PmcTools: Whole-system, low-overhead performance measurement in FreeBSD

Joseph Koshy

FreeBSD Developer

18 April 2009

Outline

Goals Of This Talk

- Introduce FreeBSD/PmcTools.
- Introduce BSD culture.

About FreeBSD

- http://www.freebsd.org/
- Popular among appliance makers, ISPs, web hosting providers:
  - Fast, stable, high-quality code, liberal license.
- FreeBSD culture in one sentence: “Shut up and code”.

About jkoshy@FreeBSD.org

- FreeBSD developer since 1998.
- Technical interests:
  - Performance analysis; the design of high performance software.
  - Low power computing.
  - Higher order, typed languages.
  - Writing clean, well-designed code.

The Three Big Questions in Performance Analysis

1. What is the system doing?
2. Where in the code does the behaviour arise?
3. What is to be done about it?

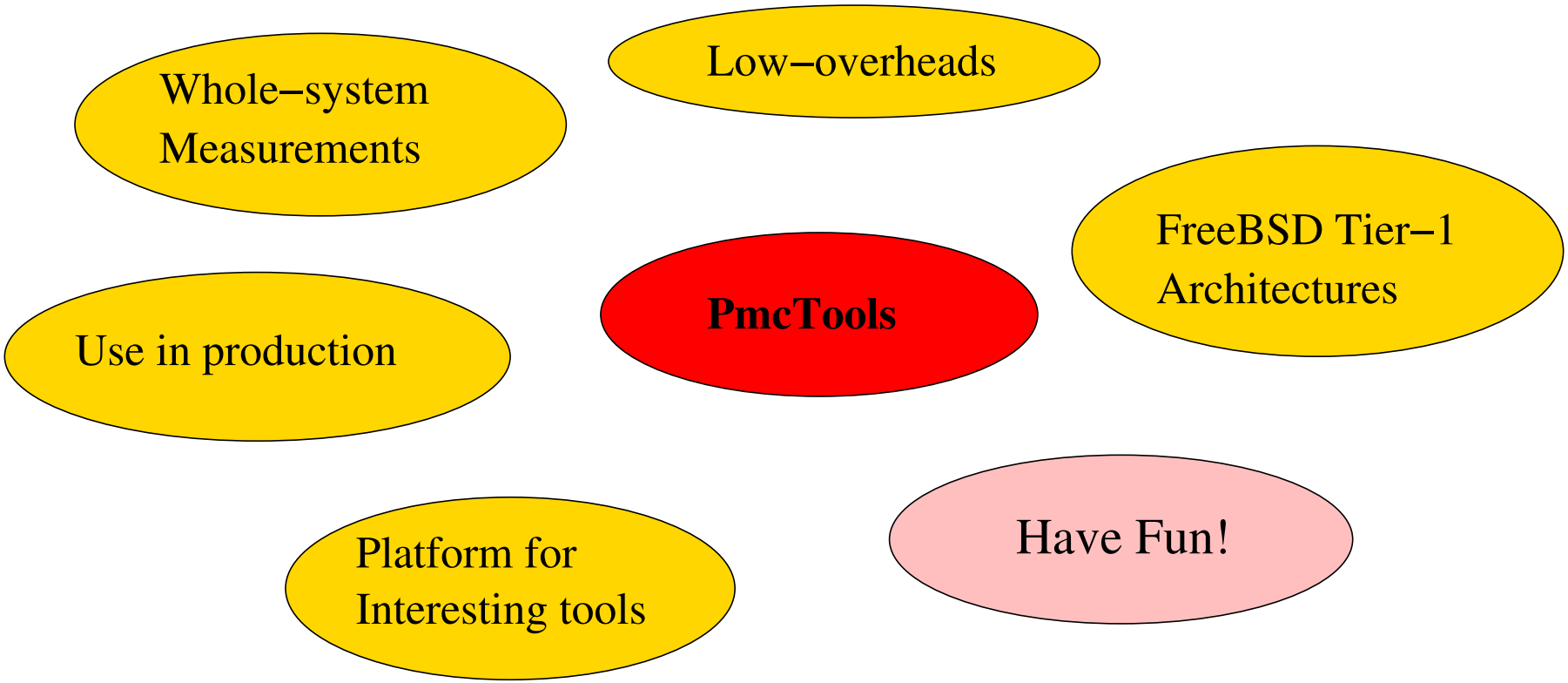
Question 1: What is the system doing?

- System behaviour:
  - Traditional UNIX tools: `vmstat`, `iostat`, `top`, `systat`, `ktrace`, `truss`.
  - Counters under the `sysctl` hierarchy.
  - Compile time options such as `LOCK_PROFILING`.
  - New tools like `dtrace`.
- Machine behaviour:
  - Modern CPUs have in-CPU hardware counters measuring hardware behaviour: bus utilization, cache operations, instructions decoded and executed, branch behaviour, floating point and vector operations, speculative execution, ...
  - Near zero overheads, good precision.

Question 2: Which portion of the system is responsible?

- Which subsystems are involved?
- Where specifically in the code is the problem arising?
- System performance is a “global” property.
  - “Local” inspection of code not always sufficient.
- As a community we are still exploring the domain of performance analysis tools:
  - Which data to collect.
  - Collecting it with low-overheads.
  - Making sense of the information collected.

