objectives was to provide an application interface to parallel data

streams such that the parallelism was maintained all the way from the storage to the application. We achieved this objective by using multiple, parallel TCP streams to transfer data and by integrating the SRB-enabled I/O library with MPI-IO.

We also had to decide on the SRB API to implement the I/O library. SRB offers two sets of C APIs to interface with the remote SRB server. The SRB high-level API handles data objects that use the MCAT server for metadata management. The high-level API registers the data object in the MCAT server and the MCAT maintains the metadata information associated with the object. The low-level SRB API does not register the data object with the MCAT server, resulting in lower overhead during I/O. SEMPLAR uses the high-level SRB API for I/O in spite of the overhead associated with maintaining metadata information in the MCAT server. The decision to use the high-level SRB API was made primarily to take advantage of the storage virtualization offered by the MCAT server.

Another design issue involved the synchronization of reads and writes on the SRB server. Multiple client nodes performing I/O simultaneously on a single file can potentially lead to incorrect ordering of data. SEMPLAR uses explicit file offsets to synchronize the multiple I/O streams on the SRB server. By specifying the offset, each client writes to a predefined section of the file. The application specifies the file offset by making a call to the MPI function, `MPI_File_set_view`.

4.3 SEMPLAR Implementation

The MPI-2 standard [14] defined a new API for parallel I/O called MPI-IO. MPI-IO is an extensive API designed specifically for portable, high-performance I/O. Thakur et al. defined an *Abstract-Device Interface for I/O (ADIO)* [23, 24] which is used to implement parallel I/O APIs. A parallel I/O API can be implemented portably across diverse filesystems by implementing it over ADIO. The ADIO interface is then implemented for each specific filesystem [3]. This provides a portable implementation of the parallel I/O API while exploiting the specific high-performance features of individual filesystems.

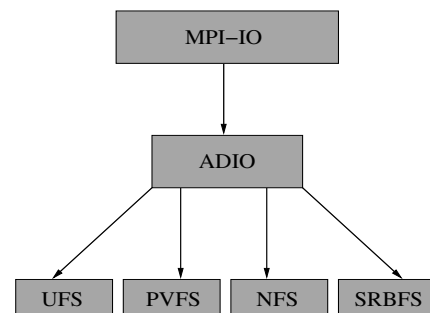


Figure 1: The ADIO Architecture

We have provided a high-performance implementation of ADIO for the SRB filesystem. The ADIO implementation connects to the remote SRB server over TCP/IP to perform parallel I/O. Each cluster node performing I/O on the file stored in the SRB repository opens an individual TCP connection to the SRB server. The connection is established during the call to the `MPI_File_open` function. The file is logically divided into segments using the `MPI_File_set_view` call and each node performs I/O on its segment independent of the other nodes. The parallelism associated with multiple nodes performing I/O simultaneously on smaller segments of a large file results in higher data throughput. Since each I/O node uses a separate TCP connection to transfer data to the SRB server, we are better able to utilize the available network bandwidth. The connection to the SRB server is terminated by the `MPI_File_close` function. The SRB ADIO implementation provides support for collective and non-collective I/O. Table 1 provides a mapping between some of the basic MPI-IO, ADIO and SRB calls.

MPI-IO API	ADIO API	SRB API
<code>MPI_File_open</code>	<code>ADIO_Open</code>	<code>srbObjOpen</code>
<code>MPI_File_read</code>	<code>ADIO_ReadContig</code>	<code>srbObjRead</code>
<code>MPI_File_write</code>	<code>ADIO_WriteContig</code>	<code>srbObjWrite</code>
<code>MPI_File_close</code>	<code>ADIO_Close</code>	<code>srbObjClose</code>

Table 1: Mapping between MPI-IO, ADIO and SRB API

5 Experimental Setup

This section describes the setup that we used to evaluate the performance of SEMPLAR.

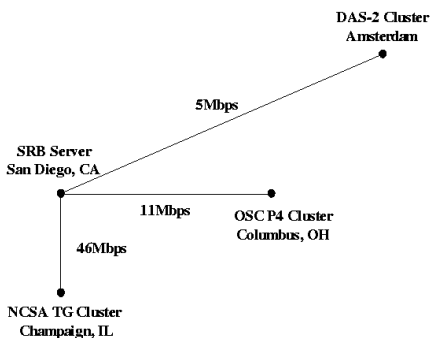


Figure 2: The Experiment Topology

SDSC SRB Server - The SDSC SRB team manages a production SRB server (version 3.2.1) on `orion.sdsc.edu`. `orion` is a high-end SUN Fire 15000 machine. It contains 36 900MHz SPARC III+ processors, 144GB of memory, 6 Gigabit Ethernet interfaces for data, 1 Gigabit Ethernet interface for control information and 16 T9940B tape drives. The machine runs the Solaris 9 operating system.

