Ubiquitous System Analysis
Performance Co Pilot

Abegail Jakop
Lukas Berk
Red Hat
Oct. 23, 2014
Introduction

- PCP Overview
  - Introduction
  - Components
- Recent Developments
  - PAPI pmda
  - pmwebd
  - Deeper metrics
- Questions?
Analyzing Performance

How is this typically/historically done?

- rsyslog/syslog-ng/journald
- top/iostat/vmstat/ps
- Mixture of scripting languages (bash/perl/python)
- Specific tools vary per platform
- Proper analysis requires more context
Introducing PERFORMANCE CO-PILOT
Performance Co-Pilot

Points of interest

- Unix-like component design
- Complements existing system functionality
- Cross platform
- Consistent unit measurement
- Extremely extensible
- Open Source!
Performance Co-Pilot

Two Underlying Components

1) Performance Metric Domain Agents
2) Performance Metric Collection Daemon
Performance Co-Pilot

Two Underlying Components

1) Agents
2) Performance Metric Collection Daemon
Performance Co-Pilot

Two Underlying Components

1) Agents
2) PMCD
Performance Co-Pilot

Agents

- Kernel
- Network
- Webserver
- Application Specific

PMCD
Performance Co-Pilot

Number of metrics exposed by agents?

- A lot! (~1500 from a default fedora install)
- Huge variation in what they're measuring
- How do you reliably and predictably name them?
Performance Co-Pilot

- Performance Metric Name Space

network.tcp.rcvpack
Performance Co-Pilot

Host A
- Kernel
- Network
- Webserver
- Application Specific
- PMCD

Host B
- pmval
- pmstat
- pmfind
- pminfo
Performance Co-Pilot

Where to start?

pminfo – display information about metrics

$ pminfo -t
Performance Co-Pilot

Where to start?

pminfo – display information about metrics

$ pminfo -t papi
Performance Co-Pilot

Where to start?

pminfo – display information about metrics

$ pminfo -t papi

    papi.system.REF_CYC  [Reference cycles]
    papi.system.L3_TCA   [L3 cache accesses]
    papi.system.L2_TCA   [L2 cache accesses]
    papi.system.L3_TCH   [L3 cache hits]
    papi.system.L2_TCH   [L2 cache hits]
Performance Co-Pilot

Where to start?

pmval – current value of a metric

$ sudo pmval papi.system.TOT_CYC

metric:    papi.system.TOT_CYC
host:      toium
semantics: cumulative counter (converting to rate)
units:     none (converting to / sec)
samples:   all

7.869E+04
9.186E+04
9.240E+04
Performance Co-Pilot

Kernel
Network
Webserver
Application Specific

PMCD

Host A

Host B

pmval
pmstat
pmfind
pminfo
Performance Co-Pilot

Host A

- Kernel
- Network
- Webserver
- Application Specific

PMCD

Host B

- pminfo
- pmlogger
Performance Co-Pilot

Kernel
Network
Webserver
Application Specific

PMCD

Host A
- pminfo
- pmlogger
- Archives

Host B
Performance Co-Pilot

pmlogger creates logs for future analysis

- Enables us to use tools on older data, retrospectively
- Default around 5mb a day, rotates and compresses
- Metrics organized, no need to stick them into elastic search
Performance Co-Pilot

- Recent and future developments
  - PAPI pmda
  - pmwebd
  - Enabling deeper system introspection
Performance Co-Pilot – Recent Developments

“Only two hard parts of computer science, cache invalidation, naming things, and off-by-one errors”

- Unknown
Performance Co-Pilot – Recent Developments

“Only two hard parts of computer science, cache invalidation, naming things, and off-by-one errors”

- Unknown
Performance Co-Pilot – Recent Developments

PAPI – Performance API

- Cross platform
- Uses dedicated hardware counters for perf metrics
  - Cache hits/misses, total instructions/cycles
- By writing a pmda (agent) for PAPI, we can expose these metrics
Performance Co-Pilot – Recent Developments

PAPI – Performance API

- Cross platform
- Uses dedicated hardware counters for perf metrics
  - Cache hits/misses, total instructions/cycles
- By writing a pmda (agent) for PAPI, we can expose these metrics
We could just view the raw values

$ sudo pmval papi.system.TOT_CYC

metric: papi.system.TOT_CYC
host: toium
semantics: cumulative counter (converting to rate)
units: none (converting to / sec)
samples: all

7.869E+04
9.186E+04
9.240E+04
We could just view the raw values

- Ratios and relative percentages are more insightful
- Perfect for the pmie tool!

