February 10th 2007

Practical Performance Analysis in Linux

IBM Linux Technology Center
Vara Prasad
prasadav@us.ibm.com

Agenda

- Common Performance Questions
- What is SystemTap?
- What can SystemTap do for you?
- SystemTap GUI
- Real Life Examples
- Conclusions
- Q&A

Common performance questions

- Occasionally jobs take significantly longer than usual to complete, or don’t complete. Why?
- An application seems to always take a long time to complete. Where is the problem?
- Is my system capable of handling additional workloads?

- *Answering these questions is often disruptive, time-consuming, and requires a high degree of OS knowledge and expertise.*

Current performance tools: Drawbacks

- Tuning high performance systems is complex
- System wide performance problems are difficult to identify
  - *many complex moving parts*
  - *Standard tools are limited in capabilities*
  - *Expert tools require customization not feasible for production systems*
- Some tools have overhead even when not in use, not ideal for production systems
- Some tools need modifying operating system
- Often, different tools are used on different hardware
  - *Many different tools and data sources but no easy way to integrate the information*

Characteristics of the ideal tool

- Available: Integrated into the OS and available on demand
- Low overhead: Has ***zero*** impact when disabled; ***insignificant*** overhead when in use
- Safe: Safe to use in a production environment
- Top to bottom: A tool that helps to solve problems from application layer to the hardware interface
- Versatile: Easy to learn and use effectively by both novices and experts

SystemTap

- One tool to analyze systemic problems all the way from applications to Operating System
- Tool for real time performance analysis
- Designed to be safe to use in production environments, no need to reproduce the problems in test environment
- Open source community project with active contributions from IBM, Intel, Hitachi, Red Hat and various community members
- A growing set of tracing applications are available on the web.
- Custom applications can be developed quickly using a familiar scripting language
- **Native code**, no interpreter and highly parallel execution
- An extensible platform and enabler for developing lots of new tools
- Enhanced through customer and development-community involvement

SystemTap

SystemTap

Realtime performance analysis

Capacity planning

Functional problem analysis (debugging)

Low overhead and safe for production systems

Continuous Performance monitoring

On Demand probing

Easy to use by system administrators

SystemTap Safety features

- Leverages well tested tool chain, no new compiler or interpreter
- Reuse well tested kernel features
- Language Safety features:
  - *No dynamic memory allocation*
  - *Types and type conversions limited*
  - *Limited pointer operations*
- Builtin safety checks
  - *Infinite loops and recursion*
  - *Invalid variable access*
  - *Division by zero*
  - *Restricted access to kernel memory*
  - *Array bound checks*
  - *Version compatibility checks*

SystemTap General Features

- Available on most common platforms
- Bundled with common enterprise distributions
- Low overhead and highly parallel execution
- Cached scripts runs are supported
- Cross compile facility is available
- GUI and command line interfaces are supported
- Fast in kernel data aggregation facilities
- Data output in both text and binary forms

