ZeptoOS – KTAU DEMO

10/2005

KTAU Development Team (ktau_team@cs.uoregon.edu)

OBJECTIVE

Using KTAU and KTAUD to provide a trace of BG/L IO node kernel activities while running iotest benchmark on 16 compute nodes.

REQUIREMENTS

- BG/L account
- ZeptoOS-v.1.2 http://www-unix.mcs.anl.gov/zeptoos/software/
- KTAU-1.6 http://www.cs.uoregon.edu/research/ktau/downloads.php
- Latest TAU release http://www.cs.uoregon.edu/research/tau/downloads.php

Note: Please configure TAU with option:

For Jumpshot : *-TRACE –slog2*

For Vampir *: -TRACE –vtf=< VTF3 Trace Generation Package>*

CONTENTS

NOTE

- Sample KTAU trace output used in this demonstration is available in *ktau/docs/ZeptoOS-KTAU_demo/*

1) ZeptoOS and KTAU Setup

First, follow ZeptoOS-KTAU installation instruction in the documentation on the KTAU website (http://www.cs.uoregon.edu/research/ktau/DOCS/KTAU_ZeptoOS_Installation-chunked/index.php).

In ZeptoOS configuration tool, enable KTAUD with following parameters:

- KTAU source path : *<full path of KTAU distribution>*
- KTAUD path : *<full path of KTAUD binary>*
- KTAUD configuration path : *<full path of KTAUD configuration file>*
- KTAUD output directory : *<full path to user's home dir>/ktaud_output/*
- KTAUD Mode of Operation : Trace
- KTAUD Sampling Period : 1 sec
- KTAUD Sampling Mode : All

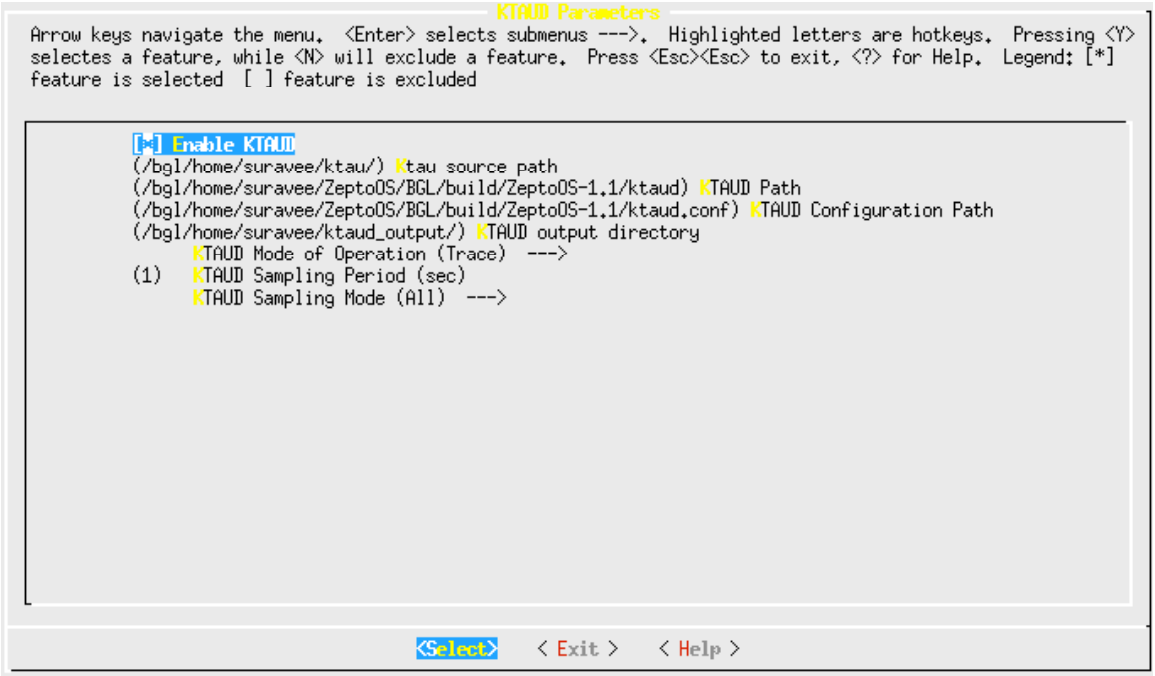


Figure 1: KTAUD configuration page of ZeptoOS configure tool

Note: Make sure that the symlink in /bgl/argonne-utils/profiles/<username>/ is setup correctly i.e. ramdisk should be linked to the ramdisk.elf, and INK should be linked to the zImage.elf that is built.

