Evaluation of the inter-cluster data transfer on Grid environment

Shoji Ogura
Tokyo Institute of Technology
Email: ogura@matsulab.is.titech.ac.jp

Satoshi Matsuoka
Tokyo Institute of Technology,
National Institute of Informatics,
Email: matsu@is.titech.ac.jp

Hidemoto Nakada
National Institute of Advanced
Industrial Science and Technology,
Tokyo Institute of Technology
Email: hide-nakada@aist.go.jp

Abstract

High-performance peer-to-peer transfer between clusters will be fundamental technology base for various Grid middleware, such as large-scale data transfer in DataGrid settings, or collective communication in Grid-wide MPIs. There, two major factors are involved: on one hand network pipes with large $RTT \times bandwidth$ typically become data-starved, resulting in bandwidth loss; on the other hand when multiple nodes on the clusters attempt simultaneous transfer, the network pipe could become saturated, resulting in packet loss which again may result in bandwidth degradation in large $RTT \times bandwidth$ networks. By dynamically and automatically adjusting transfer parameters between the two clusters, such as the number of network nodes, number of socket stripes, we could achieve optimal bandwidth even when the network is under heavy contention. In order to arrive at a proper performance model for automated adjustment, we have conducted several simulations by which we have discovered that such automatic tuning would beneficial, but the ideal number of network pipes does not exactly match the simple transfer model of traditional peer-to-peer settings between single nodes.

1 Introduction

Large-scale cluster nodes, with individual connectivity to high-speed WANs, are becoming widespread as mainstream platforms for Grid computing. There, high-performance peer-to-peer connectivity of clusters as a whole is becoming increasingly significant as the fat bandwidth on the main backbones is enabling applications with large data transfers to be much more realistic than the past. For example, in the CERN DataGrid project[1], one of whose main purpose is to construct a data processing fabric for petascale data emanating from different detectors in the LHC experiment, large-scale transfer of massive data between the compute/data clusters that constitute the DataGrid fabric would be the norm, as has been assumed by various middleware structures such as the Grid Datafarm[2]. Another example would be parallel MPIs jobs running across multiple clusters on the Grid, facilitated by middleware such as MPICH-G2[3]; there, collective communication as well as MPI-IO calls could result in massive data traffic between the peer clusters.

However, it has been widely pointed out that the limitation of TCP window size makes it difficult to naively make the best use of high-bandwidth networks in a wide-area environment. As such, several schemes have been suggested to overcome such shortcoming of TCP/IP based networking infrastructure for Grids, but these schemes focus largely on improving the performance of peer-to-peer transfer of single nodes on each end. When the peers become clusters, each with its own IP address and connectivity to the network, the proposed scheme may not scale properly, and/or make best use of the performance, since and all the nodes will try to grab the maximum bandwidth in an uncoordinated fashion. In fact, performance loss may be catastrophic in some cases, as we will observe.

Instead, we propose to come up with an efficient transfer scheme when the peers are such clusters, and coordinating the use of the networks between the nodes. The purpose of this paper is to report on the ongoing work, and specifically focus on the simulation results we have obtained so far, using the NS network simulator. The results indicate that such coordinated transfer would especially be beneficial in a (realistic) Grid environment where each network (cluster) node is running TCP Reno as the congestion control algorithm, where uncoordinated transfer may result in significant underutilization of network bandwidth. On the other hand, simple application of traditional models for determining the optimal network stripes for single-node peer-to-peer transfer does not seem to directly apply, underestimating the optimal number physically measured in the simulation. However, it seems difficult to integrate such factors into the current model directly, suggesting that automatically adaptive tuning methods would be desirable instead of static determination. We are currently working to replicate the results of the simulation using the newly facilitated SuperSINET National 10-gigabt backbone here in Japan.

