NetLogger: Distributed System Monitoring and Analysis Tools

Brian L. Tierney
Dan Gunter

Data Intensive Distributed Computing Group
Lawrence Berkeley National Laboratory

Outline

- Why are we here?
  - What is NetLogger?
  - What is NetLogger good for?
  - What is NetLogger not good for?
- NetLogger Components
  - message format
  - instrumentation library
  - system monitoring tools
  - visualization tools
- Instrumentation Techniques
- Case Studies
  - HPSS Storage Manager
  - Radiance luminosity application
  - Parallel remote data server (DPSS)
- Current Work
  - Monitoring Agents

Overview

- The Problem
  - When building distributed systems, we often observe 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, and so on
- The Solution:
    - Highly instrumented systems with precision timing information and analysis tools

Bottleneck Analysis

- Distributed system users and developers often assume the problem is network congestion
  - This is often not true

- In our experience tuning distributed applications, performance problems are due to:
  - network problems: 40%
  - host problems: 20%
  - application design problems/bugs: 40%
    - 50% client , 50% server
- Therefore it is equally important to instrument the applications

Motivation

- **To characterize the performance of distributed applications, we have developed a methodology for detailed, *end-to-end, top-to-bottom monitoring* and analysis of significant events**
  - **this allows coordinated monitoring of applications, networks, and hosts**

- **This has proven invaluable for:**
  - **isolating and correcting performance bottlenecks**
  - **debugging distributed applications**

NetLogger Toolkit

- We have developed the NetLogger Toolkit
  - A set of tools which make it easy for distributed applications to log interesting events at every critical point
  - NetLogger also includes 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.

Why “NetLogger”?

- The name “NetLogger” is somewhat misleading
  - Should really be called: “Distributed Application, Host, and Network Logger”

- “NetLogger” was a catchy name that stuck

When to use NetLogger

- When you want to:
  - do performance/bottleneck analysis on distributed applications
  - determine which hardware components to upgrade to alleviate bottlenecks
  - do real-time or post-mortem analysis of applications
  - correlate application performance with system information (ie: TCP retransmission's)
- works best with applications where you can follow a specific item (data block, message, object) through the system

When NOT to use NetLogger

- **Analyzing massively parallel programs (e.g.: MPI)**
  - **Current visualization tools don’t scale beyond tracking about 20 types of events at a time**

- **Analyzing many very short events**
  - **system will become overwhelmed if too many events**
  - **we typically use NetLogger to monitor events that take > .5 ms**
  - **e.g: probably don’t want to use to instrument the UNIX kernel**

NetLogger Components

- NetLogger Toolkit contains the following components:
  - NetLogger message format
  - NetLogger client library
  - NetLogger visualization tools
  - NetLogger host/network monitoring tools

- Additional critical component for distributed applications:
  - NTP (Network Time Protocol) or GPS host clock is required to synchronize the clocks of all systems

NetLogger Message Format

- We are using the IETF draft standard Universal Logger Message (ULM) format:
  - a list of “field=value” pairs
  - required fields: DATE, HOST, PROG, and LVL
    - —DATE = YYYYMMDDHHSS.SSSSSS
    - —PROG: program name
    - —LVL is the severity level (Emergency, Alert, Error, Usage, etc.)
  - followed by optional user defined fields
  - http://www.ietf.org/internet-drafts/draft-abela-ulm-05.txt
- NetLogger adds this required fields:
  - NL.EVNT, a unique identifier for the event being logged
    - —e.g.: SERVER_IN, VMSTAT_USER_TIME,
      NETSTAT_RETRANSSEG

NetLogger Message Format

- Sample NetLogger ULM event:

```
DATE=19980430133038.055784 HOST=foo.lbl.gov
  PROG=testprog LVL=Usage NL.EVNT=SEND_DATA
  SEND.SZ=49332
```

  – This says program named *testprog* on host *foo.lbl.gov* performed event named SEND_DATA, size = 49332 bytes, at the time given

- User-defined data elements (any number) are used to store information about the logged event - for example:
  - NL.EVNT=SEND_DATA SEND.SZ=49332
    - —the number of bytes of data sent
  - NL.EVNT=NETSTAT_RETRANSSEGS NS.RTS=2
    - —the number of TCP retransmits since the previous event

NetLogger “Mission”

- Our mission is to get everyone to use the NetLogger/ULM format for logging
  - ULM will hopefully become a “standard”
  - This way we can all share log file management and visualization tools
- Probably not realistic
  - Working on filters to convert the following to/from NetLogger format
    - Pablo
    - NWS
    - Gloperf
    - others?

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, and Perl, and Python APIs are currently supported

NetLogger API

- Only 6 simple calls:
  - NetLoggerOpen()
    - create NetLogger handle
  - NetLoggerWrite()
    - get timestamp, build NetLogger message, send to destination
  - NetLoggerGTWrite()
    - must pass in results of Unix gettimeofday() call
  - NetLoggerFlush()
    - flush any buffered message to destination
  - NetLoggerSetLevel()
    - set ULM severity level
  - NetLoggerClose()
    - destroy NetLogger handle

NetLogger API

- **Open calls:**

```
NLhandle   *lp = NULL;

/* log to a local file */
lp = NetLoggerOpen(NL_FILE, program_name, log_filename,
   NULL, 0);

/* log to syslog */
lp = NetLoggerOpen(NL_SYSLOG, program_name, NULL,
   NULL, 0);

/* log to “netlogd” on the specified host/port */
lp = NetLoggerOpen(NL_HOST, program_name, NULL,
   hostname,  DPSS_NETLOGGER_PORT);

/* log to memory, then flush to host/port */
lp = NetLoggerOpen(NL_HOST_MEM, program_name, NULL,
   hostname, DPSS_NETLOGGER_PORT);
```

NetLogger Write Call

- Creates and Writes the log event:

```
NetLoggerWrite(nl, “EVENT_NAME”,
  “EVENTID=%d F2=%d F3=%s F4=%.2f”, id,
  user_data, user_string, user_float);
```

  - timestamping is automatically done by library
  - the “event name” field is required, all other fields are optional
  - Note: not thread-safe: threaded programs must put a mutex lock around this call

- Example:

```
NetLoggerWrite(nl, “HTTPD.START_DISK_READ”,
  “HTTPD.FNAME=%s HTTPD.HOST=%s”, fname,
  hostname);
```

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);
```

netlogd

- Use *netlogd* to collect NetLogger messages at a central host
  - use to avoid the need to sort/merge several log files from several places
- Can also use *netlogd* to try to adjust time values for clock skew
  - useful if can’t get NTP installed
  - allows clients to adjust all timestamps relative to the *netlogd* host’s clock
  - accurate only to about 5 ms, and assumes all clients have the same latency to the *netlogd* host
  - basically a major HACK, but can be useful

Logging to Memory

- Use the NL_HOST_MEM option to send NetLogger events to memory if you are:
  - monitoring bursts of events with a duration < 1 ms
- Flushing of events to disk or network will occur:
  - automatically when specified memory block full
  - when calling NetLoggerFlush()
  - when calling NetLoggerClose()

- Size of memory buffer specified by NL_MAX_BUFFER in netlogger.h
  - default = 10,000 messages (typical message size is 128 bytes)

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, paging, etc.)
  - *iostat* (disk activity)
  - *ping*

- These tools have been wrapped with Perl or Java programs which:
  - parse the output of the system utility
  - build NetLogger messages containing the results

NetLogger Host Monitoring Tools

Usage:

```
nl_vmstat [-d #][-t N][-n][-f logfile] [-m # [host]]
       [-d N] output log messages every N msecs (default = 1000)
       [-t N] run for N minutes and exit (default = run for 60 min)
       [-n ] only log if value changes
       [-f logfile] write to file named logfile
       [-m N [host]] logging method: 0 = file, 1 = syslog, 2 = host
```

Sample NetLogger System Monitoring Tool

- Example: nl_vmstat -t 60 -d 5000 -m 2 logger.lbl.gov
  - Java program will exec *vmstat* every 5 seconds for 1 hour, and send the results to *netlogd* on host logger.lbl.gov
  - Generates the following information:
    - CPU usage by User
    - CPU usage by System
- NetLogger Messages:

```
DATE=19990706125055.891620 HOST=portnoy.lbl.gov
  PROG=nl_vmstat LVL=Usage NL.EVNT=VMSTAT_USER_TIME
  VMS.VAL=9
DATE=19990706125055. 891112 HOST=portnoy.lbl.gov
  PROG=nl_vmstat LVL=Usage NL.EVNT=VMSTAT_SYS_TIME
  VMS.VAL=5
```

NetLogger Network Tools

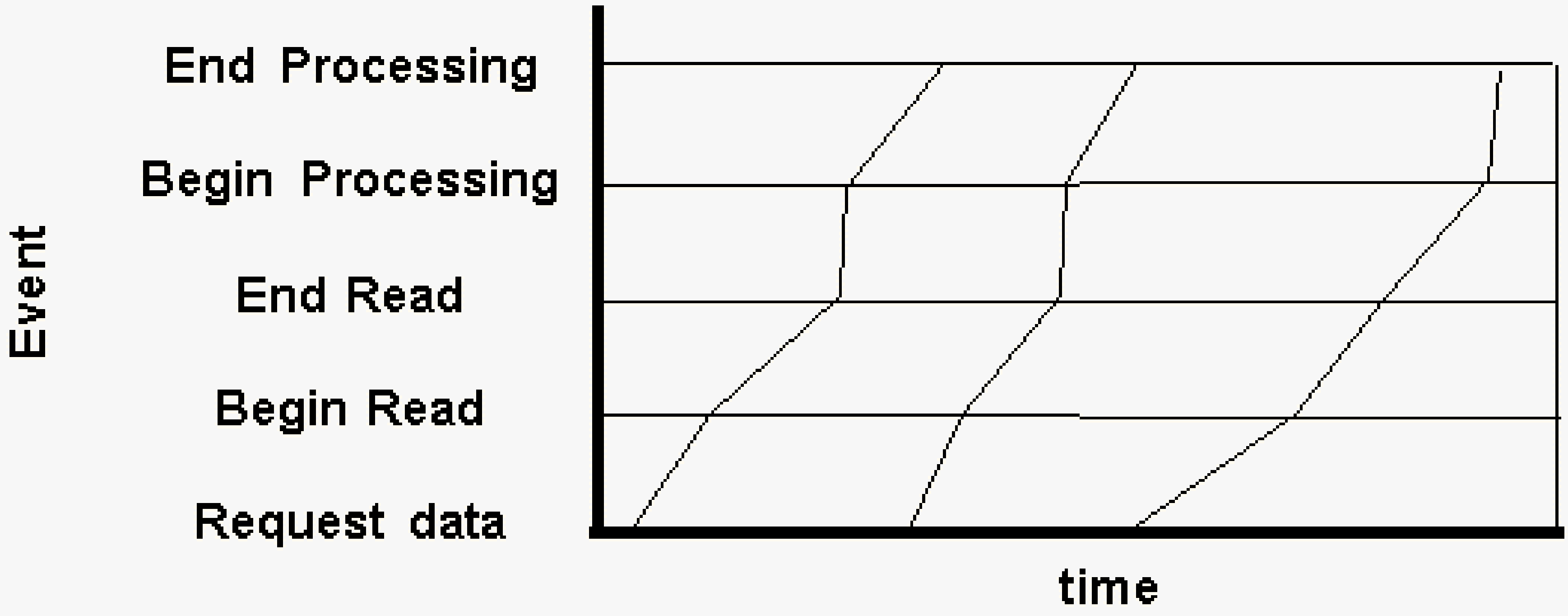
- NetLogger tool for SNMP queries
  - Usage: nl_snmpget hostname object [port]

- Examples:
  - host monitoring
    - `nl_snmpget unix_host sysName`
      - Returns: system.sysName.0 = wakko.lbl.gov
  - router monitoring
    - `nl_snmpget routername ipInDelivers 3`
      - Returns: tcp.tcpInErrs.3 = 4000
  - ATM switch monitoring
    - `nl_snmpget switchname sonetLineFEBEs`
    - `nl_snmpget switchname portTransmittedCells`

Other Tools

- NetLogger also includes a set of PERL scripts to
  - sort files by timestamp and/or other ULM field
  - merge files
  - generate *gnuplot* formatted file from a NetLogger file

NetLogger Event “Life Lines”



Event ID

- In order to associate a group of events into a "lifeline", you must assign an event ID to each NetLogger event
- Sample Event Ids
  - file name
  - block ID
  - frame ID
  - user name
  - host name
  - etc.

Sample NetLogger Use with Event IDs

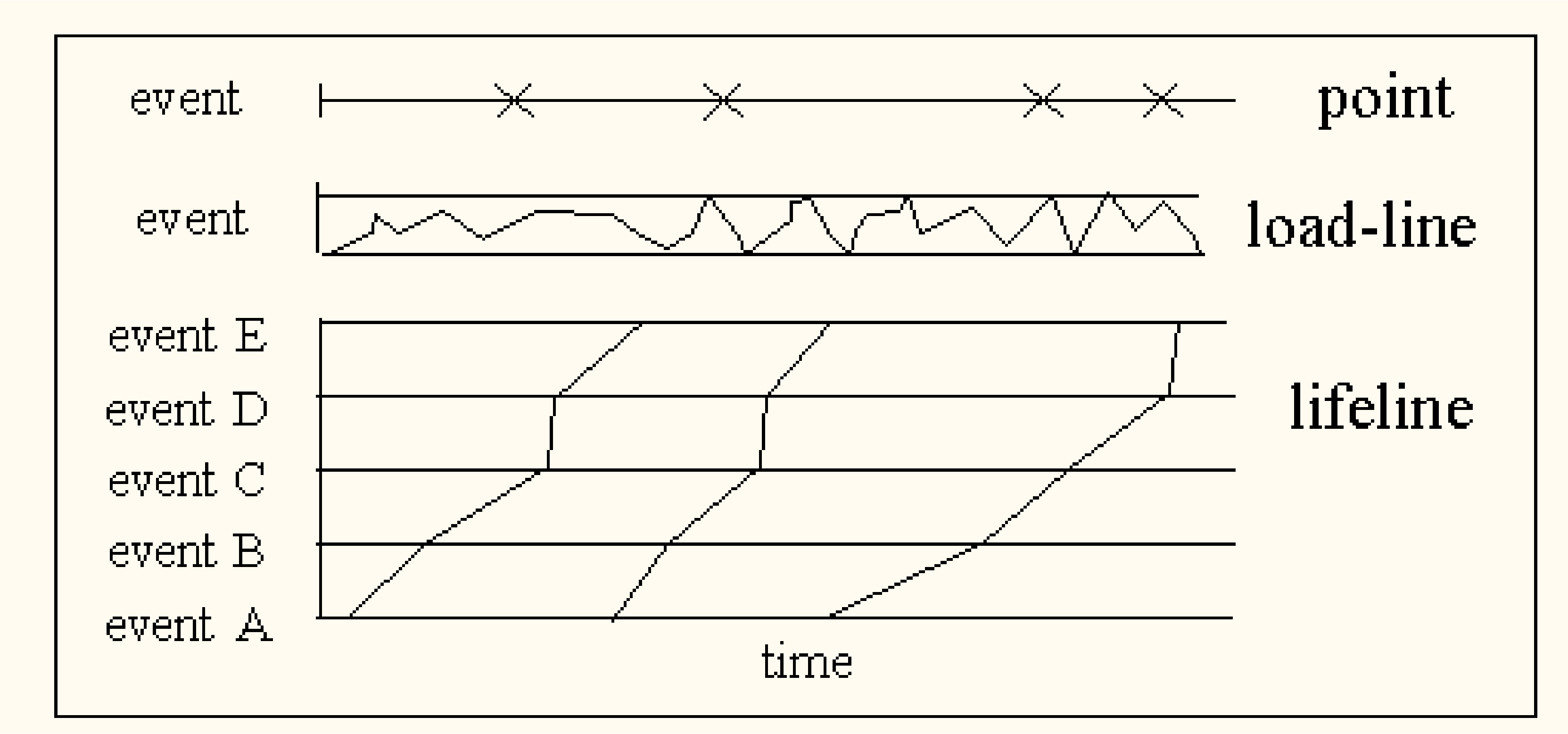
```
lp = NetLoggerOpen(method, progname, NULL, hostname, NL_PORT);
for (i=0; i< num_blocks; i++)  {
    NetLoggerWrite(lp, “START_READ”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    read_block(i);
    NetLoggerWrite(lp, “END_READ”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    NetLoggerWrite(lp, “START_PROCESS”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    process_block(i);
    NetLoggerWrite(lp, “END_PROCESS”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    NetLoggerWrite(lp, “START_SEND”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    send_block(i);