2) Running Job on BG/L

The ioteset benchmark is included in ZeptoOS distribution (ZeptoOS/BGL/ionode-ramdisk/test/). To run the benchmark,

cqsub -q short -k <user-name> -n 16 -t 30 ./iotest -f io.txt

Please refer to cqsub –h for detail of command-line option.

3) Interpreting KTAUD Trace Output

Once the job is finished, the trace output should be in the *ktaud_output* directory as specified in the configure tool. In this case, iotest benchmark use 1 IO node and 16 compute node. Therefore, KTAUD is generating one trace file, and 2 other files outputted from running *ps –ax* at the beginning and at the end of the run.

```
suravee@login1:~/ktaud_output> ls
ionode35.all  ionode35.ps_out_start  ionode35.ps_out_stop
suravee@login1:~/ktaud_output> cat ionode35.ps_out_start
  PID TTY      Uid        Size State Command
    1          root        508   S   init
    2          root          0   S   [keventd]
    3          root          0   S   [ksoftirqd_CPU0]
    4          root          0   S   [kswapd]
    5          root          0   S   [bdflush]
    6          root          0   S   [kupdated]
   53          root         68   S   /bin/utelnetd -d -l /bin/login
   70          root          0   S   [rpciod]
  152          root       1216   S   /bgl/dist/sbin/sshd
  175          1           1472   S   /sbin/portmap
  186          root       1584   S   /sbin/rpc.statd
  215          root     135048   S   /sbin/ciod_fl.440
  266          root        504   S   /bin/sh
  267          root       1384   S   /bin/ktaud -f /etc/ktaud.conf -D
  273          root        508   S   sh -c ps -ax > /bgl/home1/suravee/ktaud_out
  274          root        500   R   ps -ax
suravee@login1:~/ktaud_output>
```

Figure 2: Showing the output of running *ps –ax* command at the beginning of the run.

In this experiment, KTAUD was reading kernel trace data from the ring buffer of every process running at the time for every second, and stored output into the file *ionode35.all*. The naming convention of the trace output from KTAUD is

<hostname>.<pid | all>

Example:

ionode35.all : multiple-pid

and

ionode12.23 : single-pid

Each time KTAUD reads the kernel trace data, it appends new information to the end of the file. The following figure shows a sample of output trace. From figure 3,

- TSC : The frequency (Hz) of the PPC time-base counter on the processor.
- No Profiles : The number of pids being sample at that particular time.
- PID : The Linux process id.
- No entries : Number of trace entries being read.
- 1st column : Timestamp read from the time-base counter
- 2nd column : Memory address of the function being traced
- 3rd column : 0 = function entry, 1 = function exit

```
suravee@login1:~/ktaud_output> head -n 20 ionode35.all
TSC: 700002960.419560
No Profiles: 14
PID: 1   No Entries: 1126
317863634 c0006984 0
317883596 c0006984 1
317885078 c000f518 0
317889294 c000f518 1
317933178 c0006984 0
317947334 c0006984 1
317948001 c000f518 0
317950566 c000f518 1
317957255 c0006984 0
317972108 c0006984 1
317979913 c0006984 0
317995990 c0006984 1
317996703 c000f518 0
318001761 c000f518 1
318002304 c0006984 0
318018289 c0006984 1
318018913 c000f518 0
suravee@login1:~/ktaud_output>
```

Figure 3: Example from the KTAUD trace output

In this scheme, the tracing buffer of the process might get overflow if KTAUD does not empty out the ring buffer at a fast enough rate (default sampling information every 1 second might not be sufficient if the process is overwhelmed by the instrumentation points in the path of execution). There are several factors that contribute to the chance of buffer overflow.

- Amount of instrumentation points (configurable in the kernel)
- Process kernel activities
- Size of the ring buffer (configurable in the kernel)
- KTAUD sampling frequency (configurable in the ZeptoOS configure tool)

