Analyzing Data Movements and Identifying Techniques for Next-generation High-bandwidth Networks

Mehmet Balman

mbalman@lbl.gov
Computation Research Division
Lawrence Berkeley National Laboratory
1 Cyclotron Road, Berkeley CA 94720

2012

High-bandwidth networks are poised to provide new opportunities in tackling large data challenges in today's scientific applications. However, increasing the bandwidth is not sufficient by itself; we need careful evaluation of future high-bandwidth networks from the applications' perspective. We have investigated data transfer requirements of climate applications as a typical scientific example and evaluated how the scientific community can benefit from next generation high-bandwidth networks. We have experimented with current state-of-the-art data movement tools, and realized that there is no single solution for presetting transfer parameters for optimal use of the available bandwidth. Thus, we developed an adaptive transfer methodology for tuning and optimization in wide-area data transfers. This worked well with large files. However, typical scientific datasets may include many small files. Current file-centric data transfer protocols do not perform well with managing the transfer of small files, even when using parallel streams or concurrent transfers over high bandwidth networks. In order to overcome this problem, we develop a new block-based data movement method (in contrast to the current file-based methods) to improve data movement performance and efficiency in moving large scientific datasets that contain many small files. We implemented the new block-based data movement tool, which takes the approach of aggregating files into blocks and providing dynamic data channel management.

In our work, we also realized that one of the major obstacles in use of high-bandwidth networks is the limitation in host system resources. 100Gbps is beyond the capacity of today's commodity machine, since we need substantial amount of processing power and involvement of multiple cores to fill a 40Gbps or 100Gbps network. As a result, host system performance plays an important role in the use of high-bandwidth networks. We have conducted a large number of experiments with our new block-based method and with current available file-based data movement tools. In this white paper, we describe future research problems and challenges for efficient use of next-generation science networks, based on the lessons learnt and the experiences gained with 100Gbps network applications.

---

1. Typical Problems: Experience with Climate Data

Climate data is one of the fastest growing scientific datasets, and it is shared/distributed among many research institutions around the world. As a result, the climate science community requires high-performance data movement. This application area, which has a mixture of some large files and many small files, presents a typical load scenario. Towards that goal, we have examined the ESG (Earth System Grid) infrastructure, and experimented with various cases of data movement scenarios that are common in the ESG environment. In addition to bulk transfers of climate datasets, we have also studied climate analysis application accessing a remote data repository.

Large bandwidth provided by today's networks requires careful protocol tuning and most importantly optimization of applications to efficiently use capacity of end-systems. One common approach is to use parallel streams, so multiple threads can work simultaneously to overcome the latency cost in the network, and provide fast disk access inside the end system. Another approach is to use concurrent transfers in which multiple transfer jobs cooperate to generate high throughput data transfer in order to fill the network pipe. In both cases, concurrent transfers and parallel streams, multiple sockets are used simultaneously. We have observed that the use of many TCP sockets oversubscribes the network and causes performance degradations. Another important drawback in increasing application level parallelism is that the end system resources are over provisioned. Application level parallelism results in unnecessary CPU load, contention in memory access, increase in context switches, and as a result starvation in system resources. Therefore, it is important to understand the effects these tuning parameters and find alternatives that minimize these effects in order to obtain high throughput data transfers.

In our evaluation of transfer optimization in wide-area 10Gbps networks, we found there is no single solution to find the optimum number of streams. To achieve high throughput, the number of multiple connections needs to be adjusted according to the latency and the available capacity of the underlying network environment. Predicting the network behavior and finding a good estimation for the level of parallelism usually rely on system probes and external measurements. Further, network conditions may change over time in a shared environment, and the estimated value may not reflect the most recent state of the system. Since network is dynamic and shared, we concluded that an adaptive approach is very suitable for wide-area data movements. In this approach, we gradually adjust transfer parameters for best achievable performance. As a result of this work, we have designed and implemented an adaptive transfer adjustment methodology to dynamically set optimum number of parallel streams. However, this approach works best with large files. When there are many files and those files are not large enough to take advantage of parallel streams, we need to transfer multiple files concurrently. Concurrent transfers help minimize effects of protocol overhead and overlap waiting times in data channels, but in this case, we usually use more than optimum number of streams to fill the network pipe. Furthermore, we increase the number of concurrent transfers to overcome filesystem overheads in reading and bookkeeping many files. The number of parallel operations increases unnecessarily, and as a result, this situation causes degradations in the overall performance. For that reason, real-life performance of transferring real datasets is far below than what we have observed in test results with large files in similar network configurations.

2. Moving Datasets with Many Files Efficiently

As mentioned above, an important challenge in dealing with climate data movement is the lots-of-small-files problem. Each climate dataset includes many files, relatively small in size. In addition to network optimization, data transfers require appropriate middleware for managing and transferring a large number of small files efficiently.

The standard file transfer protocol FTP establishes two network channels. 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. 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, and as a result, the overall transfer time increases and total throughput decreases. This problem becomes more drastic for long distance connections where round-trip-time is high.

Most of the end-to-end data transfer tools perform best with large data files. The state-of-the-art data movement tools require managing each file movement separately. As a result, dealing with many 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 such as concurrent transfers, pipelining, and connection caching to their GridFTP tool to address this issue. When there are many files, multiple files are transferred concurrently in order to overlap waiting times in data channels and minimize effects of the bookkeeping overhead. However, many concurrent transfers impose extra cost in terms of system and network resources, and as explained before, may results in poor performance.