    NetLoggerWrite(lp, “END_SEND”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
}
NetLoggerClose(lp);
```

NetLogger Visualization Tools

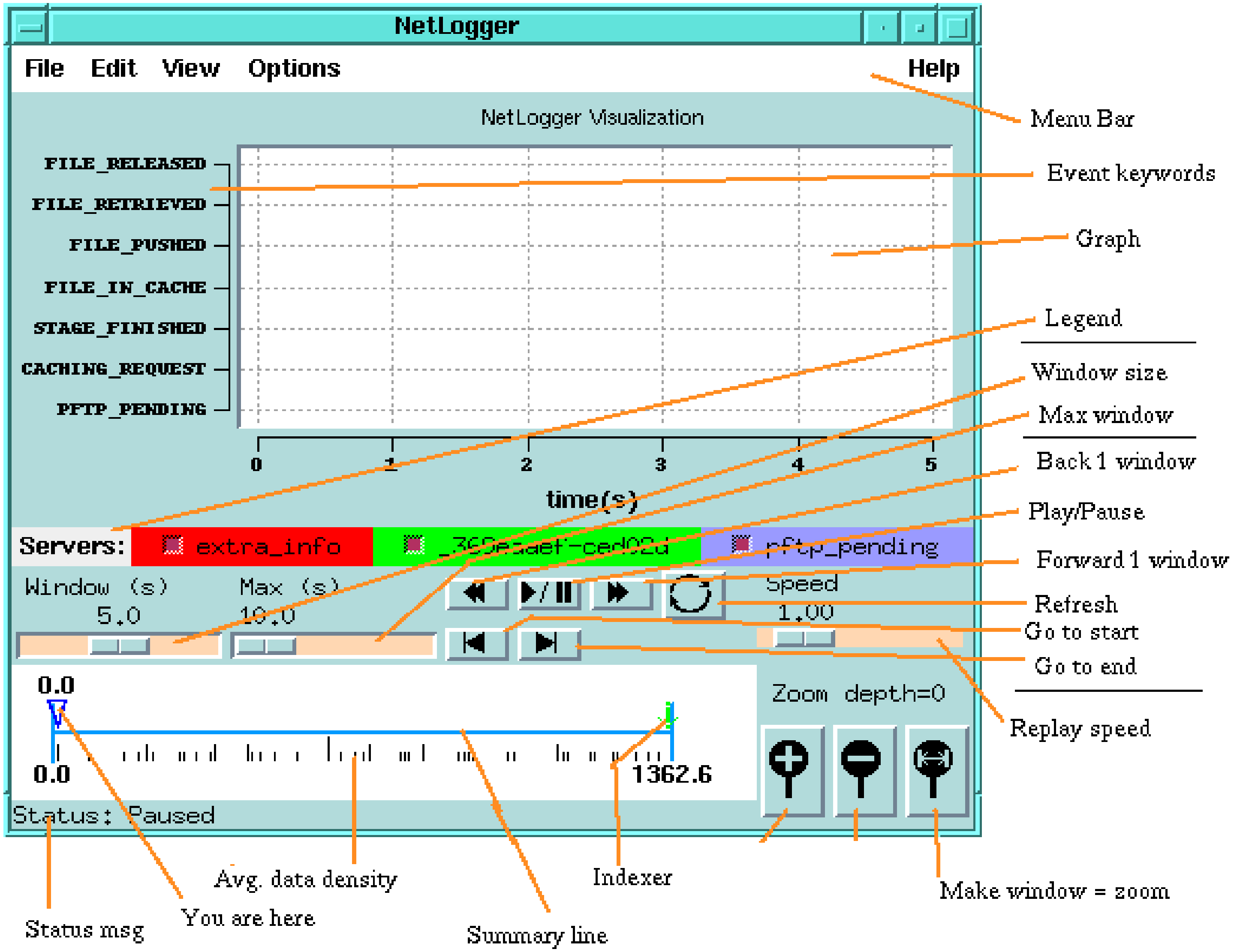
- **Exploratory, interactive analysis of the log data has proven to be the most important means of identifying problems**
  - **this is provided by *nlv* (NetLogger Visualization)**
- ***nlv* functionality:**
  - **can display several types of NetLogger events at once**
  - **user configurable: which events to plot, and the type of plot to draw (lifeline, load-line, or point)**
  - **play, pause, rewind, slow motion, zoom in/out, and so on**
  - ***nlv* can be run post-mortem or in real-time**
    - **real-time mode done by reading the output of *netlogd* as it is being written**

NLV Graph Types

- **nlv supports graphing of “points”, load-lines, and lifelines**



NLV



Window (s) 5.0

Max (s) 10.0

Speed 1.00

Zoom depth=0

Status: Paused

Menu Bar

Event keywords

Graph

Legend

Window size

Max window

Back 1 window

Play/Pause

Forward 1 window

Refresh

Go to start

Go to end

Replay speed

Make window = zoom

Indexer

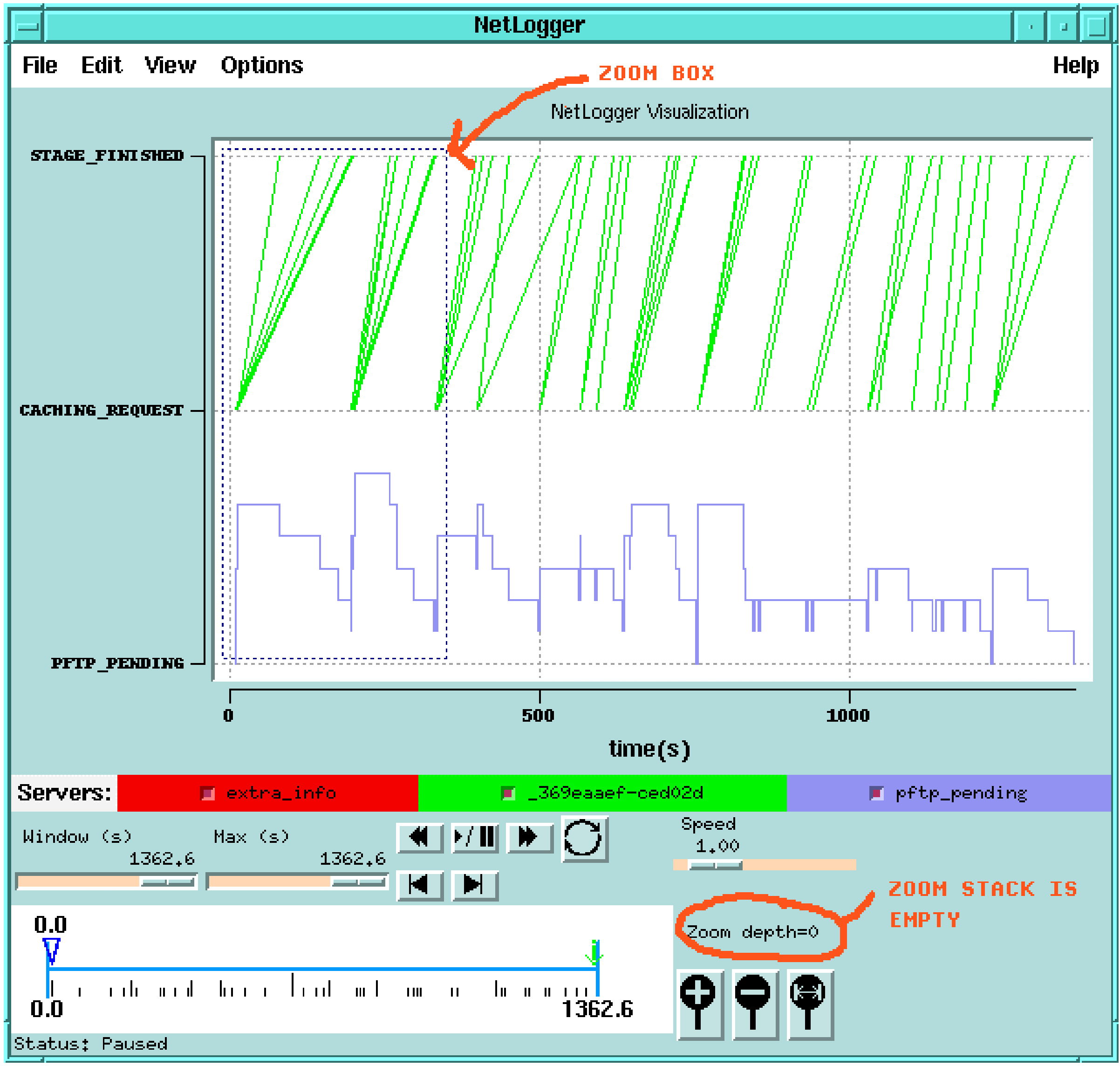
Avg. data density

Summary line

You are here

Status msg

NLV Zoom Feature



ZOOM BOX

Window (s) 1362.6

Max (s) 1362.6

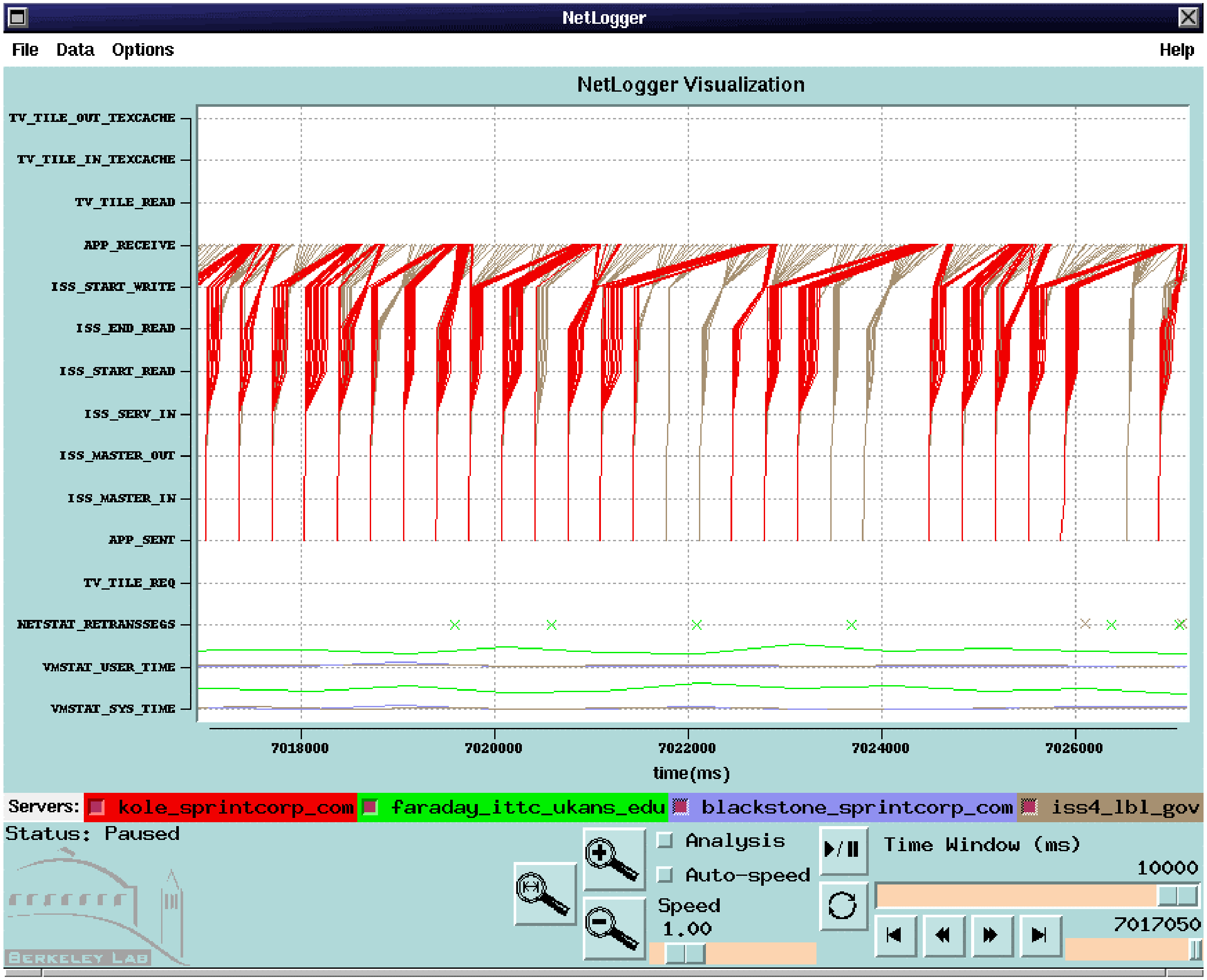
Speed 1.00

Zoom depth=0

ZOOM STACK IS EMPTY

Status: Paused

NLV Graph Types



Servers: 🟥 kole_sprintcorp_com 🟥 faraday_ittc_ukans_edu 🟥 blackstone_sprintcorp_com 🟥 iss4_lbl_gov

Status: Paused

□ Analysis

□ Auto-speed

Speed 1.00

Time Window (ms) 10000

7017050

NLV Configuration

- NLV is very flexible, with many options settable in the configuration file.
- Format:

```
set +/-eventset_name
type <line,point,load>
id [ list of ULM field names used to determine which
  NetLogger messages get grouped into the same graph
  primitive ]
group [list of ULM field names which will be mapped to the
  same color]
val field_name min_val max_val
annotate [ list of field names to display in with annotate
  option ]
[ list of all event ID’s in this lifeline ]
```

- Each nlv graph object needs to be defined by a “set”
- Events and event-sets both use "+" and "-" to indicate default visibility

NLV Configuration

- Events and eventsets are "stacked" in nlv in the order given in the configuration file
- Other Keywords:
  - groupalias A [b c d]
    - list of fields values for the “group” event that can be considered equivalent
    - e.g.: any "hostname" equal to b, c, or d will be displayed and colored as a member of group A
- Specific config file examples will be shown with each sample application later in the talk

Example NLV Configuration

```
# display vmstat info as a “loadline”
set +VMSTAT
type load
# loadline constructed from messages with the same HOST and NL.EVNT
id [ HOST NL.EVNT ]
# messages with the same HOST get the same color
group HOST
#list of NL.EVNT values in this set_
[ +VMSTAT_SYS_TIME +VMSTAT_USER_TIME ]


# display netstat TCP retransmits as a “point”
set +NETSTAT
type point
# ignore values outside the range 0 to 999
val NS.VAL 0.0 999.0
# point constructed from messages from the same HOST and PROG
id [ HOST PROG ]
# messages with the same HOST get the same color
group HOST
[ +NETSTAT_RETRANSSEGS ]
```

Example NLV Configuration