SystemTap Availability

SYSTEMTAP

AMD

intel

PowerPC™

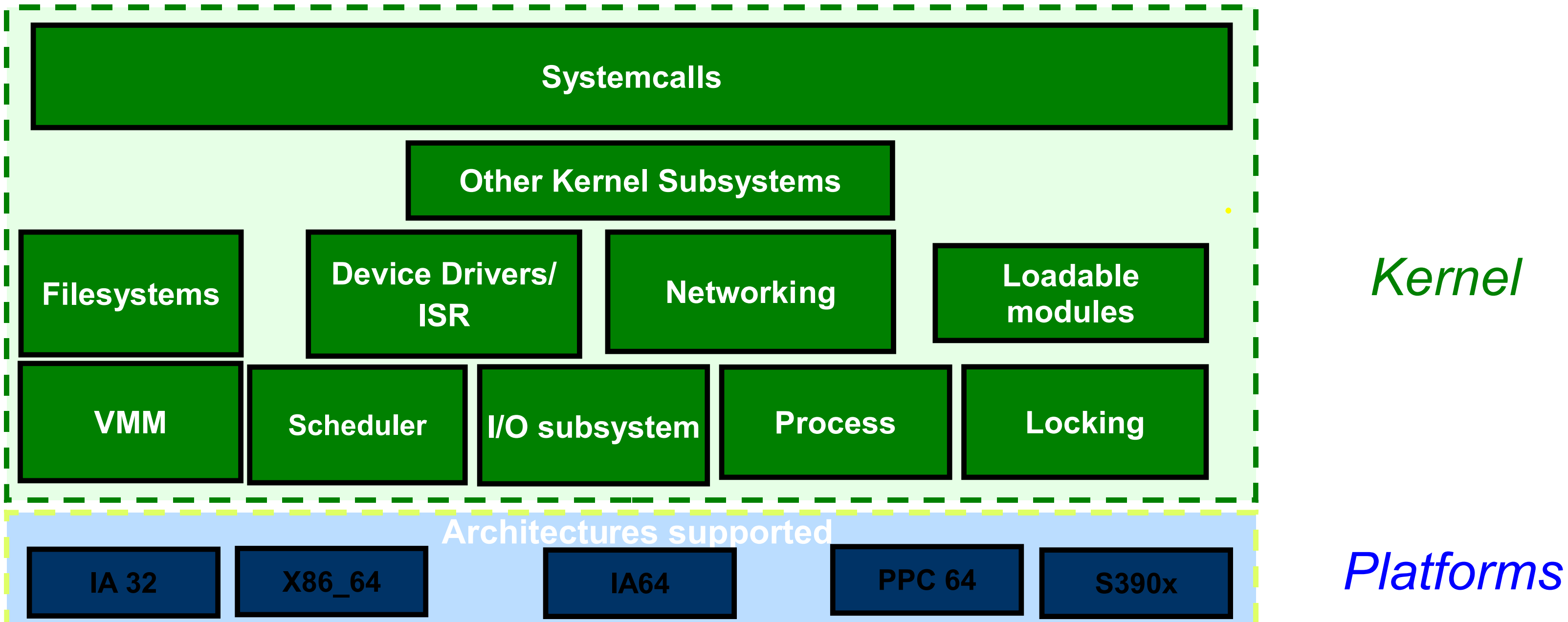
S/390

redhat

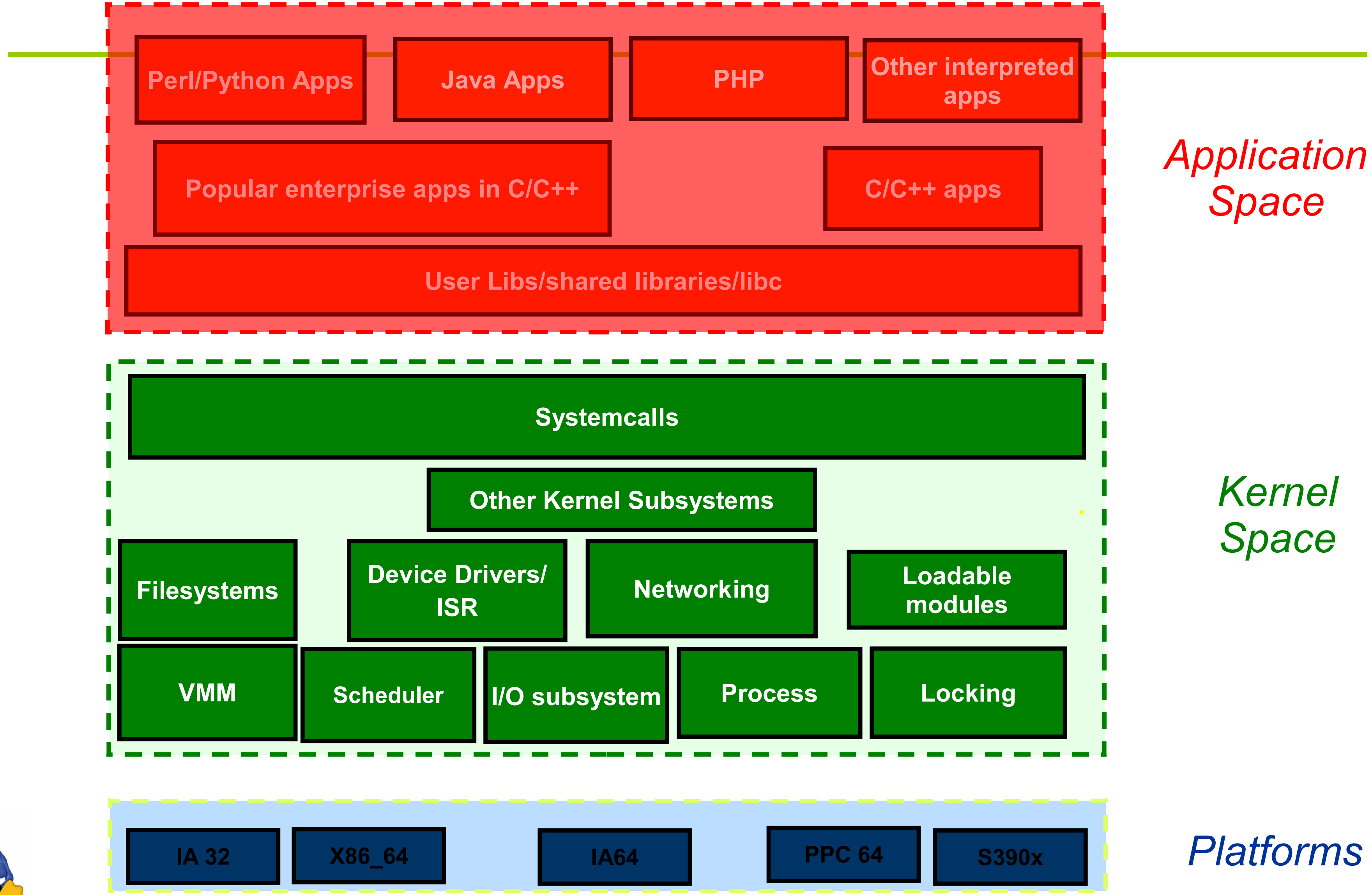
fedora™

SUSE
A NOVELL BUSINESS

What can SystemTap do for you TODAY?



What will SystemTap do for you tomorrow?



Example End User Script

```
global reads
probe begin {
  printf(“probe beginning\n”)
