End-system Performance Aware Transport over Optical Circuit-Switched Connections

Amitabha Banerjee[†], Wu-chun Feng[‡], Dipak Ghosal[†], and Biswanath Mukherjee[†]
[†]Dept. of Computer Science, University of California, Davis, CA 95616, USA
[‡] Virginia Tech, Blacksburg, VA, 24061, USA
[†] abanerjee@ucdavis.edu, ghosal@cs.ucdavis.edu, mukherje@cs.ucdavis.edu, [‡] feng@cs.vt.edu

***Abstract*— In this work we investigate the impact of the receiving end-system performance on data transfer to it via a dedicated optical circuit. Such a situation commonly exists in an e-Science application which may receive data from an optical circuit and process it simultaneously. We illustrate an end-system performance monitoring tool which can deliver the feedback of the receiving end-system performance using a transport protocol, so that the sending end-system may control the sending rate using this feedback.**

I. Introduction

The advent of large-scale collaborative science applications has demonstrated the potential for broad scientific communities to pool globally distributed resources to produce unprecedented data collections, simulations, visualizations, and analysis. The success of these large collaborations requires the ability to transfer huge amounts of data at high rates between supercomputing centers, data repositories, and end hosts.

For example, data may be aggregated from distributed information repositories that are physically located in different sites across the world, for subsequent analysis at a computing center. Alternatively, results generated at a supercomputer may be transferred to remote clients for visualization. In each of these cases, the volume of data may be hundreds of terabytes, and in some cases, petabytes, transferred at very high speeds (e.g., OC-192: 10 Gbps) and oftentimes across long distances (e.g., intra- as well as inter-continental).

To support such applications, backbone lambda-grid networks have been provisioned to provide high, on-demand bandwidth between end-points that are interconnected via optical circuits. Examples include National LambdaRail (NLR) and DOE's UltraScienceNet [1] in the United States, and CANARIE's CA*net in Canada.

While end-to-end optical circuit-switched connections eliminate congestion in the network, they push the congestion to the endpoints. For example, high priority and critical tasks may prempt the execution of the process sending or receiving data. Similarly, a compute-intensive processes such as visualization of received data may be executing at the end-system, in which case the operating system has to schedule a computationally (CPU) bound process (visualization), and an I/O bound process (receiving data) simultaneously. In such cases, packets may get dropped due to buffer overflow at the Network Interface Card (NIC) if the receiving data process is not scheduled by the operating system at the appropriate times to transfer the packets from the NIC buffer.

In this paper, we investigate how computational load on an end-system may impact its ability to receive data at high bit rates. In Section II we highlight experiments on end-system performance which have been reported in greater detail in [7] . In Section III, we illustrate a tool which extracts the end-system performance metrics and then provides an estimated bottleneck rate to the sending end-system.

TABLE I
System Configuration.

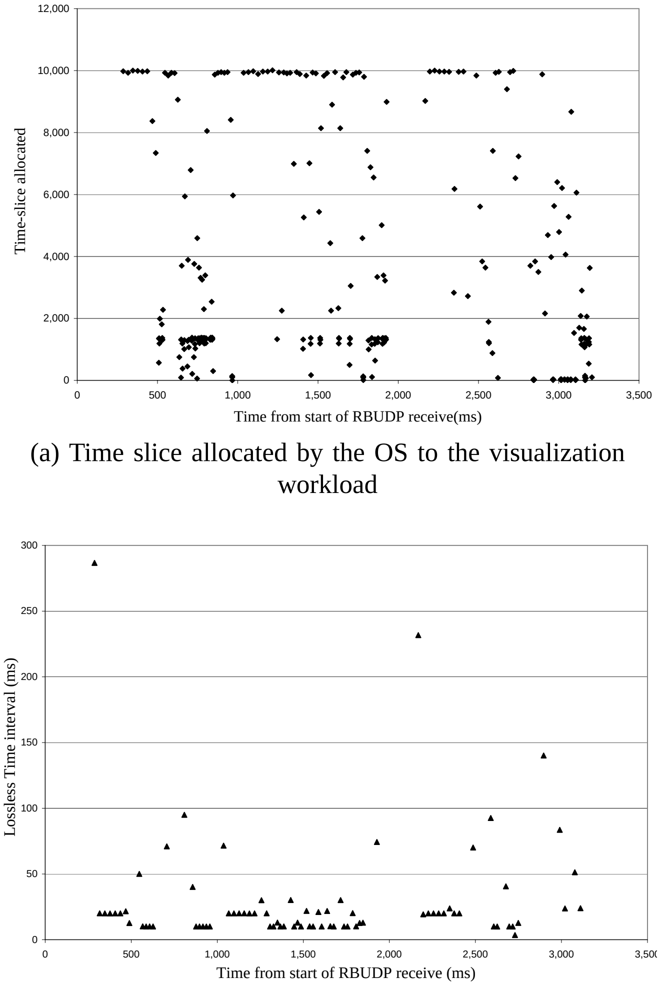
Processor	Speed	Cache Size	RAM	Iperf
Intel Pentium 4	3 GHz	512 KB	2 GB DDR RAM	2.44 Gbps