Once the ring buffer is overflow, trace events are lost. KTAU represents the period of losing events in the trace output as a function namely *ktau_lost_event* with address 0. Figure 4 show a section of the trace output that contains *ktau_lost_event*. In this figure, pid 70 has 5121 entries, which is the default maximum size specified in the KTAU configuration inside Linux. The two entries showing memory address = 0 represent the beginning and the end of *ktau_lost_event*.

```
18787704487 c000f518 0
18787923664 c000f518 1
18787924200 c0004da8 0
18787925702 c0004da8 1
18787926015 c0017f28 0
18787926928 c0017f28 1
18787927559 c000f518 0
PID: 4   No Entries: 0
PID: 5   No Entries: 0
PID: 6   No Entries: 0
PID: 53  No Entries: 0
PID: 70  No Entries: 5121
18048661950 0 0
18764823081 0 1
18764824060 c0017f28 1
18764832352 c00a1158 1
18764834792 c0004da8 0
18764836238 c0004da8 1
18764836532 c0017f28 0
18764837484 c0017f28 1
18764844345 c0004da8 0
18764846170 c0004da8 1
18764846662 c0017f28 0
```

Figure 4: Example from the KTAUD trace output showing ktau_lost_event for pid 70

4) Output Conversion and Visualization

Currently, KTAU trace output supports two formats, VTF and SLOG2. Vampir (Intel trace analyzer) is used to view VTF format, and Jumpshot for SLOG2. There are several phases of format conversion.

4.1) KTAU trace format (ktr) to TAU trace format (trc)

First KTAU trace format (ktr) is converted to TAU trace format (trc). This conversion introduces several advantages. First, it would allow traces from KTAU to be merged with TAU and produce a complete trace that contains execution path inside the kernel as well as in the application (work in progress). Furthermore, it would allow KTAU to use the existing format conversion utilities available from TAU distribution namely *tau2vtf* and *tau2slog2*.

Available conversion utilities are:

- ktau_convert : Phase 1 conversion
- ktau_convert_fix : Phase 2 conversion
- tau_merge : Merging traces from several source
- tau_convert –dump : Dumping (trc) trace in ASCII format (optional)

- ktau_convert

This tool takes KTAU trace (either single-pid or multiple-pid trace file) and output an individual (trc) file for each pid. It simply steps through the (ktr) file and parse out trace information and store them in a separate (trc) file according to the pid.

```
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$ ls
System.map      convert.sh      ionode35.all    old
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$ ../ktau_convert -i ionode35.all -m System.map -o iotest_16
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$ ls
System.map              iotest_16.175.trc       iotest_16.267.trc       iotest_16.5.trc         old
convert.sh              iotest_16.186.trc       iotest_16.275.trc       iotest_16.53.trc
ionode35.all            iotest_16.2.trc         iotest_16.276.trc       iotest_16.6.trc
iotest_16.1.trc         iotest_16.215.trc       iotest_16.3.trc         iotest_16.70.trc
iotest_16.152.trc       iotest_16.266.trc       iotest_16.4.trc         iotest_16.edf
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$
```

Figure 5: Phase 1 conversion

In figure 5, *ionode35.all* is a multiple-pid trace file. After running *ktau_convert*, each pid has its own output trace file in the (trc) format. Also, an (edf) file is created, which contains the function mapping information for all the (trc) files. At this point, memory address used for identifying the function being traced, is mapped to the corresponding function name using the *System.map* (kernel symbol table from obtained from *ZeptoOS/BGL/ionode-linux-2.4.19/linux-2.4.19/System.map*). Also, the TSC value is used to convert timestamp from number of ticks to microsecond.

```
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$ head -n 30 iotest_16.edf
81 dynamic_trace_events
# FunctionId Group Tag "Name Type" Parameters
1 KTAU_LOST_EVENT 0 "ktau_lost_event" EntryExit
2 SYSCALL 0 "sys_clone" EntryExit
3 SCHEDULE 0 "schedule" EntryExit
4 SYSCALL 0 "sys_open" EntryExit
5 SYSCALL 0 "sys_lseek" EntryExit
6 SYSCALL 0 "sys_read" EntryExit
7 SYSCALL 0 "sys_write" EntryExit
8 IRQ 0 "timer_interrupt" EntryExit
9 BOTTOMHALF 0 "do_softirq" EntryExit
10 BOTTOMHALF 0 "tasklet_hi_action" EntryExit
11 BOTTOMHALF 0 "run_timer_list" EntryExit
12 SYSCALL 0 "sys_close" EntryExit
13 SYSCALL 0 "sys_dup" EntryExit
14 SYSCALL 0 "sys_execve" EntryExit
15 SYSCALL 0 "sys_ioctl" EntryExit
16 SYSCALL 0 "sys_getpid" EntryExit
17 SYSCALL 0 "sys_rt_sigaction" EntryExit
18 SYSCALL 0 "sys_reboot" EntryExit
19 SYSCALL 0 "sys_newuname" EntryExit
20 SYSCALL 0 "sys_socketcall" EntryExit
21 SYSCALL 0 "sys_time" EntryExit
22 SYSCALL 0 "sys_fcntl" EntryExit
23 SYSCALL 0 "sys_chdir" EntryExit
24 SYSCALL 0 "sys_setsid" EntryExit
25 SYSCALL 0 "sys_brk" EntryExit
26 SYSCALL 0 "sys_sysinfo" EntryExit
27 SYSCALL 0 "sys_access" EntryExit
28 SYSCALL 0 "sys_rt_sigprocmask" EntryExit
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$
```

