High Performance Networking for Wide Area Data Grids

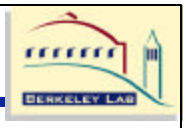
Brian L. Tierney
(bltierney@lbl.gov)

Data Intensive Distributed Computing Group
Lawrence Berkeley National Laboratory
and
CERN IT/PDP/TE

Overview

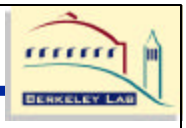
- The Problem
  - When building distributed, or “Grid” applications, one often observes unexpectedly low performance
    - the reasons for which are usually not obvious
  - The bottlenecks can be in any of the following components:
    - the applications
    - the operating systems
    - the disks or network adapters on either the sending or receiving host
    - the network switches and routers, etc.

Bottleneck Analysis



- Distributed system users and developers often assume the problem is the network
  - This is often not true
- In our experience running distributed applications over high-speed WANs, performance problems are due to:
  - network problems: 30-40%
  - host problems: 20%
  - application design problems/bugs: 40-50%
    - 50% client , 50% server

Overview



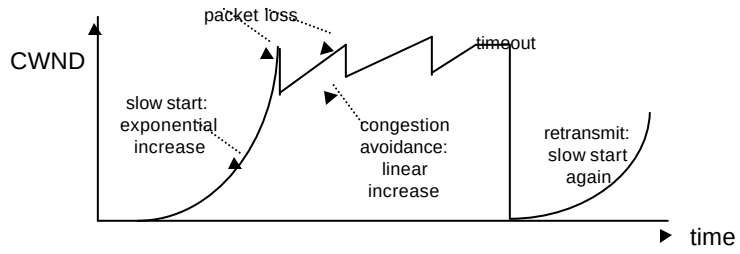
- Therefore Grid application developers must:
  - understand all possible network and host issues
  - thoroughly instrument all software.
- This talk will cover some issues and techniques for performance tuning Grid applications
  - TCP Tuning
    - TCP buffer tuning
    - other TCP issues
    - network analysis tools
  - Application Performance
    - application design issues
    - performance analysis using NetLogger

How TCP works: A very short overview

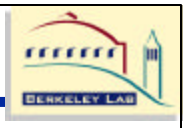
- Congestion window (cwnd)
  - The Larger the window size, higher the throughput
    - Throughput = Window size /Round- trip Time
- Slow start
  - exponentially increase the congestion window size until a packet is lost
    - this gets a rough estimate of the optimal congestion window size
- Congestion avoidance
  - additive increase: starting from the rough estimate, linearly increase the congestion window size to probe for additional available bandwidth
  - multiplicative decrease: cut congestion window size aggressively if a timeout occurs

TCP Overview

- Fast Retransmit: retransmit after 3 duplicate acks (got 3 additional packets without getting the one you are waiting for)
  - this prevents expensive timeouts
  - no need to slow start again
- At steady state, cwnd oscillates around the optimal window size
- With a retransmission timeout, slow start is triggered again

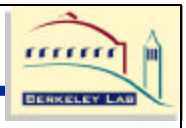


TCP Performance Tuning Issues



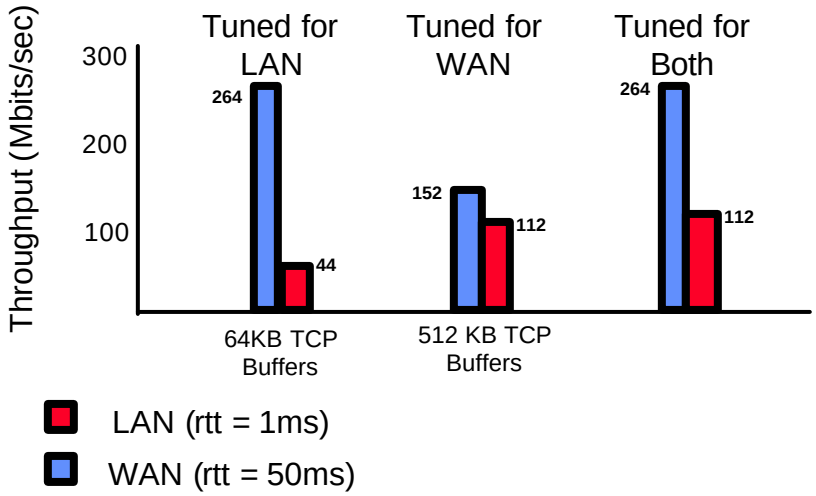
- Getting good TCP performance over high latency networks is hard!
- application must keep the pipe full, and the size of the pipe is directly related to the network latency
  - Example: from LBNL to ANL (3000km), there is an OC12 network, and the one-way latency is 25ms
    - Bandwidth = 67 MB/sec (OC12 = 622 Mb/s = ATM and IP headers = 539 Mb/s for data
    - Need 67 MBytes * .025 sec = 1.7 MB of data “in flight” to fill the pipe
  - Example: CERN to SLAC: latency = 84 ms, and bandwidth will soon be upgraded to OC3
    - assume end-to-end bandwidth of 12 MB/sec, need 1.008 MBytes to fill the pipe

Setting the TCP buffer sizes



- It is critical to use the optimal TCP send and receive socket buffer sizes for the link you are using.
  - if too small, the TCP congestion window will never fully open up
  - if too large, the sender can overrun the receiver, and the TCP congestion window will shut down
- Default TCP buffer sizes are way too small for this type of network
  - default TCP send/receive buffers are typically 24 or 32 KB
  - with 24 KB buffers, can get **only 2.2%** of the available bandwidth!