2 Background and Previous Work on High Performance Data Transfer in TCP/IP Wide-Area Networks

2.1 TCP

To prevent congestions among multiple TCP streams on a single network link, TCP regulates the rate of data transfer by controlling TCP window size, reducing the window when congestion occurs, and increasing it otherwise. The differences in the control methods give rise to different congestion control algorithms, such as TCP Tahoe, Reno, and Vegas.

In most TCP/IP settings TCP Reno is widely used. Reno controls the window size in two phases. The first phase is the slow start phase, where the TCP window size is increased exponentially until a packet loss occurs. When a loss occurs, or if the TCP window size reaches some constant value, the window size is halved, and subsequently Reno transcends into the second phase, i.e., the congestion avoidance phase. Here, TCP window size is increased one by one, and if a packet loss occurs, TCP window size is halved, and this is repeated each time packet loss occurs. The lost packets are retransmitted, but when timeout occurs on this retransmission, then Reno starts to the first (slow start) phase. Figure1 illustrates typical behaviour of TCP window size control in TCP Reno.

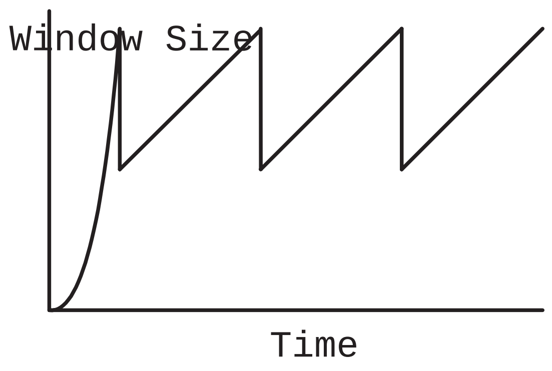

Figure 1. TCP Reno congestion window size change over time

2.2 Exploiting TCP Bandwidth in Wide-Area Networks

The problem of TCP over high-latency and high-bandwidth networks is largely caused by the limitation of TCP window size, and the subtle interactions between the congestion control algorithms. For example, in the early days of TCP/IP stack implementations, the upper limit of TCP window size was restricted to 64 kilobytes, due to memory restrictions of the machines. This severely restricts bandwidth, because data cannot sufficiently fill the network pipe before the receiver ack. Due to the recent exponential growth of the backbones as well as local router bandwidth, such restrictions quickly proved to be insufficient. As a result, various schemes have been proposed to utilize the network bandwidth fully:

1. Increase the upper limit of TCP window size. Most TCP/IP implementations now facilitate window scaling options, allowing the maximum window size to be set substantially large. This is not perfect, however; firstly, for extremely high-bandwidth networks, the TCP window size becomes substantial—the rule of thumb is twice that of the $RTT \times bandwidth$, which could amount to 10s to 100s of megabytes for networks with large delays. More serious problem is that, there will be increased probability of packet loss, and when it happens, it will take considerable amount of time for the slow start to recover the appropriate window size. There are recent extensions to TCP/IP stacks to alleviate the latter problem partially, such as High Speed TCP (HSTCP)[4]. Nonetheless, the latter could be serious because competing transfers emanating from different nodes in a single cluster may result in increased packet loss.

2. Use more bandwidth-eager UDP instead of TCP. UDP by itself does not embody any congestion control algorithm. Most recent high-performance network transfer work based on UDP literally “blasts” the UDP packets eagerly, without any congestion control, and com-

pensates for the lost packets later[5, 6]. The problem is that, it will devastate most competing TCP transmissions, as well as likely causing excessive packet losses between competing UDP transfers for inter-cluster peer-to-peer settings.

3. Transfer data in parallel, or so-called "striping". Basically, one opens multiple sockets, stripes data, and transfers data in parallel. This overcomes the limitation of TCP window size restrictions and associated problems, since a packet loss only results in partial degradation of transfer bandwidth in only one of the striped TCP connections. Moreover, it can be implemented at the application level[7], or at the library level[8]. The problem is that the ideal striped size differs depending on network bandwidth and congestion status, and could in fact result in "overstriping" when multiple nodes are involved simultaneously.