We developed a new approach, in which data movement is not file-centric, but is block-based. The idea is to combine files into a single stream on the fly. We have implemented a new block-based data movement tool based on this approach, and compared it with GridFTP in the ANI 100Gbps demo at SC11. In this tool, 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. While testing, we achieved the same performance irrespective of file sizes, without compromising on the optimum usage of the available network-bandwidth.

In the ANI (Advanced Network Initiative) 100Gbps demo at SC11 (Supercomputing Conference, Nov. 2011), we used real data from the Coupled Model Intercomparison Project (CMIP). Since it includes many files, and the files sizes are relatively small compared to available bandwidth (e.g., 100MB for 10Gbps), we needed more than 16 concurrent transfers in GridFTP to reduce the effect of having small file sizes. We have observed better performance and efficiency with our new tool in transferring large datasets especially with many files.

Our first intention while designing this tool was to decouple disk and network IO operations; so, read/write threads can work independently. We wanted to have different parallelism levels in filesystem access and in network transfer operations. In order to do that, we have developed a memory cache managements system that is accessed in blocks. Data is read directly into these memory blocks. These memory blocks are logically mapped to the memory cache that resides in the remote site. It is responsibility of back-end threads to transmit blocks over the network. The memory cache is simply a circular buffer; and, its synchronization is accomplished based on the tag header that also includes a transaction id. Main concept in this design is that application processes interact with the memory blocks, and they do not deal with the network layer. The memory-mapped network access also provides the necessary architecture for zero-copy network operations. More details on the desirability of zero-copy protocols are given in Appendix A. Although our current implementation uses regular TCP sockets at the back-end, we plan to enhance this tool and add zero-copy capability in the future. In principle, our method can easily support zero-copy because it can avoid copying from application memory to kernel memory. We call this tool MemzNet, Memory-mapped Zero-copy Network channel. More details on MemzNet's architecture are given in Section 4.

The following figure from the SC11 demo environment provides a glimpse into the performance of our new tool. Figure 1 shows early test results with GridFTP, and with our tool. The second part in the graph is with our new tool, and it gives steady performance. The first part is with GridFTP (using concurrent transfers); where the throughput value fluctuates over time. We used our tool in the SC11 demo and we were able to get 83Gbps total throughput in the ANI 100Gbps connection from NERSC (National Energy Research Scientific Computing Center) to ANL (Argonne National Laboratory). The demo configuration consisted of multiple hosts at each end, and each host has only one 10Gbps NIC

(Network Interface Card) connected to the network. The maximum achievable throughput in this environment (with memory to memory iperf transfers) was also 83Gbps. In our demo, data files were read from a GPFS system, which could provide 120Gbps read performance.

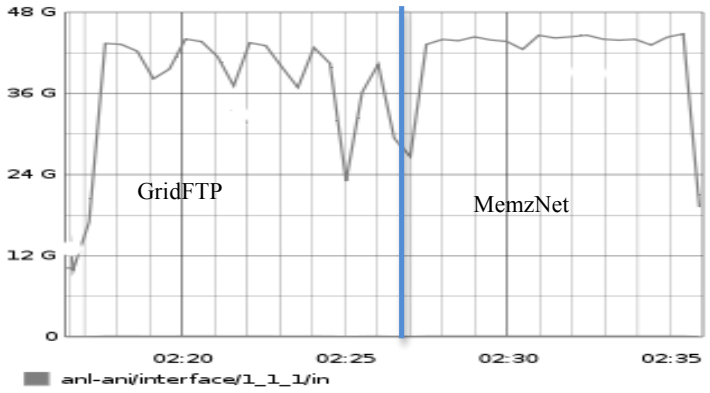

Figure1: GridFTP vs memzNet

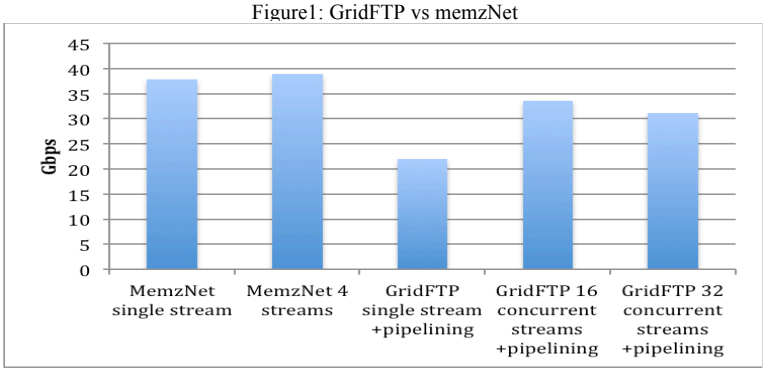

Figure 2: Throughput comparison: GridFTP vs MemzNet

After the SC11 demo, we have continued our evaluation in the 100Gbps ANI testbed. In Figure2, 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 in the ANI 100Gbps test-bed; 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 MemzNet 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 2 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). Our new tool was able to achieve 37Gbps of throughput, while GridFTP was not able achieve more than 33Gbps.