Figure 6: Example of (edf) file

- ktau_convert_fix

In case of losing event, 3 scenarios can happen to the trace data:

Case 1: Losing both entry and exit events of a function
In this case, we cannot imply anything about the traces being lost.

Case 2: Losing entry but not exit event of a function
In this case, we can imply that during the *ktau_lost_event* period, it might exist the corresponded function entry. Therefore, we can pretend that the function entry has the timestamp equal to the exit timestamp of *ktau_lost_event* period.

Case 3: Losing exit but not entry event of a function
This case is the inverse of case 2. We can assume that exit event of the function is lost inside the *ktau_lost_event* period. Therefore, we can pretend that the function exit has the timestamp equal to the starting timestamp of *ktau_lost_event* period.

ktau_convert_fix is used to insert dummy entry and exit events into each (trc) file in order to maintain the nesting level of functions inside the trace. This is a necessary step to help visualization tools render a trace that is readily apparent. However, we encourage that users configure KTAUD in a way that minimize the amount of event lost. KTAU is currently resolving this issue, and the solution will be included in the following release.

- tau_merge

When converting multiple-pid trace, multiple (trc) files are generated for each pid. In such case, *tau_merge* utility must be used to merge all (trc) files into a (trc) file, which will be converted into VTF or SLOG2 format later on. Please refer to the *tau_merge –h* for more detail.

Please note that (trc) file is in binary format. TAU also provides a utility to dump the (trc) file into a human readable ASCII format, *tau_convert -dump*. Figure 7 shows a partial output from *tau_convert –dump.*

```
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$ tau_convert -dump merge.trc merge.edf merge.dump
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$ head -n 30 merge.dump
#  creation program: tau_convert -dump
#     creation date: Oct-13-2005
#    number records: 127429
# number processors: 16
# max processor num: 276
#   first timestamp: 454088
#    last timestamp: 42230120

#=NO= ======================EVENT== ==TIME [us]= =NODE= =THRD= ==PARAMETER=
    1                     "EV_INIT"       454088      1      0           3
    2                   "sys_clone"       454088      1      0           1
    3                   "sys_clone"       454117      1      0          -1
    4                    "schedule"       454119      1      0           1
    5                     "EV_INIT"       454124      2      0           3
    6                    "schedule"       454124      2      0           1
    7                    "schedule"       454125      1      0          -1
    8                   "sys_clone"       454188      1      0           1
    9                   "sys_clone"       454208      1      0          -1
   10                    "schedule"       454209      1      0           1
   11                     "EV_INIT"       454212      3      0           3
   12                    "schedule"       454212      3      0           1
   13                    "schedule"       454213      1      0          -1
   14                   "sys_clone"       454222      1      0           1
   15                   "sys_clone"       454243      1      0          -1
   16                   "sys_clone"       454255      1      0           1
   17                   "sys_clone"       454278      1      0          -1
   18                    "schedule"       454279      1      0           1
   19                     "EV_INIT"       454282      4      0           3
   20                    "schedule"       454282      4      0           1
   21                 "FLUSH_CLOSE"       454282      4      0           0
dyna6-253:~/Software/ktau/user-src/src/tools/ktau_demo suravee$
```

Figure 7: Example of converting TAU trace format into a human readable ASCII format.

4.2) TAU trace format (trc) to VTF format

To convert, simply run

tau2vtf <TAU trace> <edf file> <out file>
vampir <outfile>.

