Appropriateness of Transport Mechanisms in Data Grid Middleware

Rajkumar Kettimuthu[1,3], Sanjay Hegde[1,2], William Allcock[1], John Bresnahan[1]

[1]Mathematics and Computer Science Division, Argonne National Laboratory, [2]Department of Computer Science, Illinois Institute of Technology, [3]Department of Computer and Information Science, The Ohio State University

Introduction

• Bulk data transfer has become one of the key requirements in many Grid applications
• GridFTP has been widely deployed for high-speed data transport services
• These services normally require reliable data transfer resulting in TCP as the preferred common base protocol
• Unfortunately TCP performs sub optimally in achieving maximum throughput on the currently available "long fat networks" over the Internet
• This work involves two phases of investigation on the impact of transport protocol on bulk data transfer:
  • Appropriate instrumentation and study of standard Linux TCP stack, incorporating the recently proposed modifications for high-speed transport
  • Evaluation the non-TCP based reliable transport mechanisms such as NETBLT, Tsunami from Indiana University, RBUDP from the Electronic Visualization Lab at University of Illinois - Chicago

Web100

• Improves TCP instrumentation by providing a simple but elegant means of understanding the underlying operation of TCP within a host.
• Includes tools for measuring performance and network diagnosis to get a dynamic view of the behavior of the TCP sessions
• Provides foundation for TCP autotuning performed in process-level code and the process-level tools designed to locate bottlenecks
• Helped identify the cause of extreme round-trip time variance in a recent bulk data transfer experiment
• Helped identify the various possible reasons for drop in the congestion window
• Does not provide information about the process id for a TCP stream; dropped packets are not instrumented

Reference: http://web100.org

Congestion Control in TCP

TCP uses two algorithm for congestion control: slow start and congestion avoidance
• Maximum data in flight is min(congestion window, advertised window)
• Slow start: Congestion window is initialized to one segment. Each time an acknowledgment is received, the congestion window is increased by one segment
• Congestion avoidance: increase in congestion window should be at most one segment each round-trip time (regardless of the number of acknowledgements that are received in that round-trip time)
• Slow-start threshold is used to switch between slow-start and congestion avoidance. Exit slow-start and enter congestion avoidance when the congestion window goes above slow-start threshold
• Fast retransmit and recovery are proposed to improve the performance of TCP by retransmitting without waiting for the retransmit timer to expire

References: RFC 2001, RFC 2581, RFC 2582, RFC 2914

Limited Slow-Start

• The Problem:
  • The current slow-start procedure effectively doubles the congestion window in the absence of delayed acknowledgments.
  • For TCP connections that are able to use congestion windows of thousands of segments, such an increase can easily result in thousands of packets being dropped in one round-trip time.
  • This is often counterproductive for the TCP flow itself and is also hard on the rest of the traffic sharing the congested link.

• The Solution – Limited Slow-Start:
  • Limits the number of segments by which the congestion window is increased during slow-start, in order to improve performance for TCP connections with large congestion windows.
  • Introduces another threshold called "limited slow start threshold"
    • Enter limited slow-start when the congestion window goes above this threshold
    • During limited slow-start, the congestion window is increased by at most half of the maximum segment size for each arriving acknowledgement
    • Exit limited slow-start and enter congestion avoidance when the congestion window goes above the "slow-start threshold"

Reference: Internet draft draft-floyd-tcp-slow-start-01.txt

High Speed TCP

• Current standard TCP places a serious constraint on the congestion windows that can be achieved by TCP in realistic environments
• High-speed TCP is a modification to TCP's current congestion control mechanism for high-delay, bandwidth networks
• It introduces a threshold value. If the congestion window is less than the threshold, it uses the normal AIMD algorithm where the additive value is 1 and the decrease factor is 0.5
• If the congestion window is greater than the threshold, it uses High Speed response function to calculate alternate values for AIMD
• Benefits:
  • Achieves high per connection throughput without requiring unrealistically low packet loss rates
  • Reaches high throughput without long delays when recovering from multiple retransmit timeouts
• The proposed change to the AIMD algorithm may impose a certain degree of unfairness as it does not reduce its transfer rate as much as standard TCP

Reference: Internet draft draft-floyd-tcp-highspeed-01.txt

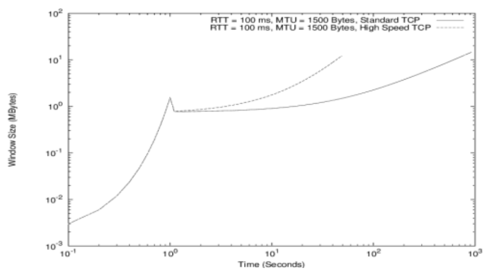


Figure 1: Comparison of the congestion window variation in standard TCP and high-speed TCP

