Profiling

Analyze where a resource, e.g. time, is spent during program execution.

Profiling Postgres with Perf

Andres Freund

PostgreSQL Developer & Committer

Email: andres@anarazel.de
Email: andres.freund@enterprisedb.com
Twitter: @AndresFreundTec
anarazel.de/talks/edb-pune-perf-2017-11-03/profilingperf.pdf

Sampling

- Measure a continuous progress in a discrete way
- Collecting a full “trace” would be too expensive
- Usually low overhead, depends on sampling rate
- Sampling:
  - Every …Seconds (perf's -F option)
  - Every … Events (perf's -c option)

Tracing

- Collect discrete events
- Full tracing of all events too expensive
- Full tracing of all events of a type often also too expensive
- static tracing: predefined event types
- dynamic tracing: new tracepoint at runtime

What's perf

An annoyingly named suite of linux tools

- sampling, tracing recording : `perf record`
- display recorded data: `perf report`
- show live events: `perf top`
- event counting: `perf stat`
- dynamic tracing: `perf probe`
- list events: `perf list`
- collect information to move perf.data: `perf archive`

Setup Perf

- Install perf:
  - debian/ubuntu: `apt-get install linux-tools`
  - Red-Hat based: `yum install perf`
- enable useful profiling for everyone:

```
sudo sysctl kernel.perf_event_paranoid=-1
sudo sysctl kernel.kptr_restrict=0
```

- make it persistent:

```
sudo tee /etc/sysctl.d/60-perf.conf <<EOF
kernel.kptr_restrict=0
kernel.perf_event_paranoid=-1
EOF
```

Prepare Applications

- Install debugging symbols

```
apt-get install libc6-dbg postgresql-9.6-dbg

debuginfo-install postgresql96
```

- Recompile with frame pointers enabled
  - framepointers allow efficient hierarchical profiling

```
./configure CFLAGS='-fno-omit-frame-pointer -ggdb -O2' …
```

  - newer debian/ubuntu packages have it enabled
  - help me lobby devrim to enable it yum.pg.o ;)

Basic Approach

- Choose Event(s) to profile. Default is 'cycles'
- perf record && perf report
- perf top

Recent Customer Example #1

Call Graph Profiling

- Sample Stack for Events
- Different methods
  - fp: efficient, default, requires compilation flag, works in VMs
  - lbr: efficient, requires new hardware, only hardware events, no tracepoints
  - dwarf: slow, large data, works always, requires debuginfo, works in VMs
- Use lbr if you can, fp otherwise, fall back to dwarf

What to record

- Everything (till ctrl-c): perf record -a
- Everyting for a while: perf record -a sleep 5
- A specific process (till ctrl-c): perf record -p $pid
- A specific process for a while: perf record -p $pid sleep 5
- A command: perf record somecommand
- Important options:
  - -a – systemwide profiling
  - -g / --call-graph $method – include stack in samples
  - -e event-desc1 – what event(s) to measure
  - -F # – sampling frequency
  - -f $file – store output in $file

What to show

- perf report options:
  - --children – include cost of children in sorting
  - --no-children – do not include cost of called functions
  - --sort comm,dso,symbol,… – fields to “group by”
  - --stdio // --tui // --gtk – frontend

Customer Example #2

Moving profiles between systems

- perf report requires debug information available
- perf archive builds package with required debug info

```
andres@alap4:~/src/postgresql$ perf archive
perf.data
Now please run:


$ tar xvf perf.data.tar.bz2 -C ~/.debug


wherever you need to run 'perf report' on
```

Events

- perf list (depends on user permissions!)
- perf help list – syntax for event descriptors
- Important Hardware Events:
  - cycles (both hard & software)
  - cache-misses
  - branch-misses
  - modifiers: pp (precise), u/k (user/kernel)
- Important OS Events
  - page-faults
  - context-switch
- Fewer Hardware events in VMs (especially “cloud”)

Sequential Scan Example #1

Static Tracepoints

- Interesting Tracepoints
  - raw_syscalls:sys_enter – look at all the tracepoints
  - syscalls:sys_enter_semop – profile lwlock waits
  - syscalls:sys_enter_select – profile spinlock waits
  - block:* – block layer tracepoints
  - sched:* – scheduler tracepoints
- Require root
- A bit faster than static tracepoints
- full trace by default, use -F to sample frequent ones

Dynamic Tracepoints

- Manage Dynamic Tracepoints
  - perf probe -l – list dynamic tracepoints
  - perf probe -x binary --add … – add tracepoint to binary
  - perf probe –del event/event*
  - perf probe -x … --line $func – show lines you can trace
- --add function/function:line/…
- Require Debug Information
- Very useful, especially for measuring contention, causes of load and such
- Multiple Matches, _1, _2, ...