```
# display server data as a “lifeline”
set +SERVER_READ
type line

# lifeline constructed from messages from the same client
  and server
id [ CLIENT_HOST DPSS.SERV ]

# messages with the same DPSS.SERV get the same color
group DPSS.SERV

[ +APP_SENT +DPSS_SERV_IN +DPSS_START_READ
+DPSS_END_READ +DPSS_START_WRITE +APP_RECEIVE ]
```

Network Time Protocol

- For NetLogger timestamps to be meaningful, all systems clocks must be synchronized.
  - NTP is used to synchronize time of all hosts in the system.
    - NTP is from Dave Mills, U. of Delaware (http://www.eecis.udel.edu/~ntp/)
  - Must have NTP running on one or more primary servers, and on a number of local-net hosts, acting as secondary time servers

- Could also place GPS clocks on every host for even more accurate clocks

How to Instrument Your Application

- You’ll probably want to add a NetLogger event to the following places in your distributed application:
  - before and after all disk I/O
  - before and after all network I/O
  - entering and leaving each distributed component
  - before and after any significant computation
    - e.g.: an FFT operation
  - before and after any significant graphics call
    - e.g.: certain CPU intensive OpenGL calls
- This is usually an iterative process
  - add more NetLogger events as you zero in on the bottleneck

Does NetLogger affect application performance?

- There are several things to be careful of when doing this type of monitoring:
  - If logging to disk, don’t log to a nfs mounted disk
    - best to log to /tmp, which may actually be RAM (Solaris)
  - Probably don’t want to send log messages to a slow (i.e.: 10BT) or congested network, as you’ll just make it worse
    - log to a local file instead

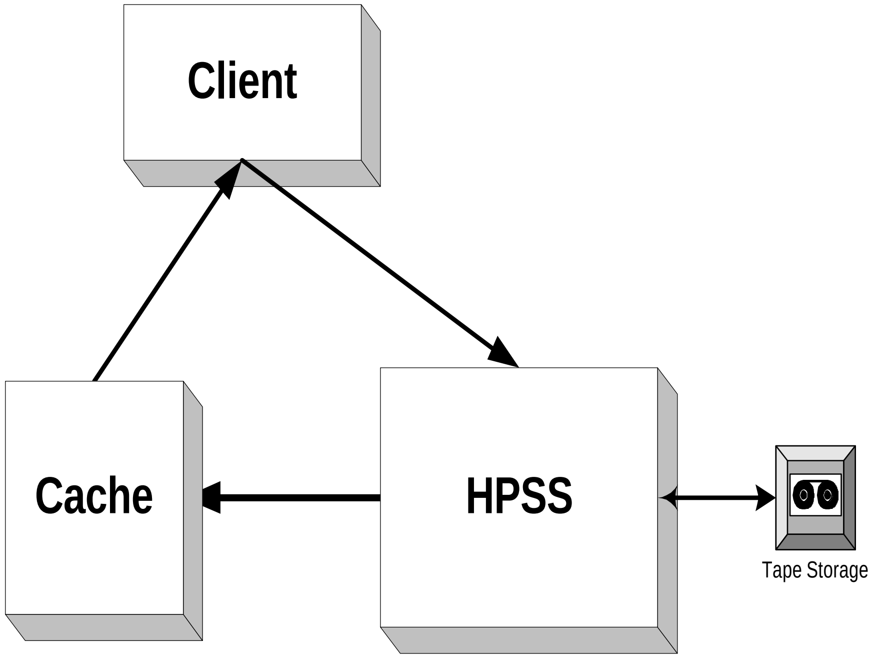
Sample NetLogger Analysis

- We next show how NetLogger was added to 3 different applications:
  - A cache manager for the HPSS
  - A remote visualization application
  - A HENP data analysis package accessing parallel remote data service

Example 1: HPSS Storage Manager Application

- NetLogger was used to test and verify the results of a Storage Access Coordination System (STACS) by LBNL’s Data Management Group
- STACS is designed to optimize the use of a disk cache with an HPSS Mass Storage system, and tries to minimize tape mount requests by clustering related data on the same tape
- NetLogger was used to look at:
  - per-query latencies
  - to show that subsequent fetches of spatially clustered data "hit" in the cache.
- (http://gizmo.lbl.gov/sm/)

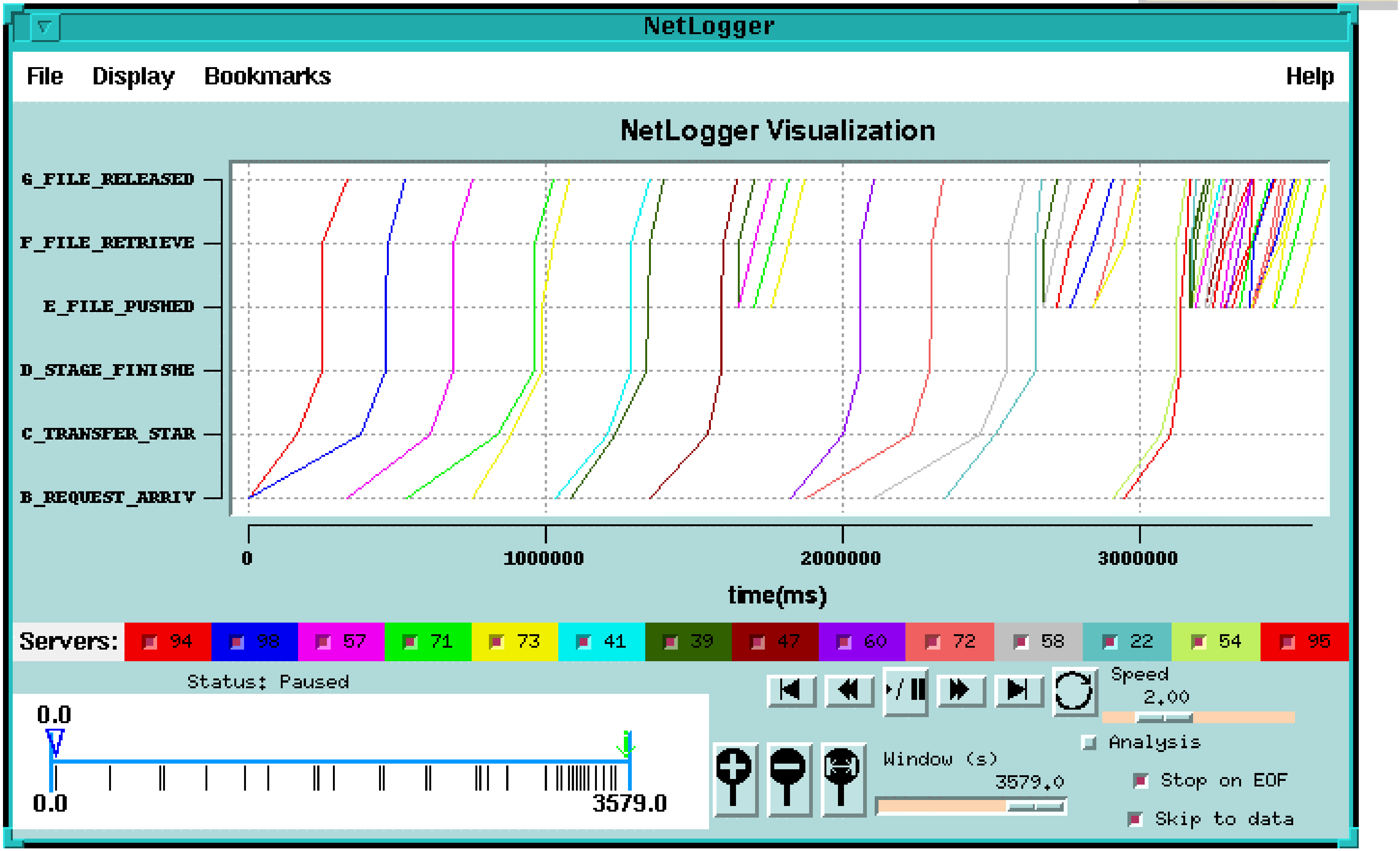
STACS Instrumentation Points



Monitoring Points:

A) request arrives at HPSS
B) start transfer from tape
C) tape transfer finished
D) file available to client
E) file retrieved by client
F) file released by client

NLV for STACS: Tracking File Requests



Status: Paused

Speed 2.00

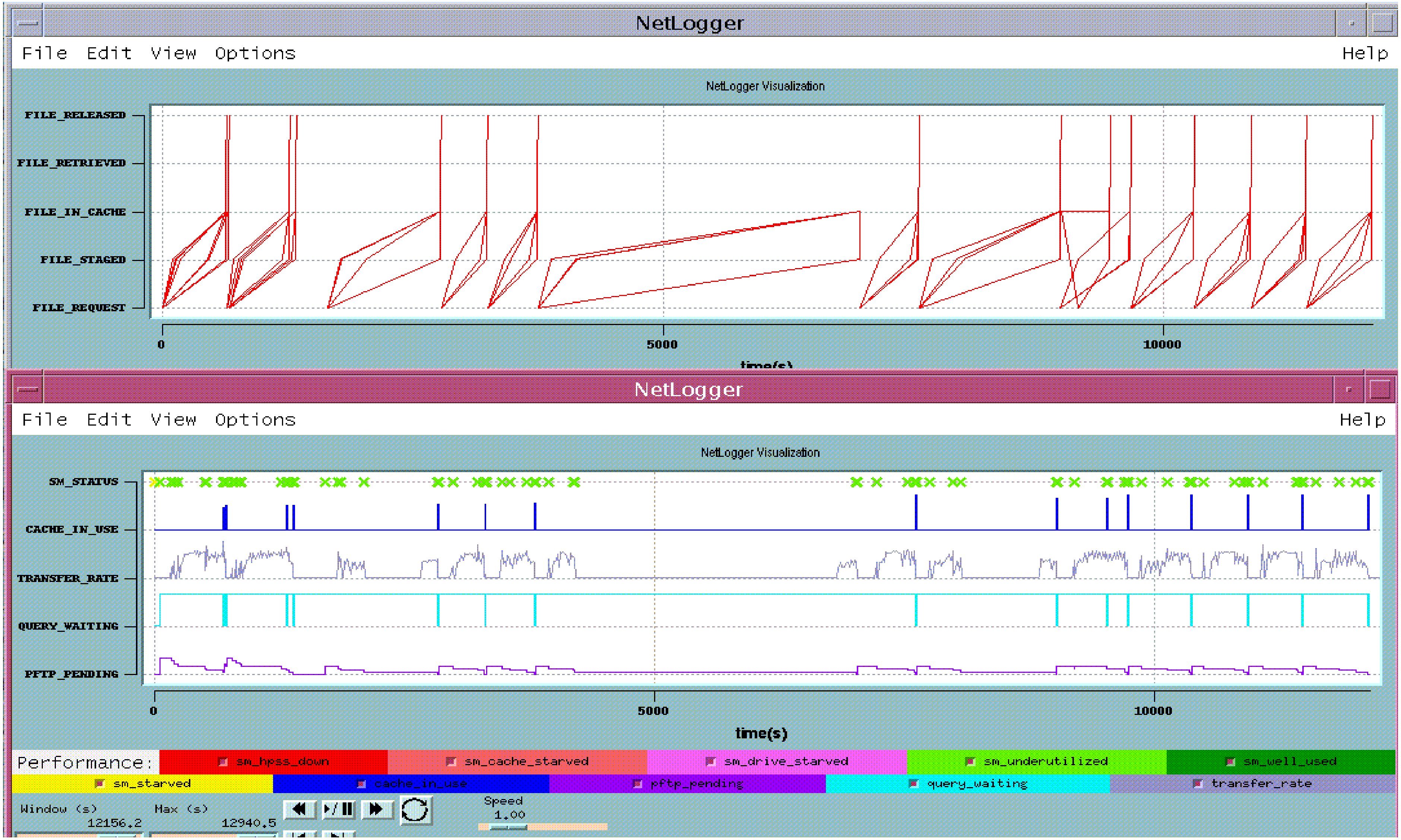
Window (s) 3579.0

□ Analysis

☑ Stop on EOF

☑ Skip to data

Tracking Files and System Performance



Window (s) 12156.2

Max (s) 12940.5

Speed 1.00

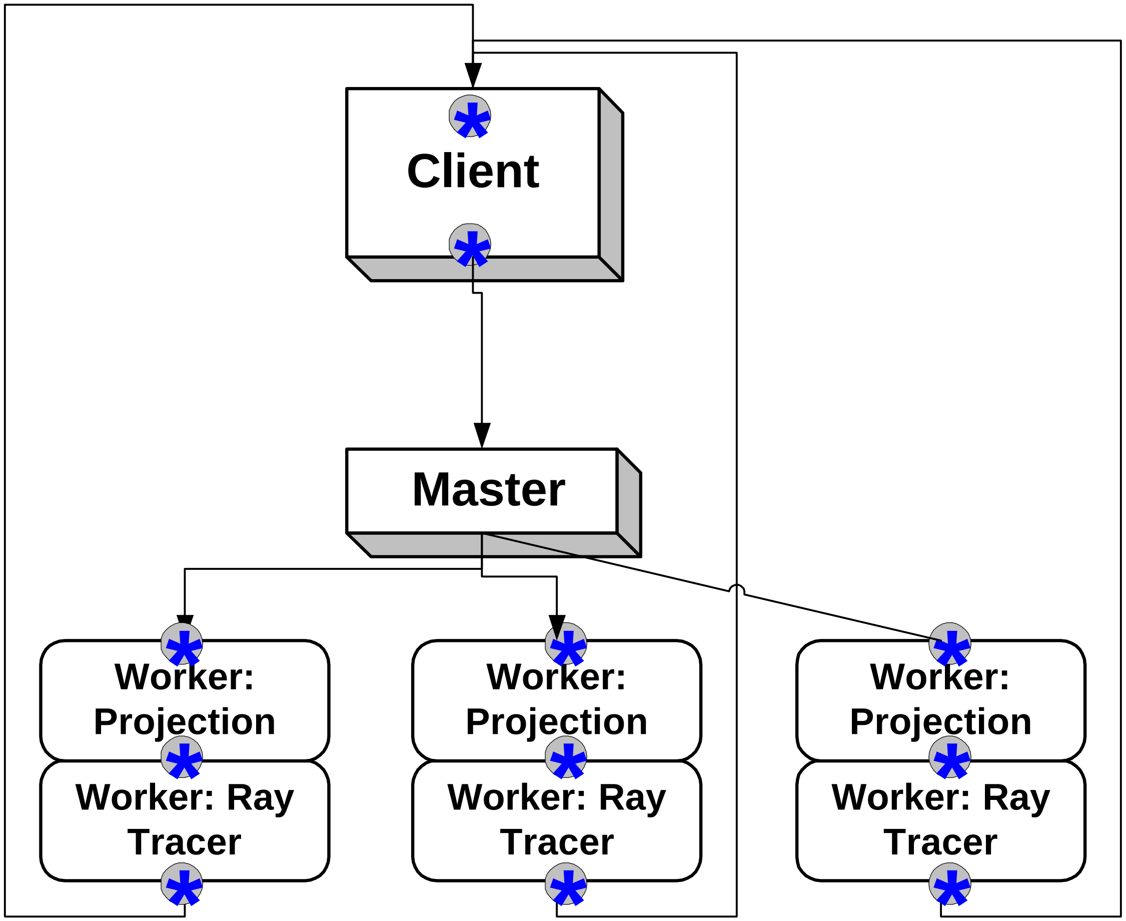
NLV Configuration File for this Application