}

probe syscall.read {
  reads[execname()] <<< count
}

probe end {
  foreach (prog_name in reads) {
    printf("Name: %s, # Reads: %d, Total Bytes: %d, Avg: %d\n",
            prog_name, @count(reads[prog_name]),
            @sum(reads[prog_name]), @avg(reads[prog_name]))
    }
}
```

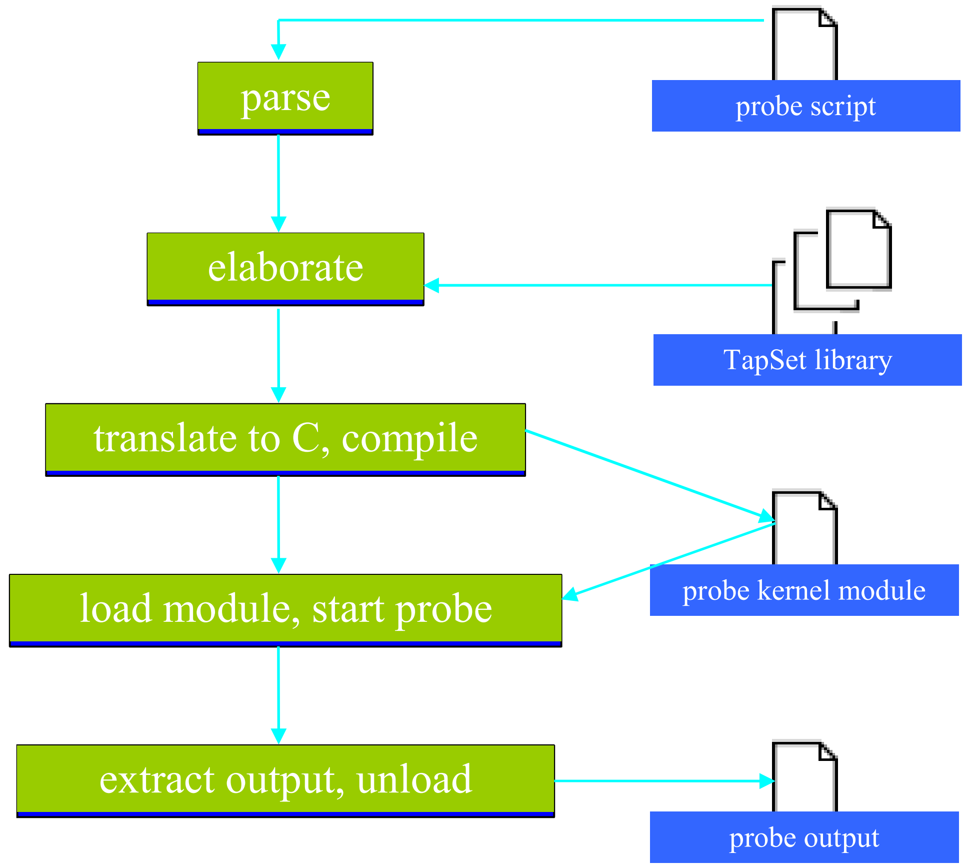
Language features:

- Global variables and builtin functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined probe library or tapsets for common probe points
- Familiar hierarchical “dot” notation for probe specification
- Probe entry and termination call-backs

TapSets

- A TapSet defines:
  - Probe Points: a set of instrumentation points for a particular subsystem
  - Data values that are available at each probe point.
- Written by experts
- Tested and packaged with SystemTap
- Tapsets are currently available for major areas of the kernel like process, systemcalls , scheduler, filesystem, networking etc.
- Currently Tapsets define thousands of probe points

How SystemTap works?



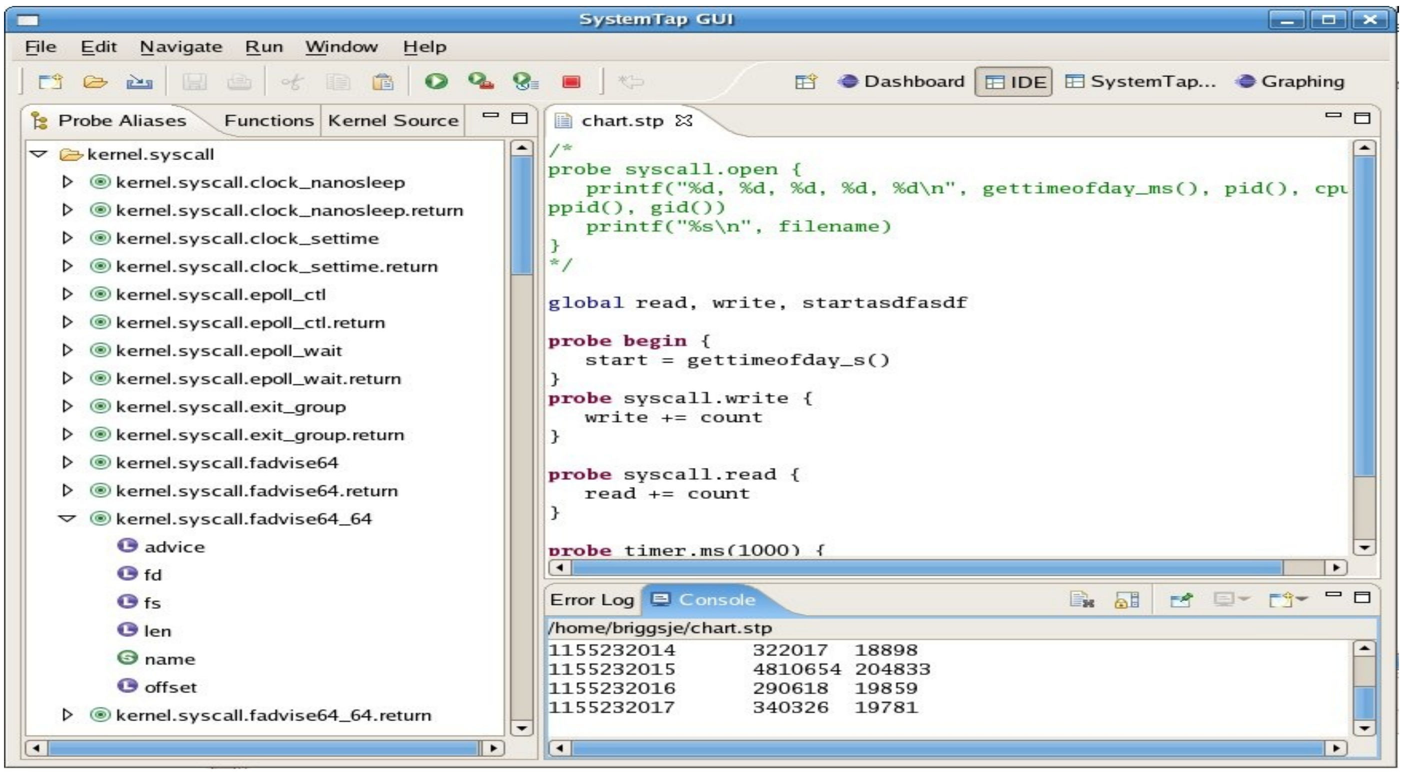
SystemTap GUI

- An Eclipse-based application intended to ease the use of SystemTap.
- Both an Integrated Development Environment for the SystemTap, as well as a data visualization and analysis tool
- Contains three unique perspectives, each with a different purpose – IDE, Graphing and Dashboard

SystemTap GUI – IDE Perspective

- Editor for creating, editing and testing SystemTap scripts, including code assist, syntax highlighting, and script execution
- Browsers:
  1. Tapset Browser – Browse and insert skeleton probes, learn available parameters.
  2. Builtin Function – Browse tapset functions/return types.
  3. Source Browser– Navigate and view source files, and using those files, place probes at arbitrary code locations