3. Performance Effects on the Servers

For the SC11 demo, we have developed a simple performance module embedded in our tool to interactively monitor the amount of data sent/received over the network. In the demo configuration, each host is connected with a 10Gbps NIC. An important problem we have observed was that some hosts were able to have almost 10Gbps throughput, but some hosts were only able to achieve less than half of that and their instant throughput usage fluctuates a lot. In a normal case, total bandwidth should be distributed equally among each connection. This unusual situation became more apparent when the number of network streams and the number of hosts connecting were increased. With UDP we were able to saturate the network but still seeing many packages lost. There was a big variance in bandwidth usage among

different hosts in TCP. We also observed an increase in the number SACK recovery in the hosts in which throughput usage fluctuates a lot. This problem has been resolved when the 100Gbps aggregator switch, a Juniper 4500, has been removed from the configuration. Balancing the load across multiple 10G circuits was a problem. We think that the aggregator switch was delaying some packages and that situation was causing problems when the host network stack is trying to adjust TCP data flow. Later, we did not have a chance to make further experiments in the SC11 demo systems. But, this first hands-on experience with a 100Gbps network, lead us to investigate further and analyze end-to-end data movements with more details. After that, there have been several updates/fixes in the network configuration based on this experience. We continued our experiments in the ANI 100Gbps testbed where we could finally get 100Gbps throughput. This prompted us to look into network performance issues and the effects of host systems' resource utilization.

We continued our performance analysis in the ANI testbed environment; we have three hosts at each end, and each host has four 10Gbps network cards. In order to achieve100Gbps throughput, we needed fine host tuning. We have observed that the host system performance can easily be the bottleneck. This prompted us to look into the host systems resource utilization.

First, we have studied application profiling including memory usage, number of context switches, time spent waiting for I/O completion, user and system time, call graph of system calls, time spent in each user operation. After several experiments, we concluded that collecting performance metrics values directly from host system gives better and more useful results. We have collected monitoring data from Linux subsystems (CPU, memory, IO) in real time, when data movement operations are performed. Our initial evaluation is that multiple NICs and concurrent transfers increase the context switch time, system CPU usage, number of interrupts and system load, both in the server and client sides for an end-to-end data movement.

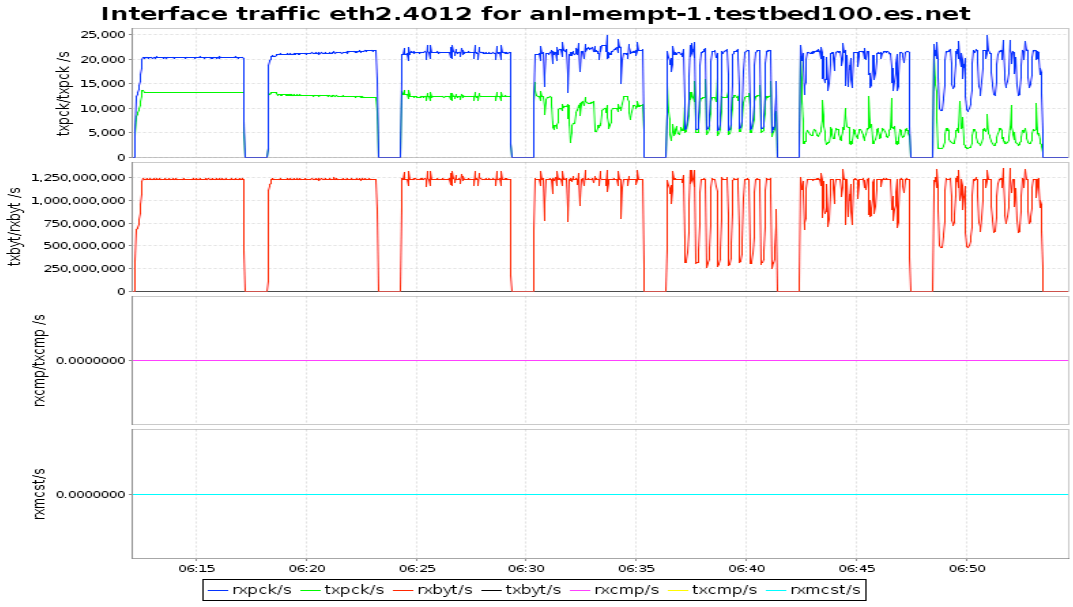

Figure 3: ANI testbed 10Gbps connection: Interface traffic vs the number of parallel streams [1, 2, 4, 8, 16, 32 64 streams -5min intervals], TCP buffer size is set to 50MB

In our experiments, we start the data transfer operation with different parameters and let it run for 5 minutes. While transfer is active, we collect system wide performance metrics and compare those results with the total throughput achieved. Performance metrics are collected both in the client and server sides. We test the effect of parallel streams for memory-to-memory transfers. In concurrent transfers, we start multiple transfer applications at the same time with single stream each. Also, we set the TCP buffer manually (we observed best performance with 50M in the testbed environment) to see how it affects throughput and system wide performance metrics.