2.3 Problems when Clusters are Peers

The common problem with all of the above schemes when they are applied naively to inter-cluster peer-to-peer transfer is that, they all attempt to optimize the transfer based on individual peer-to-peer basis between the nodes of the cluster peers, and do not account for other nodes attempting to transfer at the same time, using exactly the same strategy. As a result, this may result in excessive packets being stuffed to the network pipe, without appropriate congestion control. This will likely be very evident for the UDP-based transfers, but could also be serious for other schemes.

Feng[9] also recently addressed the problem of peer-to-peer data transfer between two large clusters on the Grid. The work simulated large clusters at two US national labs Los Alamos and Sandia, as well as the network in-between. Data transfer was simulated on NS[10] using both TCP Reno and TCP Vegas and compared. The overall result is that, TCP Vegas was relatively robust with respect to increase in the number of nodes, but for Reno, the window size became excessively large for the available (shared) bandwidth during the slow start phase, and successively resulted in considerable packet loss due to congestion. Moreover, the problem became more serious as the network bandwidth increased.

Although TCP Vegas proved to be more robust, its mixed use with TCP Reno is known to result in poor performance (which will be quite general, given existence of network nodes outside the transfer)[11, 12] . As such, the use of TCP Vegas will not be desirable for high-performance transfers in realistic settings.

3 Coordinated Striping for Peer-to-Peer Transfer between Clusters

Based on the discussions, we propose the use of striping properly coordinated across the cluster nodes as a viable scheme of maximizing network utilization when the peers are large-scale clusters. Here, we have two parameters of control: the number of nodes that participate in the parallel transfer, and the number of network stripes for each node. Assuming that the latter is the same across all the nodes, the total number of stripes connecting the two peer clusters is naturally the product of the two numbers. By coordinating across the nodes and automatically adjusting the total number of stripes depending on network status, we may sustain maximal throughput from the network, avoiding excessive packet loss and the resulting TCP slow start which would hamper throughput significantly.

The questions then are: a) What is the optimal number of stripes, i.e., can we apply the standard network model applicable to single node peer-to-peer transfer to clusters, b) when we automatically adjust the total number of stripes, do the two parameters have differing effects, or is it just that their product merely matters, i.e., it is irrelevant whether we adjust the number of nodes or stripes per node. A) is important since if the model is applicable, controlling the number of stripes would become rather straightforward, since the ideal number can be derived simply from the observed RTT. For b), if the product only matters it will be much easier to control the number of stripes, since in fact we may alter the number of stripes straightforwardly amongst the nodes to reach the ideal number.

Although the final goal is to actually construct such a peer-to-peer transfer framework for the Grid, the goal of this paper is to identify the above questions, and investigate if the assumptions on the questions would hold. For such purposes, we have conducted extensive simulations, the result of which will be presented in the next section.

4 Simulating Data Transfer Between Clusters on the Grid

4.1 Simulation using the NS Network Simulator

In order to investigate the above issues, we simulate data transfer between clusters on the Grid under various configurations using NS (Network Simulator)[10]. NS is a discrete event simulator dedicated to simulating various network behaviors, and often employed for development and verification of new network protocols, analyzing queuing systems, etc. NS provides various protocols for different networking layers, such as low-level unicast/multicast protocols, as well as standard TCP protocols such as TCP, HTTP, FTP,

Table 1. Simulation Parameters of Cluster Nodes

TCP Packet Size	1KByte
Upper Limit of TCP Window Size	64 packets
Transfer Size (per node)	8Mbyte
Bandwidth to Router	100Mbps

etc. NS also allows simulation of different network topologies. In our simulations, we employed version 2.1b8 of NS.