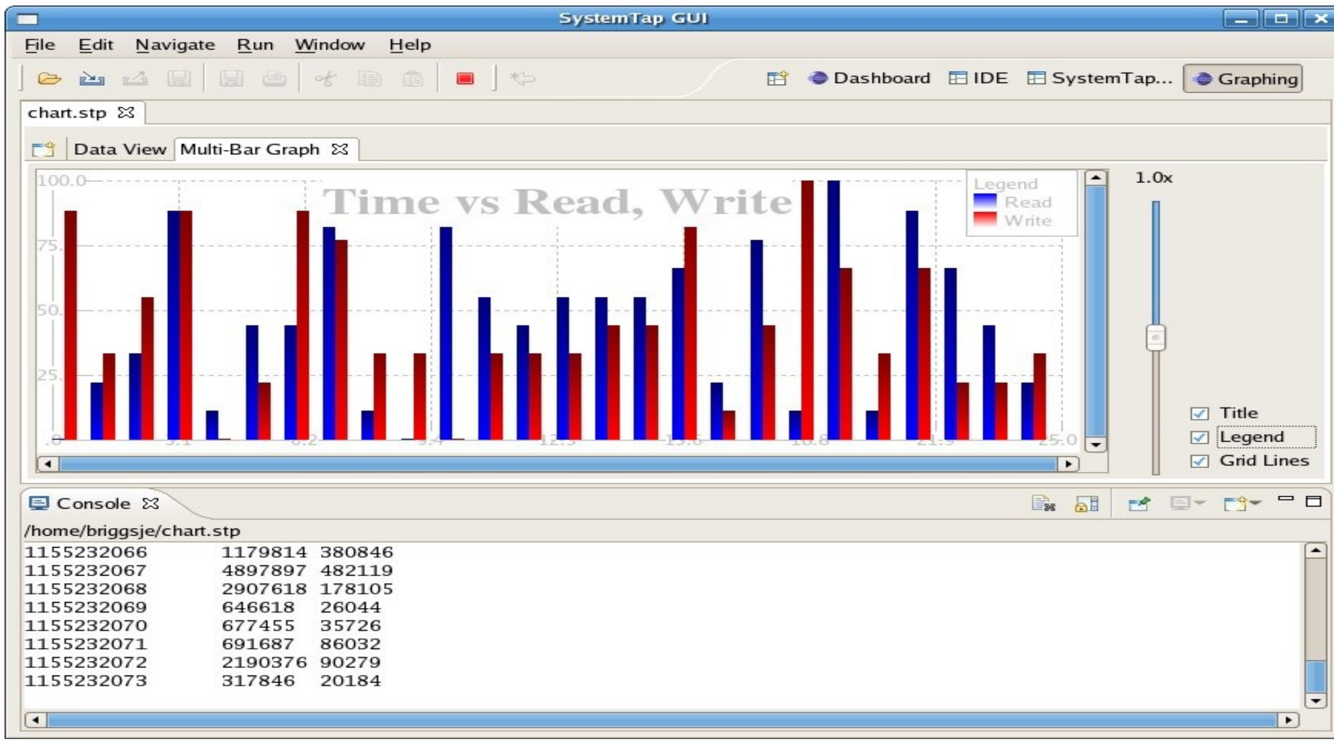
SystemTap GUI – IDE Perspective



SystemTap GUI – Graphing Perspective

- Allows users to view the output of their SystemTap scripts in graph form
- Users can run an open script, import existing data from a previous run, export data from a new run, or save the graph as an image
- Features include zooming, scrolling along the timeline, and optional legends, gridlines, etc