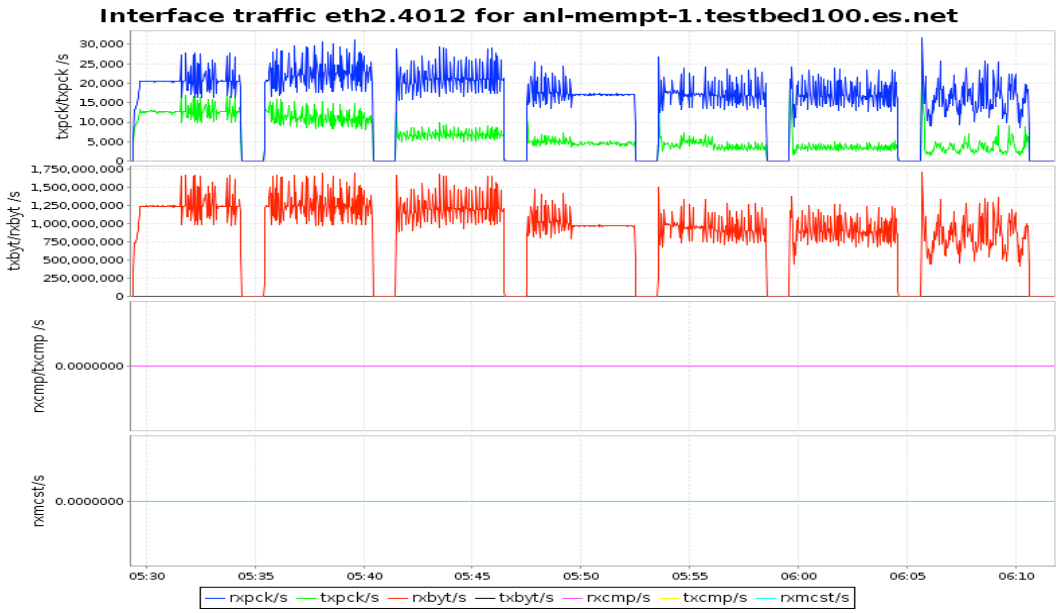

Figure 4: ANI testbed 40Gbps (4x10NICs, single host): Interface traffic vs the number of parallel streams [1, 2, 4, 8, 16, 32 64 streams - 5min intervals], TCP buffer size is set to 50MB

Furthermore, we simulate the affect of having many files by creating a memory file system (tmpfs with a size of 20G) in each host. We created files with various sizes (i.e., 10M, 100M, 1G) and transferred those files continuously while measuring the performance. At present, those tests are intended to compare performance of our new tool with GridFTP. We believe that we developed a useful methodology to analyze host system performance issues in data movement operations. We would like to extend our experiments; our current tests include only GridFTP and MemzNet transfers.

4. Measurements in the ANI 100Gbps Testbed

Figure 3 shows the interface traffic, packages per second (blue) and bytes per second, when a single NIC (eth2 in this case) at each end is used. The GridFTP is used for memory-to-memory (/dev/zero to /dev/null) transfers over a 10Gbps connection. We set the number of parallel streams, and run the data transfer operation for 5 minutes. Each peak in these graphs represents a different test with different number of parallel streams. We tested transfers with 1, 2, 4, 8, 16, 32, 64 parallel streams.

Figure 4 shows interface traffic when all four NICs were used. The maximum achievable bandwidth was 40Gbps in this test. As we increased the number of parallel streams, we observed that the total throughput dropped, as can be seen in Figure 5.

Figure 6 shows the total throughput when three hosts and a total of 10 NIC pairs were used. The maximum available bandwidth was 100Gbps. The maximum throughput was achieved with four streams. Even when TCP buffer was set properly, we observed that performance dropped after 4 parallel streams, as shown in Figure 7.

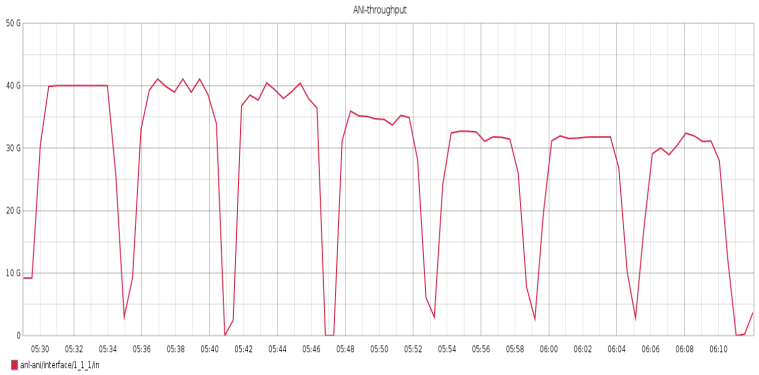

Figure 5: ANI testbed 40Gbps (4x10NICs, single host): Throughput vs the number of parallel streams [1, 2, 4, 8, 16, 32 64 streams - 5min intervals], TCP buffer size is set to 50MB