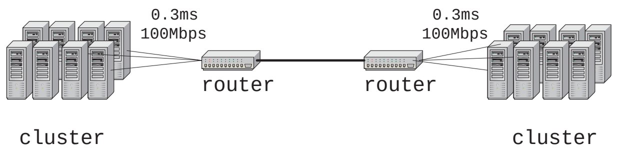

Figure 2. simulated network topology

4.2 Simulated Cluster Peer-to-Peer Environment

Figure2 illustrates the simulation network configuration we employed for Peer-to-Peer cluster data transfer. Parameters for each cluster node are shown in Table 1. Each cluster node transfers data to the corresponding node of the other cluster peer using a direct inter-node peer-to-peer TCP stream. The sizes of transmitted data are identical among the nodes (8MBytes), and the data on each node are striped to be the same size in the manner identical to PSockets[8]. We varied the network latency between the routers from 10ms to 320ms, and the network bandwidth from 1.5Mbps to 1000Mbps, although made the node to router bandwidth to be constant at 100Mbps. We also assume that the router queues are FIFO. We limit the duration of simulation to 100 seconds, since network instability was not observed for all cases.

5 Simulation Results

5.1 Single-Node Transfers—Determining Optimal Number of Stripes

We first determine the optimal number of stripes in a single node (i.e., non-cluster) peer-to-peer data transfer setting. Figure9 shows the data transfer time when we alter the number of stripes, assuming that the inter-router network bandwidth is 100Mbps. Different series varies the network delay. We observe that, for all cases the transfer time initially decreases as we increase the number of stripes, but at some point (approximately 50 stripes) we actually observe increase in transfer time. This is likely because the optimal TCP window size is overestimated during the slow start phase, causing excessive congestion. Figure3 illustrates this in another measurement. The graph shows the amount of data transmitted versus amount of data whose packets had been retransmitted packets for single-node transfer with the number of stripes being fixed at 52. We observe that, after 3 seconds from the beginning of transfer, there is significant rise in the data being retransmitted; this implies that the window size exceeded the optimal value after 3 seconds, causing significant packet loss.

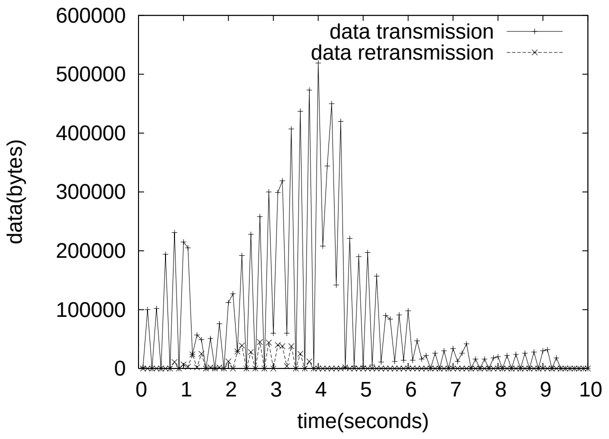

Figure 3. Data Transmitted / Retransmitted between Single Nodes with 52 Stripes over Time

5.2 Multiple Node Transfers

Figure4 shows the correlation between network latency and the optimal number of stripes, assuming the network bandwidths of 1.5Mbps, 5Mbps, and 10Mbps. Again, the optimal number of stripes is the number of stripes that finished data transfer the fastest. We observe that, the graphs are generally very irregular, and it is difficult to determine the trend given the network bandwidth/delay and the number of nodes. This is especially evident when the network latency is small.

Figure5 illustrates the data transfer time of different number of stripes over the 1.5Mbps links with varying latency. Here, we observe that, the variance of data transfer over differing number of stripes is much greater for short-latency networks compared to longer ones. Thus, for low-bandwidth and low-latency networks, striping is less essential, and in fact “overstriping” will have adverse effect on