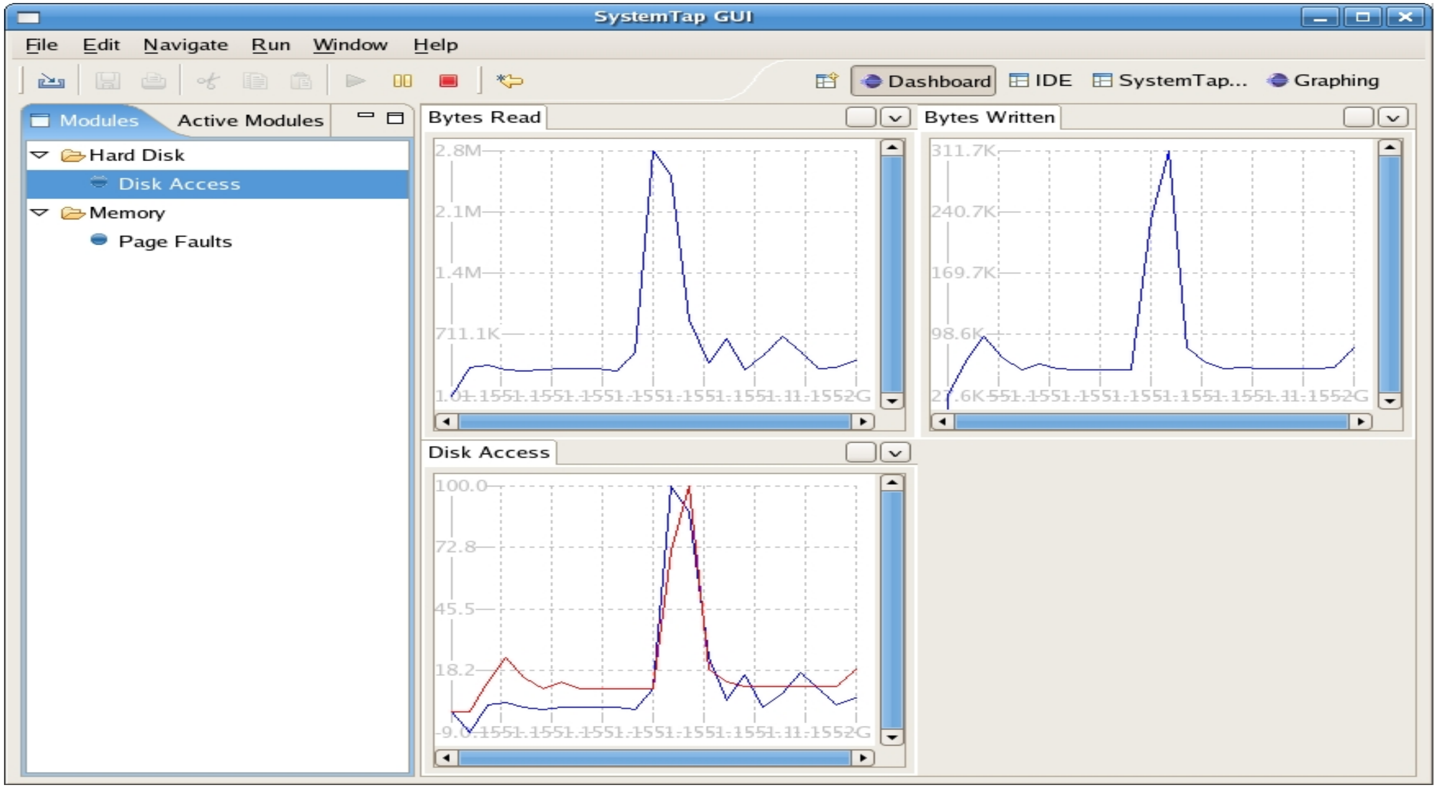
SystemTap GUI – Graphing Perspective



SystemTap GUI – Dashboard Perspective

- Enables users to import, load, and run predefined scripts.
- Allows the execution and viewing of 1 to 8 different graphs at one time, gives ideal perspective for entire system analysis.

SystemTap GUI – Dashboard Perspective



Real Life Uses of SystemTap

- SCSI request size mismatch
- UDP datagram loss
- Top I/O by users and processes

SCSI Request Sizes

Problem
In a benchmark run, we observed a mismatch between expected and actual SCSI I/O counts.

Solution
Create a simple SystemTap script to track the counts and sizes of SCSI requests to a specific device.

SCSI Request Sizes – scsi_req.stp

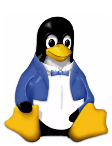
```
# Thanks to Allan Brunelle from HP
global rqs, host_no, channel, id, lun, direction

probe begin
{
    host_no   = 0
    channel   = 1
    id        = 1
    lun       = 0
    direction = 1 /* write */
}

probe scsi.iodispatching
{
    if (data_direction != direction) next
    if (lun            != lun) next
    if (id             != dev_id) next
    if (channel        != channel) next
    if (host_no        != host_no) next

    rqs[req_bufflen / 1024]++
}

probe end
{
    printf("ReqSz(KB)\t#Reqs\n")
    foreach (rec+ in rqs)
        printf("%8d\t%5d\n", rec, rqs[rec])
}
```



SCSI Request Sizes – output

```
# stap scsi_req.stp
ReqSz(KB)        #Reqs
        4            3
        8            2
       12            1
       28            1
