Experiences with 100Gbps Network Applications

Mehmet Balman, Eric Pouyoul, Yushu Yao, E. Wes Bethel
Burlen Loring, Prabhat, John Shalf, Alex Sim, and Brian L. Tierney
Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CA, 94720, USA
{mbalman,epouyoul,yyao,ewbethel,bloring,prabhat,jshalf,asim,btierney}@lbl.gov

ABSTRACT

100Gbps networking has finally arrived, and many research and educational institutions have begun to deploy 100Gbps routers and services. ESnet and Internet2 worked together to make 100Gbps networks available to researchers at the Supercomputing 2011 conference in Seattle Washington. In this paper, we describe two of the first applications to take advantage of this network. We demonstrate a visualization application that enables remotely located scientists to gain insights from large datasets. We also demonstrate climate data movement and analysis over the 100Gbps network. We describe a number of application design issues and host tuning strategies necessary for enabling applications to scale to 100Gbps rates.

Categories and Subject Descriptors

C.2.5 [**Local and Wide-Area Networks**]: High-speed
; C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Performance

Keywords

100Gbps Networking, Data Intensive Distributed Computing, Data Movement, Visualization

1. INTRODUCTION

Modern scientific simulations and experiments produce an unprecedented amount of data. End-to-end infrastructure is required to store, transfer and analyze these datasets to gain scientific insights. While there has been a lot of progress in computational hardware, distributed applications have been hampered by the lack of high-speed networks. Today, we have finally crossed the barrier of 100Gbps networking; these networks are increasingly becoming available to researchers, opening up new avenues for tackling large data challenges.

When we made a similar leap from 1Gbps to 10Gbps about 10 years ago, distributed applications did not automatically run 10 times faster just because there was more bandwidth available. The same is true today with the leap for 10Gbps to 100Gbps networks. One needs to pay close attention to application design and host tuning in order to be able to take advantage of the higher network capacity. Some of these issues are similar to those of 10 years ago, such as I/O pipelining and TCP tuning, but some are different due to the fact that we have many more CPU cores involved.

Copyright 2012 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the U.S. Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
DIDC'12, June 18, 2012, Delft, The Netherlands.
Copyright 2012 ACM 978-1-4503-1341-4/12/06 ...$10.00.

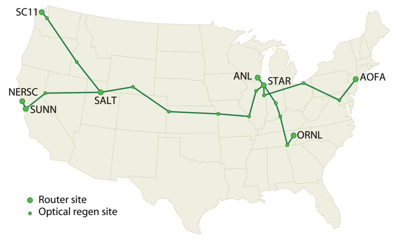

Figure 1: 100Gbps Network for Supercomputing 2011

ESnet and Internet2, the two largest research and education network providers in the USA, worked together to make 100Gbps networks available to researchers at the Supercomputing 2011 (SC11) conference in Seattle Washington, November 2011. This network, shown in Figure 1, included a 100Gbps connection between National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory (LBNL) in Oakland, CA, Argonne National Laboratory (ANL) near Chicago, IL, and Oak Ridge National Laboratory (ORNL) in Tennessee.

In this paper, we describe two of the first applications to take advantage of this network. The first application demonstrates real-time streaming and visualization of a 600 Gigabyte cosmology dataset. We illustrate how enhanced network capability enables remotely located scientists to gain insights from large data volumes. The second application showcases data distribution for climate science. We demonstrate how scientific data movement and analysis between geographically disparate supercomputing facilities can benefit from high-bandwidth networks.

The paper is organized as follows. First, we briefly review background information, and provide details about our testbed configuration for the SC11 demonstrations. Next, we provide technical information and optimization strategies utilized in the visualization demo. We then describe a climate data movement application and introduce a data streaming tool for high-bandwidth networks. We describe how the application design needed to be modified to scale to 100Gbps. We then discuss a number of Linux host tuning strategies needed to achieve these rates. Finally, we state lessons learned in end-system configuration and application design to fully utilize underlying network capacity and conclude with brief evaluation and future directions in use of 100Gbps networks.

2. BACKGROUND

2.1 The Need for 100Gbps Networks

Modern science is increasingly data-driven and collaborative in nature. Large-scale simulations and instruments produce petabytes of data, which is subsequently analyzed by tens to thousands of geographically dispersed scientists. Although it might seem logical and efficient to collocate the analysis resources with the source of the data (instrument or a computational cluster), this is not the likely scenario. Distributed solutions – in which components are scattered geographically – are much more common at this scale, for a variety of reasons, and the largest collaborations are most likely to depend on distributed architectures.

The Large Hadron Collider[1] (LHC), the most well-known high-energy physics collaboration, was a driving force in the deployment of high bandwidth connections in the research and education world. Early on, the LHC community understood the challenges presented by their extraordinary instrument in terms of data generation, distribution, and analysis.

Many other research disciplines are now facing the same challenges. The cost of genomic sequencing is falling dramatically, for example, and the volume of data produced by sequencers is rising exponentially. In climate science, researchers must analyze observational and simulation data sets located at facilities around the world. Climate data is expected to exceed 100 exabytes by 2020 [5]. The need for productive access to such data led to the development of the Earth System Grid[2] (ESG) [9], a global workflow infrastructure giving climate scientists access to data sets housed at modeling centers on multiple continents, including North America, Europe, Asia, and Australia.

Efficient tools are necessary to move vast amounts of scientific data over high-bandwidth networks, for such state-of-the-art collaborations. We evaluate climate data distribution over high-latency high-bandwidth networks, and state the necessary steps to scale-up climate data movement to 100Gbps networks. We have developed a new data streaming tool that provides dynamic data channel management and on-the-fly data pipelines for fast and efficient data access. Data is treated as first-class citizen for the entire spectrum of file sizes, without compromising on optimum usage of network bandwidth. In our demonstration, we successfully staged real-world data from the Intergovernmental Panel on Climate Change (IPCC) Fourth Assessment Report (AR4) Phase 3, Coupled Model Intercomparison Project[3] (CMIP-3) into computing nodes across the country at ANL and ORNL from NERSC data storage over the 100Gbps network in real-time.

2.2 Visualization over 100Gbps

Modern simulations produce massive amounts of datasets that need further analysis and visualization. Often, these datasets cannot be moved from the machines that the simulations are conducted on. One has to resort to in situ analysis (i.e. conduct analysis while the simulation is running), or remote rendering (i.e. run a client on a local workstation, and render the data at the supercomputing center). While these modes of operation are often desirable, a class of researchers would much rather prefer to stream the datasets to their local workstations or facilities, and conduct a broad range of visualization and analysis tasks locally. With the availability of the 100Gbps network, this mode of analysis is now feasible. To justify this claim, we demonstrate real-time streaming of a large multi-Terabyte sized dataset in a few minutes from DOE's production supercomputing facility NERSC, to four commodity workstations at SC11 in Seattle. For illustration purposes, we then demonstrate real-time parallel visualization of the same dataset.