Importance of TCP Tuning



TCP Buffer Tuning

- Must adjust buffer size in your applications:

```
int skt, int sndsize;
err = setsockopt(skt, SOL_SOCKET, SO_SNDBUF,
            (char *)&sndsize,(int)sizeof(sndsize));
```

and/or

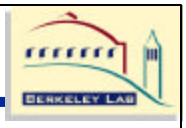
```
err = setsockopt(skt, SOL_SOCKET, SO_RCVBUF,
            (char *)&sndsize,(int)sizeof(sndsize));
```

- Also need to adjust system maximum and default buffer sizes
  - Example: in Linux, add to /etc/rc.d/rc.local

```
echo 8388608 > /proc/sys/net/core/wmem_max
echo 8388608 > /proc/sys/net/core/rmem_max
echo 65536 > /proc/sys/net/core/rmem_default
echo 65536 > /proc/sys/net/core/wmem_default
```

- For More Info, see: http://www-didc.lbl.gov/tcp-wan.html

Determining the Buffer Size



- The optimal buffer size is twice the bandwidth*delay product of the link:

  `buffer size = 2 * bandwidth * delay`

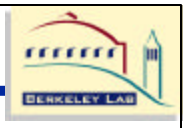
- **ping** can be used to get the delay (use the MTU size)
  - e.g.: `portnoy.lbl.gov(60)>ping -s lxplus.cern.ch 1500`

```
64 bytes from lxplus012.cern.ch: icmp_seq=0. time=175. ms
64 bytes from lxplus012.cern.ch: icmp_seq=1. time=176. ms
64 bytes from lxplus012.cern.ch: icmp_seq=2. time=175. ms
```

- pipechar or pchar can be used to get the bandwidth of the slowest hop in your path. (see next slides)
- Since ping gives the round trip time (RTT), this formula can be used instead of the previous one:

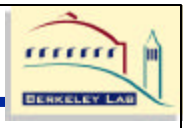
  `buffer size = bandwidth * RTT`

Buffer Size Example



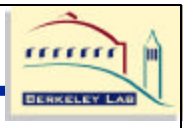
- ping time = 55 ms (CERN to Rutherford Lab, UK)
- slowest network segment = 10 MBytes/sec
  - (e.g.: the end-to-end network consists of all 100 BT ethernet and OC3 (155 Mbps)
- TCP buffers should be:
  - .055 sec * 10 MB/sec = 550 KBytes.
- Remember: default buffer size is usually only 24KB, and default maximum buffer size is only 256KB !

pchar



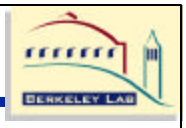
- pchar is a reimplementation of the pathchar utility, written by Van Jacobson.
  - http://www.employees.org/~bmah/Software/pchar/
  - attempts to characterize the bandwidth, latency, and loss of links along an end-to-end path
- How it works:
  - sends UDP packets of varying sizes and analyzes ICMP messages produced by intermediate routers along the path
  - estimate the bandwidth and fixed round-trip delay along the path by measuring the response time for packets of different sizes

pchar details



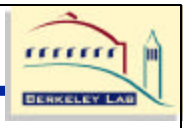
- How it works (cont.)
  - vary the TTL of the outgoing packets to get responses from different intermediate routers.
    - At each hop, pchar sends a number of packets of varying sizes
  - attempt to isolate jitter caused by network queuing:
    - determine the minimum response times for each packet size
    - performs a simple linear regression fit to the minimum response times.
    - This fit yields the partial path bandwidth and round-trip time estimates.
  - To yield per-hop estimates, pchar computes the differences in the linear regression parameter estimates for two adjacent partial-path datasets

Sample pchar output



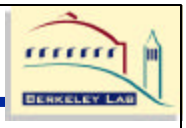

```
pchar to webr.cern.ch (137.138.28.228) using UDP/IPv4
Packet size increments by 32 to 1500
46 test(s) per repetition
32 repetition(s) per hop
 0: 131.243.2.11 (portnoy.lbl.gov)
    Partial loss:      0 / 1472 (0%)
    Partial char:      rtt = 0.390510 ms, (b = 0.000262 ms/B), r2 = 0.992548
                       stddev rtt = 0.002576, stddev b = 0.000003
    Partial queueing:  avg = 0.000497 ms (1895 bytes)
    Hop char:          rtt = 0.390510 ms, bw = 30505.978409 Kbps
    Hop queueing:      avg = 0.000497 ms (1895 bytes)
 1: 131.243.2.1 (ir100gw-r2.lbl.gov)
   Hop char:          rtt = -0.157759 ms, bw = -94125.756786 Kbps
 2: 198.129.224.2 (lbl2-gig-e.es.net)
    Hop char:          rtt = 53.943626 ms, bw = 70646.380067 Kbps
 3: 134.55.24.17 (chicago1-atms.es.net)
    Hop char:          rtt = 1.125858 ms, bw = 27669.357365 Kbps
 4: 206.220.243.32 (206.220.243.32)
    Hop char:          rtt = 109.612913 ms, bw = 35629.715463 Kbps
```


pchar output continued




```
 5: 192.65.184.142 (cernh9-s5-0.cern.ch)
    Hop char:          rtt = 0.633159 ms, bw = 27473.955920 Kbps
 6: 192.65.185.1 (cgate2.cern.ch)
    Hop char:          rtt = 0.273438 ms, bw = -137328.878155 Kbps
 7: 192.65.184.65 (cgate1-dmz.cern.ch)
    Hop char:          rtt = 0.002128 ms, bw = 32741.556372 Kbps
 8: 128.141.211.1 (b513-b-rca86-1-gb0.cern.ch)
    Hop char:          rtt = 0.113194 ms, bw = 79956.853379 Kbps
 9: 194.12.131.6 (b513-c-rca86-1-bb1.cern.ch)
    Hop char:          rtt = 0.004458 ms, bw = 29368.349559 Kbps
10: 137.138.28.228 (webr.cern.ch)
    Path length:       10 hops
    Path char:         rtt = 165.941525 ms, r2 = 0.983821
    Path bottleneck:   27473.955920 Kbps
    Path pipe:         569883 bytes
    Path queueing:     average = 0.002963 ms (55939 bytes)
```