Distributed ASCI Supercomputer 2 - DAS-2 [5] is a wide-area distributed cluster designed by the Advanced School for Computing and Imaging (ASCI) in the Netherlands. It consists of 200 Dual Pentium-III nodes. Each node contains two 1GHz Pentium-III processors, 1GB of RAM, a 20 GB local IDE disk, a Myrinet interface card and an on-board Fast Ethernet interface. Each compute node is connected to the outside world by a 100Mbps link. The nodes run the Red Hat Enterprise Linux operating system.

OSC Pentium 4 Xeon Cluster - The Ohio Supercomputer Center [2] Pentium 4 cluster is a distributed/shared memory hybrid system. The cluster consists of 512, 2.4GHz Intel Xeon processors. Each node has two 2.4GHz processors, 4 GB of memory, a 100Base-T Ethernet interface and one Gigabit Ethernet interface. The nodes run the Red Hat Enterprise Linux operating system.

NCSA TeraGrid cluster - The TeraGrid cluster at the National Center for Supercomputing Applications (NCSA), consists of 887 IBM cluster nodes. Each of the 256 *Phase 1* nodes contains dual 1.3 GHz Intel Itanium 2 processors while the remaining 631 *Phase 2* nodes contain dual 1.5 GHz Intel Itanium 2 processors each. The *Phase 2* nodes are equipped with 4GB of memory. Half of the *Phase 1* nodes contain 4GB of memory while the remaining nodes are large-memory processors with 12GB of memory per node. Each node is also equipped with a Gigabit Ethernet interface. The cluster runs the SuSE Linux operating system.

6 Benchmarks

We used two microbenchmarks and an application to evaluate SEMPLAR. This section gives a brief description of the benchmarks.

ROMIO perf - `perf` is a sample MPI-IO program included in the ROMIO source code. It measures the read and write performance of a filesystem. Each process contains a data array which is written to a fixed location in the file using `MPI_File_write`. The array is then read back using `MPI_File_read`. The location from which a process reads and writes data in the file depends on its rank. There are two variations to this benchmark. `perf` reports I/O bandwidth measurements with and without making calls to `MPI_File_sync`. The `MPI_File_sync` call is made before read operations and after write operations. `perf` essentially provides an upper-bound on the MPI-IO performance that can be expected from a filesystem [7].

NAS btio - The NAS Parallel Benchmarks (NPB) are a small set of programs derived from computational fluid dynamics (CFD) applications. The NPB applications are designed to evaluate the performance of parallel supercomputers [1]. The `btio` benchmark is a variation of the `bt` appli-