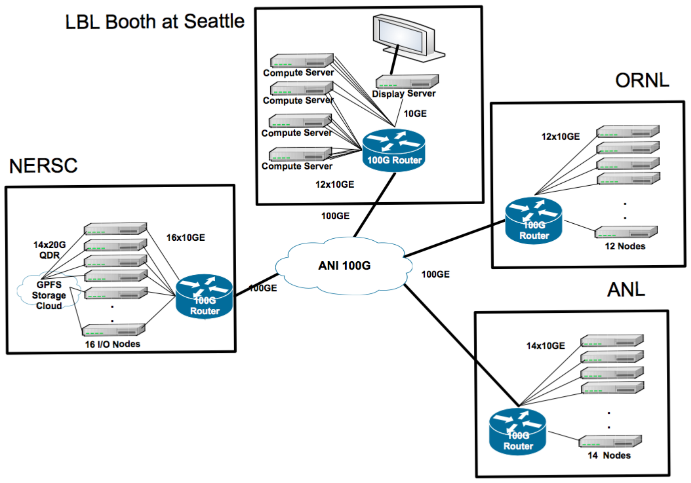

Figure 2: SC11 100Gbps Demo Configuration

3. 100Gbps TEST ENVIRONMENT

We performed our tests using a wide array of resources from DOE's Advanced Network Initiative [4] (ANI) network and testbed[5], and the DOE Magellan Project [16]. The ANI Network is a prototype 100Gbps network connecting DOE's three Supercomputer centers. These three centers include National Energy Research Scientific Computing Center[6] (NERSC), Argonne Leadership Class Facility[7], and Oak Ridge Leadership Class Facility[8]. The ANI Testbed includes high-speed hosts at both NERSC and ALCF. The Magellan project included large clusters at both NERSC and ALCF. 16 hosts at NERSC were designated as I/O nodes to be connected to the 100Gbps ANI network.

In the annual Supercomputing conference (SC11) held in Seattle, WA, ESnet[9] and Internet2[10] worked together to bring a 100Gbps link from Seattle to Salt Lake City, where it was connected to ESnet's ANI network, as shown in Figure 1.

We utilized 16 hosts at NERSC to send data, each with 2 quad-core Intel Nehalem processors and 48 GB of system memory. In addition to regular disk-based GPFS file system, these hosts are also connected via Infiniband to a Flash-based file system for sustained I/O performance during the demonstration. The complete system, including the hosts and the GPFS[11] filesystem can sustain

[1]The Large Hadron Collider http://lhc.web.cern.ch/lhc/
[2]Earth System Grid http://www.earthsystemgrid.org
[3]CMIP3 Multi-Model Dataset Archive at PCMDI http://www-pcmdi.llnl.gov/ipcc/

[4]Advanced Network Initiative http://www.es.net/RandD/advanced-networking-initiative/
[5]ANI Testbed http://sites.google.com/a/lbl.gov/ani-testbed/
[6]National Energy Research Center http://www.nersc.gov
[7]Argonne Leadership Class Facility http://www.alcf.anl.gov
[8]Oak Ridge Leadership Class Facility http://www.olcf.ornl.gov
[9]Energy Sciences Network http://www.es.net
[10]Internet2 http://www.internet2.edu
[11]GPFS http://www.ibm.com/systems/software/gpfs

an aggregated 16 GBytes/second read performance. Each host is equipped with a Chelsio 10Gbps NIC which is connected to the NERSC Alcatel-Lucent router.

We utilized 12 hosts at OLCF to receive data, each with 24GB of RAM and a Myricom 10GE NIC. These were all connected to a 100Gbps Juniper router. We used 14 hosts at ALCF to receive data, each with 48GB of RAM and a Mellanox 10GE NIC. These hosts were connected to a 100Gbps Brocade router. Each host at ALCF and OLCF had 2 quad-core Intel Nehalem processors. We measured a round-trip time (RTT) of 50ms between NERSC and ALCF, and 64ms between NERSC and OLCF. We used four hosts in the SC11 LBL booth, each with two 8-core AMD processors and 64 GB of memory. Each host is equipped with Myricom 10Gbps network adaptors, one dual port, and two single-port,connected to a 100Gbps Alcatel-Lucent router at the booth. Figure 2 shows the hosts that were used for the two 100Gbps applications at SC11.

4. VISUALIZING THE UNIVERSE AT 100Gbps

Computational cosmologists routinely conduct large scale simulations to test theories of formation (and evolution) of the universe. Ensembles of calculations with various parametrizations of dark energy, for instance, are conducted on thousands of computational cores at supercomputing centers. The resulting datasets are visualized to understand large scale structure formation, and analyzed to check if the simulations are able to reproduce known observational statistics. In this demonstration, we used a modern cosmological dataset produced by the NYX[12] code. The computational domain is 1024^3 in size; each location contains a single precision floating point value corresponding to the dark matter density at each grid point. Each timestep corresponds to 4GB of data. We utilize 150 timesteps for our demo purposes.

To demonstrate the difference between the 100Gbps network and the previous 10Gbps network, we split the 100Gbps connection into two parts. 90Gbps of the bandwidth is used to transfer the full dataset. 10Gbps of the bandwidth is used to transfer 1/8th of the same dataset at the same resolution. By comparing the real-time head-to-head streaming and rendering results of the two cases, the enhanced capabilities of the 100Gbps network are clearly demonstrated.

4.1 Demo Configuration

Figure 3 illustrates the hardware configuration used for this demo. On the NERSC side, the 16 servers described above, named "Sender 01-16", are used to send data. The data resides on a the GPFS file system. In the LBL booth, four hosts, named "Receive/Render H1-H4", are used to receive data for the high bandwidth part of the demo. Each server has two 8-core AMD processors and 64 GB of system memory. Each host is equipped with 2 Myricom dual-port 10Gbps network adaptors which are connected to the booth Alcatel-Lucent router via optical fibers. The "Receive/Render" servers are connected to the "High Bandwidth Vis Server" via 1Gbps ethernet connections. The 1Gbps connection is used for synchronization and communication of the rendering application, not for transfer of the raw data. A HDTV is connected to this server to display rendered images. For the low bandwidth part of the demo, one server, named "Low Bandwidth Receive/Render/Vis", is used to receive and render data. A HDTV is also connected to this server to display rendered images. The low bandwidth host is equipped with 1 dual-port 10Gbps network adaptor which is connected to the booth router via 2 optical fibers. The one-way latency from NERSC to the LBL booth was measured at 16.4 ms.

[12]NYX: https://ccse.lbl.gov/Research/NYX/index.html

4.2 UDP shuffling

Prior work by Bethel, et al. [6] has demonstrated that the TCP protocol is ill-suited for applications that need sustained high-throughput utilization over a high-latency network channel. For visualization purposes, occasional packet loss is acceptable, we therefore follow the approach of VisaPult[6] and use the UDP protocol for transferring the data for this demo.