pipechar

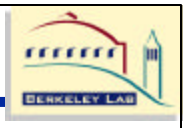


- Problems with pchar:
  - takes a LONG time to run (typically 1 hour for an 8 hop path)
  - often reports inaccurate results on high-speed (e.g.: > OC3) links.
- New tool called *pipechar*
  - http://www-didc.lbl.gov/pipechar/
  - solves the problems with pchar, but only reports the bottleneck link accurately
    - all data beyond the bottleneck hop will not be accurate
  - only takes about 2 minutes to analyze an 8 hop path

pipechar

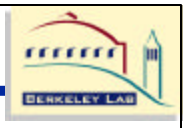
- Like pchar, pipechar uses UDP/ICMP packets of varying sizes and TTL's.
- Differences:
  - uses the jitter (caused by router queuing) measurement to estimate the bandwidth utilization
  - uses a synchronization mechanism to isolate "noise" and eliminate the need to find minimum response times
    - requires fewer tests than pchar/pathchar
  - performs multiple linear regressions on the results

Sample pipechar output



```
>pipechar pdrd10.cern.ch
From localhost:  156.522 Mbps      (157.6028 Mbps)
1: ir100gw-r2.lbl.gov               (131.243.2.1 )
|       157.295 Mbps              <4.9587% BW used>
2: lbl2-gig-e.es.net                (198.129.224.2)
|       159.364 Mbps              <21.5560% BW used>
3: chicago1-atms.es.net             (134.55.24.17)
|       45.715 Mbps               <1.6378% BW used>
4:                                  (206.220.243.32)
|       46.895 Mbps               <1.6378% BW used>
5: cernh9-s5-0.cern.ch              (192.65.184.142)
|       46.330 Mbps               <5.9290% BW used>
6: cgate2.cern.ch                   (192.65.185.1)
|       45.348 Mbps               <10.6760% BW used>
7: cgate1-dmz.cern.ch               (192.65.184.65)
|       46.041 Mbps               <10.1195% BW used>
8: b513-b-rca86-1-gb0.cern.ch       (128.141.211.1)
|       45.411 Mbps    !!!       <23.0134% BW used>
9: b513-c-rca86-1-bb1.cern.ch       (194.12.131.6)
|       46.911 Mbps               <9.3956% BW used>
10: r31-s-rca20-1-gb7.cern.ch        (194.12.129.98)
|       9.954 Mbps *** static bottle-neck 10BT
11: pcrd10.cern.ch                   (137.138.29.237)
```


Other Tools



- iperf : tool for measuring end-to-end TCP/UDP performance
  - http://dast.nlanr.net/Projects/Iperf/
- traceroute: lists all routers from current host to remote host
  - ftp://ftp.ee.lbl.gov/
- tcpdump: dump all TCP header information for a specified source/destination
  - ftp://ftp.ee.lbl.gov/

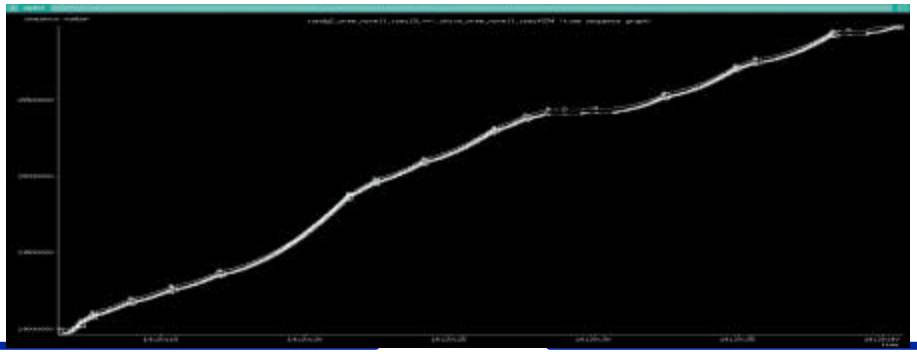
tcptrace

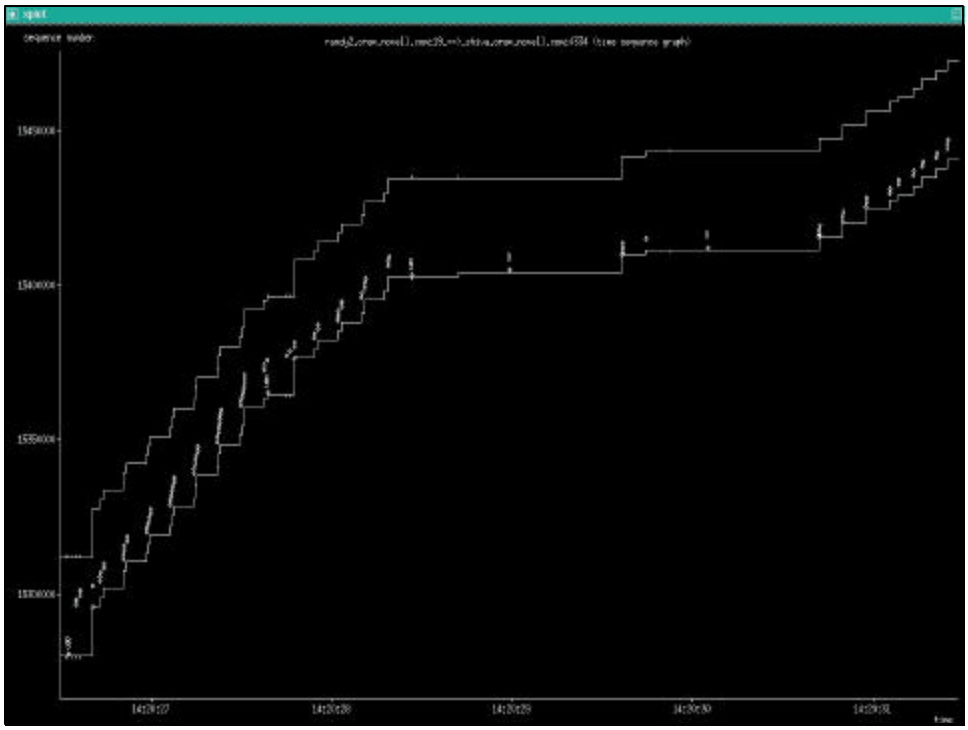
- tcptrace: format tcpdump output for analysis using xplot
  - http://jarok.cs.ohiou.edu/software/tcptrace/
  - NLANR TCP Testrig : Nice wrapper for tcpdump and tcptrace tools
    - http://www.ncne.nlanr.net/TCP/testrig/
- Sample use:

```
tcpdump -s 100 -w /tmp/tcpdump.out host hostname
tcptrace -Sl /tmp/tcpdump.out
xplot /tmp/a2b_tsg.xpl
```


