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)**

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Command</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>clear_page_c</strong></td>
</tr>
<tr>
<td>57.68%</td>
<td>program</td>
<td>[kernel.kallsyms]</td>
<td>[k] clear_page_c</td>
</tr>
<tr>
<td>7.76%</td>
<td>program</td>
<td>[kernel.kallsyms]</td>
<td>[k] page_fault</td>
</tr>
<tr>
<td>6.40%</td>
<td>program</td>
<td>[kernel.kallsyms]</td>
<td>[k] _raw_spin_lock</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>clear_page_c</strong></td>
</tr>
<tr>
<td>29.30%</td>
<td>program</td>
<td>[kernel.kallsyms]</td>
<td>[k] clear_page_c</td>
</tr>
<tr>
<td>19.67%</td>
<td>program</td>
<td>[kernel.kallsyms]</td>
<td>[k] isolate_migratepages_range</td>
</tr>
<tr>
<td>16.52%</td>
<td>program</td>
<td>[kernel.kallsyms]</td>
<td>[k] compaction_alloc</td>
</tr>
</tbody>
</table>

different sets of functions contribute to the profile
Advanced diagnosis (cont.)

isolate_migratepages_range()
compaction_alloc()

Both defined in mm/compaction.c
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

```bash
# 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.)

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Command</th>
<th>Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>60.29%</td>
<td>swapper</td>
<td>[kernel.kallsyms]</td>
<td>intel_idle</td>
</tr>
<tr>
<td>5.20%</td>
<td>swapper</td>
<td>[kernel.kallsyms]</td>
<td>acpi_os_read_port</td>
</tr>
<tr>
<td>3.54%</td>
<td>swapper</td>
<td>[kernel.kallsyms]</td>
<td>menu_select</td>
</tr>
<tr>
<td>3.34%</td>
<td>swapper</td>
<td>[kernel.kallsyms]</td>
<td>_raw_spin_lock_irqsave</td>
</tr>
</tbody>
</table>

- idling task is dominating in the profile
- no other visible time consumer
Advanced diagnosis (cont.)

CPU flame graphs to the rescue
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/cpu*/
cpuFreq/scaling_governor
Case #3

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

- Hardware problem
- Increased load
Conventional diagnosis

# iostat -x

%util: 100.00

svctm: 2.24

wAwait: 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.)

```c
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() {
```
Lots of similar events happening while our task is waiting
Looks like journaling can not advance while under heavy writeback
Case #3 remediation

- Decrease write back buffer, e.g. dirty_ratio
Thank you!