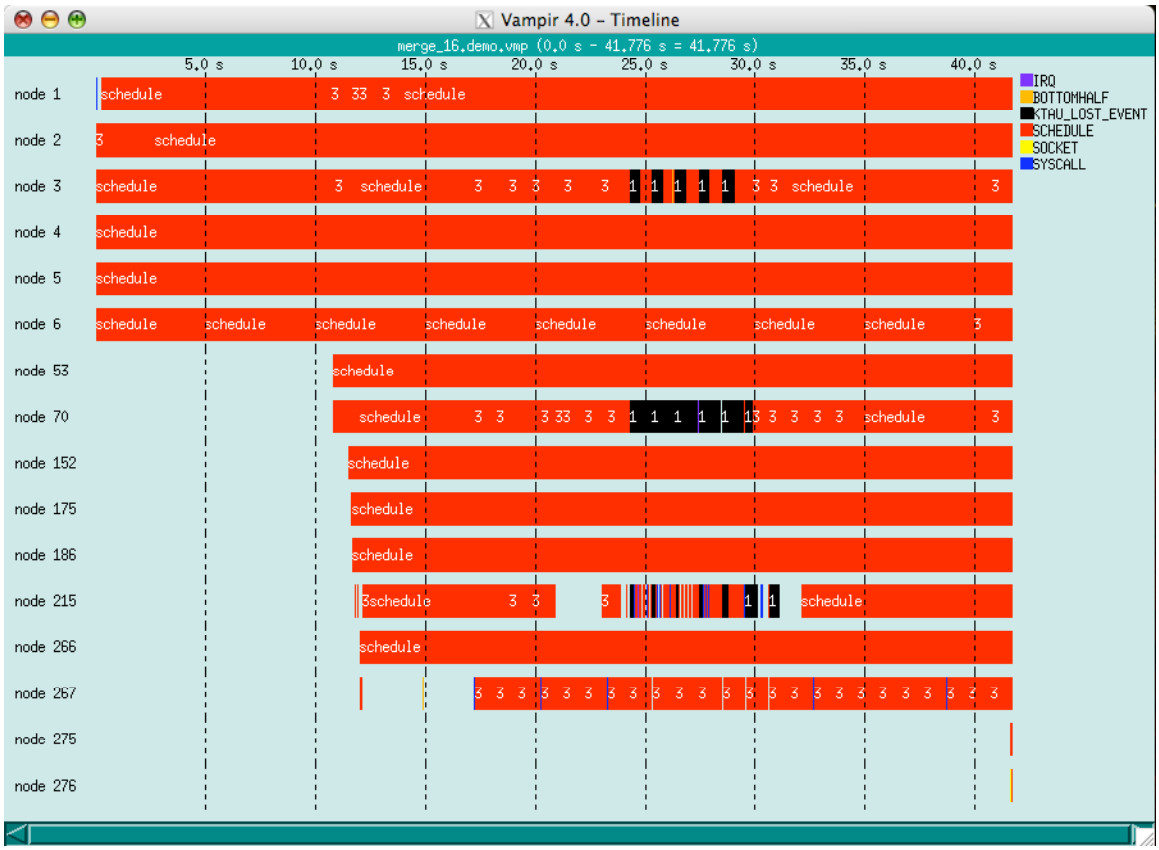


Figure 8: Example screenshot from Vampir

This screen shot shows concurrent kernel activities of process running on BG/L IO node using Vampir. Please note that the left column (node XXX) is used to identify Linux pid on a single machine. At this point, the information from *ionode35.ps_out_start* and *ionode35.ps_out_stop* can be used to help identify the name of each process.