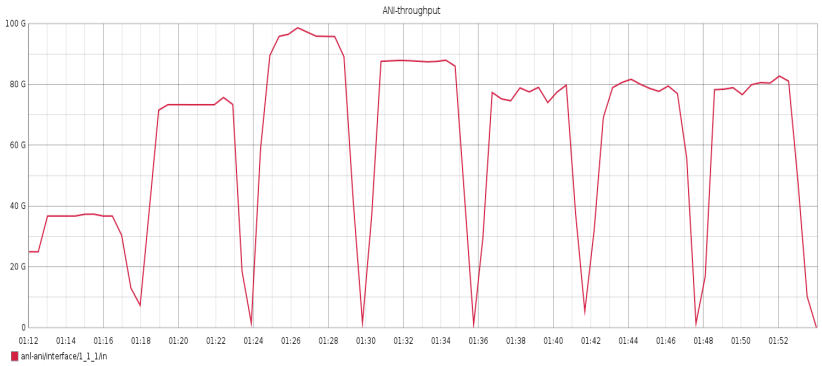

Figure 6: ANI testbed 100Gbps (10x10NICs, three hosts): Throughput vs the number of parallel streams [1, 2, 4, 8, 16, 32 64 streams - 5min intervals], TCP buffer size is default

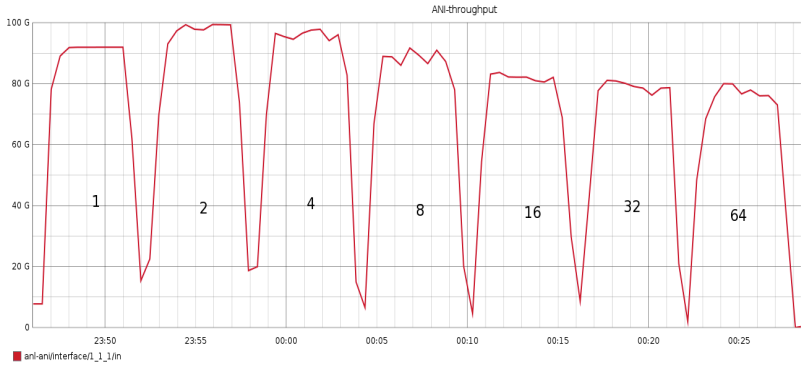

Figure 7: ANI testbed 100Gbps (10x10NICs, three hosts): Throughput vs the number of parallel streams [1, 2, 4, 8, 16, 32 64 streams - 5min intervals], TCP buffer size is 50M

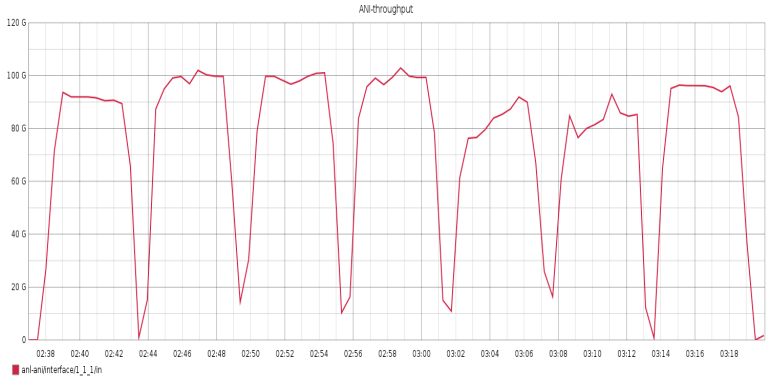

Figure 8: ANI testbed 100Gbps (10x10NICs, three hosts): Throughput vs the number of concurrent transfers [1, 2, 4, 8, 16, 32 64 concurrent jobs - 5min intervals], TCP buffer size is 50M

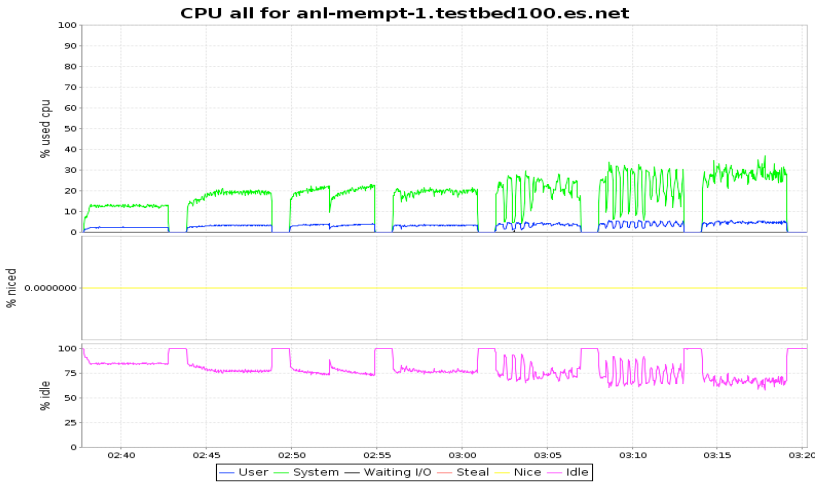

Figure 9: ANI testbed 100Gbps (10x10NICs, three hosts): CPU utilization vs the number of concurrent transfers [1, 2, 4, 8, 16, 32 64 concurrent jobs - 5min intervals], TCP buffer size is 50M

