Benchmark Noise Reduction:
How to Configure Your Machines for Stable Results

Santa Clara, California | April 23th – 25th, 2018
Privet!

2004 - 2010: Performance Engineer, Software Engineer @ MySQL AB / Sun Microsystems / Oracle

2010 - 2015: Principal Software Engineer, Project Lead @ Percona

2015 - NOW(): MySQL/InnoDB Performance Expert @ Cavium

sysbench maintainer
Why reduce benchmark noise?
CPU frequency scaling
CPU frequency scaling
CPU frequency scaling

- Modern CPU cores can scale their frequencies up and down
  - based on temperature, load & OS power saving policies
CPU frequency scaling

- Modern CPU cores can scale their frequencies up and down
  - based on temperature, load & OS power saving policies
- The most easily identified problem:

  $ grep MHz /proc/cpuinfo
  $ lscpu
  $ cpupower --c all frequency-info
CPU frequency scaling

- Modern CPU cores can scale their frequencies up and down based on temperature, load & OS power saving policies
- The most easily identified problem:

  $ grep MHz /proc/cpuinfo
  $ lscpu
  $ cpupower --c all frequency-info

- and the most frequently hit too!
CPU frequency scaling
CPU frequency scaling

Balancing power and performance:
CPU frequency scaling

Balancing power and performance:

- P-states:
  - busy cores, P0 — maximum performance, Pn — reduced voltage/frequency with n > 0
CPU frequency scaling

Balancing power and performance:

- **P-states:**
  - busy cores, P0 — maximum performance, Pn — reduced voltage/frequency with n > 0

- **C-states:**
  - idle cores, C0 — not sleeping, Cn — deeper sleep levels with n > 0
Balancing power and performance:

- **P-states:**
  - busy cores, \(P_0\) — maximum performance, \(P_n\) — reduced voltage/frequency with \(n > 0\)

- **C-states:**
  - idle cores, \(C_0\) — not sleeping, \(C_n\) — deeper sleep levels with \(n > 0\)

Higher P- and C-states are a major source of noise in benchmarks
Turbo mode

- Turbo Boost™ in Intel CPUs
- similar technologies by other vendors and in other architectures
- dynamic overclocking
- increased frequency is limited by HW limits and the number of currently active cores
- complicates core-to-core and scalability comparisons
CPU frequency scaling: What You Can Do
CPU frequency scaling: What You Can Do

- disable higher P-states by setting CPU governor to `performance`:

  ```
echo performance | sudo tee \
/sys/devices/system/cpu/cpu[0-9]*/cpufreq/scaling_governor
  ```
CPU frequency scaling: What You Can Do

- disable higher P-states by setting CPU governor to `performance`:

  ```
  echo performance | sudo tee \
  /sys/devices/system/cpu/cpu[0-9]*/cpufreq/scaling_governor
  ```

- disable higher C-states via PM QOS:

  ```
  (echo 0; cat) > /dev/cpu_dma_latency &
  ```

  or use `pmqos-static.py` from `tuned`
CPU frequency scaling: What You Can Do

- disable higher P-states by setting CPU governor to **performance**:

  ```bash
  echo performance | sudo tee /sys/devices/system/cpu/cpu[0-9]*/cpufreq/scaling_governor
  ```

- disable higher C-states via PM QOS:

  ```bash
  (echo 0; cat) > /dev/cpu_dma_latency &
  ```

  or use `pmqos-static.py` from **tuned**

- disable TurboBoost:
  - with intel_pstate
    ```bash
    echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo
    ```
  - without intel_pstate, use Machine-Specific Registers and msr-tools
    ```bash
    wrmsr -a 0x1a0 0x4000850089
    ```
CPU scheduler
CPU scheduler tuning
CPU scheduler tuning

More an art than a science

```
sysctl -a | grep sched | grep -cv domain
14
```
CPU scheduler tuning

More an art than a science

```
sysctl -a | grep sched | grep -cv domain
```

14

Inadequate settings may result in higher context switches & cache misses
CPU scheduler tuning

- More an art than a science
- `sysctl -a | grep sched | grep -cv domain`
- 14
- Inadequate settings may result in higher context switches & cache misses
- There’s no universal solution
CPU scheduler tuning

More an art than a science

```
sysctl -a | grep sched | grep -cV domain
14
```

Inadequate settings may result in higher context switches & cache misses

There's no universal solution

This is what I use for sysbench OLTP:

- CFS (the default) is best
- Disable autogrouping:

```
sysctl kernel.sched_autogroup_enabled=0
```

- Raise minimal granularity from default:

```
sysctl kernel.sched_min_granularity_ns=5000000
```
Memory management
Address space layout randomization
Address space layout randomization

- addresses of program code, libraries and data are different on each invocation
Address space layout randomization

- addresses of program code, libraries and data are different on each invocation
- enabled by default
Address space layout randomization

- addresses of program code, libraries and data are different on each invokation
- enabled by default
- can affect performance by multiple times:
  - Causes of Performance Instability due to Code Placement in X86
Address space layout randomization

- Addresses of program code, libraries and data are different on each invocation.
- Enabled by default.
- Can affect performance by multiple times:
  - Causes of Performance Instability due to Code Placement in X86
- Enabled by default.
Address space layout randomization

