Investigating and reducing latency of trading applications

Konstantin Volkov,

Tracing Summit 2017

About me

- DevOps engineer
- Infrastructure for trading applications
- Containers, configuration automation, kernel technologies, performance/tracing tools

Agenda

- **Use cases**

Case #1

- Program allocates several gigabytes of memory
- Performs math calculations
- After system software update on one of the servers, program runs ~50% slowly.

Assumptions:

- Configuration issue
- Increased load on the system
- Hardware problem

Conventional diagnosis

- `uptime(1), top(1), ps(1)`

basic investigation reveals no additional running processes or parasite load

Conventional diagnosis (cont.)

`time(1)` utility:

- healthy server:

```
0.14user 2.67system 0:02.84elapsed
```

- impacted server:

```
0.14user 4.98system 0:05.14elapsed
```

elapsed +55% increase, system +53%

Conventional diagnosis (cont.)

```
# strace -c <program>
```

```
% time   seconds usecs/call calls errors syscall
------ --------- ---------- ----- ------ -------
100.00  0.259030      43172     6         munmap
  0.00  0.000000          0     1           read
------ --------- ---------- ----- ----- --------
100.00  0.262200      43700     6         munmap
  0.00  0.000000          0     1           read
```

total time spent in syscalls increased by 3ms

Conventional diagnosis (cont.)

`mpstat(1) (%irq and %soft)`

both servers do not experience any significant interrupt load

Advanced diagnosis

```
# perf record <program>
# perf report
```

Advanced diagnosis (output)

```
# Overhead   Command     Shared Object                              Symbol
# ........  ........  ................  .............................
#
    57.68%  program  [kernel.kallsyms]  [k] clear_page_c
     7.76%  program  [kernel.kallsyms]  [k] page_fault
     6.40%  program  [kernel.kallsyms]  [k] _raw_spin_lock
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    29.30%  program  [kernel.kallsyms]  [k] clear_page_c
    19.67%  program  [kernel.kallsyms]  [k] isolate_migratepages_range
    16.52%  program  [kernel.kallsyms]  [k] compaction_alloc
```

different sets of functions contribute to the profile

Advanced diagnosis (cont.)

```
isolate_migratepages_range()
compaction_alloc()
```

Both defined in `mm/compaction.c`

Advanced diagnosis (cont.)

Documentation/sysctl/vm.txt

compact_memory

Available only when CONFIG_COMPACTION is set. When 1 is written to the file, all zones are compacted such that free memory is available in contiguous blocks where possible. This can be important for example in the allocation of huge pages although processes will also directly compact memory as required.

Case #1 remediation

```
# echo never > \
/sys/kernel/mm/transparent_hugepage/defrag
```

Case #2

- Freshly setup server constantly spends 30% of time in system
- No production software running yet

Assumptions:

- Huge amount of interrupts?