PMCD → pmie
     ↑   ↑
     ↑   ↑
    pmtools
Performance Co-Pilot – Recent Developments

Performance Metrics Inference Engine

- Allow you to form metrics-based expressions for evaluation
- Ratios, counts, aggregates, conditionals
- Raise alarms, logging entries, shell commands
- Run on live data or logs
- Run rules across data from multiple hosts
Performance Co-Pilot – Recent Developments

Example pmie expression:

(papi.system.L3_TCM / papi.system.TOT_INS)
Performance Co-Pilot – Recent Developments

Example pmie expression:

$$((\text{papi.system.L3_TCM} / \text{papi.system.TOT_INS}) * 100)$$
Performance Co-Pilot – Recent Developments

Example pmie expression:

```
((papi.system.L3_TCM / papi.system.TOT_INS) * 100) > 2
```
Performance Co-Pilot – Recent Developments

Example pmie expression:

```python
some_inst

((papi.system.L3_TCM / papi.system.TOT_INS) * 100) > 2
```
Performance Co-Pilot – Recent Developments

Example pmie expression:

some_inst

((papi.system.L3_TCM / papi.system.TOT_INS) * 100) > 2

-> syslog "Percentage of Level 3 Cache misses > 2%"
Performance Co-Pilot – Recent Developments

pmwebd

- We already ship a gui tool (pmchart)
- Several feature full graphing tools available
- PCP's architecture and design makes integration easy
Performance Co-Pilot – Recent Developments

pmwebd

- We already ship a gui tool (pmchart)
- Several feature full graphing tools available
- PCP's architecture and design makes integration easy
Performance Co-Pilot – Recent Developments

pmwebd

- We already ship a gui tool (pmchart)
- Several feature full graphing tools available
- PCP's architecture and design makes integration easy
Performance Co-Pilot – Recent Developments

![Performance Co-Pilot Graphs](image-url)

- Displayed graphs show performance metrics like CPU load and network traffic over time.
- The upper graph illustrates kernel all load over a period, highlighting spikes and trends.
- The lower graph tracks network interface traffic in bytes, showing variations and patterns.

These visualizations are critical for monitoring and optimizing system performance.
Performance Co-Pilot – Recent Developments
Performance Co-Pilot – Current Developments

PCP offers wide variety of metrics

- What if we want 'under the hood' metrics?
- Need a system-wide, tool with live data to help...
Introducing:
What is SystemTap

Tool for examining live system events

- Communicated through scripts
- Links the strengths of tracers, profilers, and debuggers.

```c
// SystemTap script
probe tcp.sendmsg { gather_info; print(info) }
```

Linux Kernel Module

Summary Report

sent packet of size ... to ...
sent packet of size ... to ...
...

the all-seeing Linux Kernel
Usage

Two major components of scripts:

  1) Probe Points
  2) Handlers
Example Probe

Simple Hello world

```java
probe begin {
    println ("Hello, World!")
}
```

Or tracking when a new bash process is started

```java
probe process("bash").function("main") {
    println("A bash process has started")
}
```
Example Probe

Or something a little more complicated:

- Listing functions in the order that a process calls them

```bash
$ cat bash_functioncalls.stp
probe process("bash").function("*").call {
    printf ("bash called function %s\n", ppfunc())
}

$ stap bash_functioncall.stp
bash called function _start
bash called function ___libc_csu_init
bash called function __init
bash called function frame_dummy
bash called function register_tm_clones
bash called function main
bash called function xtrace_init
...
```
Getting Started

• Where do you start? Figure out what you can probe.
• If you don't know what probe points types there are:

  $ stap --dump-probe-types

  java(number).class(string).method(string)
  kernel.function(number)
  module(string).statement(string)
  process(string).function(string).callees
  procfs(string).read
  timer.usec(number)
  ...

Language

What can you include in handlers?

- Ordinary features you'd find in a language:
  - Globals, locals, string, integers, loops, conditionals, functions, arrays, error handling and more

Additional, handy features:

- Associative arrays, foreach loop, aggregates, macros, regex matching
Probe Points

How does one start writing a script?

- Listing mode is a great starting point
- Lists possible probe points

```
$ stap -l 'process("stap").function("symbol_*")'

process("stap").function("symbol_fetcher@elaborate.cxx:1092")
process("stap").function("symbol_table@tapsets.cxx:424")
```
Context Variables

Probes can access context variables

$ stap -L 'kprocess.create'

kprocess.create task:long new_pid:long new_tid:long
$return:struct task_struct* $clone_flags:long unsigned int ...