We prepared UDP packets by adding position (x, y, z) information in conjunction with the density information. While this increases the size of the streamed dataset by a factor of 3 (summing up to a total of 16GB per timestep), this made the task of the placing the received element into the right memory offset trivial. Also, we experimented with different data decomposition schemes (z-ordered space filling curves) as opposed to a z-slice based ordering, and this scheme allowed us to experiment with both schemes without any change in the packet packing/unpacking logic.

As shown in Figure 4, a UDP packet contains a header followed by a series of quad-value segments. In the header, the batch number used for synchronization purposes, i.e., packets from different time steps have different batch numbers. An integer n is also included in the header to specify the number of quad-value segments in this packet. Each quad-value segment consists 3 integers, which are the X, Y and Z position in the 1024^3 matrix, and one float value which is the particle density at this position. To maximize the packet size within the MTU value of 9000, the number n is set to 560 which gives the optimal packet size of 8968 bytes, which is the largest possible packet size under 8972 bytes (MTU size minus IP and UDP headers) with the above described data structure.

For each time step, the input data is split into 32 streams along the z-direction; each stream contains a contiguous slice of the size $1024 * 1024 * 32$. Each stream is staged, streamed and received separately for the purpose of reliability and maximizing parallel throughput. Figure 4 shows the flow of data for one stream. A stager first reads the data into a memory-backed file system (*/dev/shm*), it is optimized to reach the maximum read performance of the underlying file system. The stager also buffers as many future time steps as possible, to minimize the effect of the filesystem load variation.

A shuffler then opens the staged file from */dev/shm*, and transmits UDP packets inside the file. After the shuffling is finished, the file is removed from */dev/shm*, so that the stager can stage in a future time step. To control the rate of each UDP stream, we use the Rate Control tool developed for the Visapult project [6]. Rate Control can accurately calibrate the data transmission rate of the UDP stream to the computational horsepower of the CPU core.

The receiver task allocates a region in */dev/shm* upon initialization, which corresponds to the size of the slice. For each UDP packet it receives in the transmitted stream, the receiver decodes the packet and places the particle density values at the proper offset in shared memory. The rendering software spawns 32 processes across all the Receiver/Render servers, each process opens the corresponding data slice from */dev/shm* in read-only mode, and renders the data to produce an image.

For the high bandwidth demo, the rate of the shufflers is set 2.81Gbps, so that the total of 32 streams utilizes 90Gbps of the total bandwidth. For the low bandwidth demo, 4 streams are used, transferring 1/8 of the full data set. The rate of the shufflers is set to 2.5Gbps to utilize 10Gbps of the total bandwidth.

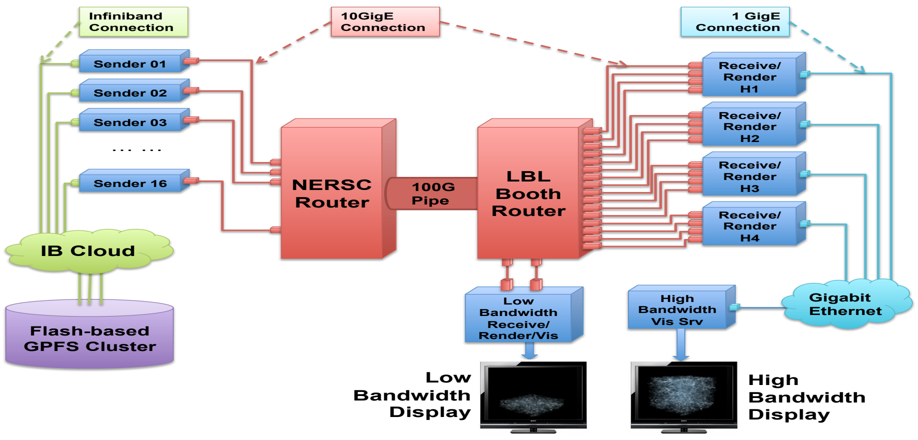

Figure 3: System diagram for the visualization demo at SC11.

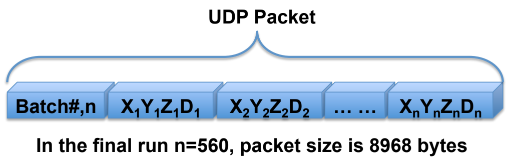

Figure 4: UDP Packet

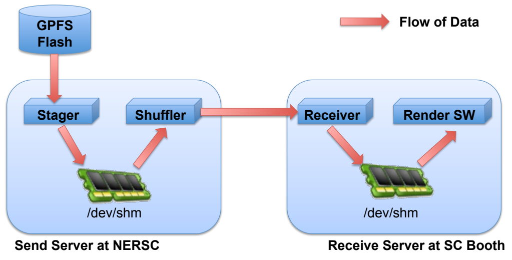

Figure 5: Flow of Data

4.3 Synchronization Strategy

The synchronization is performed at the NERSC end. All shufflers, including 32 for high bandwidth demo and 4 for low band demo, are listening to a UDP port for the synchronization packet. Sent out from a controller running on a NERSC host, the synchronization packets contains the location of the next file to shuffle out. Upon receiving this synchronization packet, a shuffler will stop shuffling the current time step (if it is unfinished), and start shuffling the next time step, until it has shuffled all data in the time step, or receives the next synchronization packet. This mechanism ensures all the shufflers, receivers, and renders are synchronized to the same time step.

We also made an important decision to decouple the streaming tasks from the rendering tasks on each host. The cores responsible for unpacking UDP packets, place the data into a memory-mapped file location. This mmap'ed region is dereferenced in the rendering processes. There is no communication or synchronization between the rendering tasks and streaming tasks on each node.

4.4 Rendering

We used Paraview[13], an open-source, parallel, high performance scientific visualization package for rendering the cosmological dataset. We used a ray-casting based volume rendering technique to produce the images shown in Figure 6. The cubic volume is decomposed in a z-slice order into 4 segments and streamed to individual rendering nodes. Paraview uses 8 cores on each rendering node to produce intermediate images and then composites the image using sort-last rendering over a local 10Gbps network. The final image is displayed on a front-end node connected to a display.

Since the streaming tasks are decoupled from the rendering tasks, Paraview is essentially asked to volume render images as fast as possible in an endless loop. It is possible, and we do observe artifacts in the rendering as the real-time streams deposit data into different regions in memory. In practice, the artifacts are not distracting. We acknowledge that one might want to adopt a different mode of rendering (using pipelining and multiple buffers) to stream data, corresponding to different timesteps, into distinct regions in memory.

4.5 Optimizations

On the 16 sending servers, only 2-3 stagers and 2-3 shufflers are running at any given time; the load is relatively light and no special tuning is necessary to sustain the 2-3 UDP streams (<3Gbps each). On both high bandwidth and low bandwidth receive/render servers, the following optimizations are implemented (as shown in Figure 7):

- Each 10Gbps NIC in the system is bound to a specific core by assigning all the interrupts to that core. For the servers with 4 ports, each NIC is bound to a core in a different NUMA node;
- Two receivers are bound to a 10Gbps NIC by binding the processes to the same core as the port;
- For each stream, the render process is bound to the same NUMA node as the receiver, but to a different core;
- To minimize the NUMA effect, for each stream, the memory region in */dev/shm* is preallocated to make sure it resides in the same NUMA nodes as the receivers and rendering processes.

