Dynamic Adaptation of Parallelism Level in Data Transfer Scheduling *

Mehmet Balman and Tevfik Kosar
Center for Computation & Technology and Department of Computer Science
Louisiana State University, Baton Rouge, LA 70803, USA
{balman, kosar}@cct.lsu.edu

Abstract

We discuss dynamic parameter tuning in wide-area data transfers for efficient utilization of available network capacity and optimized end-to-end application performance. Impacts of parallel TCP streams as well as concurrent data transfer jobs running simultaneously have been studied. We present an adaptive approach for tuning parallelism level of data placement jobs in distributed environments. The adaptive data scheduling includes dynamically setting parameters of data placement jobs. The proposed methodology operates without depending on any external profiles to adapt to changing network conditions.

1. Introduction

Efficient data access is one of the major concerns in large scale distributed applications. In a data-aware system model, data placement jobs are crucial components of the end-to-end workflow to ensure on-time arrival of data sets to the execution nodes [14]. The data placement scheduler is responsible for orchestrating the management and the coordination of data transfer jobs [12, 13]. One important goal of the data scheduler is to optimize data transfers. Therefore, we tune-up system resources and organize data placement jobs to minimize the completion time of every single operation while trying to maximize the overall throughput.

We use multiple data streams for fetching data in order to fully utilize the network bandwidth. Since we are limited by the latency in the network, the buffer size optimization and data stream parallelism have great impact on minimizing the overall data transfer time. Besides, we start multiple concurrent operations in which several data transfer jobs operate simultaneously. System resources like CPU, disk, and network are also shared by other processes. Excessive use of these resources may lead to some problems like resource starvation. So, we need to adjust the level of parallelism according to the capacity of the environment.

In this study, we focus on the level of parallelism in two stages: (1) the number of parallel data streams connected to a data transfer service for increasing the utilization of network bandwidth, and (2) the number of concurrent data transfer operations that are initiated at the same time for better utilization of system resources.

One approach is to measure the network and system statistics and come up with a good estimation for the parallelism level. However, those techniques require extensive measurements and usage of historical data. Besides, the estimated value might not reflect the best possible current settings due to the dynamic characteristics of the distributed environment.

We have designed an alternative approach in which, instead of making external measurements over the network, we make use of the information gathered from the data transfer operations that are activated by the scheduler. We propose an adaptive approach in which we set the level of parallelism for data transfer operations according to the application level throughput of previous and current operations inside the scheduler. Number of concurrent jobs running at the same time is increased gradually as long as there is any performance gain for the overall throughput. We have also implemented a transfer module that adaptively sets the number of parallel streams in a single transfer.

2. Related Work and Discussion

Buffer size tuning is an important issue in wide-area TCP performance to fully use the network pipe for high throughput. The optimal buffer size is related to bandwidth-delay product which is the product of the data link capacity (bandwidth of the bottleneck link) with end-to-end delay (the round trip time of the connection) [5, 6, 7]. There have been many studies that investigate diagnosis techniques to measure bandwidth and round-trip time of a given con-

*This project is in part sponsored by National Science Foundation under award numbers CNS-0619843 (PetaShare) and EPS-0701491 (CyberTools), and by the Board of Regents, State of Louisiana, under Contract Numbers DOE/LEQSF (2004-07), NSF/LEQSF (2007-10)-CyberRII-01, and LEQSF(2007-12)-ENH-PKSFI-PRS-03

nection [4]. The bandwidth-delay product can vary in a shared wide-area network environment. Thus, initial measurements and estimated buffer size values may not reflect the optimal value for the time a data transfer operation is active. A dynamic auto-tuning approach in user space has been proposed [16], and an adaptive methodology to set the buffer size dynamically has been implemented in [7, 6].

There are several studies for configuring the GridFtp (a high performance data transfer protocol [2]) control parameters like the number of parallel streams [9, 8]. In order to optimize the level of parallelism, we need information such as packet loss rate, end-to-end delay and available bandwidth. Moreover, several models have been studied in the literature to estimate accurate parameter configuration for parallelism [17]. The GridFtp automatic parameter configuration described in [10, 9], measures throughput for each chunk transferred and adaptively increases the number of parallel TCP connections if there is performance gain. Since we need to reset the number of TCP data channels to be used in the transfer, there is an overhead in reconfiguring the GridFTP connection. In order to minimize the effect of this overhead for reconfiguring the GridFTP control parameter, large blocks of data chunks are transferred at every data operation call.