```
set +SMANAGER
type line
# lifeline defined by messages for the same file and
  a given query ID number
id [ QUERY FID ]
# color lines by query ID
group QUERY
[
+B_REQUEST_ARRIVED
+C_TRANSFER_STARTED
+D_STAGE_FINISHED
+E_FILE_PUSHED
+F_FILE_RETRIEVED
+G_FILE_RELEASED
]
```

Example 2: Parallel Visualization Application

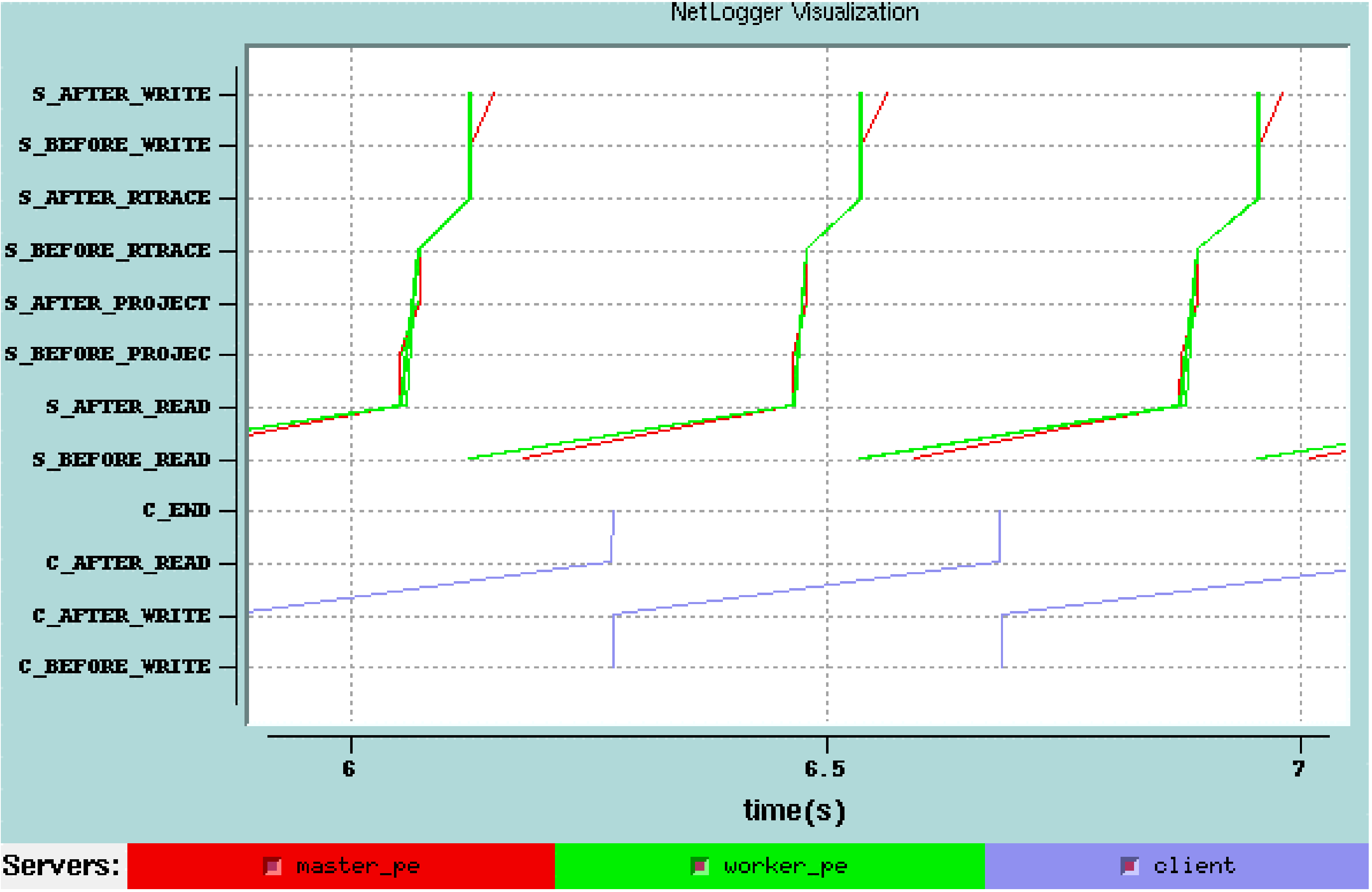
- **Radiance is a suite of programs for the analysis and visualization of lighting in design.**
  - **Input includes the scene geometry, materials, luminance, time, date, and sky conditions**
- **Radiance has been adapted at LBNL to run on multiple cluster nodes**
  - **The image is broken into many small pieces, and illumination calculations are performed for each piece independently**
- **Used NetLogger to measure:**
  - **overall system throughput**
  - **latency for each stage of getting data, processing it, and writing it**
  - **patterns of latency which reflect resource contention and other interaction delays**

Parallel Ray Tracing (Radiance): Instrumentation Points

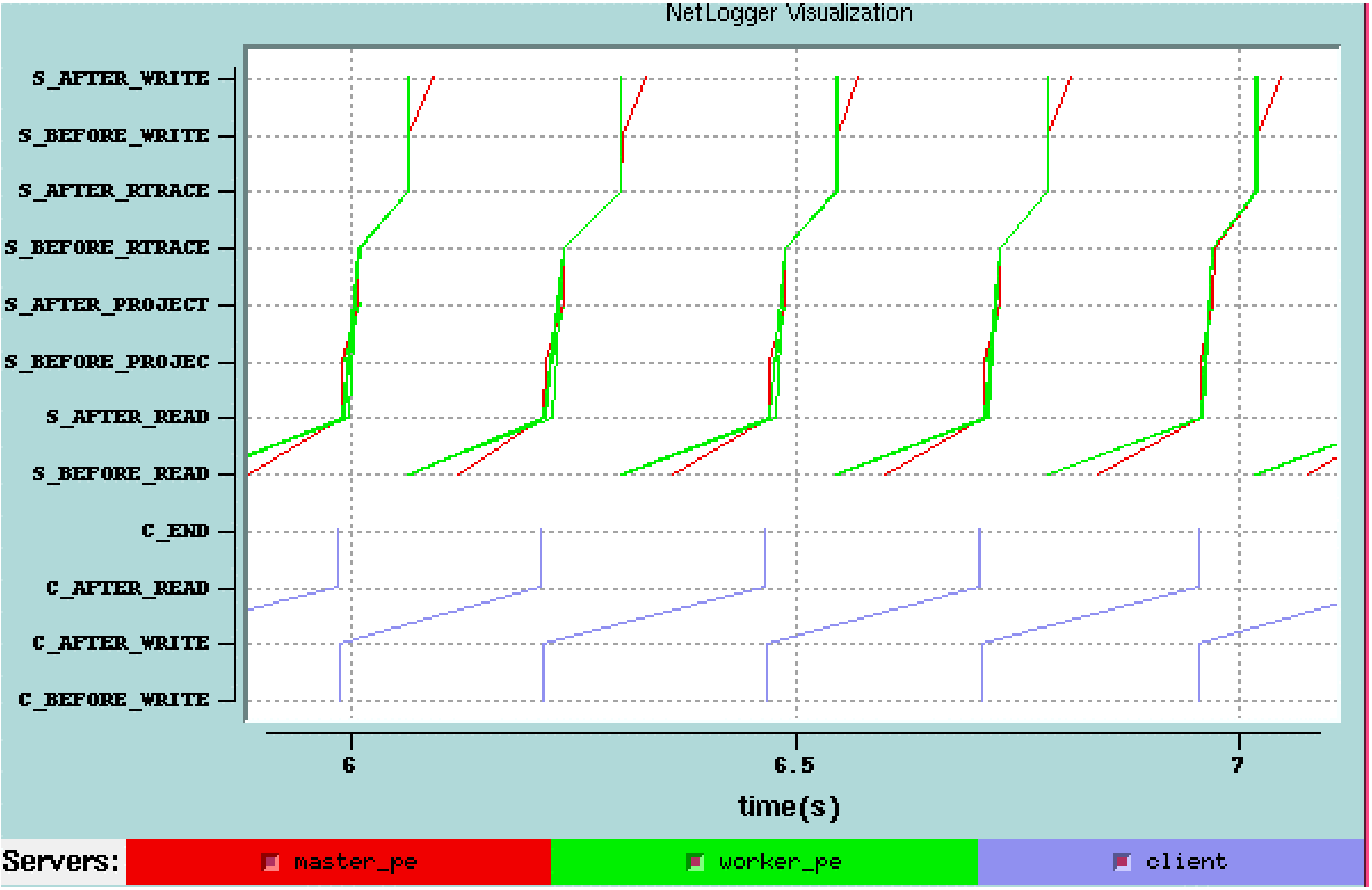


✱ = monitoring point

NetLogger Radiance Results: Before Tuning



NetLogger Radiance Results: After Tuning



NLV Configuration File for this Application

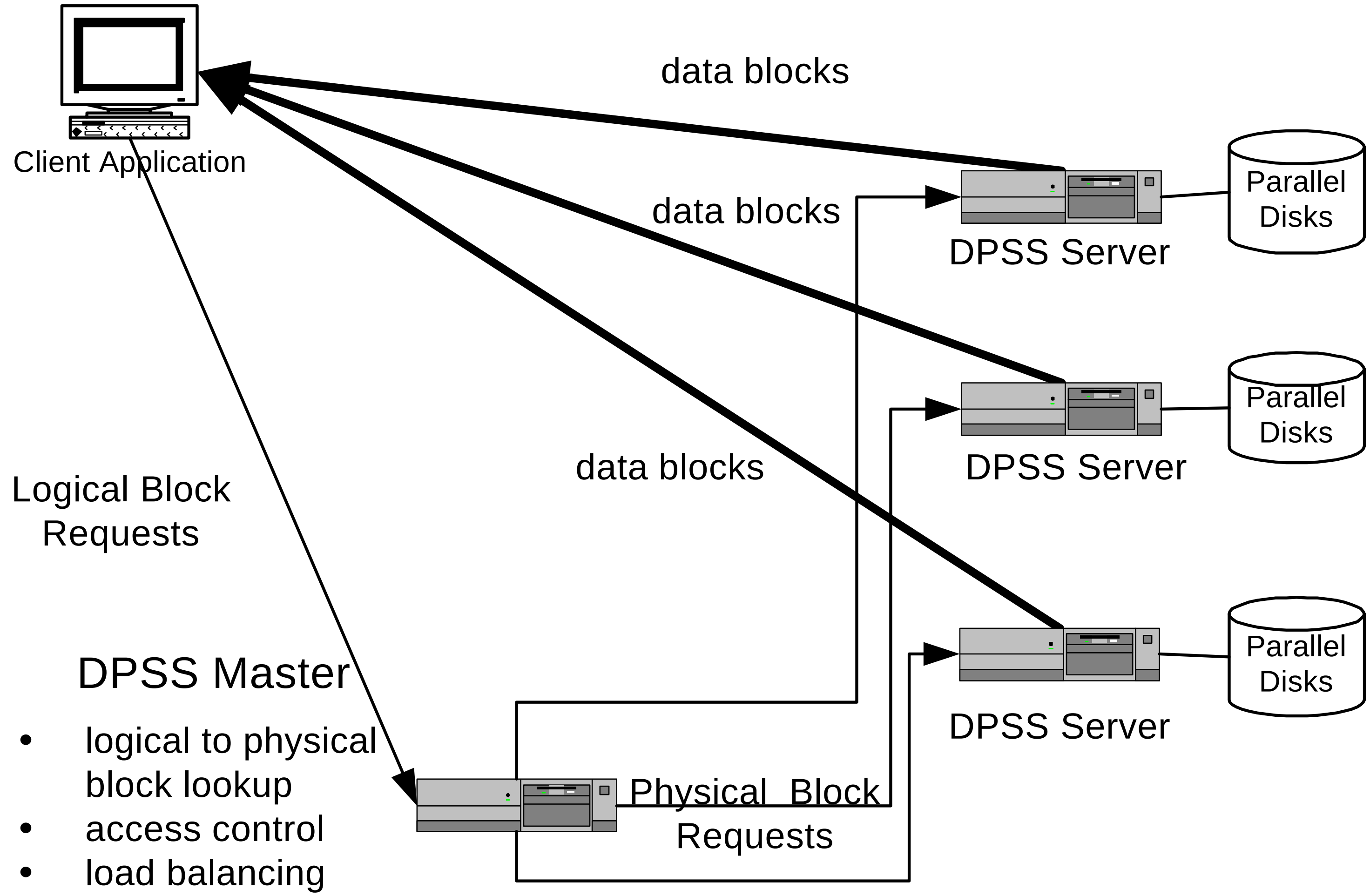
```
set +RADSERVER
type line
# lifeline defined by processing element id
id PE
# color lifelines by LTYPE (1=server, 2=client)
group LTYPE
[ +S_BEFORE_READ +S_AFTER_READ +S_BEFORE_PROJECTION
  +S_AFTER_PROJECTION +S_BEFORE_RTRACE +S_AFTER_RTRACE
  +S_BEFORE_WRITE S_AFTER_WRITE ]