But there’s no load yet applied

Advanced diagnosis

- `perf` to collect execution profile of the whole system

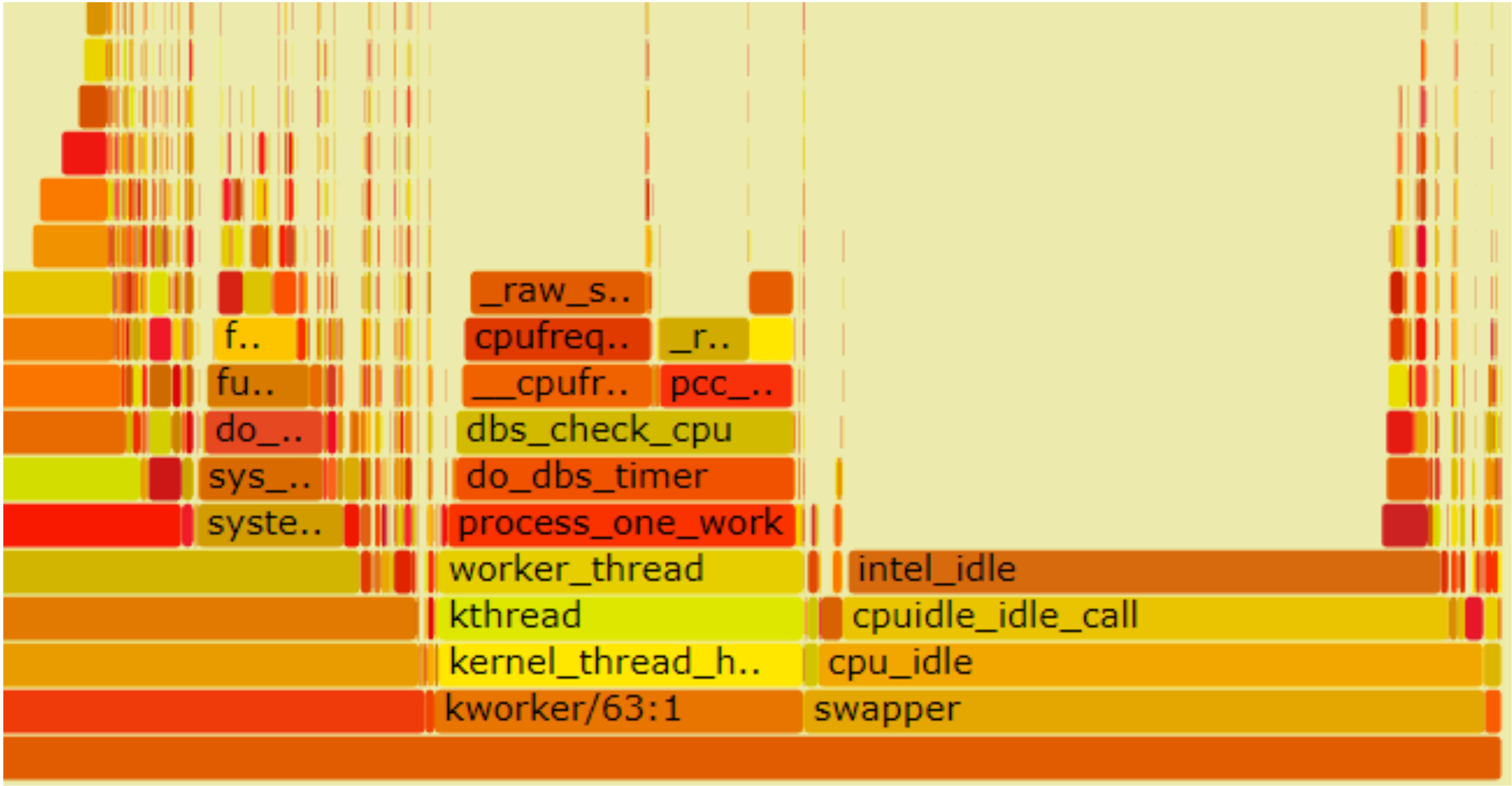
Advanced diagnosis (cont.)

```
# Overhead    Command       hared Object                     Symbol
# ........  .........  ................. ..........................
#
    60.29%    swapper  [kernel.kallsyms] [k] intel_idle
     5.20%    swapper  [kernel.kallsyms] [k] acpi_os_read_port
     3.54%    swapper  [kernel.kallsyms] [k] menu_select
     3.34%    swapper  [kernel.kallsyms] [k] _raw_spin_lock_irqsave
```

- idling task is dominating in the profile
- no other visible time consumer

Advanced diagnosis (cont.)

CPU flame graphs to the rescue


_raw_s..
cpufreq..
_r..
f..
fu..
__cpufr..
pcc_..
do_..
dbs_check_cpu
sys_..
do_dbs_timer
syste..
process_one_work
worker_thread
intel_idle
kthread
cpuidle_idle_call
kernel_thread_h..
cpu_idle
kworker/63:1
swapper

Advanced diagnosis (cont.)

- `_raw_spin_lock_irqsave()` comes from CPU frequency scaling code
- looks like cpufreq code has one global lock, on the system with 64 CPUs this leads to a sensible contention

Case #2 remediation

```
# echo performance > \
/sys/devices/system/cpu/cpu*/
cpufreq/scaling_governor
```

Case #3

- synchronous writes take to much time to complete (10 sec).

Assumptions

- Hardware problem
- Increased load

Conventional diagnosis

```
# iostat -x

%util:   100.00

svctm:   2.24

w_await: 322.17
```

Advanced diagnosis

- `ftrace events via trace-cmd(1)`

```
3094618.749527: block_rq_insert:   386645440
3094618.753639: block_rq_complete: 386645440
```

it takes 4ms to service IO request

Advanced diagnosis (cont.)

- `ftrace function_graph`

```
3094618.749248: funcgraph_entry: SyS_fsync()
3094628.729051: funcgraph_exit:
```

`fsync()` system call takes 10 sec to complete

Advanced diagnosis (cont.)

```
jbd2_log_wait_commit() {
  _raw_read_lock();
  __wake_up() {
    _raw_spin_lock_irqsave();
    __wake_up_common();
    _raw_spin_unlock_irqrestore();
  }
  prepare_to_wait_event() {
    _raw_spin_lock_irqsave();
    _raw_spin_unlock_irqrestore();
  }
  schedule() {
```

Advanced diagnosis (cont.)

```
kworker/u8:2-1718  [000] 3094619.035436: block_rq_insert:
kworker/u8:2-1718  [000] 3094619.035463: kernel_stack:
=> blk_flush_plug_list (ffffffff81285258)
=> blk_queue_bio (ffffffff812854ca)
=> generic_make_request (ffffffff81280cb0)
…….
=> __writeback_single_inode (ffffffff811d1c09)
=> writeback_sb_inodes (ffffffff811d2964)
=> __writeback_inodes_wb (ffffffff811d2c56)
=> wb_writeback (ffffffff811d2f03)
```

Lots of similar events happening while our our task is waiting

Advanced diagnosis (cont.)

Looks like journaling can not advance while under heavy writeback

Case #3 remediation

- Decrease write back buffer, e.g.

  `dirty_ratio`

Thank you!