Additive increase, multiplicative decrease, and multiplicative increase are some of the techniques for numerically searching best parameter optimization [9, 8]. After initializing the number of parallel TCP connections, a fixed chunk of data is transferred and the instant throughput is measured along with the round-trip time. If there is gain in terms of performance, the parallelism level, N, is increased by a constant factor. If the number of parallel streams is larger than the optimal value, we will see a performance decrease in throughput. The upper limit for the measured instant throughput is defined in [9, 10] as $(NW)/R$, where R is the round-trip time, N is the number of optimum parallel streams, and W is the TCP buffer size.

We have also implemented a similar methodology which benefits from a very simple adaptive approach. We transfer data chunks of the file and gradually increase the number of parallel streams. The adaptive transfer module, described in the following section, makes use of the current throughput of each chunk, and tries to reach to a near optimum value gradually, instead of finding the best parameter achieving the highest throughput at once.

Adaptive data placement has also been studied in various data management architectures like [15]. Beyond that, adaptive data transfer protocols are used in wireless networks for better energy consumption [3]. Data transfer protocol adapts itself to the changing environment conditions such that data transfer rate is increased when communicating parties are closer to each other to reduce power consumption. And, data transfer rate is decreased when there is long distance between communicating parties. Data transfer rate is adjusted using the estimated, $D_{estimated}$, and measured, $D_{measured}$, values in each transfer period [3]: $D_{estimated}(t+1) = \alpha D_{measured}(t)+(1-\alpha)D_{estimated}(t)$ Similarly, we also try to utilize the underlying resources and fill the network pipe as much as possible in order to minimize completion time of a transfer.

One recent work studies run-time adaptation of data placement jobs [11]. This work comes with a prototype to monitor environment conditions and to update the data transfer scheduler with required information to tune up control parameters. The data placement scheduler uses this information for performing run-time adaptation. There is a tuning infrastructure in which environment information is provided by external profiler like memory, disk, network profilers [11]. The parameter tuner executes periodically and requests information from profilers to come up with a decision which is to be used by the data transfer scheduler to set data transfer parameters.

Our approach differs from the above method by not depending on any external measurements or profiling results. Moreover, adaptability is embedded inside the scheduler such that performance information is collected to be used for tuning subsequent requests. Instead of probing the system in order to get profiling information, we just use performance metrics from actual data transfers for parameter tuning. This approach also does not require any complex model for parameter optimization. The scheduler's decision adapts itself to the environment conditions. In summary, memory shortage or insufficient network latency might be the actual reason for not letting us to increase the number of parallel streams or the number of concurrent jobs. But, gradually improving parallelism level brings a near optimal value without the burden of dealing with complex optimization steps to find the major bottleneck in a data transfer.

3. Dynamic Parameter Tuning

Instead of making measurements with external profiler to set the level of concurrency and parallelism in data transfer scheduling, we propose to calculate parameters using information from previous or current running data transfer operations. Thus, we do not pollute the network with extra packets and do not put extra load to the system due to extraneous calculations for exact parameter settings. We simply set the level of parallelism dynamically (both the number of concurrent jobs and the number of parallel streams) by observing the achieved application throughput for each transfer operations and gradually tuning parameters according to previous or current performance merit.

Using multiple parallel streams in data transfer is simply aggregation of TCP connections [9]. Instead of a single connection at a time, multiple TCP streams are opened to a

single data transfer service in the destination host. We gain larger bandwidth in TCP especially in a network with less packet loss rate; parallel connections better utilize the TCP buffer available to the data transfer, such that N connections might be N times faster than a single connection [9, 10].