tcptrace and xplot

- X axis is time
- Y axis is sequence number
  - Data packets are indicated with double arrows
  - Window and Acknowledgement numbers as staircases
- Huge range of important scales

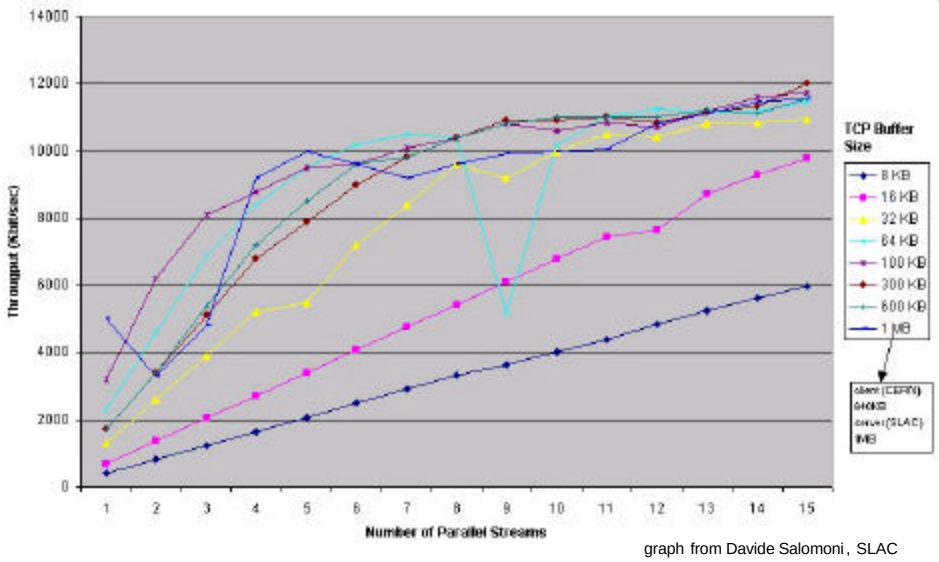




Other Tools

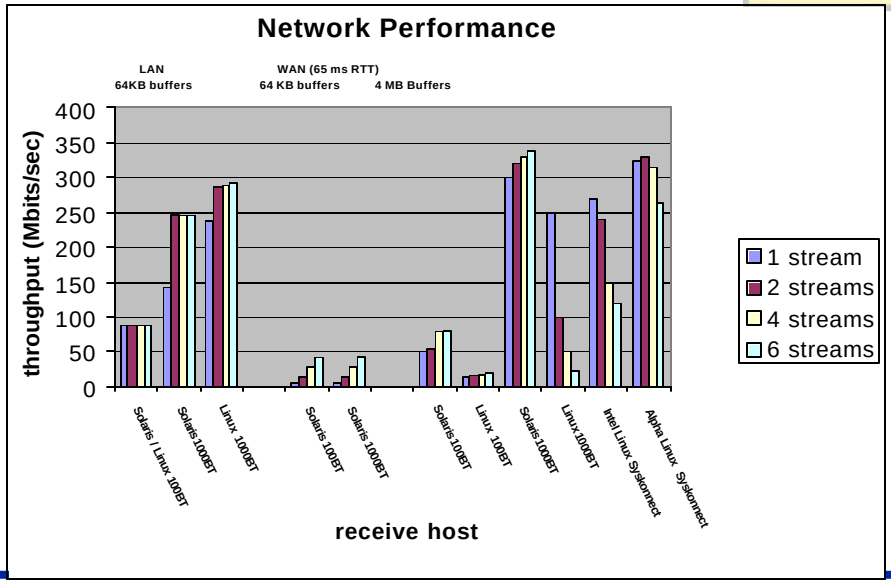
- NLANR Tools Repository:
  - Lots more network analysis tools
  - http://www.ncne.nlanr.net/tools/