PmcTools Project Goals



Performance Analysis: Conventional vs. PmcTools

Description	Conventional	PmcTools
Need special binaries	Yes	No
Dynamically loaded objects	No	Yes
Profiling scope	Executable	Process & System
Need restart	Yes	No
Measurement overheads	High	Low
Profiling tick	Time	Many options
Profile inside critical sections	No	Yes (x86)
Cross-architecture analysis	No	Yes
Distributed profiling	No	Yes
Production use	No	Yes

Related open-source projects

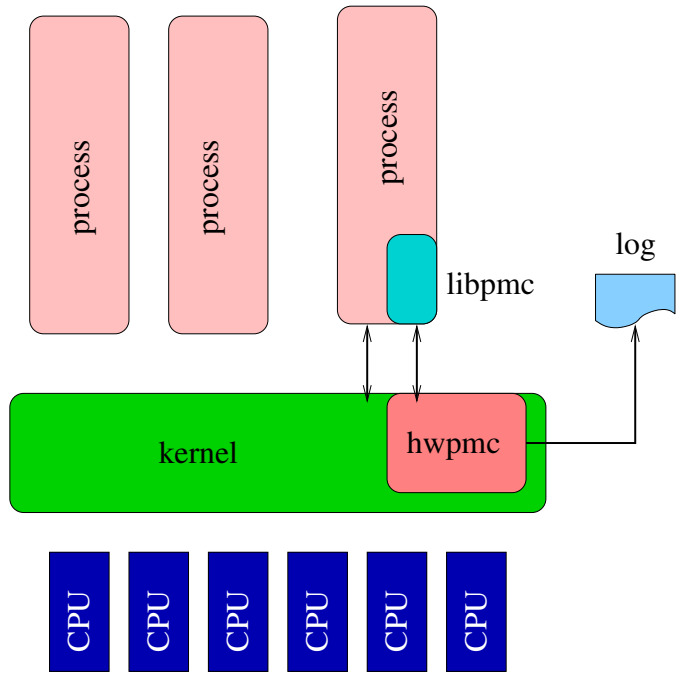
Linux Many projects related to PMCs:

- Oprofile: http://oprofile.sourceforge.net/
- Perfmon: http://perfmon2.sourceforge.net/
- Perfctr: http://sourceforge.net/projects/perfctr/
- Rabbit: http://www.scl.ameslab.gov/Projects/Rabbit/

Solaris CPC(3) library.

NetBSD A pmc(3) API.

Overview: Architecture

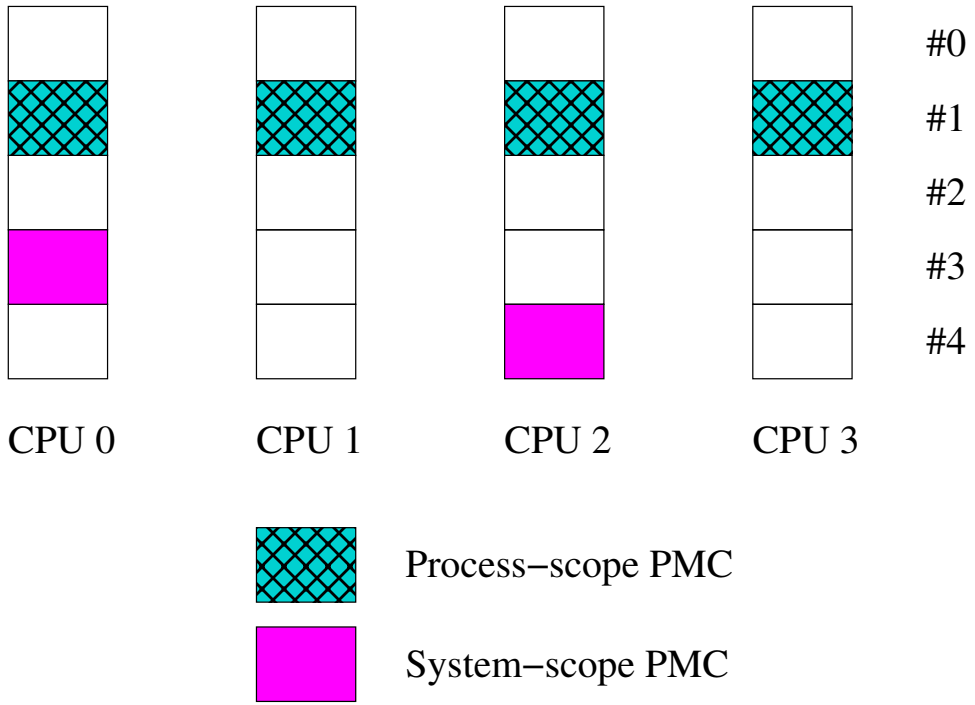


- A *platform* to build tools that use PMC data.