In addition to setting the number of parallel streams for a data transfer inside the data transfer scheduler, we have also implemented and tested a special data transfer module which automatically sets the number of parallel streams.The methodology is similar to the automatic parallelism setting described in [9, 10]. We use a less complicated but effective algorithm. It measures the transfer time of each chunk transferred and calculates the current throughput. The dynamic feature of this GridFTP transfer module enables us to transfer data by chunks and also set control parameters on the fly. We keep the record of best throughput for the current parallelism level. The actual throughput value of the data chunk transferred is calculated and the number of parallel streams is increased by one if this throughput value is larger than the best throughput seen so far. We gradually increase the number of parallel streams till it comes to an equilibrium point. Figure 1 gives a glimpse of the algorithm used to implement the transfer module which set parallelism level adaptively.

3.1. Adaptive Scheduling

The data transfer scheduler accepts multiple jobs in a nondeterministic order, say: $< J1, J2, J3, \ldots >$. Completion time of a job also depends on environment conditions such as network and system load. In order to increase the throughput, we use multiple streams, so completion time T, depends on the number of parallel streams used for the transfer. For simplicity, we reduce parameters affecting T into two key attributes: (1) the number of parallel streams to fetch data from destination host, and (2) the number of concurrent jobs scheduled to access resources of the source and destination hosts. We underline the fact that our focus

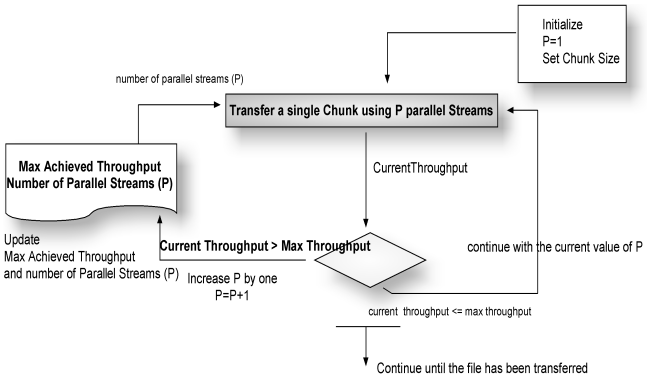

Figure 1: Setting the Parallelism Level inside the Data Transfer Module

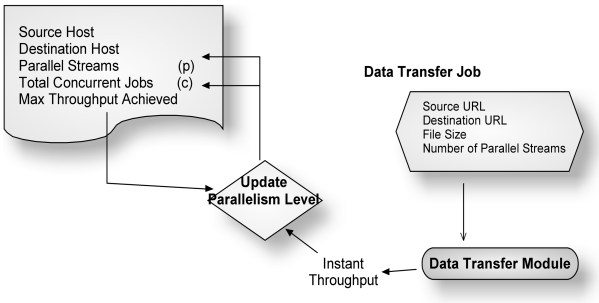

Figure 2: Adaptive Parameter Tuning in Data Transfer Scheduler

is on application level tuning such that we do not deal with low level network and server optimization. The goal of the scheduler is to minimize the execution time for each data transfer while preserving the user fairness based on submission order. Therefore, we try to find the best possible settings, $T =< J, c, p >$, for c (concurrent jobs) and p (parallel streams) in an adaptive manner without using the external measurements from network prediction tools.

We give the control of setting the level of parallelism to the scheduler. Figure 2 shows the overall architecture of adaptive data scheduling. We tune the number of parallel streams according to the environmental conditions, and also taking into consideration the fact that there are other jobs currently running on the system. For each *source-destination pair*, we keep track of jobs with all possible parameters to be used in subsequent scheduling decisions as follows: $J_L(source, destination) = < total\ \#\ of\ parallel\ streams, \#\ of\ concurrent\ jobs, current\ throughput\ >$. We keep information about the number of concurrent running jobs using the same source and destination hosts during the transfers. We maintain a record about the current throughput which is calculated from the last successful data transfer operation. This value reflects the current state of the system. We use it to make a decision on whether to increase the parallelism level for transfer jobs using this source-destination pair. Moreover, we maintain another entry inside the scheduler for each remote host, which keeps the total number of data transfer jobs scheduled at the current time-period associated with that host: $J_H = < total\ \#\ of\ jobs, total\ \#\ of\ connections >$. J_H enables the scheduler to prevent resource starvation and to enhance the scheduling decision by controlling and limiting the number of concurrent access to a single location.