With the modified values for the AIMD, the high-speed TCP is able to reach the bandwidth delay product much faster than the standard TCP

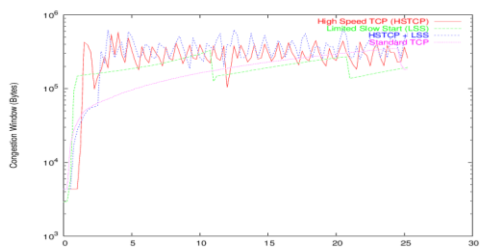


Figure 2: Comparison of the congestion window variation for various schemes

High-speed TCP schemes achieve higher congestion window than the other schemes

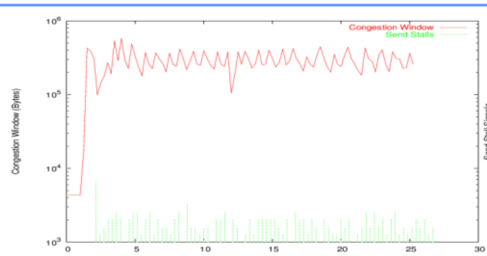


Figure 3: Effect of send stalls on the congestion window for high-speed TCP

Even though there is no congestion in the network, Linux TCP treats the local resource stalls as congestion signals

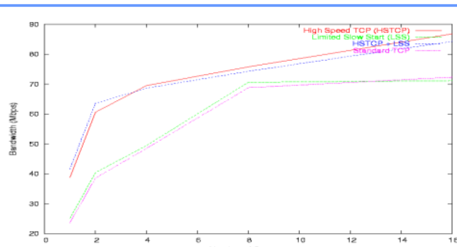


Figure 4: Variation of bandwidth with parallel streams for different schemes

High-speed TCP schemes outperform the other schemes

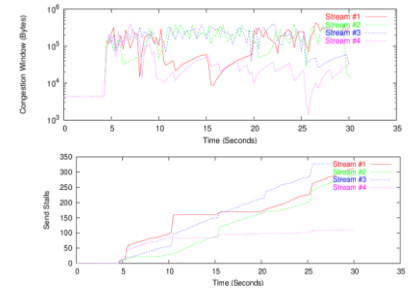


Figure 5: Interaction of multiple streams for high speed TCP with send stalls

Note that some of the streams are not able to achieve a higher congestion window

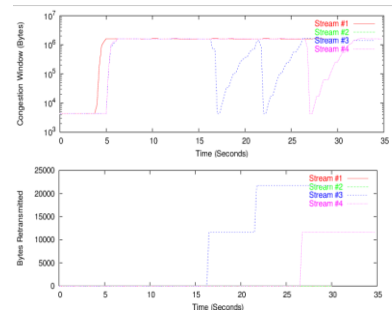
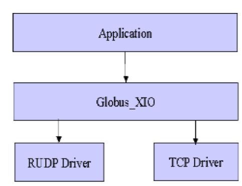


Figure 6: Interaction of multiple streams for high-speed TCP with no send stalls

Globus XIO



• Provides a standard API for the applications
• Drivers are responsible for all file access and data transporting

Other Transport Protocols

• NETBLT (NETwork BLock Transfer):
  • Transfer the data in a series of large data aggregates called "buffers". The sending NETBLT must inform the receiving NETBLT of the transfer size during connection setup
• RUDP (Reliable UDP)
  • Layered on UDP/IP protocols and provides reliable in-order delivery. EACK is used to specify the out-of-order segments received and unlike TCP the receiver RUDP receiver cannot discard the out-of-order segments
• RBUDP (Reliable Blast UDP)
  • UDP augmented with aggregated acknowledgements to provide reliable bulk data transmission. Acknowledgements are delivered at the end of the transmission phase using a bit vector
• Tsunami
  • A hybrid TCP/UDP based file transfer protocol. It uses UDP for payload and TCP for signaling including request for retransmission

Conclusion

• Web 100 is a useful tool for TCP instrumentation and trouble shooting
• Current TCP is not suitable for long fat networks.
• High Speed provides better throughput than the standard TCP but the fairness of it needs to be evaluated
• Local resource stalls imposed by Linux adds additional constrains on throughput