II. Experiments on the End-system Performance

We conducted experiments with two machines connected back-to-back with Chelsio 10-Gigabit Ethernet adapters [2] using optical fiber. The system configuration and the maximum bandwidth achieved by a TCP connection measured using Iperf 2.0.2 is shown in Table I. The IPerf measurements are with a 9000-byte Maximum Transfer Unit (MTU). TCP offloading was not enabled on the Chelsio adapters. Since the speed of a single regular hard disk is substantially lower than the transfer rate of 10 Gbps, we emulated an end-system to end-system file transfer by transferring a file between two RAMdisks (which use the RAM for storing data). RAMDisks allow for RAM access times. We used Reliable Blast UDP (RBUDP) as a transport protocol in our experiments [4].

We used MAGNET (Monitoring Apparatus for General kerNel-Event Tracing) [3] to analyze the end-system performance. MAGNET is a low-overhead tool that provides fine-grained monitoring of kernel- and user-space events by allowing any event to be monitored, and by time stamping each event with the CPU-cycle counter which is the highest-resolution time source available on most machines. Optionally, additional information can be exported to give a more detailed account of kernel operations.

The context-switch times between the different processes was monitored by MAGNET at the receiving end-system. We transferred a 500 MByte file using the experimental setup described above. We observed that the sending rate at 1.6 Gbps resulted in the fastest end-system to end-system transfer



(a) Time slice allocated by the OS to the visualization workload

(b) *Lossless time interval* duration

Fig. 1. Context-switch time analysis for the *visualization* workload.

time when the receiving end-system was under no additional computational load. In order to emulate a computational load on the receiving end-system, we ran an isosurface extraction visualization process of a knee joint image, which is expected to impose varying computational and I/O load, depending on the stage of computation. We hereafter refer to the latter as a *visualization workload*.

Figure 1(a) shows the duration of the time slices that is allocated to the visualization workload by the operating system when the end-system is receiving data simultaneously. For simplicity, we only showed the results for the time duration of the first UDP blast (iteration of transmitting UDP packets) of RBUDP. We observed that many time slices allocated by the OS to the visualization workload are of duration of as much as 10 ms. Most of the packets arriving at the receiving end-system in such a time interval when the visualization workload is executing would be lost, because the network interface card may not support enough buffer to store the arriving packets. At a line rate of 1.6 Gbps, 2 MByte of data would be lost in a duration of 10 ms, if not handled properly by the end-system.

We define time intervals when no packet losses occur at the receiving end-system as **lossless time intervals**. We measure the duration of the lossless time interval by comparing packet losses with MAGNET traces. Figure 1(b) shows the duration of lossless time intervals. We observed that most of the lossless time intervals are distributed between 10 ms and 20 ms.

We explain these observations as follows. Most OS schedulers differentiate between an I/O-bound process and a CPU-bound process. One of the goals of the OS scheduler is to improve the interactivity and response time of the system. The OS intends to favor the I/O process when scheduling. Different OSs have different means of classifying I/O and CPU processes, and favoring the I/O-bound processes. For example, the Linux 2.6 scheduler classifies between the above two, based on the average sleep time of a process, which is updated every time the process is context switched. A dynamic bonus proportional to the average sleep time is awarded to the task priority. I/O-bound processes have higher average sleep time than CPU-bound processes, and they are thus favored.