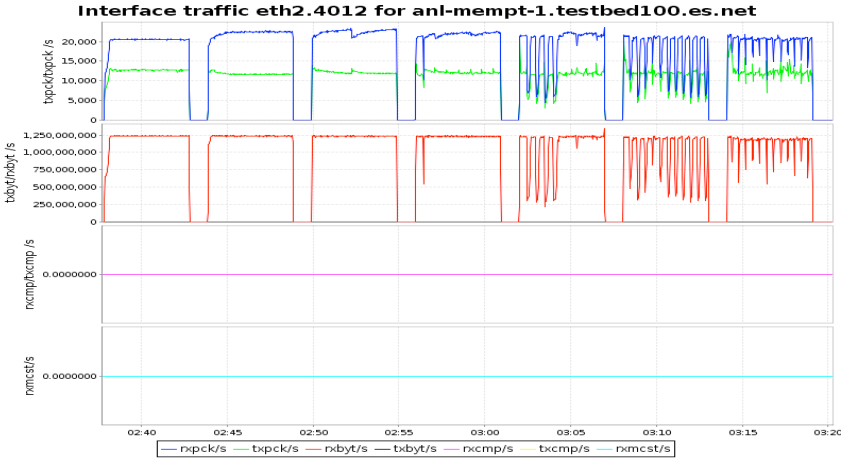

Figure 10: ANI testbed 100Gbps (10x10NICs, three hosts): Interface traffic vs the number of concurrent transfers [1, 2, 4, 8, 16, 32 64 concurrent jobs - 5min intervals], TCP buffer size is 50M

Figure 11: ANI testbed 100Gbps (10x10NICs, three hosts): Interrupts vs the number of concurrent transfers [1, 2, 4, 8, 16, 32 64 concurrent jobs - 5min intervals], TCP buffer size is 50M

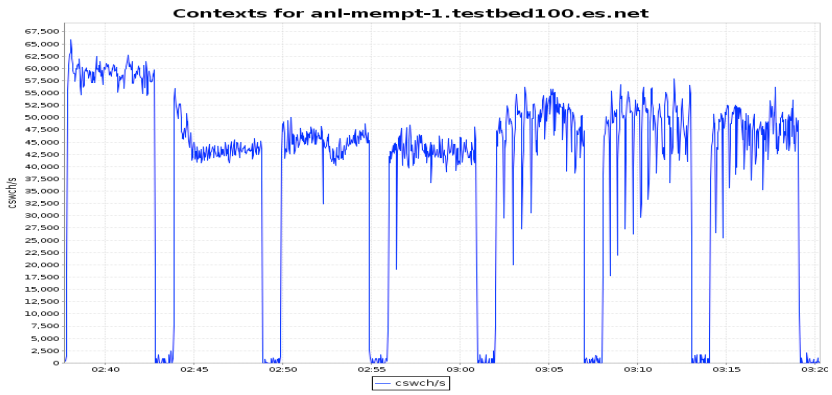

Figure 12: ANI testbed 100Gbps (10x10NICs, three hosts): Context switches vs the number of concurrent transfers [1, 2, 4, 8, 16, 32 64 concurrent jobs - 5min intervals], TCP buffer size is 50M

Figure 9 shows CPU utilization for the memory-to-memory concurrent transfers. We initiated multiple data movement jobs (each uses a single stream) at the same time. Each peak in these graphs represents a different test with different number of concurrent operations. As can be seen in Figure 10 (concurrent transfers), the interface traffic shows different characteristics as compared to Figure 4 (parallel streams). Figure 11 shows interrupts per second, and Figure 12 shows the number of context switches per second. In these tests, three hosts and 10 NIC pairs were used; the maximum achievable bandwidth was 100Gbs.

Hence we conclude that parallel streams and concurrent transfers have different effects if more than one NIC is involved. The total throughput drops as we increase the number of parallel streams. However, we do not see the same affect with concurrent transfers, though logically the number of sockets used is the same (parallel streams and concurrent transfers use multiple TCP sockets). Even though the CPU utilization is similar, concurrent streams perform better in most cases, as shown in Figure 8. On the other hand, context switches increases as concurrency level increases.

When we collect traffic information from the interface, we can see the effect of multiple streams clearly; with single stream and TCP buffer set, we see a steady performance peak, usually the best possible throughput value. Setting up the TCP buffer precisely also slightly reduces context switching.

When we set the interrupt affinity (each NIC is assigned to separate core, instead of sending all NIC interrupts to a single core), we observed increased CPU usage and increased context switches.

With these results, it seems evident that using 100Gpbs efficiently is a feasible goal for the future, but careful evaluation is necessary to avoid host system bottlenecks. We still lack a good understanding of how host performance affects the overall bandwidth usage.

5. MemzNet (Memory-mapped Zero-copy Network Channel)

In addition to experiments with file-centric state-of-the-art transfer tools such as GridFTP, we have also tested our new tool that aggregates files into blocks. We have evaluated its performance in 100Gbps demo and tests, and we observed better efficiency with this tool in transferring large datasets with many small files. This tool introduces a different concept. We call it memory-mapped zero-copy network channel (memZnet). Here, we briefly explain its architecture.

The architecture of MemzNet consists of two layers: a front-end and a back-end. Each layer works independently so that we can tune each layer separately. Transmitting data over the network is logically separated from the reading/writing of data blocks. Having separate front-end and back-end components has other benefits - we are able to have different parallelism levels in each layer. Those layers are tied to

each other with a block-based pre-allocated memory cache, 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 component receives data blocks and feeds the memory cache, so the corresponding front-end can get and process data blocks. These memory caches are logically mapped between client and server. Figure 13 gives the overall idea behind MemzNet.