[13]Paraview http://www.paraview.org

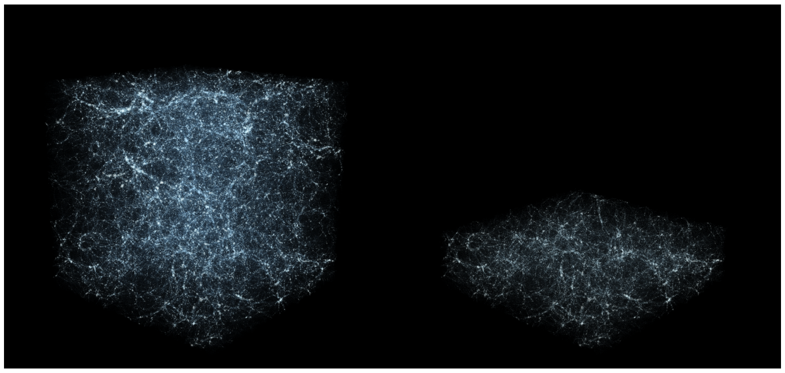
Figure 6: Volume rendering of a timestep from the cosmology dataset. The 90Gbps stream is shown on the left, 10Gbps on the right

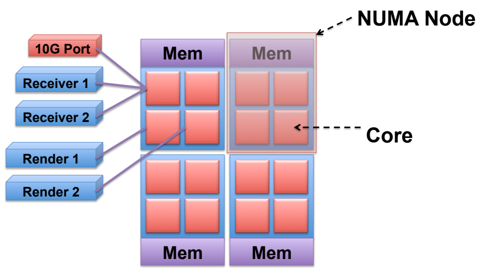

Figure 7: NUMA Binding

We experimented with alternative core-binding strategies, however, the packet loss rate is minimized when binding both the NIC and 2 receivers to the same core. We suspect that this is due to cache misses. For this reason, the main performance limitation is the computational horsepower of the CPU core. A max receiving rate of $\approx$ 2.7Gbps/stream can be reached when one core is handling two receiving streams and all the interrupts from the corresponding NIC port. This causes a $\approx 5\%$ packet loss in the high bandwidth demo, when the shuffling rate is 2.81Gbps/stream. For the low bandwidth demo, the packet loss is smaller than 1 percent.

4.6 Network Performance Results

We streamed 2.3TB of data from NERSC to SC11 show floor in Seattle in $\approx$ 3.4 minutes during live demonstrations at SC11. Each timestep, corresponding to 16GB of data, took $\approx$ 1.4 seconds to reach the rendering hosts at our SC11 booth. The volume rendering took an additional $\approx$ 2.5 seconds before the image was updated. Aggregating across the 90Gbps and 10Gbps demonstrations, we were able to achieve a peak bandwidth utilization of $\approx$ 99Gbps. We observed an average performance of $\approx$ 85Gbps during various time periods at SC11. The bandwidth utilization information was obtained directly from the 100Gbps port statistics on the Alcatel-Lucent router in the LBL booth.

5. CLIMATE DATA OVER 100Gbps

High-bandwidth connections help increase throughput of scientific applications, opening up new opportunities for sharing data that were simply not possible with 10Gbps networks. However, increasing the network bandwidth is not sufficient by itself. Next-generation high-bandwidth networks need to be evaluated carefully from the applications' perspectives. In this section, we explore how climate applications can adapt and benefit from next generation high-bandwidth networks.

Data volume in climate applications is increasing exponentially. For example, the recent "Replica Core Archive" data from the IPCC Fifth Assessment Report (AR5) is expected to be around 2PB [9], whereas, the IPCC Forth Assessment Report (AR4) data archive is only 35TB. This trend can be seen across many areas in science [2, 8]. An important challenge in managing ever increasing data sizes in climate science is the large variance in file sizes [3, 20, 10]. Climate simulation data consists of a mix of relatively small and large files with irregular file size distribution in each dataset. This requires advanced middleware tools to move data efficiently in long-distance high-bandwidth networks. We claim that with such tools, data can be treated as first-class citizen for the entire spectrum of file sizes, without compromising on optimum usage of network-bandwidth.

To justify this claim, we present our experience from the SC11 ANI demonstration, titled 'Scaling the Earth System Grid to 100Gbps Networks'. We used a 100Gbps link connecting National Energy Research Scientific Computing Center (NERSC), Argonne National Laboratory (ANL) and Oak Ridge National Laboratory (ORNL). For this demonstration, we developed a new data streaming tool that provides dynamic data channel management and on-the-fly data pipelines for fast and efficient data access.

The data from IPCC Fourth Assessment Report (AR4) phase 3, CMIP-3, with total size of 35TB, was used in our tests and demonstrations. In general, it took approximately 30 minutes to move CMIP-3 data over 100Gbps. This would have taken around 5 hours over a 10Gbps network, which is the expected 10-times gain in data transfer performance. In the demo, CMIP-3 data was staged successfully into the memory of computing nodes across the country at ANL and ORNL from NERSC data storage over the 100Gbps network on demand.

5.1 Motivation

Climate data is one of the fastest growing scientific data sets. Simulation results are accessed by thousands of users around the world. Many institutions collaborate on the generation and analysis of simulation data. The Earth System Grid Federation[14] (ESGF) [9, 8] provides necessary middleware and software to support end-user data access and data replication between partner institutions. High performance data movement between ESG data nodes is an

[14]Earth System Grid Federation: http://esgf.org/

important challenge, especially between geographically separated data centers.

In this study, we evaluate the movement of bulk data from ESG data nodes, and state the necessary steps to scale-up climate data movement to 100Gbps high-bandwidth networks. As a real-world example, we specifically focus on data access and data distribution for the Coupled Model Intercomparison Project (CMIP) from Intergovernmental Panel on Climate Change (IPCC).

IPCC climate data is stored in common NetCDF data files. Metadata from each file, including the model, type of experiment, and the institution that generated the data file are retrieved and stored when data is published. Data publication is accomplished through an Earth System Grid (ESG) gateway server. Gateways work in a federated manner such that the metadata database is synchronized between each gateway. The ESG system provides an easy-to-use interface to search and locate data files according to given search patterns. Data files are transferred from a remote repository using advanced data transfer tools (e.g., GridFTP [1, 7, 20]) that are optimized for fast data movement. A common use-case is replication of data to achieve redundancy. In addition to replication, data files are copied into temporary storage in HPC centers for post-processing and further climate analysis.