The context variables start with “$”
Tapsets

A library for systemtap scripts

- Their purpose is provide a level of abstraction
- Users don't have to know the exact details

For example:

```plaintext
kprocess.create = kernel.function("copy_process").return
```

For a list of all the aliased probes

```plaintext
stap --dump-probe-alias
```
Tapsets

There are also helper functions

$ cat kprocess_list.stp
probe kprocess.create {
    printf ("Process %s was started\n", pid2execname(new_pid))
}

$ stap kprocess_list.stp
Process bash was started
Process bash was started
Process soffice.bin was started
Process soffice.bin was start
Process udisksd was started
Process firefox was started

For a list of available helper functions

stap --dump-functions
Tapsets

And helper variables

$ cat helper_vars.stp
probesyscall.* {
    printf("syscall: %s, parameters: %s\n", name, $$parms$$)
}

$ stap helper_vars.stp
syscall: read, parameters: fd=4 buf=140736613309360 count=8196
syscall: fcntl, parameters: fd=4 cmd=4 arg=32770
syscall: kill, parameters: pid=4200 sig=10
syscall: fcntl, parameters: fd=4 cmd=4 arg=34818
syscall: read, parameters: fd=4 buf=140736613309360 count=8196
syscall: fcntl, parameters: fd=4 cmd=4 arg=32770
syscall: pselect6, parameters: n=5 inp=140736613308976 outp=0
    exp=0 tsp=0 sig=140736613308864
...
Example
#include <stdlib.h>
#include <stdio.h>

int sleeper () {
  static int num = 0;
  sleep(1);
  return num;
}

int main () {
  int num = 0;
  while (num < 10) {
    num = sleeper ();
    printf("a second has passed\n");
  }
  printf("10 seconds have passed\n");
  return 0;
}
$ ./terminator
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed

10\textsuperscript{th} line == 10\textsuperscript{th} second
Figure out what's going on

- Where to probe?
- Does function X ever call function Y? And with what parameters?
- What can be done if something's not quite right?
Where to probe?

$ stap -L 'process("terminator").function("*")'

Does main() ever call sleeper()?

- Check what functions main() calls.

$ stap -L 'process("terminator").function("main").callee("*")'

process("terminator").function("main@terminator.c:10").callee("sleeper@terminator.c:4") $num:int const
That was just a starting point!

- What determines when the loop ends?
- What about the return value from `sleeper`?

```bash
$ cat sleeper_return_check.stp

global old_num=-1

probe process("./terminator").function("sleeper").return {
  if ($num <= old_num)
    error("num is not increasing!")
    old_num = $num
}
```
Running the script

$ stap sleeper_return_check.stp -c ./terminator

a second has passed
a second has passed
ERROR: num is not increasing!
WARNING: Number of errors: 1, skipped probes: 0
WARNING: /home/ajakop/work/codebase/install/bin/staprun exited with status: 1
Pass 5: run failed. [man error::pass5]

Well, that explains things.

• So how can we fix this?
#include <stdlib.h>
#include <stdio.h>

int sleeper () {
    static int num = 0;
    sleep(1);
    return num;
}

int main () {
    int num = 0;
    while (num < 10) {
        num = sleeper ();
        printf("a second has passed\n");
    }
    printf("10 seconds have passed\n");
    return 0;
}
Can't do that if the program can't be stopped.

- Alternative: write a script to do it!

```bash
$ cat fix_terminator.stp

global actual_num=-1

probe process("./terminator").function("sleeper").return {
  if ($num <= actual_num)
  {
    actual_num++
    $return = actual_num
  }
  else
    actual_num = $num
}
```
terminator.c

The result:

```
$ stap -g fix_terminator.stp -c ./terminator
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
a second has passed
10 seconds have passed
```

Yay! It worked!
Performance Co-Pilot – Current Developments

Systemtap fits the bill for what we need

- Malleable output
- Able to specify various probe points
- Exposes low level information, safely

![Diagram showing relationships between PAPI, Application Specific, and PMCD]
Performance Co-Pilot – Current Developments

Systemtap fits the bill for what we need

- Malleable output
- Able to specify various probe points
- Exposes low level information, safely

PAPI

PMCD

systemtap
Performance Co-Pilot – Current Developments

Example

- Can we determine network latency on a network device?
Performance Co-Pilot – Current Developments

# stap ./net_xmit.stp eth0 dev1 dev2

# pminfo -df stap_json

stap_json.json.net_xmit_data.xmit_latency
   Data Type: 64-bit int   InDom: 130.0 0x20800000
   Semantics: counter    Units: none
   inst [0 or "dev1"] value 0
   inst [1 or "dev2"] value 0
   inst [2 or "eth0"] value 319

stap_json.json.net_xmit_data.xmit_count
   Data Type: 64-bit int   InDom: 130.0 0x20800000
   Semantics: counter    Units: none
   inst [0 or "dev1"] value 0
   inst [1 or "dev2"] value 0
   inst [2 or "eth0"] value 2304551
Performance Co-Pilot

Questions?
Get Involved!

IRC: irc.freenode.net
  #pcp
  #systemtap

Web:
  http://pcp.io
  http://sourceware.org/systemtap

Email:
  systemtap@sourceware.org
  pcp@oss.sgi.com
Get Involved!

IRC: irc.freenode.net
   #pcp
   #systemtap

Web:
   http://pcp.io
   http://sourceware.org/systemtap

Email:
   systemtap@sourceware.org
   pcp@oss.sgi.com