       44            1
       88            1
      164            1
      204            1
      216            1
      308            1
      448            1
      508            1
      512           36
```

UDP Datagram Loss

Problem

A customer wanted to see UDP statistics for both the sending and receiving sides and how many UDP datagrams were dropped.
Existing tools don't provide all of this data:

- *netstat -su doesn't show how many datagrams are dropped when sending.*
- *iptraf doesn't show statistics on datagram loss.*

Solution

Create a SystemTap script that records how many UDP datagrams have been sent and received and how many were dropped.

UDP Datagram Loss - udpstat.stp

```
# Thanks to Eugene Teo from Red Hat

global udp_out, udp_outerr, udp_in, udp_inerr, udp_noport

probe begin {
  /* print header */
  printf("%11s  %10s  %10s  %10s  %10s\n",
         "UDP_out", "UDP_outErr", "UDP_in", "UDP_inErr", "UDP_noPort")
}

probe kernel.function("udp_sendmsg").return {
  $return >= 0 ? udp_out++ : udp_outerr++
}

probe kernel.function("udp_queue_rcv_skb").return {
  $return == 0 ? udp_in++ : udp_inerr++
}

probe kernel.function("icmp_send")  {
  /* destination not reachable and port not reachable */
  if (type == 3 && code == 3) {
    /* UDP Protocol = 17 */
    if (skb_in->nh->iph->protocol == 17)
      udp_noport++
  }
}

/* print data every sec */
probe timer.ms(1000){
  printf("%11d  %10d  %10d  %10d  %10d\n",
         udp_out, udp_outerr, udp_in, udp_inerr, udp_noport)
}
```