Advantage of Parallel Transfers

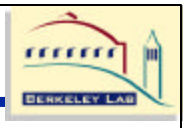


graph from Davide Salomoni, SLAC

TCP WAN Performance: Host Issues

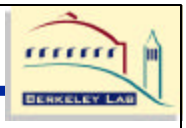


Things to Notice in Previous Slide



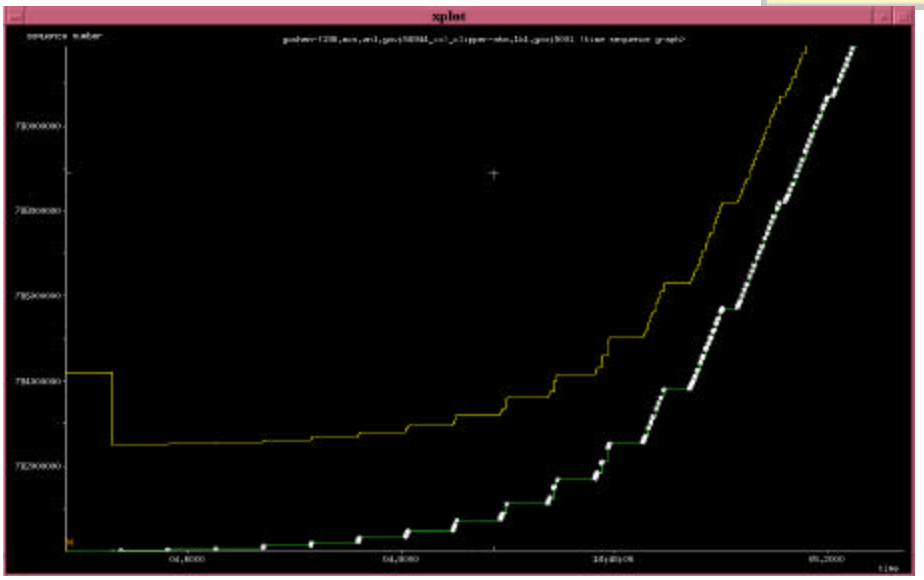
- Parallel Streams help a lot with un-tuned TCP buffers
  - and help a little with large buffers on Solaris
- Problems sending from a 1000BT host to a 100BT Linux host
- Problems sending multiple streams to a 1000BT Linux system, especially with cheap 1000BT hardware

Other TCP Issues

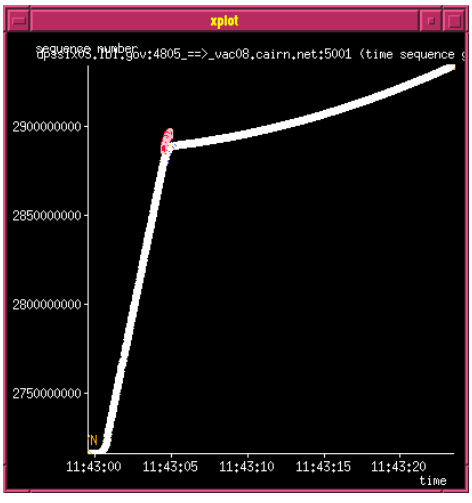


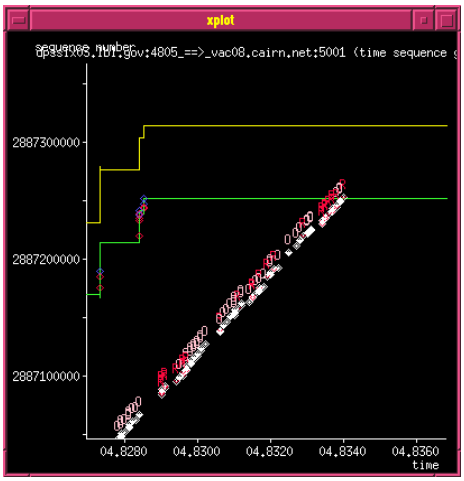
- Things to be aware of:
  - TCP slow-start
    - On the LBL to ANL link, it takes 12 RTT's to ramp up to full window size, so need to send about 10 MB of data before the TCP congestion window will fully open up.
  - router buffer issues
  - host issues