Important Dynamic Tracepoints

- s_lock – unavailable spinlock
- LWLockWakeup – blocked others in lwlock
- ProcSleep – waiting for other backend, e.g. heavyweight lock
- WaitLatchOrSocket – waiting for something, client commands or e.g. a proc wakeup
- XLogInsert()

Secret Workload #17

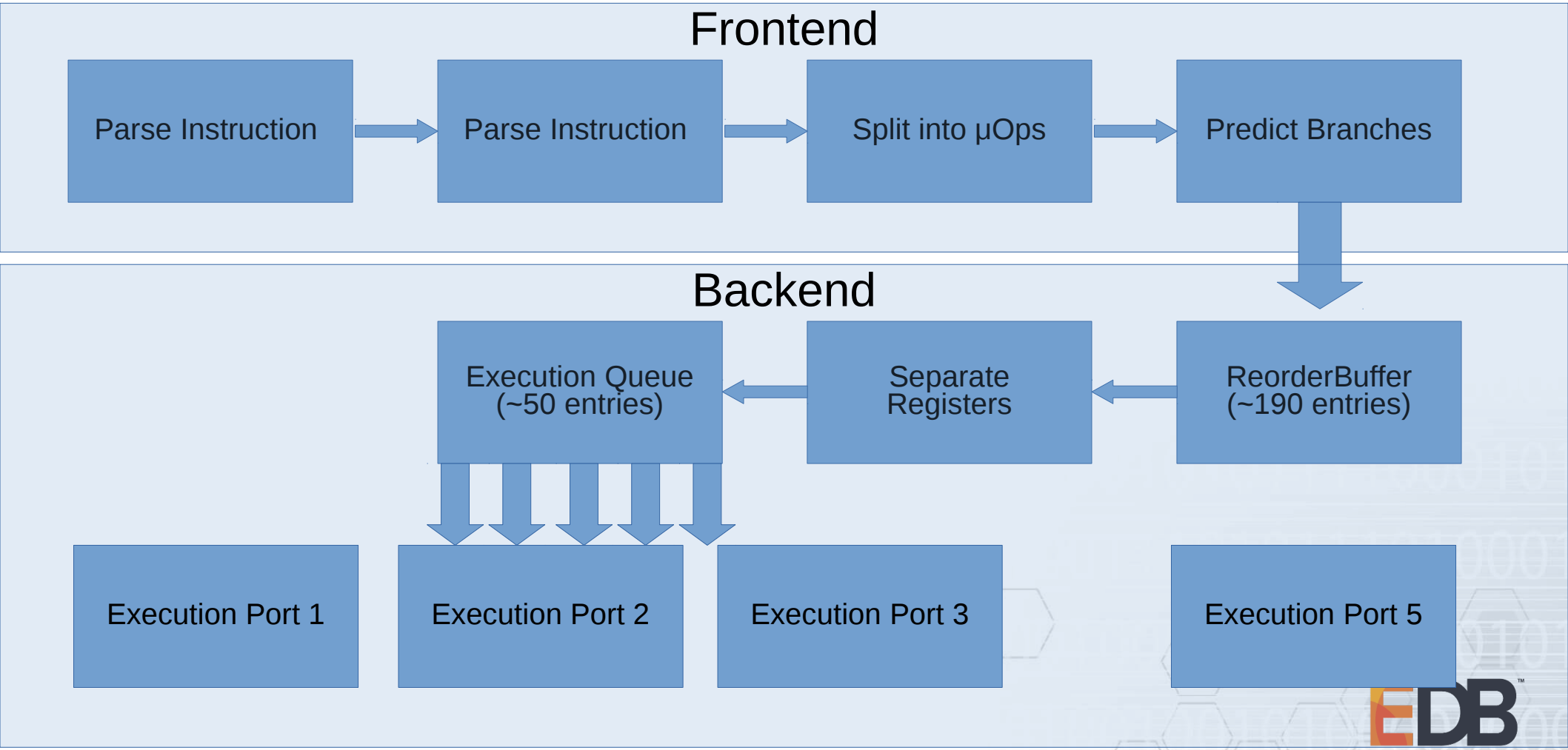
```
Available samples
0 probe_postgres:XLogWrite
0 probe_postgres:XLogInsert
185 probe_postgres:WaitLatchOrSocket
25K probe_postgres:s_lock
0 probe_postgres:ProcWakeup_1
0 probe_postgres:ProcWakeup
0 probe_postgres:ProcSleep
0 probe_postgres:LWLockWakeup

ESC: exit, ENTER|->: Browse histograms
```

Additional Tools

- flame graph generator
  - https://github.com/brendangregg/FlameGraph
  - shows profile over time in a graphical manner
- BPF based tracing
  - https://github.com/iovisor/bcc
  - http://archives.postgresql.org/message-id/20170622210845.d2hsbqv6rxu2tiye%40alap3.anarazel.de
  - http://anarazel.de/t/2017-06-22/pgsemwait_8_async.svg
- pmu-tools
  - ocperf list – show low level intel hardware events
  - toplev – look for “pipeline bottleneck”
    - highlevel, not line level profile

Quick Intro into modern CPUs



EDB POSTGRES

Consequences of modern CPUs

- Out-of-Order hides latencies
- Hidden latencies make profiling much harder
  - sometimes a cache miss is fata
  - most of the time a cache miss is harmless
- Independent instructions allow reordering
- Stalling the entire pipeline is extremely expensive
- Should have it's own talk