Depending on the characteristics of the experiments and simulations, files may have small sizes such as several hundreds of megabytes, or they can be as large as several gigabytes [9]. IPCC data files are organized in a hierarchical directory structure. Directories are arranged according to experiments, metadata characteristics, organization lists, and simulation models. In addition to having many small files, bulk climate data consists of many directories. This puts extra burden on filesystem access and network transfer protocols. An important challenge in dealing with climate data movement is the lots-of-small-files problem [14, 22, 7]. Most of the end-to-end data transfer tools are designed for moving large data files. State-of-the-art data movement tools require managing each file movement separately. Therefore, dealing with small files imposes extra bookkeeping overhead, especially over high latency networks.

The Globus Project also recognized the performance issues with small files, and added a number of features to their GridFTP tool to address these [7]. This includes an option to do multiple files concurrently (-concurrency), and an option to do pipelining (-pipeline). They also have the -fast option, which reuses the data channel operations. Other similar parallel data mover tools include FDT [17] from Caltech and bbcp from SLAC [12].

5.2 Climate Data Distribution over 100Gbps

Scientific applications for climate analysis are highly data-intensive [5, 2, 9, 8]. A common approach is to stage data sets into local storage, and then run climate applications on the local data files. However, replication comes with its storage cost and requires a management system for coordination and synchronization. 100Gbps networks provide the bandwidth needed to bring large amounts of data quickly on-demand. Creating a local replica beforehand may no longer be necessary. By providing data streaming from remote storage to the compute center where the application runs, we can better utilize available network capacity and bring data into the application in real-time. If we can keep the network pipe full by feeding enough data into the network, we can hide the effect of network latency and improve the overall application performance. Since we will have high-bandwidth access to the data, management and bookkeeping of data blocks would play an important role in order to use remote storage resources efficiently over the network.

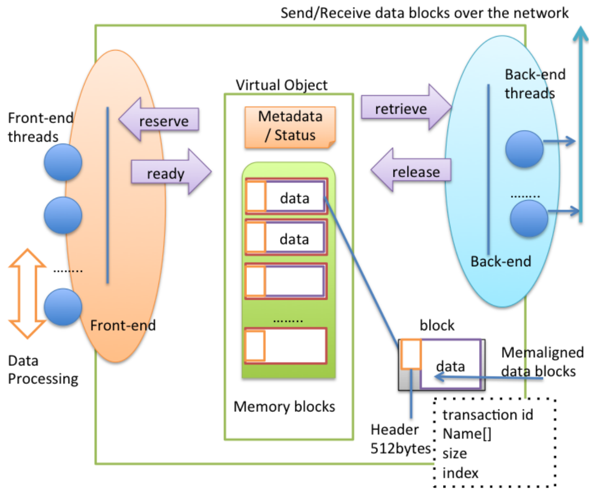

Figure 8: Climate Data Mover Framework

The standard file transfer protocol FTP establishes two network channels [18, 22]. The control channel is used for authentication, authorization, and sending control messages such as what file is to be transferred. The data channel is used for streaming the data to the remote site. In the standard FTP implementation, a separate data channel is established for every file. First, the file request is sent over the control channel, and a data channel is established for streaming the file data. Once the transfer is completed, a control message is sent to notify that end of file is reached. Once acknowledgement for transfer completion is received, another file transfer can be requested. This adds at least three additional round-trip-times over the control channel [7, 22]. The data channel stays idle while waiting for the next transfer command to be issued. In addition, establishing a new data channel for each file increases the latency between each file transfer. The latency between transfers adds up, as a results, overall transfer time increases and total

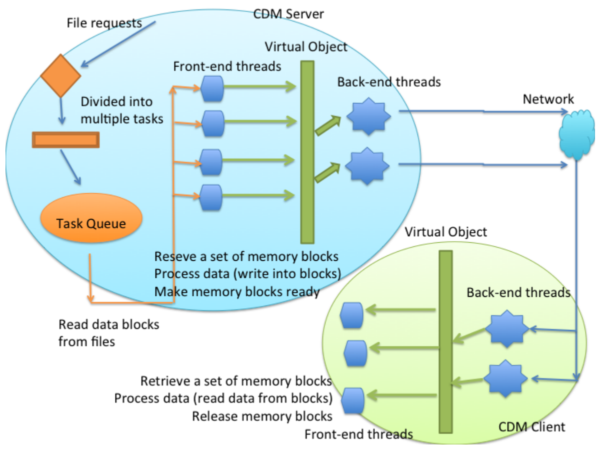

Figure 9: Climate Data Mover Server/Client Architecture

throughput decreases. This problem becomes more drastic for long distance connections where round-trip-time is high.

Keeping the data channel idle also adversely affects the overall performance for window-based protocols such as TCP. The TCP protocol automatically adjusts the window size; the slow-start algorithm increases the window size gradually. When the amount of data sent is small, transfers may not be long enough to allow TCP to fully open its window, so we can not move data at full speed.

On the other hand, data movement requests, both for bulk data replication and data streaming for large-scale data analysis, deal with a set of many files. Instead of moving data from a single file at a time, the data movement middleware could handle the entire data collection. Therefore, we have developed a simple data movement utility, called the Climate Data Mover, that provides dynamic data channel management and block-based data movement. Figure 8 shows the underlying system architecture. Data files are aggregated and divided into simple data blocks. Blocks are tagged and streamed over the network. Each data block's tag includes information about the content inside. For example, regular file transfers can be accomplished by adding the file name and index in the tag header. Since there is no need to keep a separate control channel, it does not get affected by file sizes and small data requests. The Climate Data Mover can be used both for disk-to-disk data replication and also for direct data streaming into climate applications.

5.3 Climate Data Mover

Data movement occurs in two steps. First, data blocks are read into memory buffers (disk I/O). Then memory buffers are transmitted over the network (network I/O). Each step requires CPU and memory resources. A common approach to increase overall throughput is to use parallel streams, so that multiple threads (and CPU cores) work simultaneously to overcome the latency cost generated by disk and memory copy operation in the end system. Another approach is to use concurrent transfers, where multiple transfer tasks cooperate together to generate high throughput data in order to fill the network pipe [24, 4]. In standard file transfer mechanisms, we need more parallelism to overcome the cost of bookkeeping and control messages. An important drawback in using application level tuning (parallel streams and concurrent transfers) is that they cause extra load on the system and resources are not used efficiently. Moreover, the use of many TCP streams may oversubscribe the network and cause performance degradations.

In order to be able to optimally tune the data movement through the system, we decoupled network and disk I/O operations. Transmitting data over the network is logically separated from the reading/writing of data blocks. Hence, we are able to have different parallelism levels in each layer. Our data streaming utility, the *Climate Data Mover*, uses a simple network library consisting of two layers: a front-end and a back-end. Each layer works independently so that we can measure performance and tune each layer separately. Those layers are tied to each other with a block-based virtual object, implemented as a set of shared memory blocks. In the server, the front-end is responsible for the preparation of data, and the back-end is responsible for the sending of data over the network. On the client side, the back-end components receive data blocks and feed the virtual object, so the corresponding front-end can get and process data blocks.