TCP Slow Start

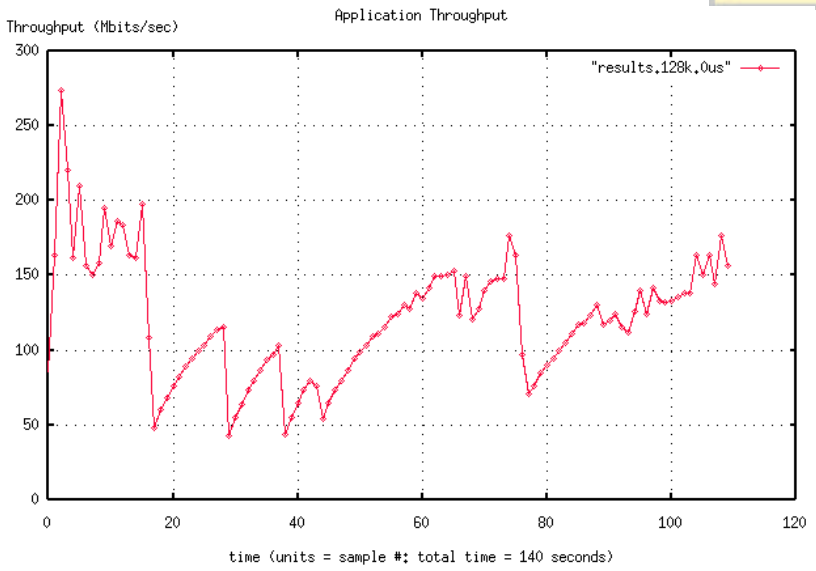


Problems with TCP over NGI-like Networks





TCP Throughput on DARPA SuperNet



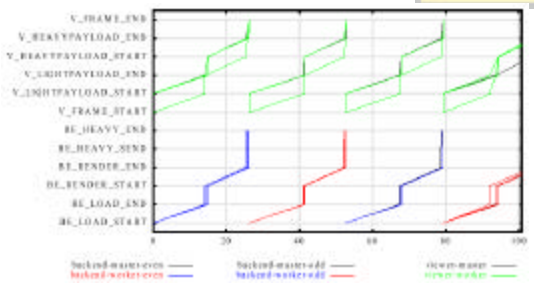
Application Performance Issues

Other Techniques to Achieve High Throughput over a WAN

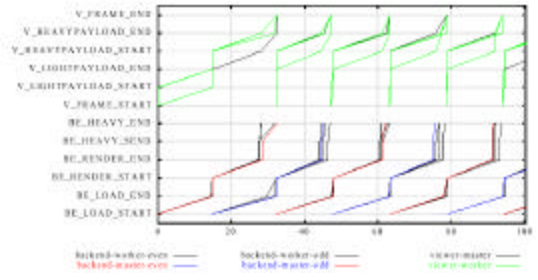
- Use multiple TCP sockets for the data stream
  - but only if your receive host is fast enough
- Use a separate thread for each socket
- Keep the data pipeline full
  - use asynchronous I/O
    - overlap I/O and computation
  - read and write large amounts of data (> 1MB) at a time whenever possible
  - pre-fetch data whenever possible
- Avoid unnecessary data copies
  - manipulate pointers to data blocks instead

Use Asynchronous I/O

- I/O followed by processing

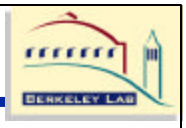


- overlapped I/O and processing

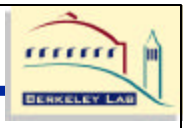


almost a 2:1 speedup

Throughput vs. Latency

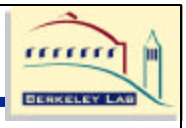


- Most of the techniques we have discussed are designed to improve throughput
- Some of these might even increase latency
  - with large TCP buffers, OS will buffer more data before sending it out.
- Goal of a Grid application programmer
  - hide latency
- However, there are some ways to help latency:
  - use separate control and data sockets
  - use TCP_NODELAY option on control socket
    - But: combine control messages together into 1 larger message whenever possible on TCP_NODELAY sockets



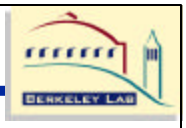
Application Analysis Using The NetLogger Toolkit

NetLogger Toolkit



- We have developed the NetLogger Toolkit (short for Networked Application Logger), which includes:
  - tools to make it easy for distributed applications to log interesting events at every critical point
  - tools for host and network monitoring
- The approach is novel in that it combines network, host, and application-level monitoring to provide a complete view of the entire system.
- This has proven invaluable for:
  - isolating and correcting performance bottlenecks
  - debugging distributed applications

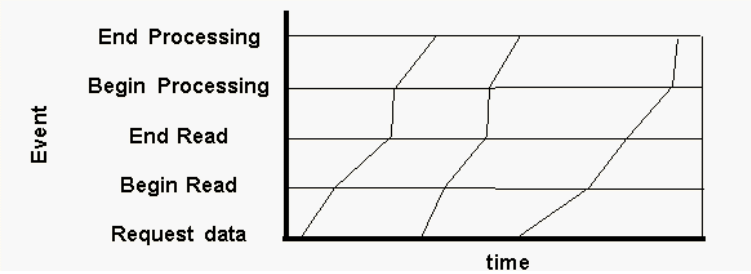
NetLogger Components



- NetLogger Toolkit contains the following components:
  - NetLogger message format
  - NetLogger client library (C, C++, Java, Perl, Python)
  - NetLogger visualization tools
  - NetLogger host/network monitoring tools
- Source code and binaries are available at:
  - http://www-didc.lbl.gov/NetLogger/
- Additional critical component for distributed applications:
  - NTP (Network Time Protocol) or GPS host clock is required to synchronize the clocks of all systems

Key Concepts

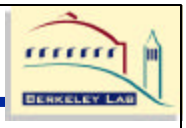
- NetLogger visualization tools are based on time correlated and/or object correlated events.
- NetLogger client libraries include:
  - precision timestamps (default = microsecond)
  - ability for applications to specify an “object ID” for related events, which allows the NetLogger visualization tools to generate an object “lifeline”



NetLogger API

- NetLogger Toolkit includes application libraries for generating NetLogger messages
  - Can send log messages to:
    - file
    - host/port (*netlogd*)
    - syslogd
    - memory, then one of the above
- C, C++, Java, Fortran, Perl, and Python APIs are currently supported

Sample NetLogger Use

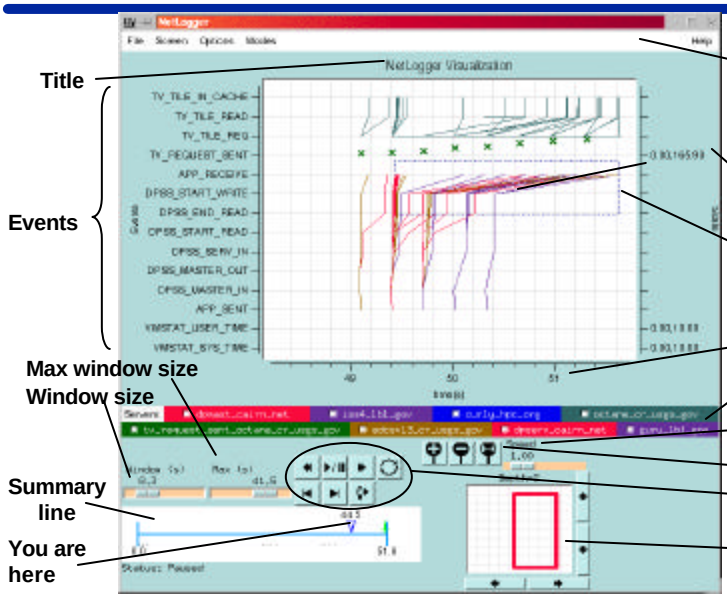


```
lp = NetLoggerOpen(method, progname, NULL,
                   hostname, NL_PORT);

while (!done)
{
        NetLoggerWrite(lp, "EVENT_START",
                       "TEST.SIZE=%d", size);

        /* perform the task to be monitored */
        done = do_something(data, size);

        NetLoggerWrite(lp, "EVENT_END");
}
NetLoggerClose(lp);
```