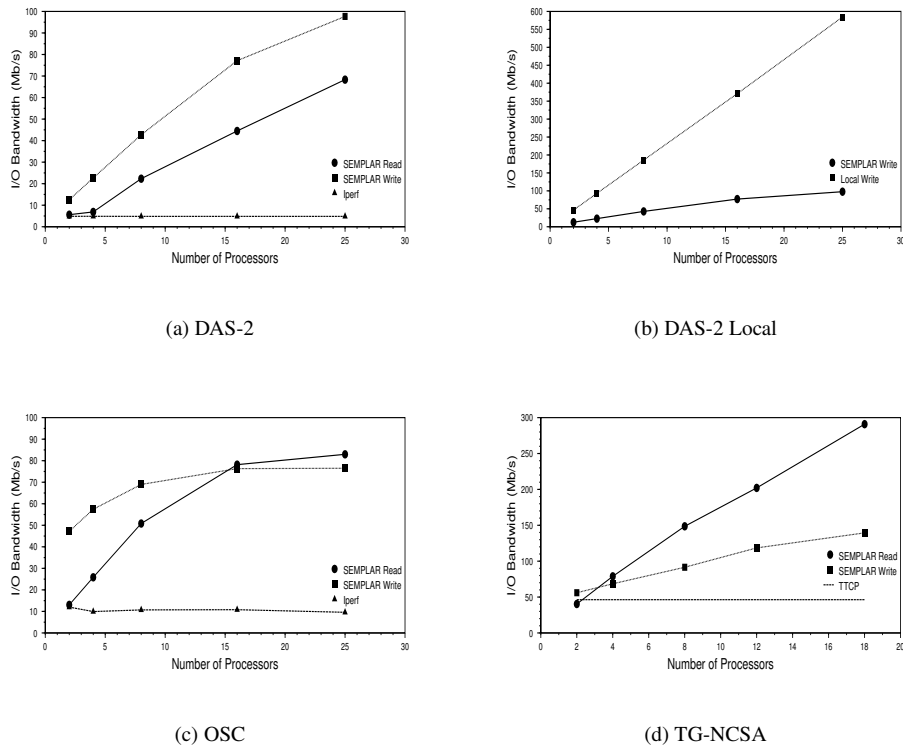


Figure 3: perf I/O Performance

cation from the NPB suite. The `bt` application solves systems of block-tridiagonal equations in parallel. The `btio` benchmark added several different methods of doing periodic solution checkpointing in parallel, including Fortran direct unformatted I/O and MPI-IO [7].

Ground Motion Simulation - Ground Motion Simulations are used to model geological phenomena. Besides being compute-intensive, these simulations also produce large datasets. We have used a basic version of the Dynamic Fault Model (DFM) rupture dynamics simulation as one of our benchmarks. This version does not contain the entire earthquake modeling code but the MPI-IO functions are the same as the ones used in the full DFM simulation.

DFM models spontaneous fractures on a planar fault within a 3D isotropic, viscoelastic solid [13]. The simulation uses domain decomposition to achieve parallelization. It performs I/O using the collective MPI-IO API.

7 Results

This section presents the I/O performance results for `perf`, `btio` and the DFM benchmarks. We ran the benchmarks on the DAS-2, NCSA TeraGrid and the OSC P4 Xeon clusters.

The SDSC SRB server (version 3.2.1) running on

`orion.sdsc.edu` was used to interface with the data repository at SDSC. We have integrated our remote I/O library with `mpich-1.2.6`. The MPI library uses TCP/IP for communication between the nodes. We have also compared SEMPLAR’s I/O bandwidth with the available network bandwidth between the individual clusters and the SRB server ¹. We measured the available network bandwidth using `Iperf` and `TTCP` ². `Iperf` enabled us to measure the network bandwidth using multiple, parallel TCP streams. We were restricted to a single TCP stream with `TTCP`. We used the default value of the TCP buffer size on each cluster.

ROMIO perf - Figure 3(a) shows the SEMPLAR I/O results for the `perf` benchmark on the DAS-2 cluster. The compute nodes in DAS-2 are connected to the outside world over a 100Mbps link. Each node reads and writes an array of size 32MB to the remote SRB server. The `perf` benchmark achieves an aggregate read and write bandwidth of 68Mbps and 98Mbps respectively for 25 processors. These

¹The network bandwidth measurements were conducted between the clusters and another machine on the same LAN as the SRB server. This was done because we did not have ssh access to the machine running the SRB server.

²The `Iperf` binary is not available on the NCSA TeraGrid cluster. We have used `TTCP` instead to measure the available network bandwidth between NCSA and SDSC.