The front-end component requests a contiguous set of memory blocks from the virtual object. Once they are filled with data, those blocks are released, so that the back-end components can retrieve and transmit the blocks over the network. Data blocks in the virtual object include content information, i.e., file id, offset and size.

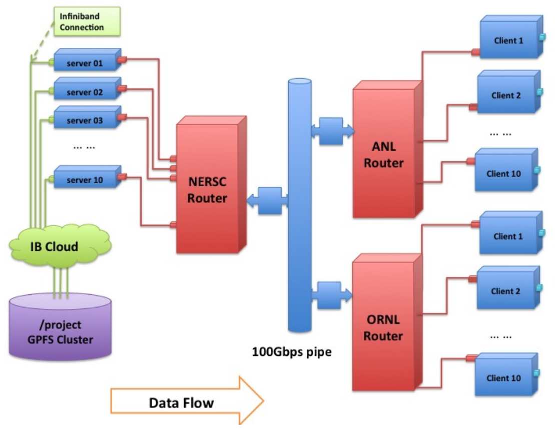

Figure 10: System diagram for the demo at SC11.

Therefore, there is no need for further communication between client and server in order to initiate file transfers. This is similar to having an on-the-fly 'tar' approach bundling and sending many files together. Moreover, by using our tool, data blocks can be received and sent out-of-order and asynchronously. Figure 9 shows client/server architecture for data movement over the network. Since we do not use a control channel for bookkeeping, all communication is mainly over a single data channel, over a fixed port. Bookkeeping information is embedded inside each block. This has some benefits for ease of firewall traversal over wide-area [15].

In our test case, we transfer data files from the NERSC GPFS filesystem into the memory of ALCF and OLCF nodes. The Climate Data Mover server initiates multiple front-end and back-end threads. The front-end component reads data, attaches a file name and index information, and releases blocks to be sent to the client. The client at the remote site receives data blocks and makes them ready to be processed by the corresponding front-end threads. For a disk-to-disk transfer, the client's front-end can simply call file write operations. The virtual object also acts as a large cache. For disk to memory, the front-end keeps the data blocks and releases them once they are processed by the application. The main advantage with this approach is that we are not limited by the characteristics of the file sizes in the dataset. Another advantage over FTP-based tools is that we can dynamically increase/descrease the parallelism level both in the network communication and I/O read/write operations, without closing and reopening the data channel connection (as is done in regular FTP variants).

5.4 Test Results

Figure 10 represents the overall system details for the SC11 demo. We used 10 host pairs, each connected to the network with a 10 Gbps link. We used TCP connection between host pairs; default settings have been used so that TCP window size is not set specifically. We have tested the network performance between NERSC and ANL/ORNL with various parameters, such as, total size of virtual object, thread count for reading from GPFS filesystem, and multiple TCP streams to increase the utilization of the available bandwidth. According to our test results, we have manually determined best set of parameters for the setup. Specific host tuning issues about IRQ binding and interrupt coalescing described in the following section have not been applied, and are open to future explorations.

Our experiment moved data stored at NERSC to application nodes at both ANL and ORNL. We staged the Coupled Model

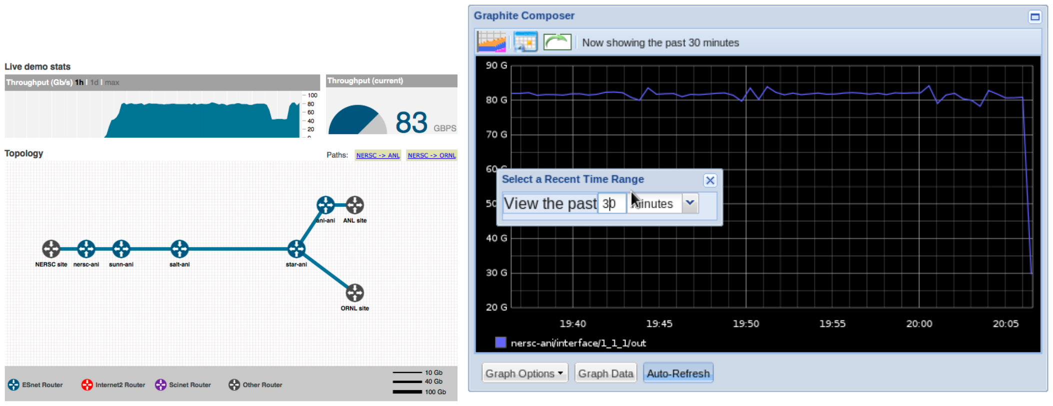

Figure 11: SC11 Climate100 demonstration results, showing the data transfer throughput

Intercomparison Project (CMIP) data set from Intergovernmental Panel on Climate Change (IPCC) from the GPFS filesystem at NERSC. A total 35TB was transferred in about 30 minutes. The default filesystem block size was set to 4MB, so we also used 4MB blocks in Climate Data Mover for better read performance. Each block's data section was aligned according to the system pagesize. Total size of the virtual object was 1GB both at the client and the server applications. The servers at NERSC used eight front-end threads on each host for reading data files in parallel. The clients used four front-end threads for processing received data blocks. In the demo, four parallel TCP streams (four back-end threads) were used for each host-to-host connection. We observed 83 Gbps total throughput both using both NERSC to ANL and NERSC to ORNL, as shown in Figure 11.

5.5 Performance of Climate Data Mover

The Climate Data Mover is able to handle both small and large files with high performance. Although a detailed analysis of it is beyond the scope of this paper, we present a brief comparison with GridFTP, for evaluating performance of the Climate Data Mover in transferring small files. Two hosts, one at NERSC and one at ANL, were used from the ANI 100Gbps Testbed. Each host is connected with four 10Gbps NICs. Total throughput is 40Gbps between two hosts. We did not have access to a high performance file system; therefore, we simulate the effect of file sizes by creating a memory file system (*tmpfs* with a size of 20G). We created files with various sizes (i.e., 10M, 100M, 1G) and transferred those files continuously while measuring the performance. In both Climate Data Mover and GridFTP experiments, TCP buffer size is set to 50MB in order to get best throughput. The pipelining feature was enabled in GridFTP. A long file list (repeating file names that are in the memory filesystem) is given as input. Figure 12 shows performance results with 10MB files. We initiated four server applications at ANL node (each running on a separate NIC), and four client applications at NERSC node. In the GridFTP tests, we tried both 16 and 32 concurrent streams (-cc option). The Climate Data Mover was able to achieve 37Gbps of throughput, while GridFTP was not able achieve more than 33Gbps.