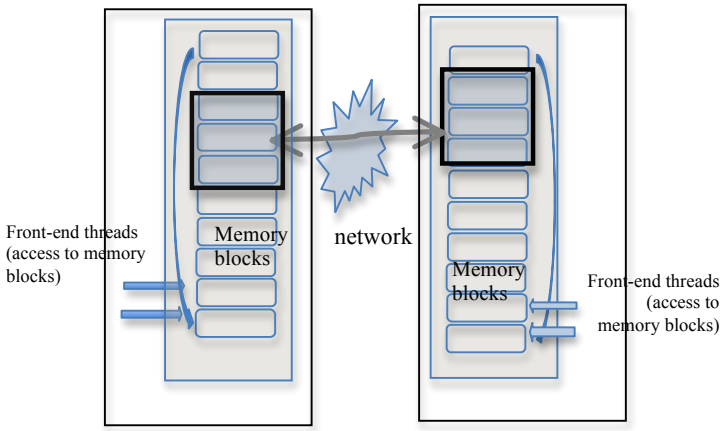

Figure 13: MemzNet's Design Principle

MemzNet introduces dynamic data channel management and asynchronous movement of data blocks. In file-transfers, the front-end component requests a contiguous set of memory blocks. Once they are filled with data read from the filesystem, those blocks are released, so that the back-end component can retrieve and transmit the blocks over the network. Data blocks include content information, i.e. file-id, offset and size. Therefore, there is no need for further communication between client and server in order to initiate file transfers. The memory management layer helps to send the data in large chunks when it is ready. A single stream is sufficient to fill up the network pipe. This is analogous in concept to on-the-fly 'tar' approach bundling and sending many files together.

Moreover, the data blocks can be received and sent out-of-order and asynchronously. 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 also. Besides, we can increase/decrease the number of parallel streams (if necessary) without the need to setup a connection channel, since each block includes its bookkeeping information.

In our new tool, every block includes its own metadata, and connection management is dynamic. Different from other file transfer protocols, there is almost no cost for changing the number of parallel streams; hence, it perfectly matches adaptive parameter tuning.

Although we tested MemzNet only for file transfers, the concept can also be extended to build efficient end-to-end communication channel for general network applications. We plan to enhance our implementation and encapsulate MemzNet's components as a library that can be used as a plug-in or a driver by external tools and applications. These libraries can be used to deliver streaming over the network for remote data analysis. We would like to test it for remote data access and with real data analysis applications. Finally, we plan to package this software as a production tool for general use.

6. Open Problems in End-to-end Transfers

Considering that the bandwidth will continue to increase but latency will still be the same, the next-generation high-bandwidth networks need to be evaluated carefully from the applications perspective. We believe that the next-generation 40Gbps/100Gbps will enforce new designs and enhancements in network applications. The research questions we would like to investigate can be classified as follows:

The effect of tuning parameters: We would like to understand the effects of tuning parameters in high-performance data transfers. With single stream and TCP buffer precisely set, we see a steady performance peak, usually the best possible throughput value. However, in real life, there will be many users at one point of time; hence, many streams will be used simultaneously. Our tests show that the total throughput decreases as the number of streams increases. We would like to examine how parallel streams, concurrent transfers, and TCP tuning affect network performance. We want to understand how to enhance network performance and make data movement more efficient in the application domain, without low-level changes in the network stack.

The effect of host performance: From our test results, we noticed that intensive tuning and optimizations are required to achieve 100Gbps performance. We observed that the host system performance could easily be the bottleneck. We still lack a thorough understanding of how host performance affects the overall bandwidth usage. We would like to explore the effects of resource utilization on the overall network performance. It is also important to identify bottleneck in current transfer protocols. Besides, the understanding of host performance issues will help design new communication frameworks and tune future network applications.

The effect of multiple streams on the host systems: We observed that parallel streams and concurrent transfers have different performance results and different effects on the host systems if more than one NIC is involved. The number of context switches increases as concurrency level increases. Another important parameter is the interrupt handling. We would like to investigate how parallel streams and concurrent transfer affect resource utilization in servers.

The effect of multiple NICs and multiple cores: Multiple NICs and multiple cores are involved in 40Gbps or 100Gbps data movements. Therefore, we need to understand the effects of using multiple NICs in multi-core environments. In our initial tests, we observed that setting the interrupt affinity properly increases network performance and also decreases the load in the host systems, by reducing the number of context switches and CPU usage. Analyzing performance statistics from the host systems will help us see the effects of using tuning parameters and multiple NIC cards in an end-to-end data transfer. Transfer tools and network applications should take multiple NICs and multiple cores into consideration, in order to adapt and benefit from high-bandwidth networks efficiently.

The effect of the application design: Another important aspect is whether we need enhancements in network applications to scale to the next generation high-bandwidth networks. It is likely that we may need radical changes in the current middleware tools to take advantage of future networking frameworks. MemzNet is an example specifically designed for high throughput data movements. We would like to explore what types of enhancements are necessary in network applications to take advantage of high-bandwidth networks.

We think that we need intelligent application designs that support high throughout network operations and efficient resource utilization in the host systems, and better system management in servers for automated tuning and optimization for high performance network access.

Besides, if new communication frameworks are developed in the future in order to support ongoing efforts to improve the networking stack, we want to identify important steps from application's perspective (explained briefly in Appendix A).