set +RADCLIENT
type line
id PROG
group LTYPE
[ +C_BEFORE_WRITE +C_AFTER_WRITE +C_AFTER_READ +C_END ]
```

Example 3: Parallel Data Block Server

- The Distributed Parallel Storage Server (DPSS)
  - provides high-speed parallel access to remote data
  - Unique features of the DPSS:
    - On a high-speed network, can actually access remote data faster that from a local disk
      - 57 MB/sec vs 10 MB/sec
    - Only need to send parts of the file currently required over the network
      - e.g.: client may only need 100 MB from a 2 GB data set
      - analogous to http model
- NetLogger was used for performance tuning and debugging of the DPSS

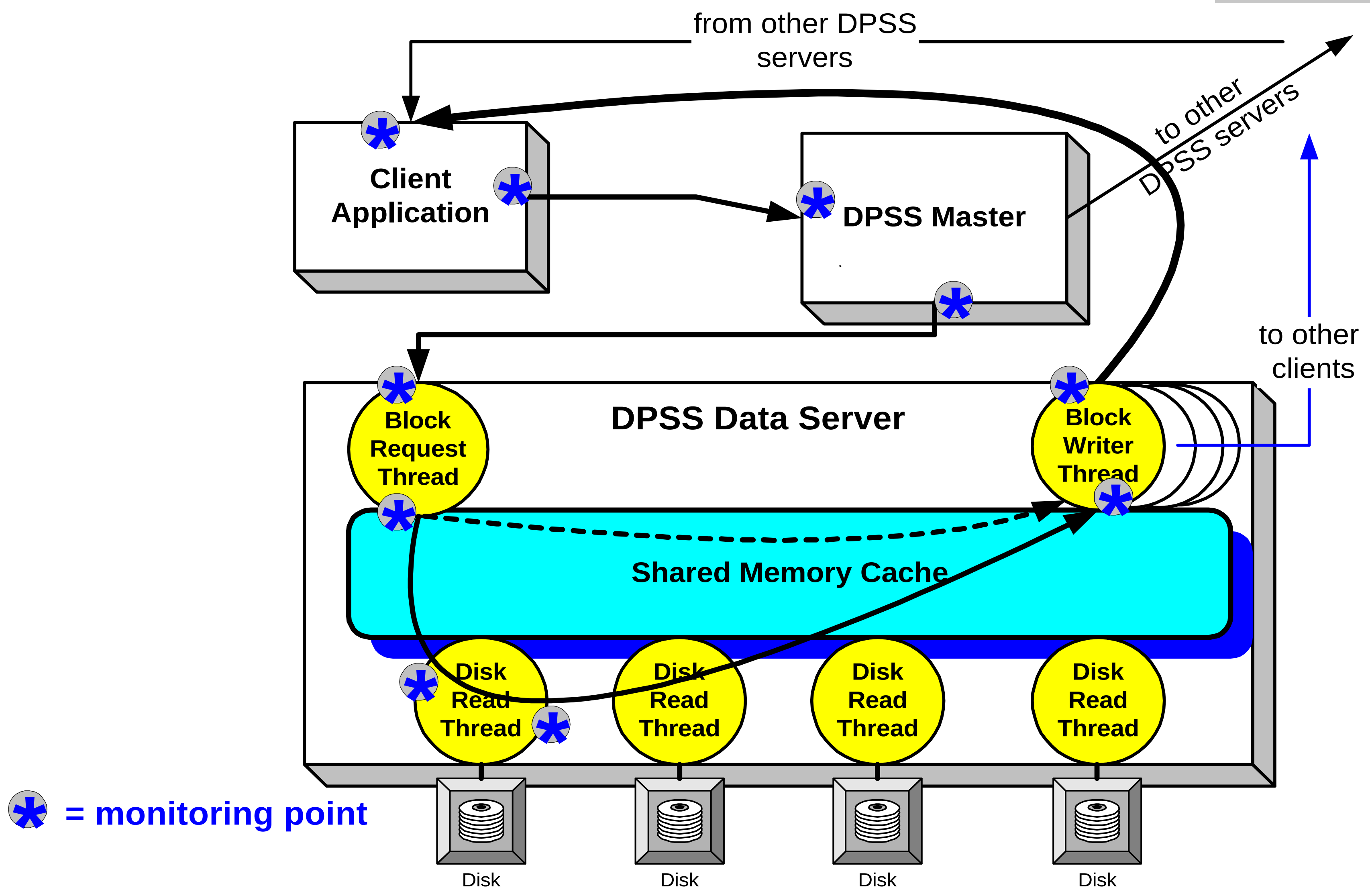
DPSS Cache Architecture



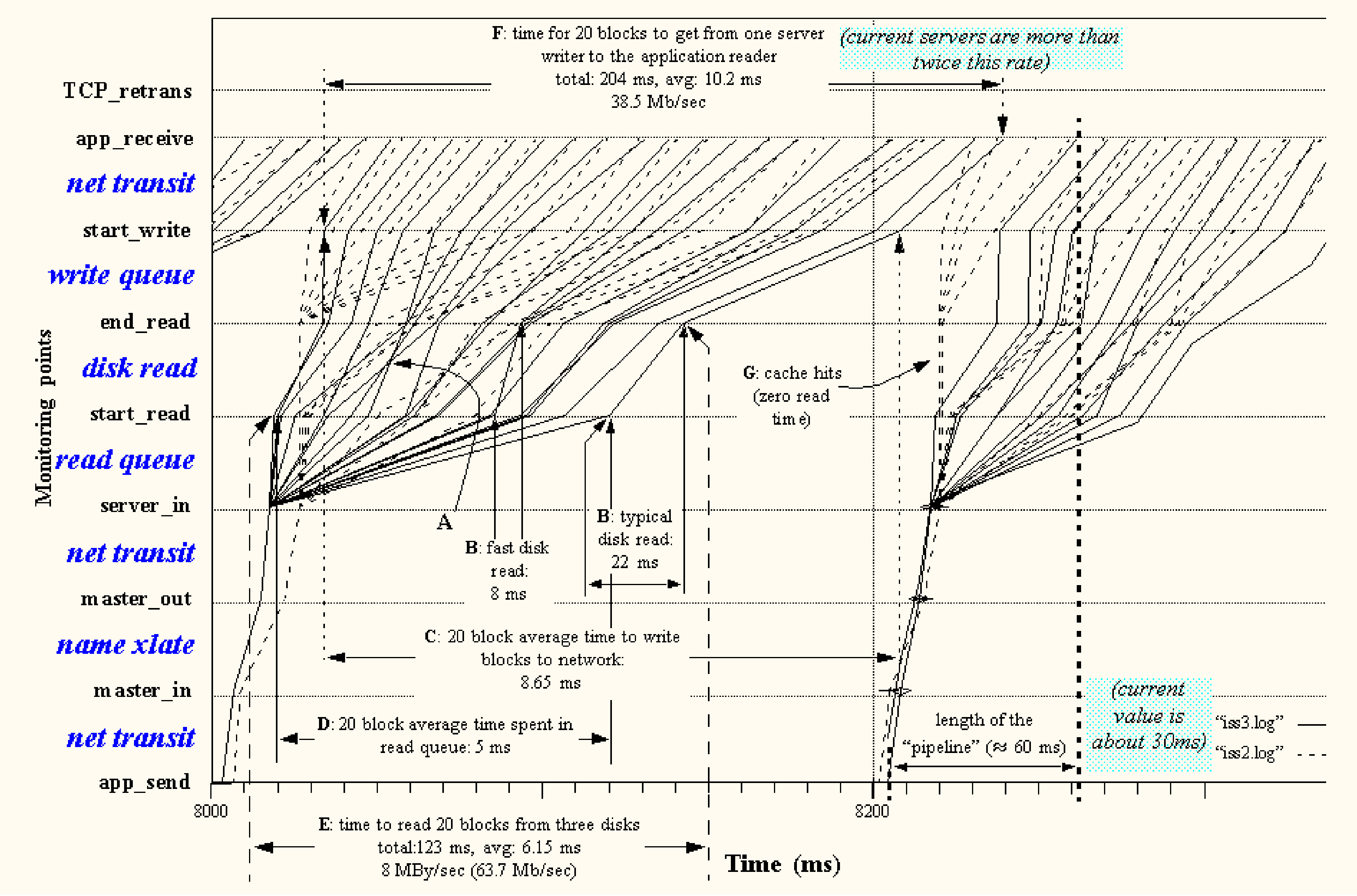
DPSS Master

- logical to physical block lookup
- access control
- load balancing

DPSS Instrumentation



NetLogger Results for the DPSS



F: time for 20 blocks to get from one server writer to the application reader total: 204 ms, avg: 10.2 ms 38.5 Mb/sec

(current servers are more than twice this rate)

G: cache hits (zero read time)

B: fast disk read: 8 ms

B: typical disk read: 22 ms

C: 20 block average time to write blocks to network: 8.65 ms

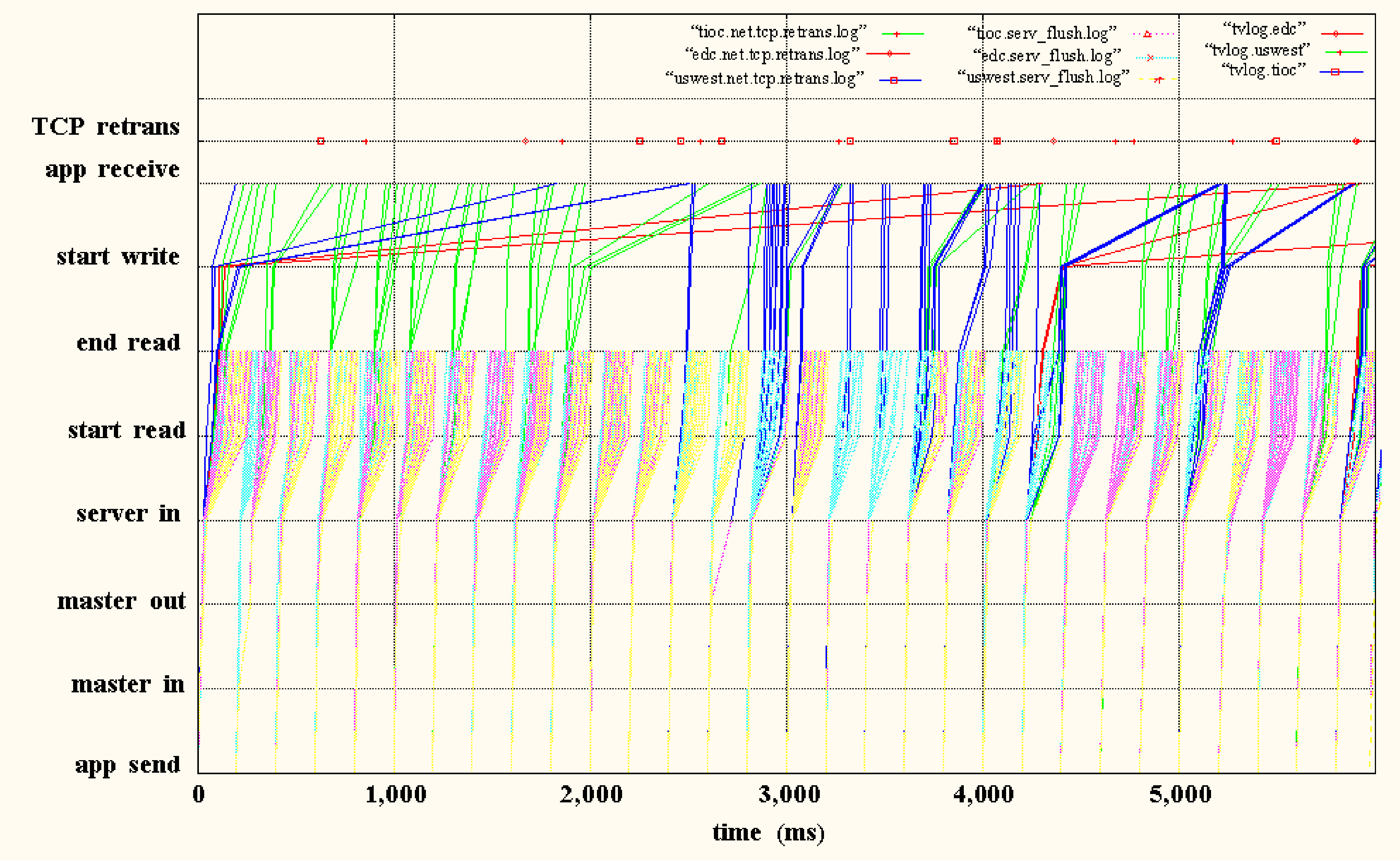
D: 20 block average time spent in read queue: 5 ms

length of the "pipeline" (≈ 60 ms)

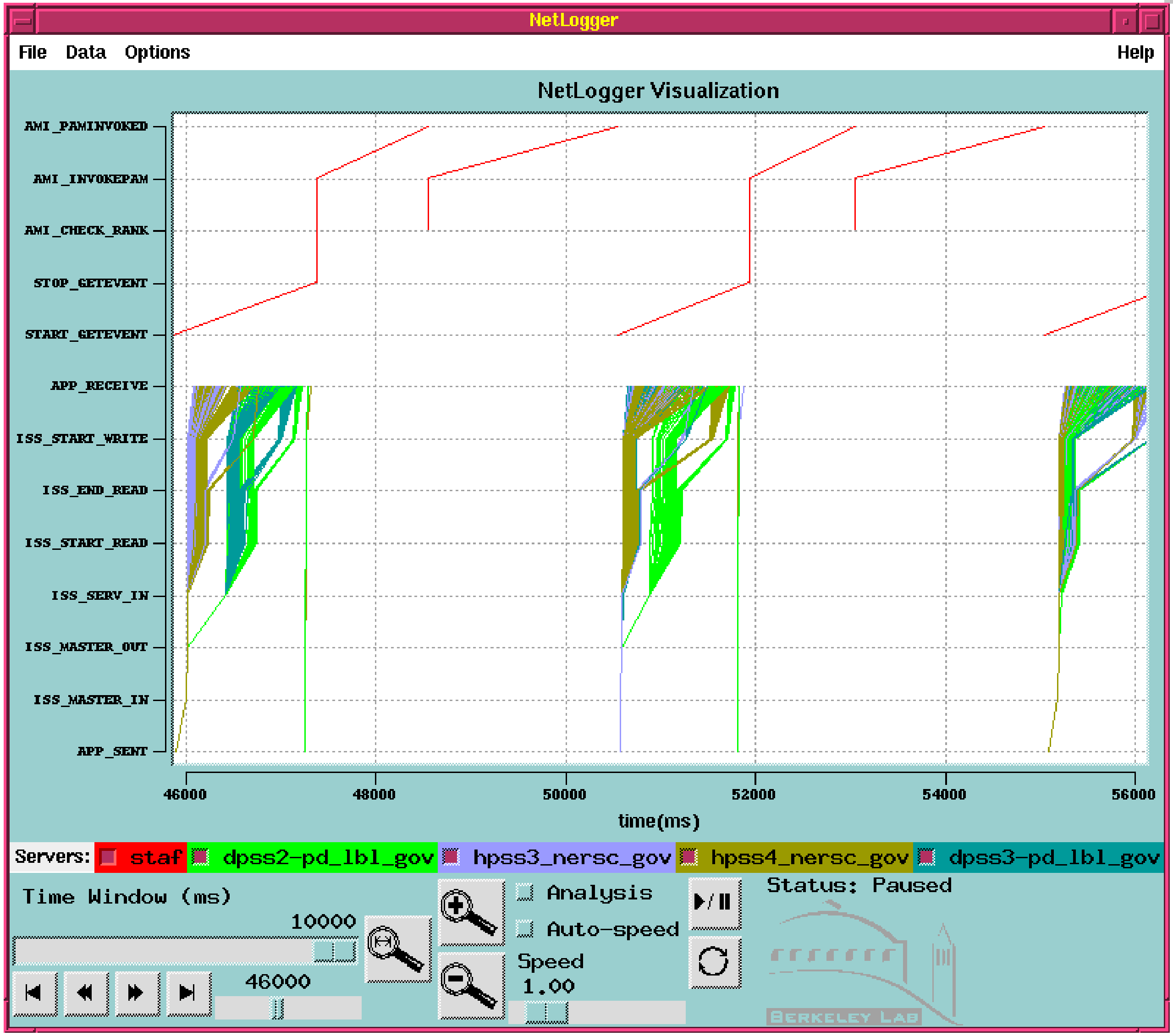
(current value is about 30ms)

E: time to read 20 blocks from three disks total:123 ms, avg: 6.15 ms 8 MBy/sec (63.7 Mb/sec)

NetLogger Results for the DPSS over a WAN



NLV of DPSS with a HENP client



Servers: 🟥 staf 🟩 dpss2-pd_lbl_gov ▨ hpss3_nersc_gov ▨ hpss4_nersc_gov ▨ dpss3-pd_lbl_gov

Time Window (ms) 10000

46000

Speed 1.00

Status: Paused

NLV Configuration File for this Application