6. HOST TUNING ISSUES

Optimal utilization of the network bandwidth on modern linux hosts requires a fair amount of tuning. There are several studies on

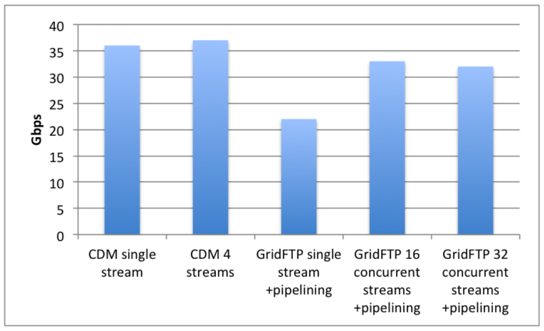

Figure 12: GridFTP vs. Climate Data Mover (CDM)

network performance optimization in 10Gbps networks [21, 23]. However, only a few recent studies have tested high-speed data transfers in a 100Gbps environment. One of these is the team at Indiana University's testing of the Lustre filesystem over the 100Gbps network at SC11 [13]. Other recent studies include presentations by Rao [19] and by the team at Nasa Goddard [11]. They all have found that a great deal of tuning was required. In this section we specifically discuss modifications that helped us increase the total NIC throughput. These additional host tuning tests were done after the SC11 conference on ESnet's ANI 100GbpsTestbed, shown in Figure 13.

We conducted these series of experiments on three hosts at NERSC connected with a 100Gbps link to three hosts at ANL. After adjusting the tuning knobs, we were able to effectively fill the

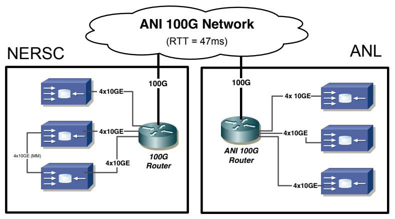

Figure 13: ANI 100Gbps Testbed Configuration used for Host Tuning Experiments

100Gbps link with only 10 TCP sockets, one per 10Gbps NIC. We used four 10GE NICS in two of the hosts, and two (out of 4) 10GE NICS in the third host, as shown in Figure 13. The results of the two applications described in this paper used some, but not all of these techniques, as they both used more than 10 hosts on each end instead of just 3, thereby requiring fewer tuning optimizations. Using all of the tuning optimizations described below, we were able to achieve a total of 60 Gbps throughput (30 Gbps in each direction) on a single host with four 10GE NICS. 60 Gbps is the limit of the PCI bus. We were also able to achieve 94 Gbps TCP and 97 Gbps UDP throughput in one direction using just 10 sockets. UDP is CPU bound on the send host, indicating that just under 10Gbps per flow is the UDP limit of today's CPU cores regardless of NIC speed.

6.1 TCP and UDP Tuning

We observed a latency of 23 ms on the path from NERSC to ANL each way, therefore the TCP window size needed to be increased to hold the entire bandwidth delay product of 64 MB, as described on *fasterdata* [15] For UDP, the optimal packet size to use is 8972 bytes (path MTU minus the IP and UDP header). For UDP, it was also important to increase SO_SNDBUF and SO_RCVBUF using the *setsockopt* system call to 4MB in order to saturate a 10GE NIC. Since these tests were conducted on a dedicated network with no competing traffic or congestion, there was no reason to use multiple TCP flows per NIC; a single flow was able to fill the 10G pipe.

6.2 IRQ and Thread Binding

With modern multi-socket multi-core architectures, there is a large performance penalty if the NIC interrupts are being handled on one processor, while the user read process is on a different processor, as data will need to be copied between processors. We solved this problem using the following approach. First, we disabled *irqbalance*. *irqbalance* distributes interrupts across CPUs to optimize for L2 cache hits. While this optimization is desirable in general, it does not maximize performance in this test configuration. Therefore we explicitly mapped the interrupt for each NIC to a separate processor core, as described on *fasterdata*, [16] and used the *sched_setaffinity* system call to bind the network send or receive process to that same core. For a single flow, *irqbalance* does slightly better than the static binding, but, for four streams, *irqbalance* performs much worse than our static binding. This is because as the overall CPU load goes up, the IRQ distribution policy becomes less likely to provide the right binding to the ethernet interrupts, leading to the host dropping packets and TCP backing off. Figure 14 shows the results for TCP and UDP performance of 4 streams using irqbalance compared to manual IRQ binding. There is an 20 percent improvement for TCP, and 38 percent for UDP.

6.3 NIC Tuning

We used two well-known techniques to reduce the number of interrupts to improve performance: Jumbo frames (9000 byte MTUs), and interrupt coalescing. We also increased *txqueuelen* to 10000. Figure 14 shows that enabling interrupt coalescing and setting it to 100 milliseconds provides a 93 percent performance gain for UDP and 53 percent for TCP.

6.4 BIOS Tuning

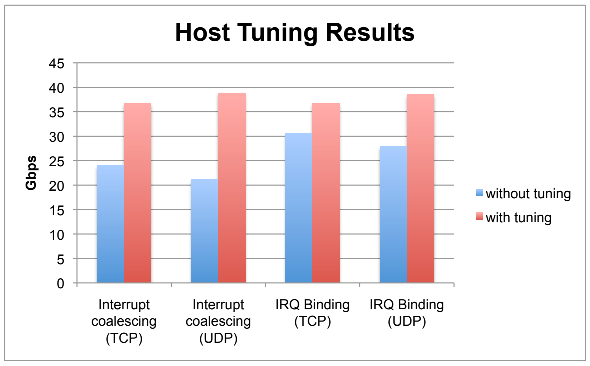

Figure 14: Host Tuning Results

It is important to verify that a number of BIOS settings are properly set to obtain maximum performance. We modified the following settings in the course of our experiments:

- **hyper-threading:** We disabled hyper-threading. While it simulates more cores than are physically present, this can reduce performance under variable load conditions
- **memory speed:** The default BIOS setting did not set the memory bus speed to the maximum.
- **cpuspeed:** The default BIOS setting did not set the CPU speed to the maximum.
- **energy saving:** We disabled this to ensure the CPU was always running at the maximum speed.

6.5 Open Issues

This paper has shown that one needs to perform a lot of hand-tuning in order to saturate a 100Gbps networks. We need to optimize CPU utilization in order to achieve higher networking performance. The Linux operating system provides a service, *irqbalance*, but as this paper has shown, its generic algorithm fails under certain workloads. Statically binding IRQs and send/receive threads to a dedicated core is a simple solutions that works well in a well defined, predictable, environment, but quickly becomes difficult to manage in a production context where many different applications may have to be deployed.

PCI Gen3 motherboards are just becoming available, allowing up to 8Gbps per lane. These motherboards will allow a single NIC to theoretically go up to 64Gbps. In this environment more parallel flows will be needed, as current CPU speed limits UDP flows to around 10Gbps, and TCP flows to around 18Gbps. This problem will be further exacerbated when PCIe Gen4, slated for 2015, will further increase PCI bandwidth.

Last but not least, the various experiments reported in the paper were running in a closed network with no competing traffic. More thorough testing will be needed to identify and address issues in a production network.