NetLogger Host/Network Tools

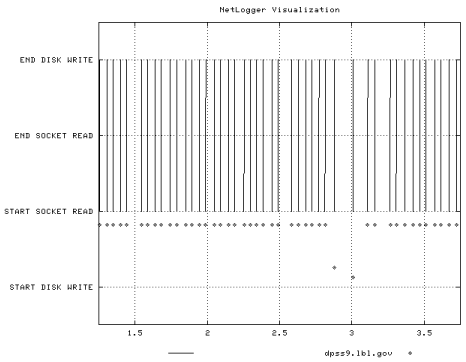
- Wrapped UNIX network and OS monitoring tools to log "interesting" events using the same log format
  - *netstat* (TCP retransmissions, etc.)
  - *vmstat* (system load, available memory, etc.)
  - *iostat* (disk activity)
  - *ping*
- These tools have been wrapped with Perl programs which:
  - parse the output of the system utility
  - build NetLogger messages containing the results

NetLogger Visualization Tool: nlv

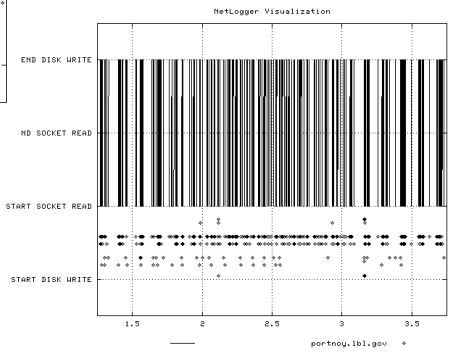


NetLogger Case Studies

Example: NetLogger of ncftp client

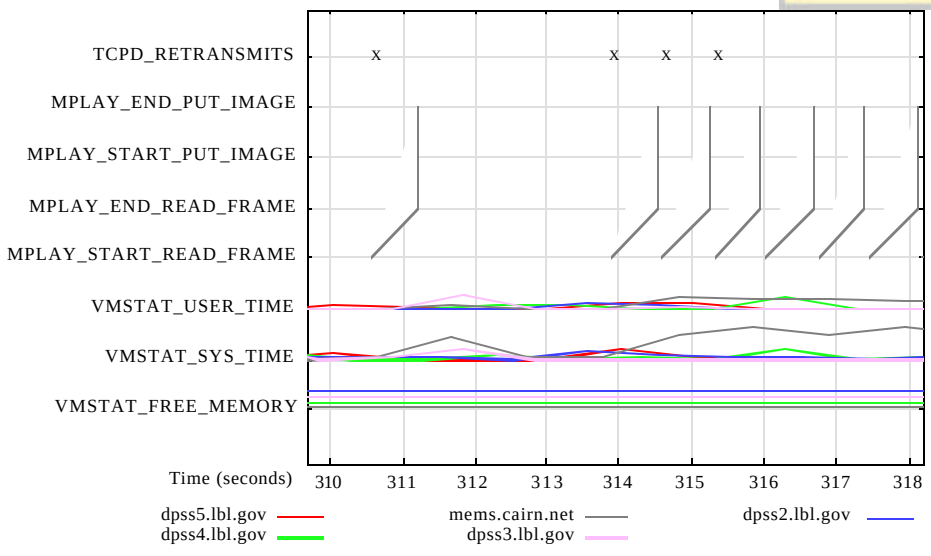


- ncftp client on a 10BT ethernet host

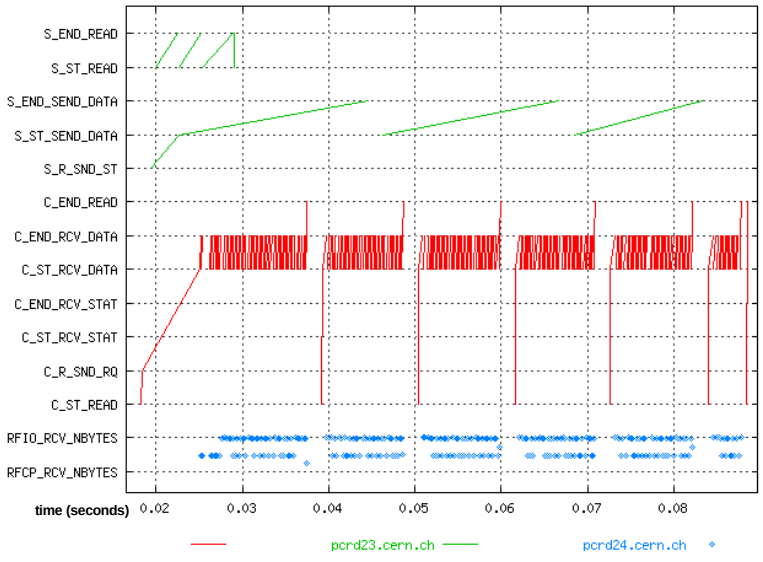


- ncftp client on a 1000BT ethernet host

Example: Combined Host and Application Monitoring

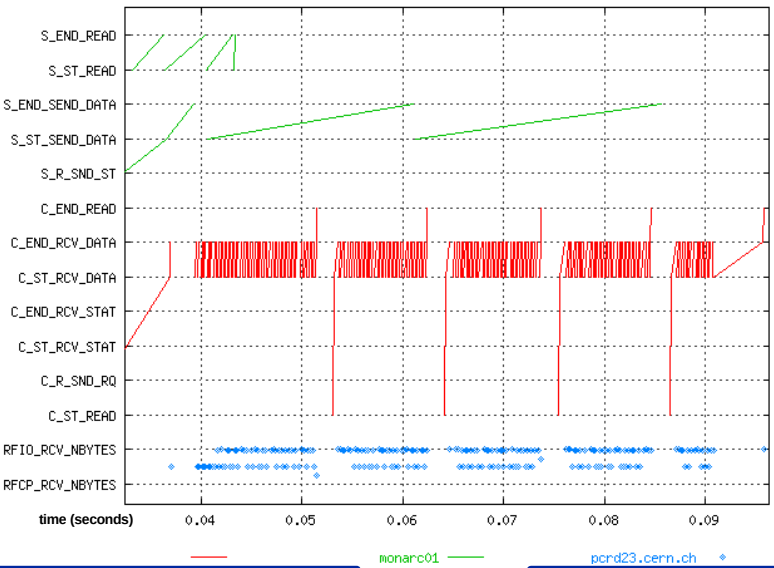


rfio get: Linux client and server



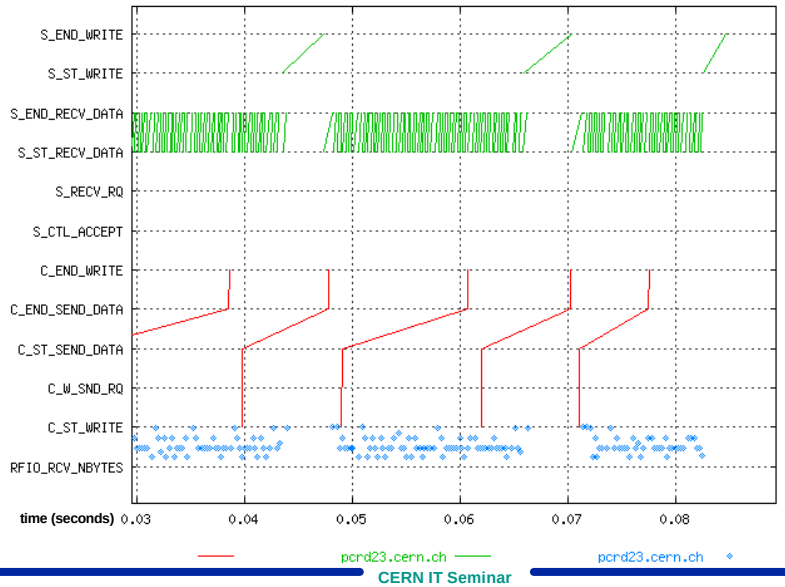
Notice: 2ms pause between network sends

rfio get: Linux client and Solaris server



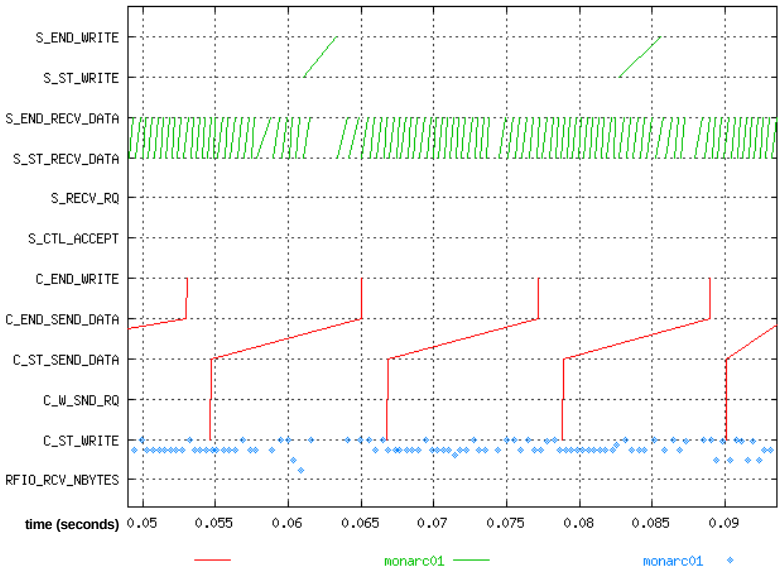
Notice: only .2ms pause between network sends

rfio put: Linux client and server



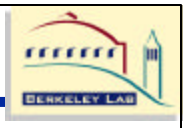
Notice: server network receive and disk write are NOT overlapped

rfio put: Linux client and Solaris server



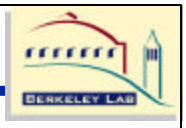