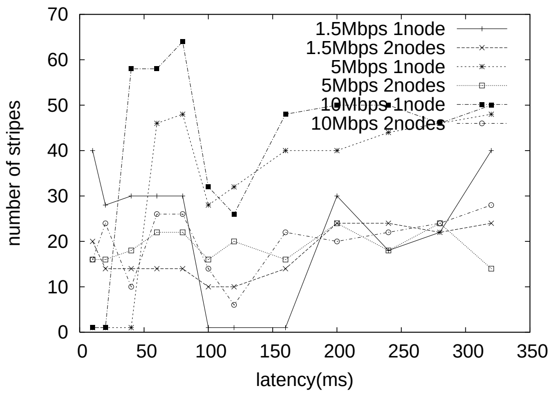

Figure 4. Optimal number of stripes for 1.5Mbps, 5Mbps, 10Mbps

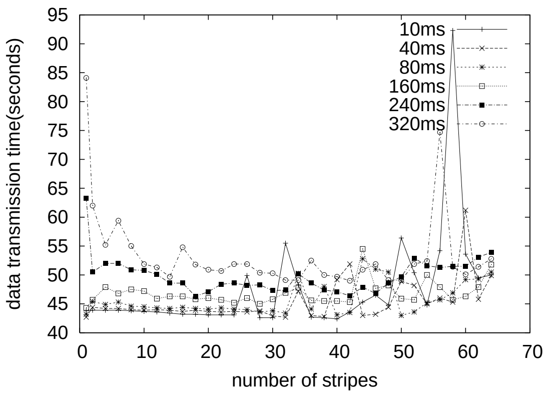

Figure 5. Single-Node Data Transfer Time for 1.5Mbps

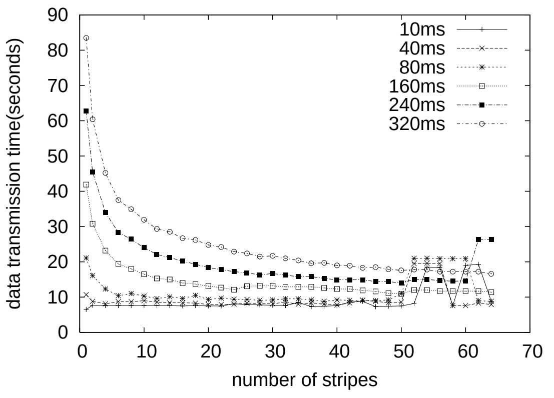

Figure 6. Single-Node Data Transfer Time for 10Mbps

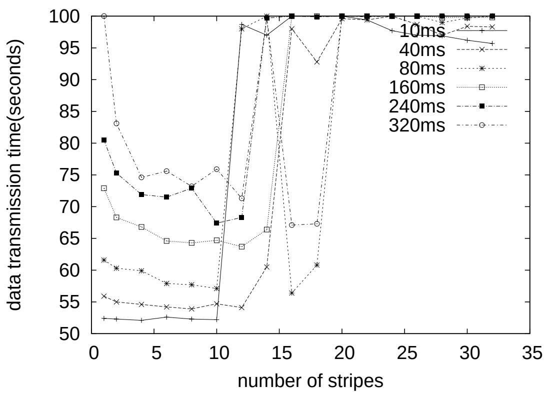

Figure 7. 8-Node Data Transfer time for 10Mbps

bandwidth utilization.

Now we move onto the real heart of the issues, i.e., multi-node peer-to-peer transfers. For 10Mbps, we observe the difference between the single-node and 8-node settings. Here, the network bottleneck is entirely between the routers, since the aggregate node-router bandwidth is 800Mbps on each end. For single node, we see in Figure6 that the transfer time steadily decreases, until there is a small ‘rift’ upon which the above overstriping occurs. On the other hand, for 8-node configuration in Figure7, we observe catastrophic loss in performance when we exceed the optimal number of stripes. This verifies that, even with striping, uncoordinated transfer that would saturate the network will have unacceptable performance due to explosive increase in packet loss.

The single-node series in the Figure 8 graph shows the situation where bandwidth increases to 100Mbps and 1000Mbps, and the optimal number of stripes increases as network latency grows. This implies that the number of stripes will increase as the delay-bandwidth product grows.