```
set +STAF
type line
id [ HOST_PROG]
group HOST
[ +STAF_OPEN_R +START_GETEVENT +STOP_GETEVENT
  +STAF_CLOSE_R ]



set +DPSS_READ
type line

#lifeline defined by DPSS.BID and HOST
id [DPSS.BID HOST]

# color lines by DPSS.SERV
group DPSS.SERV

[ +APP_SENT +DPSS_MASTER_IN +DPSS_MASTER_OUT
+DPSS_SERV_IN +DPSS_START_READ +DPSS_END_READ
+DPSS_START_WRITE +APP_RECEIVE ]
```

Current Work: JAMM

- **Java Agents for monitoring and management (JAMM)**
  - **Java RMI-based agents are used to start up NetLogger versions of system tools**
    - **netstat, vmstat, uptime, xntpdc, ping, netperf, etc.**
- **Monitoring can be based on application use**
  - **e.g.: only do monitoring while a client is connected to a server**
- **For more info see: http://www-didc.lbl.gov/JAMM/**

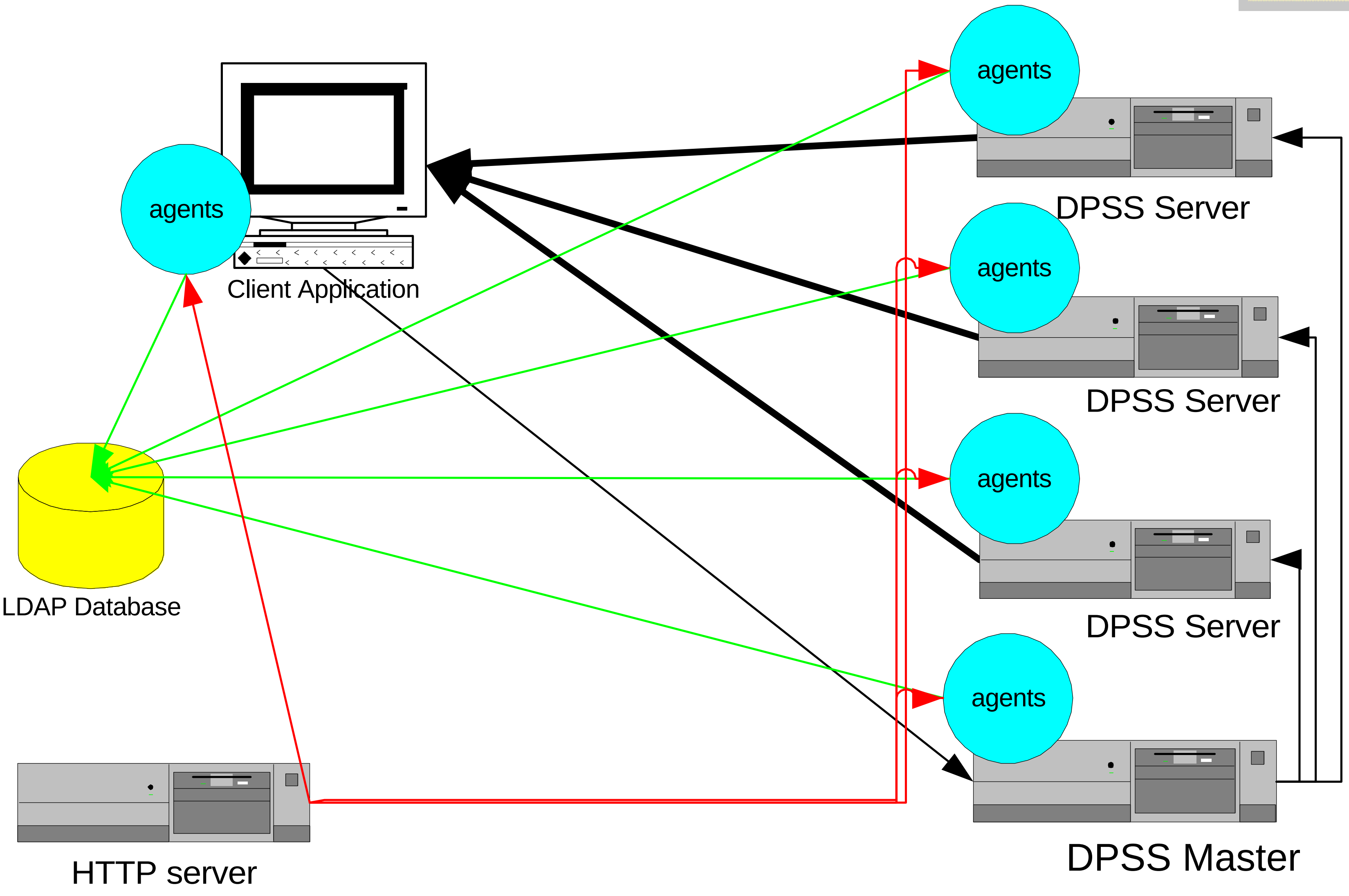
JAMM for active Network Monitoring

- **Network performance data is measured using netperf (http://www.netperf.org) and ping, and results are published in an LDAP database**

- **JAMM agents are used to monitor server activity, and automatically start netperf and ping experiments between client and server hosts**

- **Applications can query LDAP for this information, and set the optimal TCP buffer size based on this.**
  - **Optimal buffer size equal 2 x (bandwidth * delay)**

Java Agents For Monitoring and Management (JAMM)



Current Work

- NetLogger enhancements:
  - adding Globus security
    - plan to use GlobusIO for sending NetLogger socket connections
  - binary transmission/storage format

Grid Monitoring Service

- Our goal is to make this sort of monitoring a standard "grid service"
- Before this can happen, we need to define:
  - archive system
    - standard interface to archive system (probably LDAP?)
  - Network monitoring system
    - Surveyor, NWS, pingER, OCXmon, GloPerf,…
    - SNMP security issues (SNMP proxy?)

- Grid Forum "end to end monitoring" working group
- DOE NGI monitoring / instrumentation working group
  - goal is to deploy something by the end of the year

Getting NetLogger

- Source code and some precompiled binaries are available at:
  - http://www-didc.lbl.gov/NetLogger

- Solaris, Linux, and Irix versions of nlv are currently supported