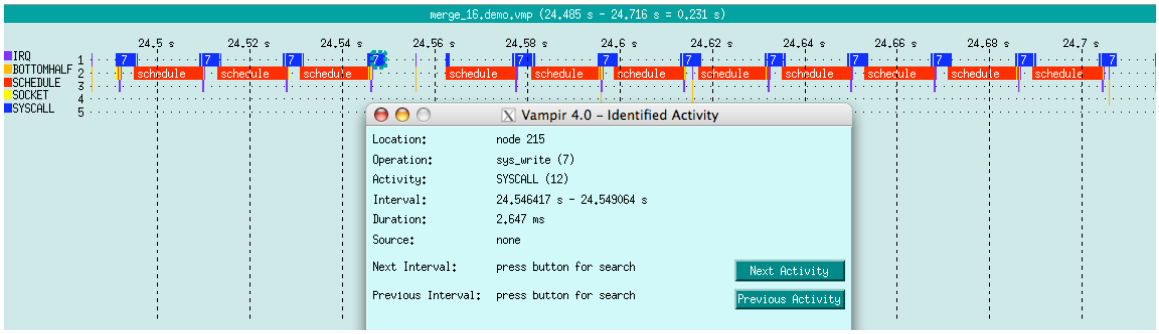


Figure 9: Example screenshot from Vampir showing CIOD kernel activities

4.3) TAU trace format (trc) to SLOG2 format

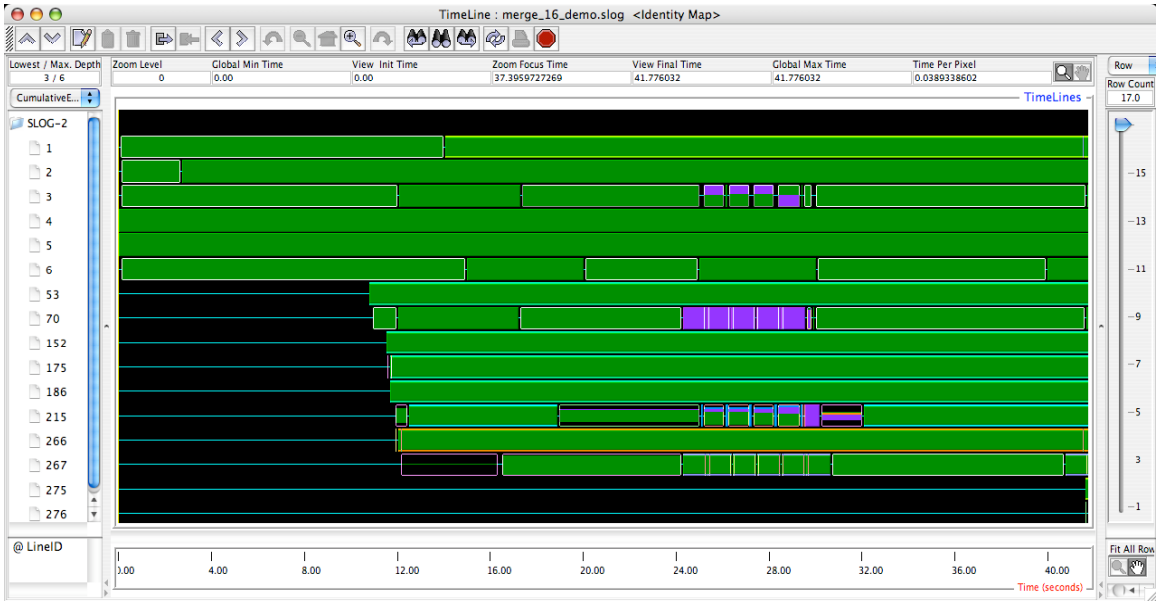
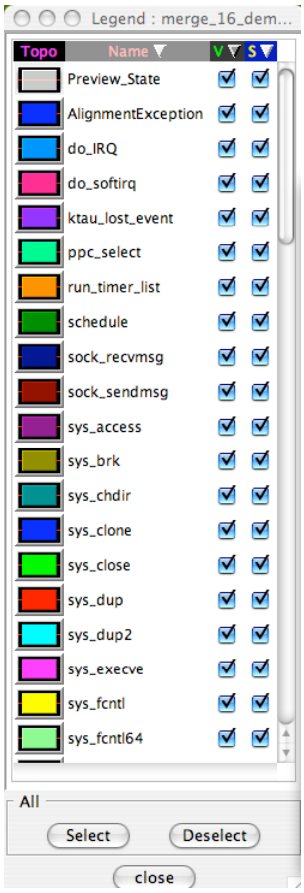


Figure 10: Example screenshot from Jumpshot



To convert, simply run

tau2slog2 <tau_tracefile> <edf_file> -o <slog_tracefile>.
Jumpshot <slog_tracefile>

This screen shot shows concurrent kernel activities of process running on BG/L IO node using Jumpshot. Please note that the left column (node XXX) is used to identify Linux pid on a single machine. At this point, the information from *ionode35.ps_out_start* and *ionode35.ps_out_stop* can be used to help identify the name of each process.

For figure 9 and 10, please note that IO node is a single processor machine. Therefore, when a process is running, others will be scheduled out. This portion is shown with red portion in figure 9, and green portion in figure 10.

Black portion in figure 9, and purple portion in figure 10 shows the *ktau_lost_event* period. rpciod (pid 70) is a kernel daemon supporting NFS implementation. Since iotest is an IO intensive application (writing to a file over NFS), a lot of kernel activities forwarded from compute node to ciod (pid 215) are also affecting rpciod. In this case, KTAUD could not catch up with the overwhelmed trace events being generated, and resulted in *ktau_lost_event.*