In the above example, when the process receiving data starts, it is classified as an I/O process. Interrupts are frequently generated for the incoming packets. While the OS is handling these interrupts, the CPU-bound visualization workload is held up. After a certain period of time (280 ms in this case), the OS begins to treat the process receiving data as a CPU-bound process, since it has been running continuously. Thereafter, both processes are handled equally.

III. End-system Performance Monitoring Tool

From these experiments, we conclude that a receiving end-system is prone to dropping packets when it is subject to a computational load in addition to receiving data. Figure 2 shows a model of the various components of an integrated tool that we are developing to track the end-system performance and deliver the feedback to the sender so that the sender may throttle the sending rate appropriately under such circumstances. The integrated tool consists of four components (i) a fine grained, lightweight Kernel-level Event Monitoring Tool (KEMT), (ii) a lightweight Event Prediction Tool (EPT), (iii) a Queuing Network-model based Tool (QNT) for performance analysis of the end-system so as to compute and predict the end-system bottleneck rate of the receiving process, and (iv) a Feedback Generator (FG) which integrates with the transport protocol and delivers the feedback to the sender. We briefly describe the role of each of the tools as follows:

A. Kernel-level Event Monitoring Tool (KEMT)

In order to model the network operation and to predict substantial changes in end-system behavior, various event traces need to be collected. Events must be monitored in the kernel rather than in the application because many events that negatively affect performance occur while the application is not running and hence the application is unable to take note. An example is the kernel scheduler.

Therefore, the KEMT must be a light-load kernel event monitoring tool, which can timestamp events such as context-switch between processes, hardware and software interrupts,

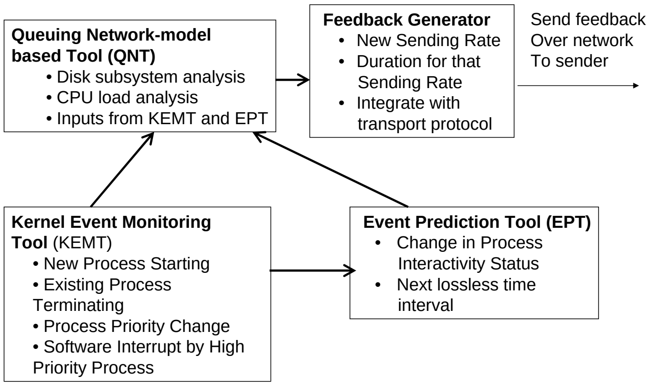

Queuing Network-model based Tool (QNT)
• Disk subsystem analysis
• CPU load analysis
• Inputs from KEMT and EPT
Feedback Generator
• New Sending Rate
• Duration for that Sending Rate
• Integrate with transport protocol
Send feedback Over network To sender
Kernel Event Monitoring Tool (KEMT)
• New Process Starting
• Existing Process Terminating
• Process Priority Change
• Software Interrupt by High Priority Process
Event Prediction Tool (EPT)
• Change in Process Interactivity Status
• Next lossless time interval

Fig. 2. Various components of the integrated tool

changes in the task priority, etc. and supply it to the EPT and QNT for useful extraction. MAGNET is an example.

B. Event Prediction Tool (EPT)

Optical circuits may span across large geographical distances, resulting in large Round-Trip Times (RTTs). Hence any feedback that is sent from the receiving end-system encounters a feedback delay. The purpose of the event prediction tool is to address the feedback delay. The goal is to predict events that are likely to change the workload and hence the bottleneck rate of the receive process. Examples of such events that may be predicted are change in the dynamic task priority, disk I/O interrupts, etc. QNT may be invoked at such predicted times to analyze the system and determine the bottleneck rate which may be fed back to the sender.