As for multi-node setting, 100Mbps and 1000Mbps behave differently. 100Mbps behaves much like the 10Mbps case, where overstriping causes catastrophic bandwidth loss as we observe in Figures 11 and 13, whereas graphs in 1000Mbps Figures 10, 12, and 14 remain rather similar. This is largely because in the latter case the bottleneck resides in the network between the nodes and the router (100Mbps), and as a result individual node saturates its bandwidth at the level where their aggregate largely matches the inter-router bandwidth. This suggests that, we must be aware of the actual network topology and the associated bandwidth of not only the trunk network, but also the cluster interconnect to the router and the outside world.

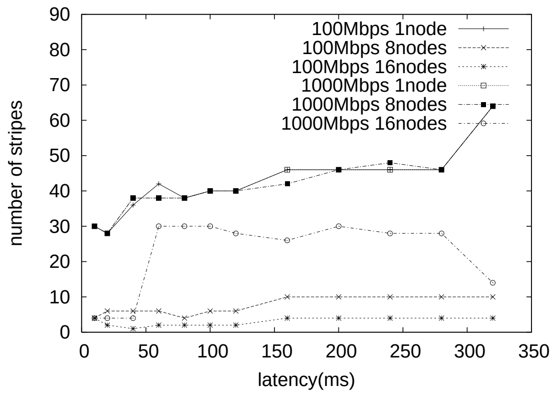

Figure 8. Optimal number of stripes for 100Mbps, 1000Mbps

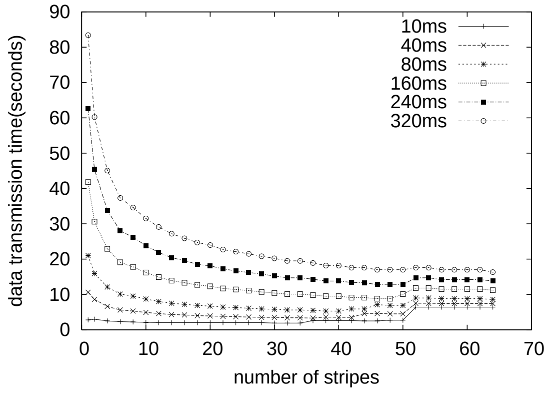

Figure 9. Single-node data transfer time for 100Mbps

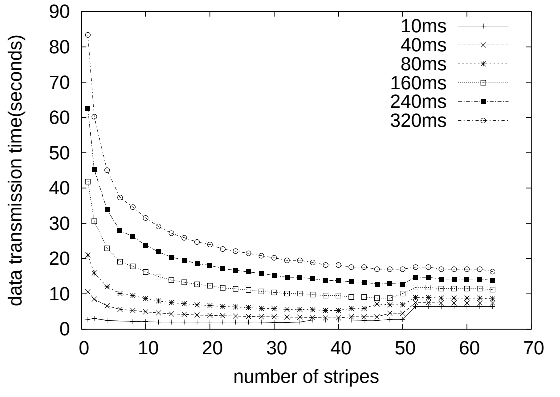

Figure 10. Single-node data transfer time for 1000Mbps

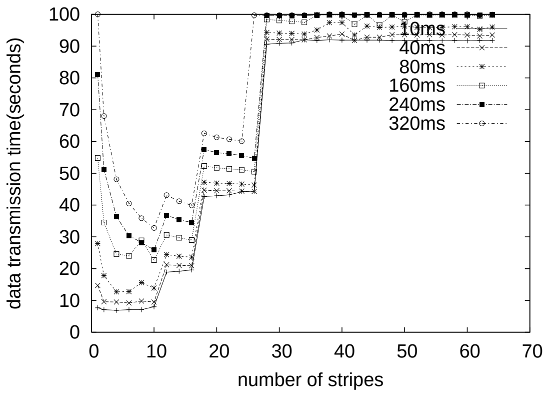

Figure 11. 8-node data transfer time for 100Mbps

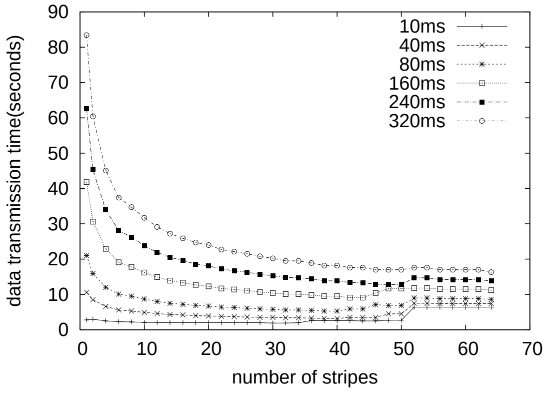

Figure 12. 8-node data transfer time for 1000Mbps

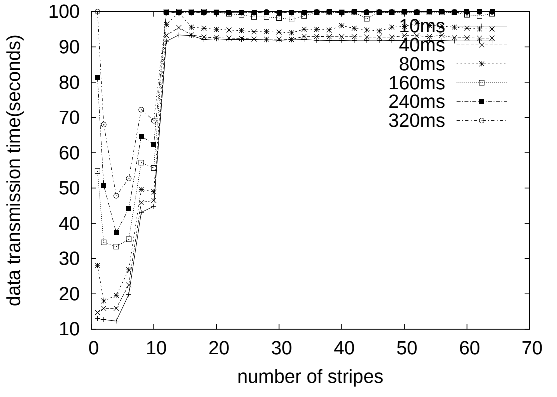