Components

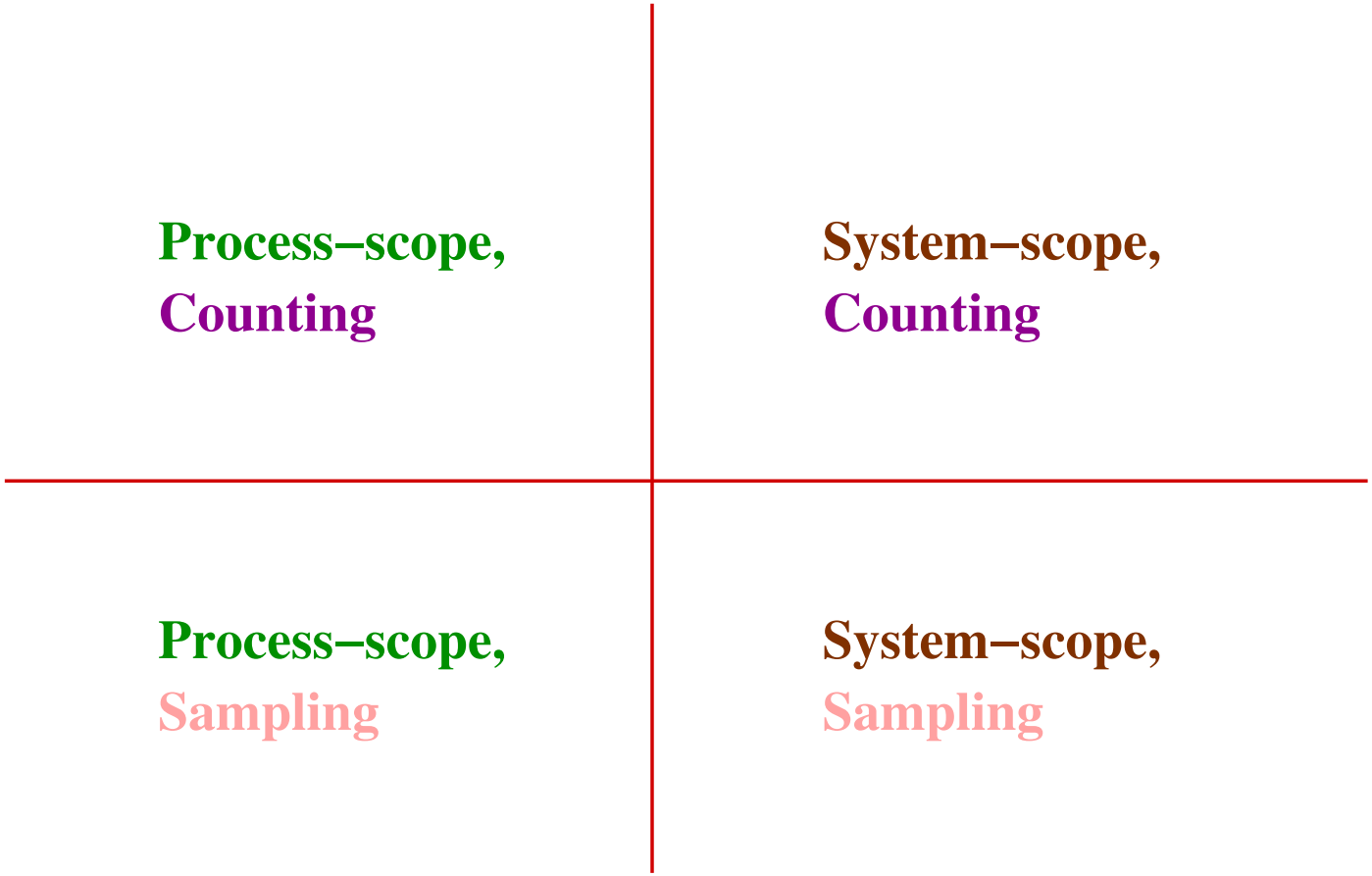
hwpmc	kernel bits
kernel changes	(see later)
libpmc	application API
pmccontrol	management tool
pmcstat	proof-of-concept application
pmcannotate	contributed tool
etc...	others in the future

PMC Scopes

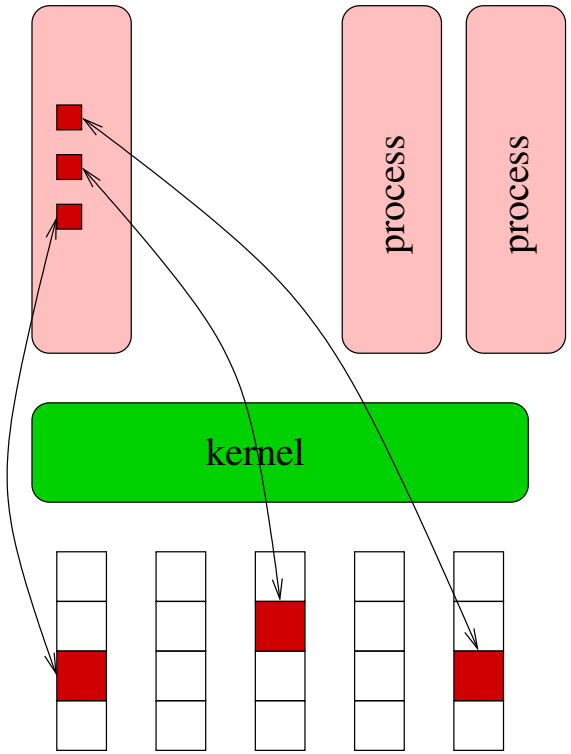


1 process-scope PMC & 2 system-scope PMCs simultaneously active.