7. CONCLUSION

100Gbps networks have arrived, and with careful application design and host tuning, a relatively small number of hosts can fill a 100Gbps pipe. Many of the host tuning techniques from 1Gbps to 10Gbps transition still apply. These include TCP/UDP buffer tuning, using jumbo frames, and using interrupt coalescing. With the current generation of multi-core systems, IRQ binding also is now essential for maximizing host performance.

[15] http://fasterdata.es.net/host-tuning/background/
[16] http://fasterdata.es.net/host-tuning/interrupt-binding/

While application of these tuning techniques will likely improve the overall throughput of the system, it is important to follow an experimental methodology in order to systematically increase performance. In some cases a particular tuning strategy may not achieve the expected results. We recommend starting with the simplest core I/O operation possible, and then adding layers of complexity on top of that. For example, one can first tune the system for a single network flow using a simple memory to memory transfer tool such as Iperf[17] or nuttcp[18]. Next optimize the multiple concurrent streams, trying to model the application behavior as closely as possible. Once this is done, the final step is to tune the application itself. The main goal of this methodology is to verify performance for each component in the critical path. Applications may need to be redesigned to get the best out of high-bandwidth networks. In this paper, we demonstrated two such applications to take advantage of this network, and described a number of application design issues and host tuning strategies necessary for enabling those applications to scale to 100Gbps.

Acknowledgments

We would like to thank Peter Nugent and Zarija Lukic for providing us with the cosmology datasets used in the visualization demo. Our thanks go out to Patrick Dorn, Evangelos Chaniotakis, John Christman, Chin Guok, Chris Tracy and Lauren Rotman for assistance with 100Gbps installation and testing at SC11. Jason Lee, Shane Canon, Tina Declerck and Cary Whitney provided technical support with NERSC hardware. Ed Holohan, Adam Scovel, and Linda Winkler provided support at ALCF. Jason Hill, Doug Fuller, and Susan Hicks provided support at OLCF. Hank Childs, Mark Howison and Aaron Thomas assisted with troubleshooting visualization software and hardware.

This work was supported by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the ESnet Advanced Network Initiative (ANI) Testbed, which is supported by the Office of Science of the U.S. Department of Energy under the contract above, funded through the The American Recovery and Reinvestment Act of 2009.

References

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The globus striped gridftp framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 54–, Washington, DC, USA, 2005. IEEE Computer Society.

[2] M. Balman and S. Byna. Open problems in network-aware data management in exa-scale computing and terabit networking era. In *Proceedings of the first international workshop on Network-aware data management*, NDM '11, pages 73–78, 2011.

[3] M. Balman and T. Kosar. Data scheduling for large scale distributed applications. In *Proceedings of the 5th ICEIS Doctoral Consortium, in conjunction with the International Conference on Enterprise Information Systems (ICEIS'07)*, 2007.

[4] M. Balman and T. Kosar. Dynamic adaptation of parallelism level in data transfer scheduling. *Complex, Intelligent and Software Intensive Systems, International Conference*, 0:872–877, 2009.

[5] BES Science Network Requirements, Report of the Basic Energy Sciences Network Requirements Workshop. Basic Energy Sciences Program Office, DOE Office of Science and the Energy Sciences Network, 2007.

[6] E. W. Bethel. Visapult – A Prototype Remote and Distributed Application and Framework. In *Proceedings of Siggraph 2000 – Applications and Sketches*. ACM/Siggraph, July 2000.

[7] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster. Gridftp pipelining. In *Proceedings of the 2007 TeraGrid Conference*, June 2007.

[8] D. N. Williams et al. Data Management and Analysis for the Earth System Grid. Journal of Physics: Conference Series, SciDAC 08 conference proceedings, volume 125 012072, 2008.

[9] D. N. Williams et al. Earth System Grid Federation: Infrastructure to Support Climate Science Analysis as an International Collaboration. Data Intensive Science Series: Chapman & Hall/CRC Computational Science, ISBN 9781439881392, 2012.

[10] S. Doraimani and A. Iamnitchi. File grouping for scientific data management: lessons from experimenting with real traces. In *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC '08, pages 153–164, 2008.

[11] P. Gary, B. Find, and P. Lang. Introduction to GSFC High End Computing: 20, 40 and 100 Gbps Network Testbeds. `http://science.gsfc.nasa.gov/606.1/docs/HECN_10G_Testbeds_082210.pdf`, 2010.

[12] A. Hanushevsky, A. Trunov, and L. Cottrell. Peer-to-peer computing for secure high performance data copying. In *Proceedings of computing in high energy and nuclear physics*, September 2001.

[13] IU showcases innovative approach to networking at SC11 SCinet Research Sandbox. `http://ovpitnews.iu.edu/news/page/normal/20445.html`, 2011.

[14] R. Kettimuthu, S. Link, J. Bresnahan, M. Link, and I. Foster. Globus xio pipe open driver: enabling gridftp to leverage standard unix tools. In *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, TG '11, pages 20:1–20:7. ACM, 2011.

[15] R. Kettimuthu, R. Schuler, D. Keator, M. Feller, D. Wei, M. Link, J. Bresnahan, L. Liming, J. Ames, A. Chervenak, I. Foster, and C. Kesselman. A Data Management Framework for Distributed Biomedical Research Environments. e-Science Workshops, 2010 Sixth IEEE International Conference on, 2011.

[16] Magellan Report On Cloud Computing for Science. `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Magellan_Final_Report.pdf`, 2011.

[17] Z. Maxa, B. Ahmed, D. Kcira, I. Legrand, A. Mughal, M. Thomas, and R. Voicu. Powering physics data transfers with fdt. *Journal of Physics: Conference Series*, 331(5):052014, 2011.

[18] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), Oct. 1985. Updated by RFCs 2228, 2640, 2773, 3659, 5797.

[19] N. Rao and S. Poole. DOE UltraScience Net: High-Performance Experimental Network Research Testbed. `http://computing.ornl.gov/SC11/documents/Rao_UltraSciNet_SC11.pdf`, 2011.

[20] A. Sim, M. Balman, D. Williams, A. Shoshani, and V. Natarajan. Adaptive transfer adjustment in efficient bulk data transfer management for climate dataset. In Parallel and Distributed Computing and Systems, 2010.

[21] T. Yoshino et al. Performance Optimization of TCP/IP over 10 gigabit Ethernet by Precise Instrumentation. Proceedings of the ACM/IEEE conference on Supercomputing, 2008.

[22] D. Thain and C. Moretti. Efficient access to many samall files in a filesystem for grid computing. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 243–250, Washington, DC, USA, 2007. IEEE Computer Society.

[23] W. Wu, P. Demar, and M. Crawford. Sorting reordered packets with interrupt coalescing. *Comput. Netw.*, 53:2646–2662, October 2009.

[24] E. Yildirim, M. Balman, and T. Kosar. Dynamically tuning level of parallelism in wide area data transfers. In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, DADC '08, pages 39–48. ACM, 2008.

[17]iperf: http://iperf.sourceforge.net/
[18]nuttcp: http://www.nuttcp.net