- addresses of program code, libraries and data are different on each invocation
- enabled by default
- can affect performance by multiple times:
  - [Causes of Performance Instability due to Code Placement in X86](#)
- enabled by default
- to disable:

  ```bash
  sysctl kernel.randomize_va_space=0
  ```
Address space layout randomization

- addresses of program code, libraries and data are different on each invocation
- enabled by default
- can affect performance by multiple times:
  - [Causes of Performance Instability due to Code Placement in X86](#)
- enabled by default
- to disable:
  - `sysctl kernel.randomize_va_space=0`
- Security feature, don’t try this at home in production!
NUMA
NUMA

- NUMA auto-balancing
  - moves memory and tasks to avoid remote memory access
  - works by unmapping pages and handling page faults
  - potentially improves performance at the cost of latency jitter
  - enabled by default on most systems
NUMA

- NUMA auto-balancing
  - moves memory and tasks to avoid remote memory access
  - works by unmapping pages and handling page faults
  - potentially improves performance at the cost of latency jitter
  - enabled by default on most systems

- Disable for benchmarks:
  
  ```
sysctl kernel.numa_balancing=0
  ```
NUMA

- NUMA auto-balancing
  - moves memory and tasks to avoid remote memory access
  - works by unmapping pages and handling page faults
  - potentially improves performance at the cost of latency jitter
  - enabled by default on most systems
- Disable for benchmarks:
  - `sysctl kernel.numa_balancing=0`
  - Don’t forget about `innodb_numa_interleave=1` in `my.cnf`
Swap
Swap

- Minimize (not disable) swapping:
  
  `sysctl vm.swappiness=1`

  "In defence of swap: common misconceptions" by Chris Down
Swap

- Minimize (not disable) swapping:
  
  ```
  sysctl vm.swappinnes=1
  ```

  "In defence of swap: common misconceptions" by Chris Down

- `innodb_numa_interleave=1` in `my.cnf`
Swap

- Minimize (not disable) swapping:
  
  ```
  sysctl vm.swappinnes=1
  ```

- "In defence of swap: common misconceptions" by Chris Down

- `innodb_numa_interleave=1` in `my.cnf`

- To ensure allocation fairness between nodes:
Minimize (not disable) swapping:

```
sysctl vm.swappiness=1
```

"In defence of swap: common misconceptions" by Chris Down

```
inoddb_numa_interleave=1 in my.cnf
```

To ensure allocation fairness between nodes:

```
sync; sysctl vm.drop_caches=3
```
Transparent Huge Pages

- **Disable:**
  
  ```
  echo never > /sys/kernel/mm/transparent_hugepage/enabled
  echo never > /sys/kernel/mm/transparent_hugepage/defrag
  ```
Memory allocators
Memory allocators

MySQL is a heavy `malloc()` user
Memory allocators

- MySQL is a heavy `malloc()` user
- glibc/jemalloc/tcmalloc performance & scalability heavily depend on the version
Memory allocators

- MySQL is a heavy malloc() user
- glibc/jemalloc/tcmalloc performance & scalability heavily depend on the version
- glibc is improving, but…
  - scalability & fragmentation is a problem in LTS distributions
  - tcmalloc is good enough in LTS distributions, broken in Ubuntu Artful
  - jemalloc gives the most stable & consistent results, sane default behavior
Memory allocators

- MySQL is a heavy `malloc()` user
- glibc/jemalloc/tcmalloc performance & scalability heavily depend on the version
- glibc is **improving**, but…
  - scalability & fragmentation is a problem in LTS distributions
  - tcmalloc is good enough in LTS distributions, broken in Ubuntu Artful
  - jemalloc gives the most stable & consistent results, sane default behavior
- for benchmarks, make sure to use the **same version of same library** with **same settings**!
Spectre and Meltdown mitigations
Spectre and Meltdown mitigations

- major headache for benchmarks
Spectre and Meltdown mitigations

- major headache for benchmarks
- overhead up to hundreds of %, depends on:
  - the workload
  - kernel version
  - CPU microcode version
  - compiler version and flags
Spectre and Meltdown mitigations

-_major headache for benchmarks
- overhead up to hundreds of %, depends on:
  - the workload
  - kernel version
  - CPU microcode version
  - compiler version and flags
- no runtime tuning
Spectre and Meltdown mitigations

- major headache for benchmarks
- overhead up to hundreds of %, depends on:
  - the workload
  - kernel version
  - CPU microcode version
  - compiler version and flags
- no runtime tuning
- make sure mitigations are as close as possible between compared systems
sysbench tune
sysbench tune

$ sysbench tune list
$ sudo sysbench tune apply --profile=mysqllbench

Inspired by tuned, python-perf system and Krun
but more portable
already available from rocks.sysbench.io
pull requests are welcome!
Summary

- achieving stable & consistent benchmark results gets increasingly harder
- all existing knowledge is fragmented & mostly scratches the surface
- YMMV, test on your workloads
- feedback on sysbench tune is welcome!
Links

- Victor Stinner’s articles on benchmark stability (with focus on microbenchmarks):
  - https://vstinner.readthedocs.io/benchmark.html
- Brendan Gregg’s talk on tuning production cloud instances:
  - "How Netflix Tunes EC2 Instances for Performance"
- Øystein Grøvlen’s post on improving stability of MySQL benchmarks:
- Upcoming series of posts in my blog
  - http://kaamos.me/blog
Thank You!

Questions?