Counting vs. Sampling

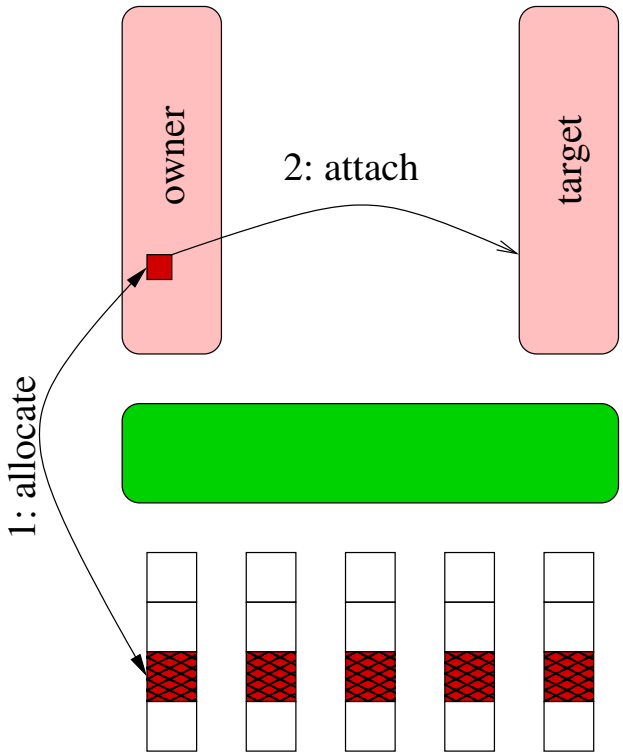


Using system-scope PMCs



- Three system scope PMCs, on three CPUs.
- Measure behaviour of the system as a whole.

Using process-scope PMCs



- A process-scope PMC is allocated & attached to a target process.
- Entire “row” of PMCs reserved across CPUs.

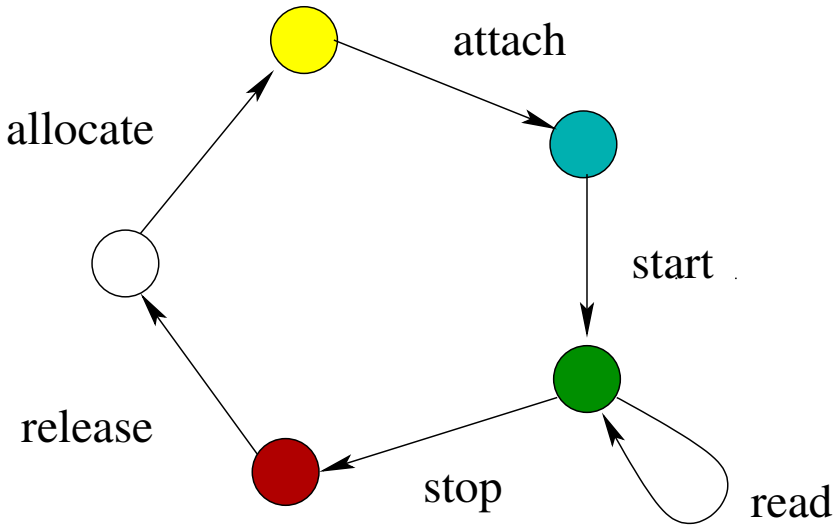
API Overview

Categories:

- Administration (2).
- Convenience Functions (8).
- Initialization (1).
- Log file handling (3).
- PMC Management (10).
- Queries (7).
- Arch-specific functions (1).

32 functions documented in 15 manual pages. 10 manual pages for supported hardware events.

Example: API Usage



`pmc_allocate()`	Allocate a PMC; returns a handle.
`pmc_attach()`	Attach a PMC to a target process.
`pmc_start()`	Start a PMC.
`pmc_read()`	Read PMC values.
`pmc_stop()`	Stop a PMC.
`pmc_release()`	Release resources.

PMCs Vary A Lot

AMD Athlon XP	4 PMCs, 48 bits wide.
AMD Athlon64	4 PMCs, Different set of hardware events.
Intel Pentium MMX	2 PMCs. 40 bits wide. Counting only.
Intel Pentium Pro	2 PMCs, 40 bits for reads, 32 bits for writes.
Intel Pentium IV	18 PMCs shared across logical CPUs. Entirely different programming model.
Intel Core	Number of PMCs and widths vary. Has programmable & fixed-function PMCs.
Intel Core/i7	As above, but also has per-package PMCs.

- PMCs are closely tied to CPU micro-architecture.
- PMC capabilities, supported events, access methods, programming constraints can vary across CPU generations.

API Design Issues

Issues:

- Designing an extensible programming interface for application use.
- Allowing knowledgeable applications to make full use of hardware.

PmcTools philosophy:

- Make simple things easy to do.
- Make complex things possible.

Current "UI" uses `name=value` pairs:

```
% pmcstat -p k8-bu-fill-request-l2-miss,\
             mask=dc-fill+ic-fill,usr
```