Figure 13. 16-node data transfer time for 100Mbps

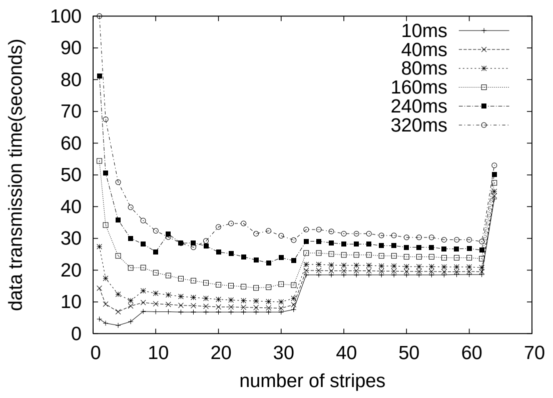

Figure 14. 16-node data transfer time for 1000Mbps

5.3 Comparison with Simple Modeling, and Proposals for Dynamic Parameter Tuning

We briefly compare our results with a simple model of optimal stripe number and sizing. We denote the roundtrip time as RTT, the bandwidth as $bandwidth$, the upper limit of TCP window size as max_wnd_size, and the number of stripes as $stripe$. If no packet losses occur, the amount of data transferred is $stripe \times max\_wnd\_size$, and the available data in flight on the link is $RTT \times bandwidth$. If we fully utilize the link, the optimal number of stripes would be as follows:

$$\frac{RTT \times bandwidth}{max\_wnd\_size}$$

For example, for 100Mbps bandwidth and 160ms RTT, the optimal number of stripes would be 6. However, in the simulation, the optimal number of stripes measured was 10; as such there is definitely a non-negligible discrepancy. Moreover, the formula denotes that the optimal number of stripes is proportional to RTT, but is not so in the simulation, especially for the case when the available theoretical maximum bandwidth is small, where optimal number of stripes may not be monotonically increasing along with growth in the network delay.