Acknowledgements

This work is accomplished in collaboration with Eric Pouyoul, Yushu Yao, E. Wes Bethel, Burlen Loring, Prabhat, John Shalf, Alex Sim, Arie Shoshani, Dean N. Williams, and Brian L. Tierney. We would like to thank 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. John Dugan and Gopal Vaswani provided monitoring tools for 100Gbps demonstrations at SC11.

This research used resources of the ESnet Testbed, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231. This work is supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract no. DE-AC02-05CH11231.

Bibliography:

M. Balman, MemzNet: Memory-Mapped Zero-copy Network Channel for Moving Large Datasets over 100Gbps Networks, Technical poster in ACM/IEEE international Conference For High Performance Computing, Networking, Storage and Analysis, 2012 (SC'12)

D. N. Williams et al., Earth System Grid Federation: Infrastructure to Support Climate Science Analysis as an International Collaboration. A Data-Driven Activity for Extreme-Scale Climate Science, Data Intensive Science Series: Chapman & Hall/CRC Computational Science, ISBN 9781439881392, 2013

M. Balman, Streaming exa-scale data over 100Gbps networks, IEEE Computing Now Magazine, Oct 2012.

M. Balman et al., Experiences with 100Gbps network applications. In Proceedings of the Fifth international Workshop on Data-intensive Distributed Computing, in conjunction with ACM High Performance Distributing Computing (HPDC), 2012

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, in conjunction with ACM/IEEE international Conference For High Performance Computing, Networking, Storage and Analysis, 2011

M. Balman and T. Kosar, Dynamic Adaptation of Parallelism Level in Data Transfer Scheduling, in Proceedings of International Workshop on Adaptive Systems in Heterogeneous Environments (ASHEs), in conjunction with the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS'09) and the IEEE International Conference on Availability, Reliability and Security (ARES'09), Fukuoka, Japan, March 2009.

Mehmet Balman, Data Placement in Distributed Systems: Failure Awareness and Dynamic Adaptation in Data Scheduling, VDM Verlag, January 2009. ISBN 978-3-639-12433-0

E. Yildirim, M. Balman, T. Kosar, Dynamically Tuning Level of Parallelism in Wide Area Data Transfers, in Proceedings of DADC'08, in conjunction with the IEEE International Symposium on High Performance Distributed Computing (HPDC 2008),Boston, MA, June 2008, pp. 39-38, ISBN:978-1-60558-154-

Appendix A: The desirability of zero-copy operations

Memory performance is a major bottleneck in today's computer systems. Loading data from memory to cache is a dominant factor compared to the increased processor speed in current hardware. Therefore, cache behavior is crucial especially in the performance of kernel code, including the networking layer. In a typical networking stack, packets are passed through multiple layers. It starts with moving packets to the kernel by interrupt handlers, encapsulation of packets by device drivers into buffer containers, metadata management, processing of packets, and copying data to the application. Memory copy and system call overheads reduce the overall performance. Bringing data to the user space requires context changes since it happens within the kernel. Especially in multicore systems, moving the task to another core as a result of a context change causes expensive cache penalties. Moving data over a network channel is not a simple task, if we take all these parameters into account.

There are several efforts to bring data as quickly as possible to the user space. One recent study is Rizzo's netmap (*info.iet.unipi.it/~luigi/netmap*) that proposes a new API to send/receive data over the network. In netmap, applications access the network card directly without involvement of the host stack. The netmap framework provides zero-copy packet processing, direct buffer access, and reduces number of system calls by processing multiple packets at once. It slightly modifies traditional socket API, and defines memory buffers mapped to the NIC's ring.

Another interesting technology is RDMA (Remote Direct Memory Access). It has been successfully implemented in HPC systems to read/ write from/to remote memory directly using the NIC. It has been also implemented on top of the IP layer, called iWARP (The Internet Wide Area RDMA Protoco) in which TCP, IP, RDMA headers are all processed in hardware. Recently, RDMA over Converged Enhanced Ethernet (RoCE) is gaining attention. RoCE defines RDMA semantics on top of Ethernet packets; using commodity interconnects in the same broadcast domain as an alternative to Infiniband.

The main principle both in RoCE and netmap is to provide user-level, low-latency, zero-copy networking. Different from traditional socket APIs, RDMA API (or so called verbs) defines memory registration, events, and asynchronous notifications. Similarly, netmap uses mmap to access zero-copy buffers and ioctl calls for synchronization and notifications. The asynchronous programming model in RDMA requires extra effort to adapt current applications and file transfer protocols. Since access method to the underlying networking stack is different, an important questions is whether we should consider redesigning current state-of-the-art tools based on these new semantics.

Based on our experience with RDMA programming models, and especially RoCE tests, we believe that applications or transfer tools also require optimizations to support fast flow of data without extra memory copies. Simply replacing network calls with RDMA equivalents is not sufficient. We require intelligent memory management techniques. MemzNet provides circular buffers that are logically mapped to remote corresponding buffers, in order to create a network channel. This design naturally complements the RDMA semantics. Therefore, we would like to test our tool over RoCE. We believe that MemzNet is beneficial over regular TCP/IP networks, but it also has great potential with future networking frameworks such as RDMA and netmap API.