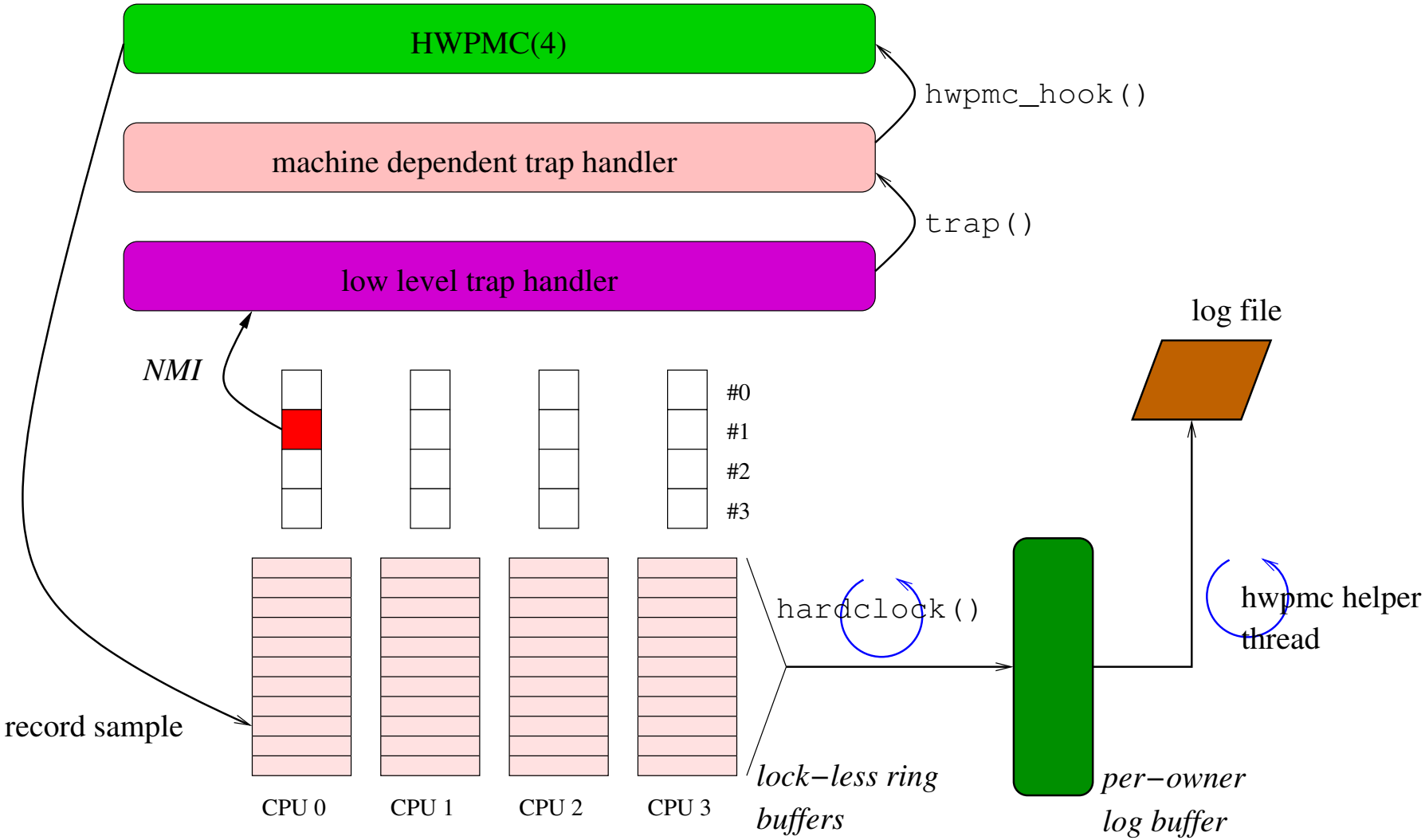
Conventional Statistical Profiling

- Needs specially compiled binaries (`cc -pg`).
- Sampling runs off the clock tick.
  - Cannot profile inside kernel critical sections.
- “In-place” record keeping.
- Call graph is approximated.