C. Queuing Network-model based Tool (QNT)

Given the kernel level data, the goal is to determine the bottleneck rate of the process receiving data. Queuing network models have been extensively used for system performance analysis. For example, DiskSim [5] uses system-level traces obtained from instrumenting the operating system with queuing network models to estimate the response time of an I/O subsystem against a given workload. The predicted response times are demonstrated in [5]to be very accurate to the actual measurements. An analytical model to predict the throughput of a disk array for a given workload is discussed in [6]. Along the above lines, QNT will analyze the system as well as the network I/O performance, to determine a bottleneck for receiving data. This bottleneck is then communicated to Feedback Generator.

D. Feedback Generator (FG)

FG will integrate with the transport protocol to send the bottleneck rate to the sender. The end-system performance monitoring tool must be lightweight so that it does not impose any additional load on the receiving end-system. In addition, it is favorable for KEMT and EPT to be a real time or soft real-time processes so that they are appropriately scheduled by the operating system.

We have designed a simple prototype for the above model which we call the Rate-Adaptive Protocol for Intelligent Data-transfer (RAPID) [7]. We chose RBUDP as a transport protocol. The interactive status of the process receiving data is polled at fixed polling intervals. A prediction is made as to when the process receiving data may reach such a state at which the OS treats it at a lower priority than an alternate CPU-bound process. At the predicted time compensated with the known RTT, a feedback is sent to *sending agent* to suspend sending for some duration of time, which is define as the *suspend interval*. The motivation is to allow the process receiving data to be idle for the corresponding time, during which the OS may schedule an alternate (CPU-bound) process.

Figure 3 shows the transfer time for a 800 MB file transfer between two machines on the testbed reported in Section II. We compare the file transfer time between RBUDP and RAPID at different values of RTT. At 0 RTT, RAPID shows an improvement of as much as 25%. The transfer time for RAPID increases with RTT because the prediction algorithm must compensate for the RTT. RAPID outperforms the fastest RBUDP transfer possible (by manually tuning the transfer rate to 1.2 Gbps) till an RTT of 80ms. We believe that better prediction algorithm will compenstae for longer RTTs. These results demonstrate that tuning the sending rate depending on feedback of the receiving end-system performance may be an effective solution.

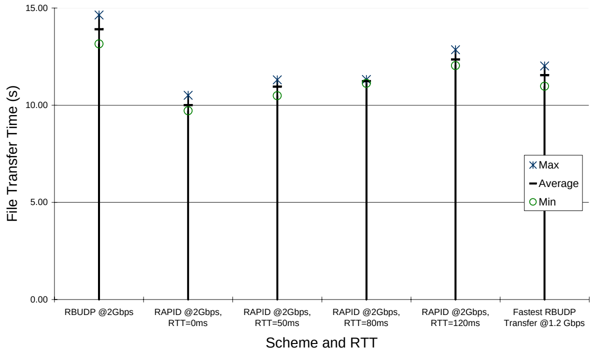


Fig. 3. File-Transfer Time for a visualization workload at MTU=9000

References

[1] DoE UltraScience Net at http://www.csm.ornl.gov/ultranet/

[2] Chelsio 10 Gigabit Etehrnet Adapter at http://www.chelsio.com/products/T210.php

[3] M. Gardner, W. Feng, M. Broxton, A Engelhart, and G. Hurwitz, "MAGNET: A Tool for Debugging, Analysis and Adaptation in Computing Systems," *Proc., CCGrid 2003*, Tokyo, Japan, May 2003.

[4] E. He, J. Leigh, O.Yu, T. A. DeFanti, "Reliable Blast UDP : Predictable High Performance Bulk Data Transfer," *Proc. IEEE Cluster Computing*, Chicago, Illinois, 2002.

[5] G. Ganger and Y. Patt, "Using System-Level Models to Evaluate I/O Subsystem designs", *IEEE Transactions on Computers*, vol.47, no. 6, June 1998.

[6] M. Uysal, G. A. Alvarez, and A. Merchant, "A Modular, Analytical Throughput Model for Modern Disk Arrays", *Proc., International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 183–192, August 2001.

[7] A. Banerjee, W. Feng, B. Mukherjee, and D. Ghosal, "RAPID: An End-System Aware Protocol for Intelligent Date Transfer over Lambda Grids", *Proc., IEEE IPDPS 2006*, Rhodes, Greece, April 2006, to appear.