Although we have not fully confirmed the results of our simulation with live measurements (we have conducted several experiments, but due to problems with the misbehaving gigabit switch our results were not conclusive), nonetheless it does seem to indicate that it is difficult to apply the simple model above for determining the optimal number of stripes. Instead, we propose the following three possible approaches, all of which adjust the maximum stripe size *dynamically* in a coordinated fashion across the nodes:

- Tune the parameters such as the number of nodes dynamically during transfer, using observable dynamic values of the transfer (e.g., packet loss ratio), and apply an extended and corrected model to derive new parameters.
- Tune the parameters such as the number of nodes dynamically during transfer by running a simulation (such as the one we have conducted for this paper) alongside the actual transfer. This may allow more accurate and faster compliance with the optimal setting, but could be more difficult to manage.
- Determine the parameters such as the number of nodes dynamically during transfer, using periodic observable dynamic values of the transfer (e.g., packet loss ratio) and searching the database that indicates the optimal values. The database may be constructed from real observation or from simulation. This may have the benefit of both accuracy and lightweight of the schemes above.

6 Conclusion and Future Work

For inter-cluster peer-to-peer transfer in Grid settings, we pointed out that traditional means of high-performance data transfers between single node peers may cause suboptimal performance due to uncoordinated bandwidth optimization of individual peer pairs. Although some may claim that striped transfer could alleviate the problem to some degree, we nonetheless claimed that even so some situations the packet loss could have devastating effect on performance. We have actually confirmed this in our simulation, and especially when the available bandwidth between the routers are much smaller than the aggregate bandwidth of the nodes to the router, the optimal number of stripes per node is much smaller than single-node peer-to-peer transfers, and in fact “overstriping” the transfer results in catastrophic degradation of performance. Moreover, the optimal number does not match the simple model of stripe determination, and as such we suggested several schemes where basically the total number of network stripes emanating from the cluster is controlled in coordinated and dynamic fashion.

As a future work, we plan to employ the new SuperSINET Japanese academic multi-gigabit networking infrastructure to conduct physical experiments to verify the validity of our simulation. Already we have connected some of our fleet of clusters to SuperSINET, including the 512-processor Presto III[13]. Also, based on the findings we plan to extend the data transfer portion of our GFarm DataGrid system in order to incorporate the necessary control and dynamic parameter tuning scheme we have suggested above, and measure the results. Our results hopefully will be applicable to numerous inter-cluster peer-to-peer transfers in general, including Grid middleware tools such as GridFTP.

References

[1] http://eu-datagrid.web.cern.ch/eu-datagrid/.

[2] Osamu Tatebe, Youhei Morita, Satoshi Matsuoka, Noriyuki Soda, and Satoshi Sekiguchi. Grid datafarm architecture for petascale data intensive computing. *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid(CCGrid 2002)*, 2002.

[3] http://www.hpclab.niu.edu/mpi/.

[4] Sally Floyd. Highspeed tcp for large congestion windows. *Internet draft*, 2002. work in progress.

[5] M. Mazzucco, H. Sivakumar, Y. Pan, and Q. Zhang. Simple available bandwidth utilization library for high-speed wide area networks. *Submitted to Journal of SuperComputing.*

[6] Phillip M. Dickens and William Gropp. An evaluation of a user-level data transfer mechanism for high-performance networks. *Proc. of 11th IEEE Symposium on High-Performance Distributed Computing(HPDC)2002*, 2002.

[7] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. Gridftp: Protocol extensions to ftp for the grid, 2001. http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf.

[8] H. Sivakumar, S. Bailey, and R. L. Grossman. PSockets:the case for application-level network striping for data intensive applications using high speed wide area networks. *SC2000*, 2000.

[9] W. Feng and P. Tinnakornsrisuphap. The failure of tcp in high-performance computational grids. *SC2000*, 2000.

[10] ns. Network simulator. http://www-mash.cs.berkeley.edu/ns.

[11] Jeonghoon Mo, La Richard J., Venkat Anantharam, and Jean Walrand. Analysis and comparison of TCP Reno and Vegas. *Proceedings of INFOCOM'99*, March 1999.

[12] Kenji Kurata, Go Hasegawa, and Masayuki Murata. Fairness comparisons between tcp reno and tcp vegas for future deployment of tcp vegas. *SSE99*, 1999. (in Japanese).

[13] http://cluster-team.is.titech.ac.jp/.