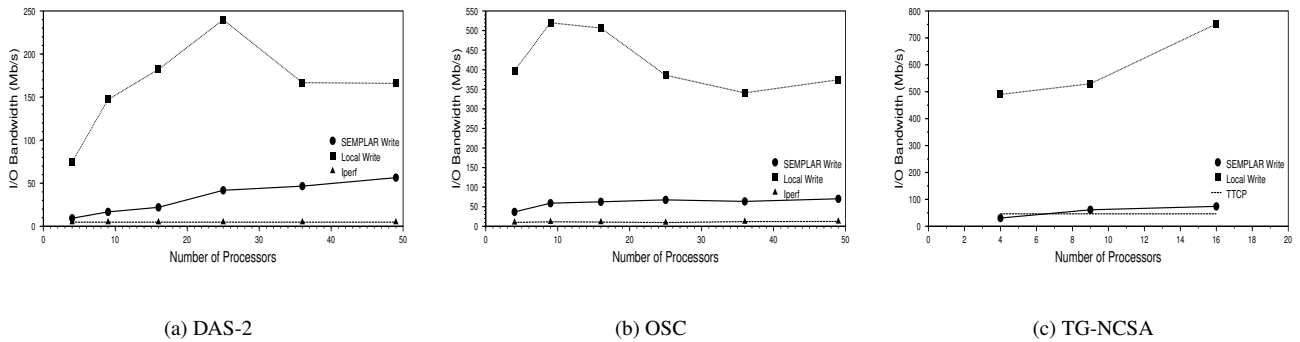


Figure 4: `btio` Class C Write Performance

numbers are encouraging considering the high average network latency (~ 182 ms) between the DAS-2 cluster and the SDSC SRB server. The aggregate write bandwidth of `perf` is 20 times the available network bandwidth between the individual DAS-2 nodes and the SRB server (5Mbps). Similarly, the aggregate read bandwidth is 14 times the available network bandwidth. These numbers put into perspective the significant performance potential of SEMPLAR. It is also interesting to note that SEMPLAR's aggregate, remote I/O bandwidth scales with the number of processors.

Figure 3(b) compares the local I/O performance of `perf` with remote I/O over wide area networks. We configured `perf` to write to the local filesystem for this experiment. As expected, the local I/O results are better than the remote I/O results. The maximum SEMPLAR I/O bandwidth (98Mbps) is about 6 times smaller than the maximum local I/O bandwidth (584Mbps).

We ran the `perf` benchmark on the OSC P4 Xeon cluster to evaluate the performance of SEMPLAR in a high-bandwidth, low-latency environment. Each node in the P4 cluster is equipped with a Gigabit Ethernet interface. The average network latency between OSC and SDSC is around 31ms. Figure 3(c) shows the SEMPLAR I/O performance on the OSC P4 cluster. The I/O bandwidth increases with the number of processors till the curves flatten around 25 processors. The network is almost completely saturated at this point and increasing the number of processors does not increase the I/O bandwidth. The maximum `perf` write bandwidth (76Mbps) is 7 times the average observed network bandwidth (11Mbps). The maximum `perf` read bandwidth (83Mbps) is 7.5 times the average observed network bandwidth at each node.

The average SEMPLAR I/O bandwidth observed on the OSC P4 cluster (80Mbps) is less than the average I/O bandwidth observed on the DAS-2 cluster (83Mbps). The reason why `perf` reports low I/O bandwidth on the OSC cluster is that the individual cluster nodes do not have a public IP address. The nodes are connected to the outside world via a

Network Address Translation (NAT) host. As the number of processors increase, the NAT host becomes the bottleneck.

Figure 3(d) shows the performance of the `perf` benchmark on the NCSA TeraGrid cluster. `perf` achieves an aggregate read and write bandwidth of 291Mbps and 139Mbps respectively for 18 processors. The maximum `perf` write bandwidth is 6 times the average observed network bandwidth (46Mbps). The maximum `perf` read bandwidth is 3 times the average observed network bandwidth at each node. These results are significantly better than the other two clusters because of the relatively low latency (~ 30 ms) and high-bandwidth between the TeraGrid cluster and the SRB server.

NAS `btio` - The NAS `btio` data access pattern is non-contiguous and is therefore difficult to handle efficiently with the traditional POSIX I/O interface. We used the *full* version of `btio` in our experiments. The *full* version of this benchmark uses collective I/O calls to perform the I/O. The collective I/O functions merge the data accesses of different processes and make large, well-formed I/O requests to the filesystem [11]. We used the `btio` Class C benchmark to measure the write performance of SEMPLAR. The Class C benchmark uses a $162 \times 162 \times 162$ element array. It writes about 6.7GB of data to the filesystem.

Figure 4 shows the `btio` performance results for the DAS-2, OSC P4 and the NCSA TeraGrid cluster. We see a marked increase in the aggregate SEMPLAR I/O bandwidth with the increase in the number of processors. The SEMPLAR I/O bandwidth peaks at 56Mbps for 49 processors on the DAS-2 cluster. This is 11 times the available network bandwidth at each node (5Mbps). The `btio` curve for the local `ext3` filesystem peaks at 240Mbps for 25 processors. The local I/O bandwidth decreases with the subsequent increase in the number of processors suggesting the saturation of the local disk bandwidth.

The `btio` benchmark on the OSC P4 cluster achieves a peak I/O bandwidth of 70Mbps for the SRB filesystem.