4. Evaluation

In order to test our methodologies, we use the LONI [1] environment. Table 1 presents the network characteristics of our test system, where Queenbee (queenbee.loni.org) is

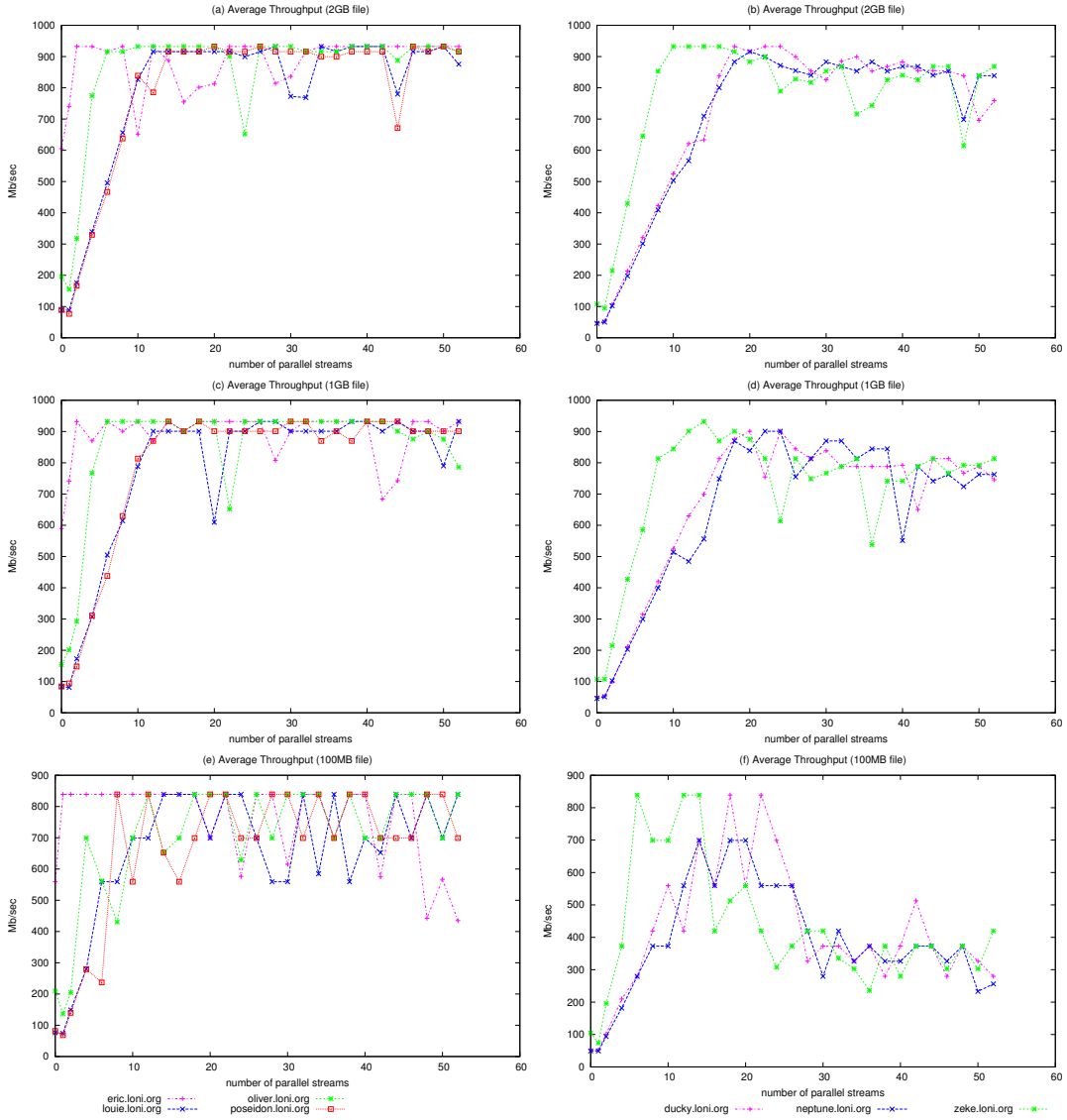

Figure 3: Average Throughput of file transfers to Queenbee (on LONI) using Parallel Streams. Get, (a) 2GB file from Linux machines, (b) from IBM machines, (c) 1GB file from Linux machines, (d) from IBM machines, (e) 100MB file from Linux machines, (f) from IBM machines.