UDP Datagram Loss - udpstat.stp output

UDP_out	UDP_outErr	UDP_in	UDP_inErr	UDP_noPort
0	0	0	0	0
0	0	0	0	0
4	0	0	0	0
5	0	0	0	0
5	0	0	0	0
6	0	1	0	0
7	0	1	0	0
7	0	1	0	0
7	0	1	0	2
7	0	1	0	2
8	0	1	0	2
9	0	2	0	5
10	0	2	0	6
11	0	2	0	6
15	0	5	0	6
19	1	9	0	6
19	1	10	0	6
19	1	10	0	6
19	1	10	0	6
19	1	10	0	6

Top IO Users by User ID

Problem
Which user is doing the most IO on the system? iostat does not provide statistics on a per user basis.

Solution
Write a simple SystemTap script that probes file system read() and write() and records the bytes of IO for each user.

uid-iotop.stp

```
global reads, writes

function print_top () {
    cnt=0
    printf ("%-10s\t%10s\t%15s\n", "User ID", "KB Read", "KB Written")
    foreach (id in reads-) {
        printf("%-10s\t%10d\t%15d\n", id, reads[id]/1024,
               writes[id]/1024)
        if (cnt++ == 5)
            break
    }
    delete reads
    delete writes
}

probe kernel.function("vfs_read")  {
    reads[sprintf("%d", uid())] += count
}

probe kernel.function("vfs_write") {
    writes[sprintf("%d", uid())] += count
}

# print top 5 IO users by uid every 5 seconds
probe timer.ms(5000) {
    print_top ()
}
```

uid-iotop.stp output

```
User ID               KB Read              KB Written
504                     14237                    3163
505                     11208                     929
502                     11175                     889
503                     12469                     866
0                        1778                     183
```

Top IO Users by Process ID

Problem
Which process is doing the most IO on the system?

Solution
Convert the uid-iotop.stp script to record IO for each process instead of each user. Changes shown on next slide in ***bold italics***. Ease of changes demonstrate the flexibility of SystemTap.

pid-iotop.stp

```
global reads, writes

function print_top () {
    cnt=0
    printf ("%-10s\t%10s\t%15s\n", “Process ID", "KB Read", "KB
  Written")
    foreach (id in reads-) {
        printf("%-10s\t%10d\t%15d\n", id, reads[id]/1024,
  writes[id]/1024)
        if (cnt++ == 5)
            break
    }
    delete reads
    delete writes
}

probe kernel.function("vfs_read") {
    reads[sprintf("%d", pid())] += count
}

probe kernel.function("vfs_write") {
    writes[sprintf("%d", pid())] += count
}

# print top 5 IO users by pid every 5 seconds
probe timer.ms(5000) {
    print_top ()
}
```

pid-iotop.stp output

Process ID	KB Read	KB Written
13839	2827	25
10608	1318	303
10587	1298	314
10627	1219	454
10633	1219	438

Future Work

- Support for analyzing compiled applications
- Support for probing interpreted applications like Java
- Support for watch point probes
- Support for processor performance monitoring hardware.
- Enhanced GUI
- Speculative tracing
- Flight recorder

Conclusions

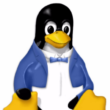
- **One tool**: SystemTap is a new performance tool for analyzing systemwide performance problems.
- **Safe**: Safety is builtin to use in production systems.
- **Realtime**: Low overhead suitable for continuous performance monitoring production systems.
- **Easy**: Easy to use by all levels of users with its familiar scripting language and intuitive GUI.
- **Effective**: Identify bottlenecks all the way from applications to OS in hours vs days to weeks.
- **On Demand**: New probe points can be added on demand, not limited to what is shipped.
- **Available**: Available on most common h/w platforms and enterprise distributions.

References

- SystemTap Project *http://sourceware.org/systemtap/*
- SystemTap GUI *http://stapgui.sourceforge.net/*
- SystemTap Wiki *http://sourceware.org/systemtap/wiki*

Disclaimers and Trademarks

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and the IBM logo are registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.



Q & A