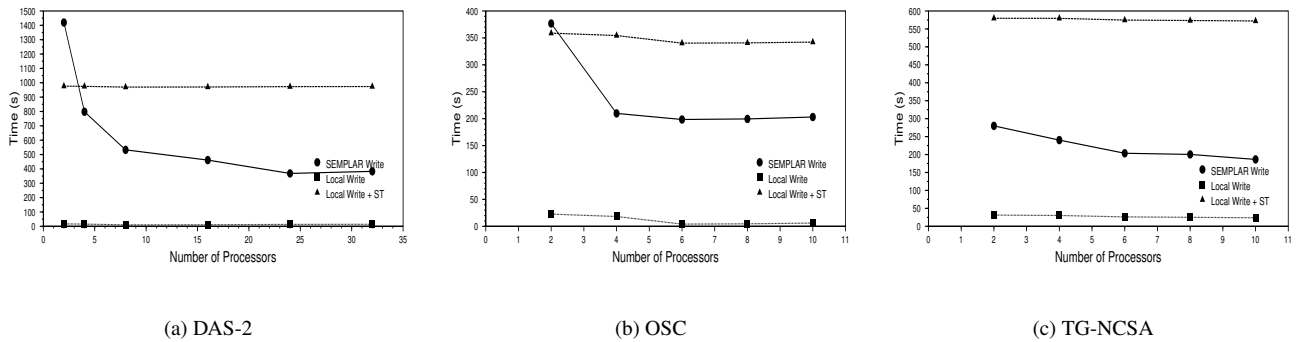


Figure 5: DFM Performance

This is 6.3 times the available network bandwidth at each node (11Mbps). Since the individual compute nodes do not have a public IP address, the NAT host is once again the bottleneck for the OSC P4 cluster. The I/O performance of SEMPLAR on the NCSA TeraGrid cluster is significantly better than the other two clusters. This is mostly due to the fact that the TeraGrid cluster is connected to a high-bandwidth (40Gbps) backbone. `btio` attains an aggregate write bandwidth of 74Mbps with 16 processors, achieving about 1.6 times the available network bandwidth (46Mbps) at each compute node.

The results for the `btio` benchmark show that SEMPLAR consistently provides applications with high I/O bandwidth across different network topologies. The I/O bandwidth also scales with the number of processors due to the parallelism incorporated in the design of SEMPLAR.

Dynamic Fault Model - We have used DFM as an example of a real-world scientific application. We ran the simulation on the SRB as well as the local filesystem. The DFM simulation used a model size of 141x221x221 nodes integrated over 10 time steps. During the first run, DFM was configured to write its dataset to the remote SRB filesystem. In the second run, we configured DFM to write to the local filesystem. We noted the total execution time of the simulation in both the cases. We also noted the time required to manually transfer DFM’s dataset to a remote repository after having stored it in the local filesystem.

Figure 5 shows the execution time of DFM on the DAS-2, OSC P4 and the NCSA TeraGrid cluster. DFM’s average execution time (660s) on the DAS-2 cluster for the SRB filesystem was about one order of magnitude more than on the local filesystem (13s). We also noticed a marked reduction in DFM’s execution time with the increase in the number of processors for the SRB filesystem. This can be attributed to the higher data throughput resulting from multiple, parallel TCP streams. DFM took the longest (973s) when data was written to the local filesystem and then man-

ually transferred to the remote SRB repository. The application execution time was 47% more than in the case when data was written to the SRB filesystem directly.

We observed similar results on the OSC P4 cluster. DFM’s average execution time (237s) on the SRB filesystem was about one order of magnitude more than on the local filesystem (11s). However, the application execution time when data was written to the SRB filesystem directly was 46% less than the time taken when data was written to the local filesystem and then manually transferred to the remote SRB repository. The average execution time of DFM for the SRB filesystem on the NCSA TeraGrid cluster (222s) was 39% less than the time taken for its execution on the local filesystem followed by *staging* the data to the remote SRB repository.

8 Conclusions and Future Work

High-performance computing applications increasingly need to access large datasets which are of the order of terabytes and petabytes. These datasets are usually stored in remote data repositories. One of the challenges in high-performance computing is to provide users with reliable, remote data and metadata access in a distributed, heterogeneous environment.

We have designed a high-performance, remote I/O library that provides parallel applications with high I/O bandwidth over wide area networks. Our library is based on the SDSC Storage Resource Broker (SRB). SRB is a middleware that provides applications with a uniform interface to distributed and heterogeneous storage resources. We have leveraged the high aggregate network bandwidth resulting from multiple, parallel TCP streams with the data ubiquity provided by SRB to develop a remote, scalable, high-performance I/O library.