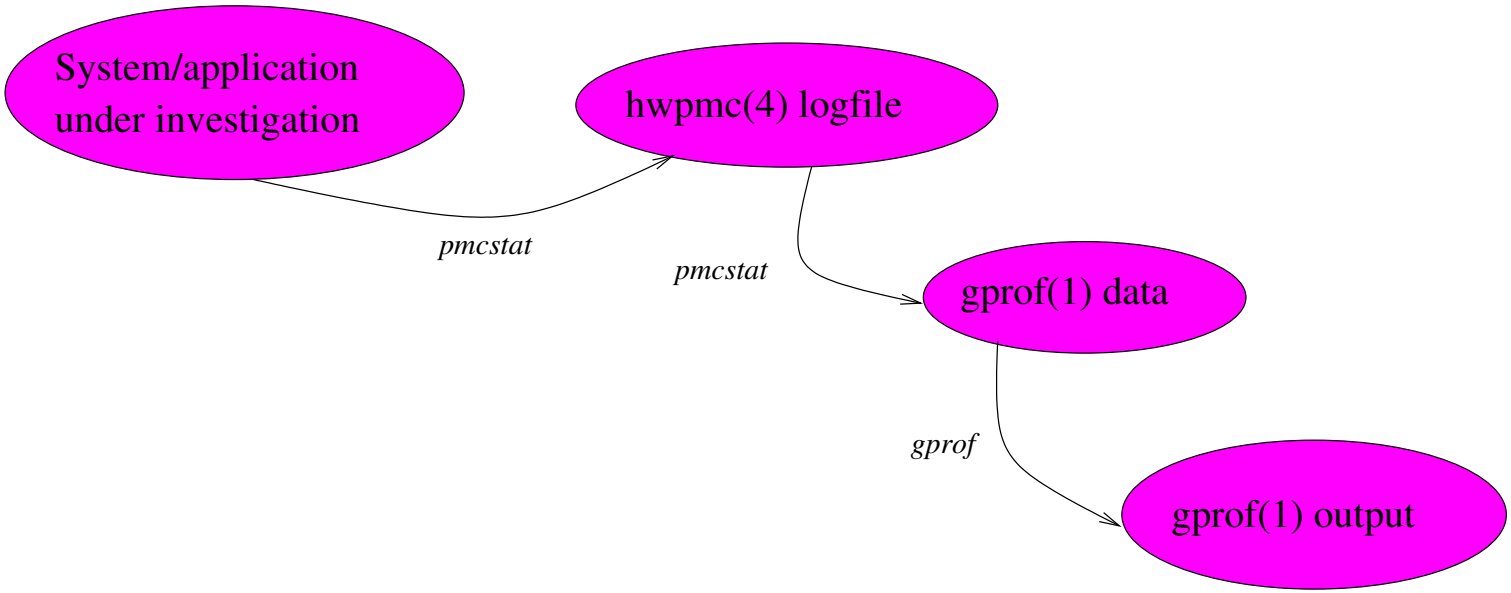
PmcTools' Statistical Profiling

- Sets up PMCs to interrupt the CPU on overflow.
- Uses an NMI to drive sampling (on x86):
  - Can profile inside kernel critical sections.
  - Needs lock-free implementation techniques.
- Separates record keeping from data collection.
- Captures the exact callchain at the point of the sample.

Profiling with NMIs

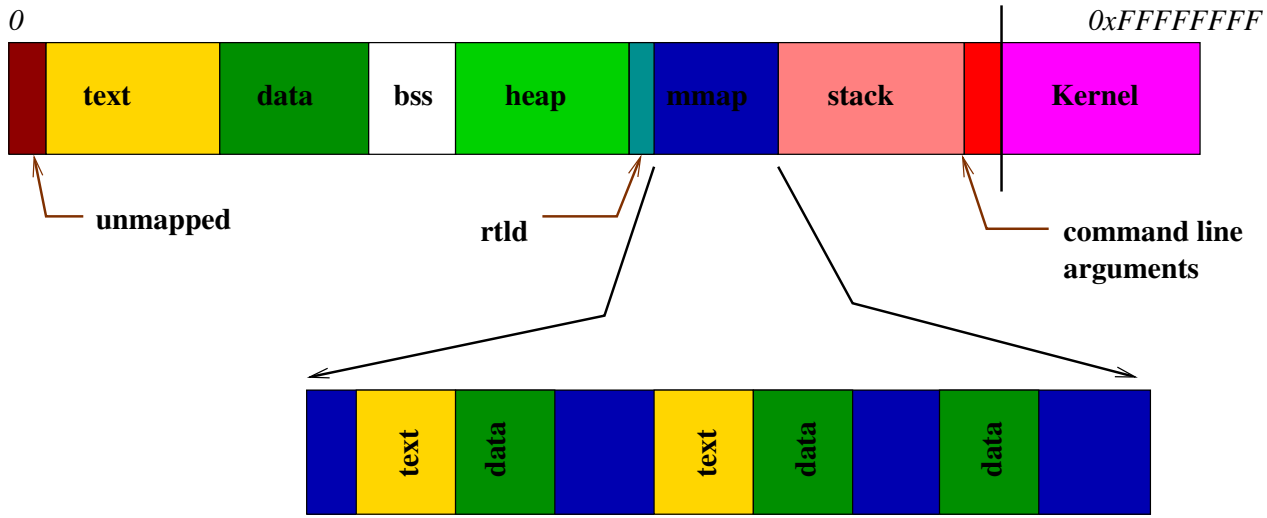


Profiling Workflow



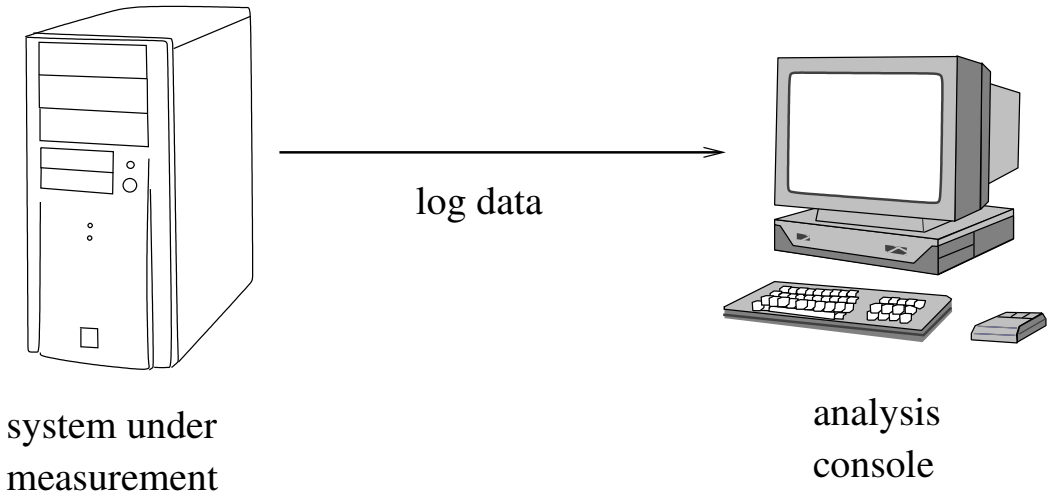
- Uses `gprof(1)` to do user reports (currently):
  - Needs to be redone: `gprof(1)` limitations.
- Call chains are captured and used to generate call graphs.

Profiling of Shared Objects



- Each executable object in the system gets its own `gmon.out` file:
  - kernel
  - kernel modules
  - executables
  - shared libraries
  - run time loader

Remote Profiling



- `pmc_configure_log()` takes a file descriptor.
- Can log to a disk file, a pipe, or to a network socket.
- Events in log file carry timestamps for disambiguation.