Notice: server network receive and disk write are overlapped

Getting NetLogger



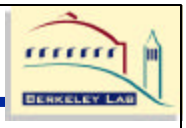
- Source code and binaries are available at:
  - http://www-didc.lbl.gov/NetLogger
- Client libraries run on all Unix platforms
- Solaris, Linux, and Irix versions of *nlv* are currently supported

Conclusions



- Tuning Grid Applications is hard!
  - usually not obvious what the bottlenecks are
- Tuning TCP is hard!
  - no single solution fits all situations
    - need to be careful TCP buffer are not too big or too small
    - sometimes parallel streams help throughput, sometimes they hurt

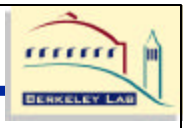
Conclusions



So what to do?

- design your grid application to be as flexible as possible
  - make it easy for clients/users to set the TCP buffer sizes
  - make it possible to turn on/off parallel socket transfers
    - probably off by default
- design your application for the future
  - even if your current WAN connection is only 45 Mbps (or less), some day it will be much higher, and these issues will become even more important

For More Information



Email:bltierney@lbl.gov

http://www-didc.lbl.gov/NetLogger/

- download NetLogger components
- tutorial
- user guide

http://www-didc.lbl.gov/tcp-wan.html

- links to all network tools mentioned here
- sample TCP buffer tuning code, etc.,