We evaluated our library using two microbenchmarks and an application. On the NCSA TeraGrid cluster, the ROMIO `perf` benchmark attained an aggregate read

bandwidth of 291Mbps with 18 processors. The NAS `btio` benchmark achieved an aggregate write bandwidth of 74Mbps with 16 processors. The bandwidth observed by these benchmarks scaled with the number of processors.

In the future, we plan to work on client-side data caching to improve SRB's available data bandwidth. We would also like to quantitatively analyze the performance offered by asynchronous parallel I/O on the SRB filesystem.

9 Acknowledgments

We wish to thank Reagan Moore, Marcio Faerman and Arcot Rajasekar of the Data Intensive Group (DICE) at the San Diego Supercomputer Center for giving us access to the SRB source. We would also like to thank Geoffrey Ely, UCSD for providing us with parts of the DFM source code. We are especially indebted to Henri Bal of Vrije Universiteit, Amsterdam for giving us access to the DAS-2 cluster. We also wish to thank Rob Pennington and Ruth Aydt at the National Center for Supercomputing Applications (NCSA) for allowing us to use the NCSA TeraGrid cluster for our experiments.

References

- [1] NAS Parallel Benchmarks. <http://www.nas.nasa.gov/Software/NPB>.
- [2] Ohio Supercomputer Center. <http://www.osc.edu>.
- [3] ROMIO: A High-Performance, Portable MPI-IO Implementation. <http://www.mcs.anl.gov/romio>.
- [4] SDSC Storage Resource Broker. <http://www.sdsc.edu/srb>.
- [5] The Distributed ASCI Supercomputer 2. <http://www.cs.vu.nl/das2>.
- [6] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data Management and Transfer in High-Performance Computational Grid Environments. *Parallel Computing*, 28(5):749–771, 2002.
- [7] T. Baer. Parallel I/O Experiences on an SGI 750 Cluster. http://www.osc.edu/~troy/cug_2002.
- [8] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, Canada, 1998.
- [9] K. Bell, A. Chien, and M. Lauria. A High-Performance Cluster Storage Server. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, pages 311–320, 2002.
- [10] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. In *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, pages 78–88, May 1999.
- [11] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System for Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, 2000.
- [12] D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *IEEE Communications*, 27(6):23–29, 1989.
- [13] G. Ely, Q. Xin, M. Faerman, G. Kremenek, B. Shkoller, S. Day, K. Olsen, B. Minster, and R. Moore. Data Handling of a High Resolution Ground Motion Simulation. http://epicenter.usc.edu/cmeportal/docs/RDM_Description.pdf.
- [14] M. P. I. Forum. MPI-2: Extensions to the Message-Passing Interface. <http://www.mpi-forum.org/docs/docs.html>, 1997.
- [15] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Super-computer Applications and High Performance Computing*, 11(2):115–128, 1997.
- [16] I. Foster, D. Kohr, Jr., R. Krishnaiyer, and J. Mogill. Remote I/O: Fast Access to Distant Storage. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 14–25, San Jose, CA, 1997.
- [17] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke. Applied Techniques for High Bandwidth Data Transfers Across Wide Area Networks. In *Proceedings of International Conference on Computing in High Energy and Nuclear Physics*, Beijing, China, September 2001.
- [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, June 1988.
- [19] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith. Andrew: A Distributed Personal Computing Environment. *Communications of the ACM*, 29(3):184–201, 1986.
- [20] E. Nallipogu, F. Ozguner, and M. Lauria. Improving the Throughput of Remote Storage Access through Pipelining. In *Proceedings of the Third International Workshop on Grid Computing*, pages 305–316, 2002.
- [21] L. Qiu, Y. Zhang, and S. Keshav. On Individual and Aggregate TCP Performance. In *Proceedings of the Seventh Annual International Conference on Network Protocols*, pages 203–212, 1999.
- [22] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network Filesystem. In *Proceedings of the USENIX 1985 Summer Technical Conference*, pages 119–130, Portland OR, 1985.
- [23] R. Thakur, W. Gropp, and E. Lusk. An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, pages 180–187, 1996.
- [24] R. Thakur, W. Gropp, and E. Lusk. On Implementing MPI-IO Portably and with High Performance. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pages 23–32, 1999.