connected over a 1Gbps link to other machines. First, we have performed experiments to see the effect of number of parallel streams in GridFTP transfers for small and large file sizes. As can be seen in Figure 3, we present average throughput results for data transfers with parallel TCP streams. Next, we have tested the effect of concurrent jobs (performing transfer operations at the same time) using single data stream per job. The scheduler is expected to decide on the concurrency level and to initiate simultaneously multiple data transfer jobs. Figure 4 presents average throughput of total data transferred versus the number concurrently running jobs. Here, test results in experimenting the effect of concurrency level show more inconsistent behavior as compared to the graphs in Figure 3. According to our first hand experience, data transfer jobs hang and stop responding due to resource starvation usually after 25-30 concurrent jobs.

Figure 5.a and Figure 5.b show results from our adaptive GridFtp transfer module in which we gradually increase the number of parallel streams inside the data transfer module. The adaptive setting of parallel streams makes use of the best throughput results achieved so far while transferring data chunks, and adjusts the parallelism level adaptively. Figure 5.c and Figure 5.d show fluctuations in instant throughput of data chunks during the entire transfer.

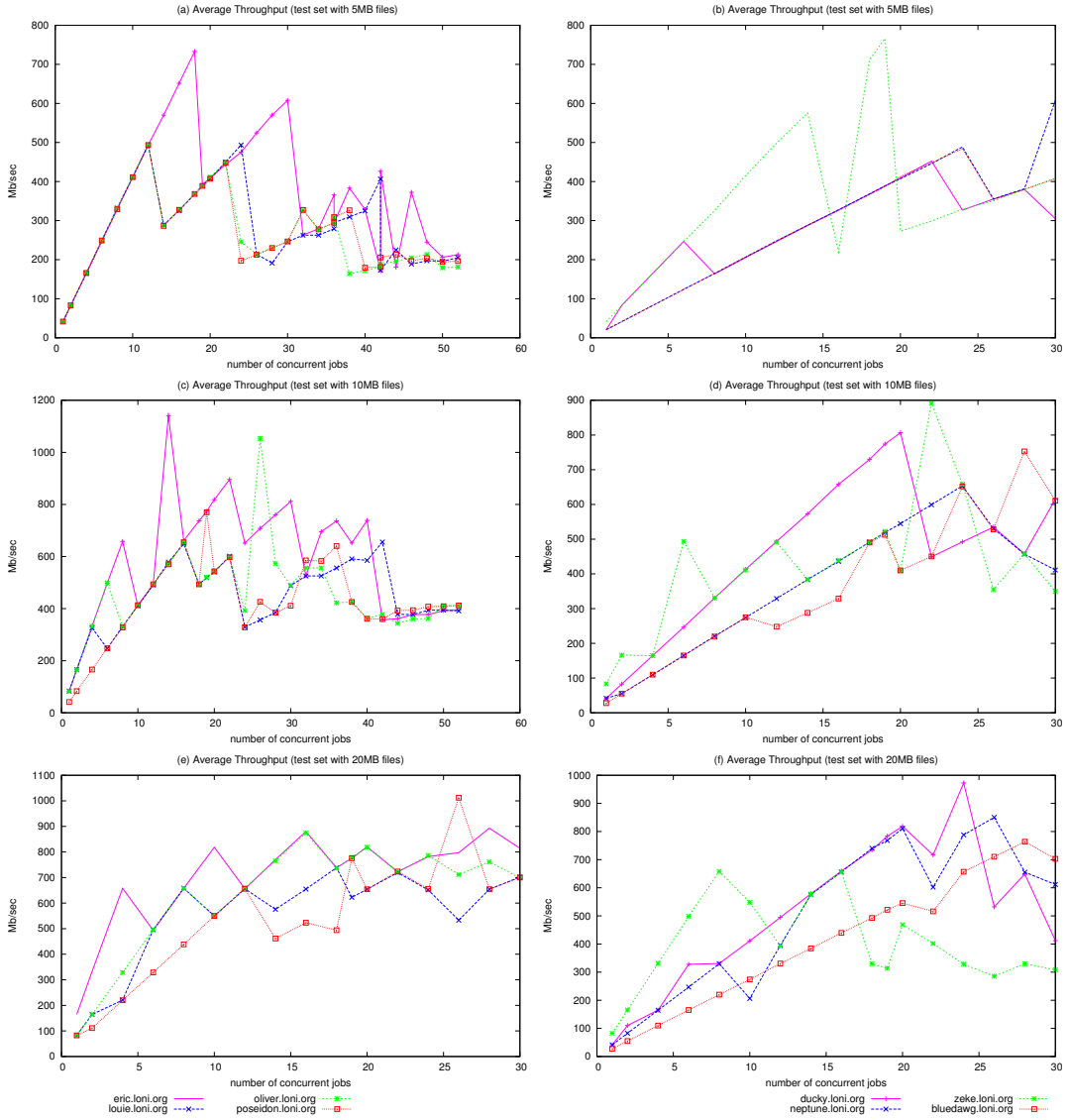

Figure 4: Average Throughput of Concurrent transfer jobs to Queenbee (on LONI) (getting multiple files concurrently), (a) with 5MB files from Linux machines, (b) from IBM machines, (c) with 10MB files from Linux machines, (d) from IBM machines, (e) with 20MB files from Linux machines, (f) from IBM machines.

source host = queenbee.loni.org		RTT(ms)		
	Destination	min	avg	max
Linux	eric.loni.org	0.541	0.548	0.555
machines	louie.loni.org	5.105	5.131	5.159
	oliver1.loni.org	2.438	2.456	2.466
	poseidon1.loni.org	5.32	5.334	5.343
IBM	ducky.loni.org	5.107	5.129	5.142
machines	neptune.loni.org	5.325	5.34	5.355
	zeke.loni.org	2.473	2.505	2.525
	bluedawg.loni.org	7.992	8.005	8.034

Table 1: Test Environment

5. Conclusion

In this paper, we have presented an adaptive approach for tuning parallelism level of data placement jobs in distributed environments. The proposed methodology operates without depending on any external profiles to adapt to changing network conditions. Our initial experiments emphasize the importance of tuning parallelism level in application level data transfer performance, and support that an adaptive approach can be a feasible alternative to the models which heavily depend on historical and/or profiling data. Our future work will include improvement of our adaptive tuning methodology as well as testing the dynamic scheduling framework in a heterogeneous testbed.