Implementation Information

Module	Comments
`sys/dev/hwpmc`, `sys/sys/pmc*.h`	31K LoC, i386&amd64
`lib/libpmc`	3.3K LoC
`usr.sbin/*`	5.4K LoC
documentation	29 manual pages, 11K LoD

- All public APIs have manual pages.
- All hardware events, and their modifiers are documented.
- The internal API between libpmc and hwpmc is also documented.

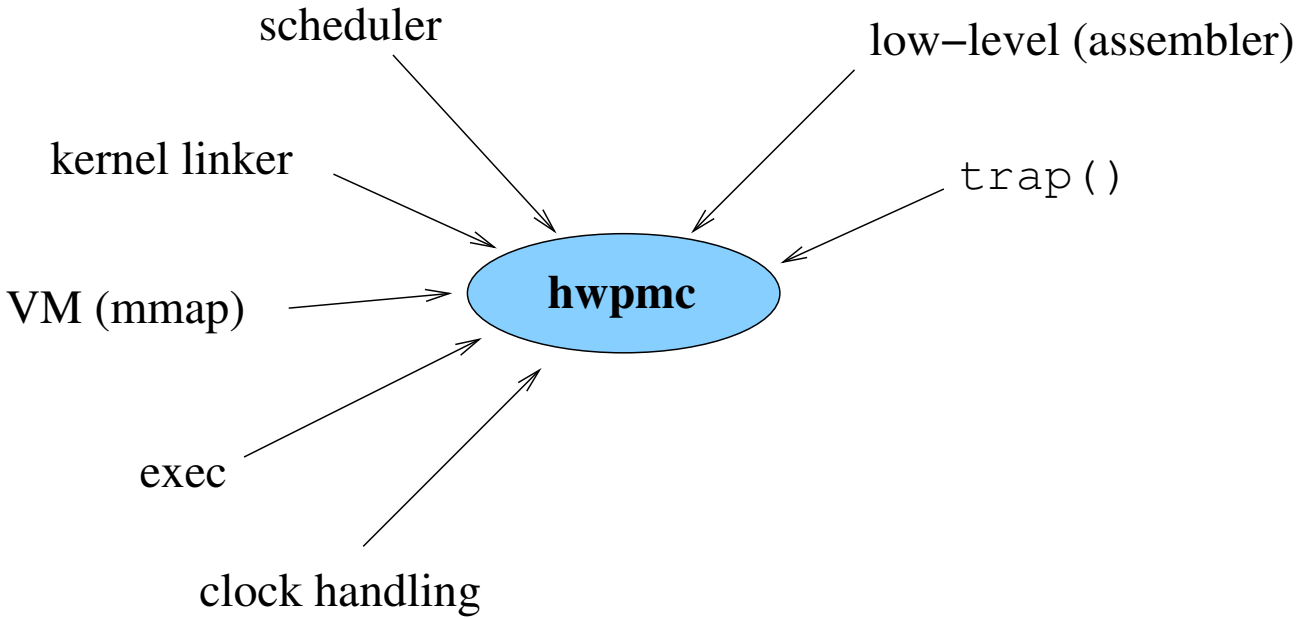
See also: “Beautiful Code Exists, If You Know Where To Look”, CACM, July 2008.

Impact on Base Kernel

Space Requirements 2 bits (`P_HWPMC`, `TDP_CALLCHAIN`). Uses free bits in existing flags words.

Kernel Changes Clock handling, kernel linker, MD code, process handling, scheduler, VM (options `HWPMC_HOOKS`).

Kernel Callbacks



Portability

Application Portability High.

Portability of libpmc Moderate. Requires a POSIX-like system.

Adding support for new PMC hardware Moderate.

- **Kernel bits** Low.

