Profiling CMS production

Giulio Eulisse

Northeaster University, Boston (MA), U.S.A.

oprofile

- Non intrusive.
- Low overhead (with proper sampling rates)
- Can profile different quantities, other then raw speed: cache misses, mispredicted branches, memory accesses.
- Can profile kernel as well.
- Will be part of next stable kernel (already in 2.5.x)
- Cross platform: ports to IA-64, x86-64, Alpha, PA-RISC, sparc64, and ppc64 at various stage of completion

- Modern CPU have internal counters for various profiling related information:
  - Number of operation performed by different operational units.
  - Mispredicted branches.
  - Cache and memory access.
- The kernel can instruct the CPU so that a NMI is generated whenever one of the counter overflows a certain user decided level.
- Information on where (in which symbol) the program counter was when the NMI was thrown is then saved in some private memory area by the kernel module.
- Whenever the user requests it (by writing to `/proc/sys/dev/oprofile/dump` a userspace daemon fetches the information from kernel space and dumps them to disk in `/var/lib/oprofile/samples/`.

- IGUANA, since version 4.2.2, provides a GUI to oprofile commandline tools.
- The GUI is logically divided in two parts. A backend which fetches the information using the standard oprofile tools and a QT frontend. This was done envisaging the possibility of allowing remote operations in which the backend and the frontend are not run on the same machine.

- A kernel module (`oprofile`)
- An userspace daemon (`oprofiled`). (run as root)
- Several userspace tools:
  - `opcontrol` (needs sudo)
  - `op_time` (run by users)
  - `oprofpp` (run by users)
  - `op_to_source` (run by users)
  - `op_help` (run by users)
- A QT GUI for configuration.

Oprofile requires the presence of some paths:

- `/proc/sys/dev/oprofile/`: must be readable by users and user must be able to write to /proc/sys/dev/oprofile/dump .
- `/var/lib/oprofile/`: must be writeable by the oprofile daemon and readable by users.

- Please build with Qt support (not necessary, but eases the configuration).

- We wish to do a global performance analysis by profiling a fraction of the production.
- Our immediate wishes would be satisfied by about 10 batch nodes with oprofile installed.

- Monitoring: it would be nice to run it for a few hours a day on random machine to look for misbehaviour.
- On demand profiling: it would be nice to start the profiling remotely on the machine of their choice and profile their own jobs.

Your input is very welcomed on such topics.

The GUI is already logically divided in to two parts: the backend would run (as user) on the cluster node collecting profiling data. The frontend, most likely running on developer/user machine, gets and displays the data, either at runtime, but also offline. How the two should communicate is an open question and your input is welcome:

- Push mode? The GUI backend would be started as a common batch job, collect all the informations and send them to a server machine which provides access to the profiling information via HTTP or similar interface.
  - Pros: very low security concerns.
  - Cons: non interactive.
- Pull mode? Maybe via python remote objects/clarens/custom HTTP server?
  - Pros: interactive.
  - Cons: a (non root) daemon running on the target machine.

Especially for the following questions:

- What to profile (besides raw speed)?
- How to implement the communication between GUI backend and frontend?
- Push, pull or both?

oprofile WEB site:

http://oprofile.sf.net