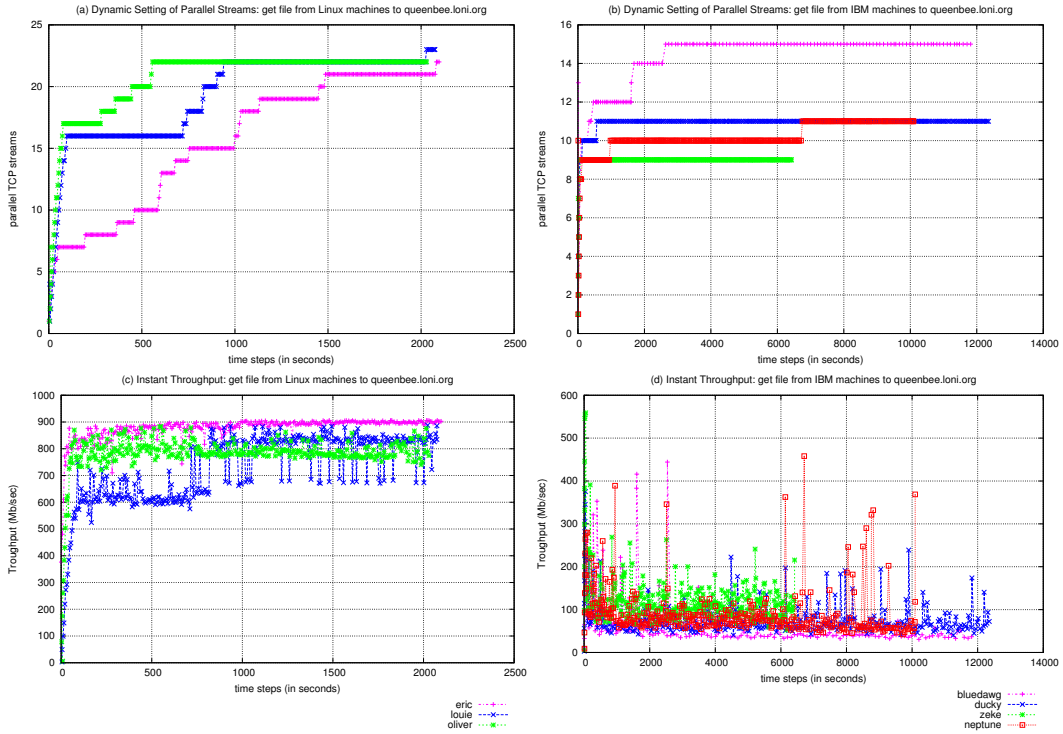

Figure 5: Dynamic Setting of Parallel Streams: transfer file to Queenbee (on LONI), (a) from Linux machines, (b) from IBM machines. Instant Throughput: transfer file to Queenbee, (c) from Linux machines, (d) from IBM machines.

References

[1] Louisiana optical network initiative. http://www.loni.org/.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link. The globus striped gridftp framework and server. In *Proceedings of the ACM/IEEE conference on Supercomputing*, 2005.

[3] M. Conti, E. Monaldi, and A. Passarella. An adaptive data-transfer protocol for sensor networks with data mules. In *Proc. IEEE International Symposium on a World of Wireless, Mobile, and Multimedia Networks*, 2007.

[4] T. Dunigan, M. Mathis, and B. Tierney. A tcp tuning daemon. In *Proceedings of SuperComputing: High-Performance Networking and Computing*, 2002.

[5] W.-C. Feng, M. Fisk, M. K. Gardner, and E. Weigle. Dynamic right-sizing: An automated, lightweight, and scalable technique for enhancing grid performance. In *Proceedings of the 7th IFIP/IEEE International Workshop on Protocols for High Speed Networks*, 2002.

[6] M. K. Gardner, W.-C. Feng, and M. Fisk. Dynamic right-sizing in ftp (drsftp): Enhancing grid performance in user-space. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[7] M. K. Gardner, S. Thulasidasan, and W.-C. Feng. User-space auto-tuning for tcp flow control in computational grids. *Computer Communications*, 27:1364–1374, 2004.

[8] T. Ito, H. Ohsaki, and M. Imase. On parameter tuning of data transfer protocol gridftp in wide-area grid computing. In *Proceedings of Second International Workshop on Networks for Grid Applications, GridNets*, 2005.

[9] T. Ito, H. Ohsaki, and M. Imase. Automatic parameter configuration mechanism for data transfer protocol gridftp. In *Proceedings of the International Symposium on Applications on Internet*, 2006.

[10] T. Ito, H. Ohsaki, and M. Imase. Gridftp-apt: Automatic parallelism tuning mechanism for data transfer protocol gridftp. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006.

[11] G. Kola, T. Kosar, and M. Livny. Run-time adaptation of grid data placement jobs. In *Proceedings of Int. Workshop on Adaptive Grid Middleware*, 2003.

[12] T. Kosar and M. Balman. A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems*, In Press, DOI: 10.1016/j.future.2008.09.006.

[13] T. Kosar and M. Livny. Stork: Making Data Placement a first class citizen in the grid. In *Proceedings of the 24th Int. Conference on Distributed Computing Systems,*, 2004.

[14] M. Balman and T. Kosar. Data scheduling for large scale distributed applications. In *the 5th DCEIS, International Conference on Enterprise Information Systems*, 2007.

[15] K. Voruganti, M. T. Özsu, and R. C. Unrau. An adaptive data-shipping architecture for client caching data management systems. *Distrib. Parallel Databases*, 2004.

[16] E. Weigle and W.-C. Feng. Dynamic Right-Sizing: A Simulation Study. In *10th International Conference on Computer Communication and Networking*, 2001.

[17] E. Yildirim, M. Balman, and T. Kosar. Dynamically tuning level of parallelism in wide area data transfers. In *Proceedings of the International workshop on Data-aware distributed computing*, 2008.