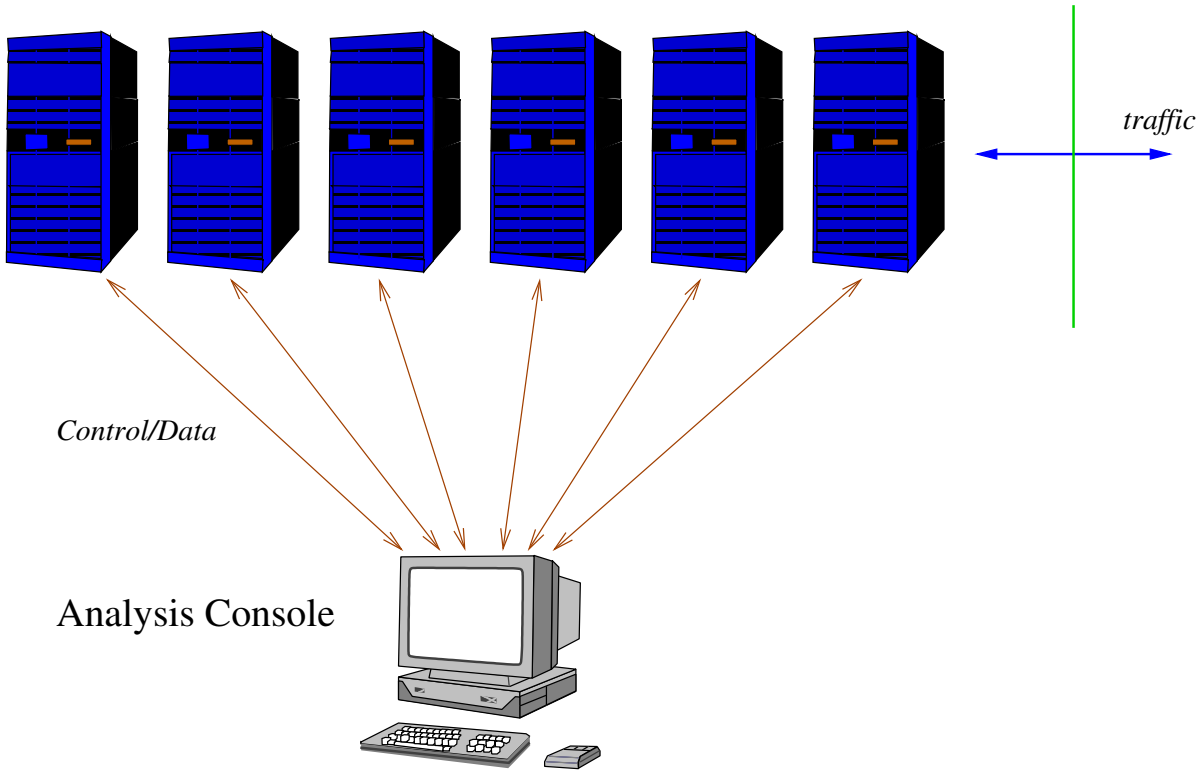
Current State

- Proof-of-concept application `pmcstat` is the current "user interface".
  - Crufty.
- Low overheads (design goal: 5%) and tunable.
- In production use. Being shipped on customer boxes by appliance vendors.
- Support load on the rise (esp. requests for support of new hardware).

Support load

- Volunteer project. Initial hardware bought from pocket, or on loan.
- Current hardware support:
  - 12 combinations of $\{$PMC hardware $\times$ 32/64 bit OS variants$\}$ $\times$ 9 OS versions [FreeBSD 6.0 $\cdots$ 6.4, 7.0 $\cdots$ 7.2, 8.0] = 108 combinations!
  - Need a hardware lab to manage testing and bug reports.
- Need an automated test suite that is run continuously.
  - Also useful for detecting OS & application performance regressions early.
- Email support load is on the rise:
  - Rise in FreeBSD adoption.
  - FreeBSD users and developers worldwide now chipping in with features, bug fixes, offering tutorials and spreading the word.

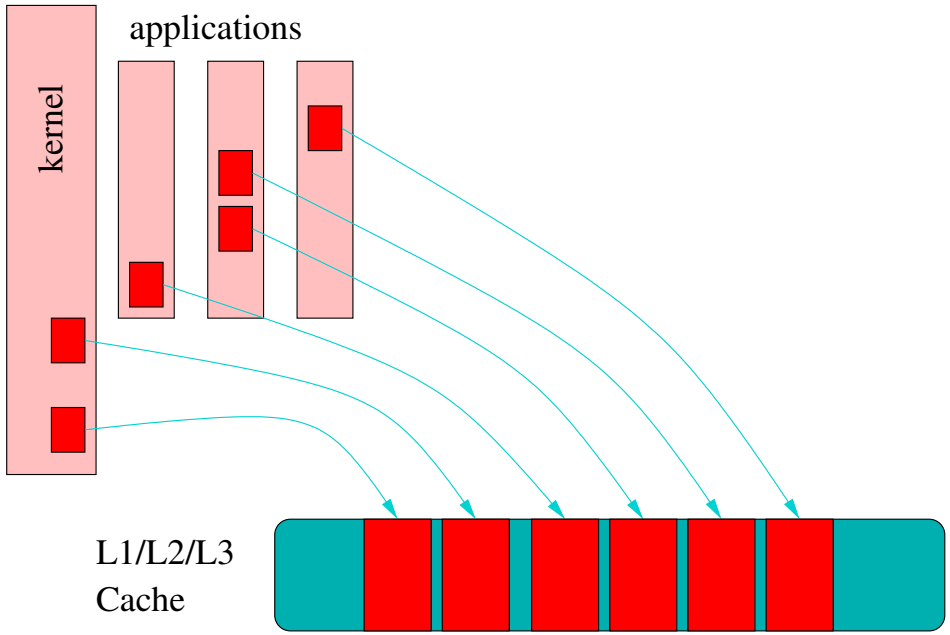
Profiling the Cloud



Other project ideas

- A graphical visualizer “console”.
- Enhance `gprof`, or write a report generator afresh.
- Link up with existing profile based optimization frameworks.
- Allow performance analysis of non-native architectures.
- Support non-x86 PMCs.
- Integrate PmcTools and DTrace.
- Port to other BSDs and/or OpenSolaris.

Profile guided system layout



- Lay out the whole system to help “hot” portions remain in cache.
- Would require an augmented toolchain (http://elftoolchain.sourceforge.net/) & enhancements to hwpmc(4).
- Useful for low end devices using direct-mapped caches.

Detection of SMP data structure layout bugs

```
struct shared
   ...
   char sh_foo;
   int sh_bar;
   char sh_buzz;
   ...
;
```

- Would use a combination of static analysis & hwpmc(4) data.
- Detection of the poor cache line layout behaviour.
  - Cache line ping-ponging between CPUs.

Profiling for power use

- What part of the system consumes power?
- Where in the code is power being spent?

Talk Summary

- FreeBSD/PmcTools was introduced.
- The design & implementation of PmcTools was looked at.
- Possible future development and research directions for the project were touched upon.