Plan

Overview
Plan

Overview

Resources

→ Execution scheduling
→ Memory management
→ Storage IO
Plan

Overview

Resources
→ Execution scheduling
→ Memory management
→ Storage IO

Abstracted layers
→ Virtualization
→ Containerization
Overview

Bidirectional resource management
IPC mechanisms
New hardware support

...
Execution scheduling
# Experiment 1
transaction type: pg_long.sql
latency average = 1312.903 ms

# Experiment 2
SQL script 1: pg_long.sql
  - weight: 1 (targets 50.0% of total)
  - latency average = 1426.928 ms

SQL script 2: pg_short.sql
  - weight: 1 (targets 50.0% of total)
  - latency average = 303.092 ms
Scheduling

T1  

C

T2  

C
Scheduling

T1  C  T3  C

T2
Scheduling
Scheduling

T2

C

C

T3
# Experiment 1
12,396,382,649 cache-misses # 28.562%
2,750 cpu-migrations

# Experiment 2
20,665,817,234 cache-misses # 28.533%
10,460 cpu-migrations
<table>
<thead>
<tr>
<th>time</th>
<th>cpu</th>
<th>01234</th>
<th>task name [tid/pid]</th>
<th>wait time (msec)</th>
<th>sch delay (msec)</th>
<th>run time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4227.834476 [0003] s postgres[12935]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.834895 [0003] s postgres[12935]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.418</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.835478 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.040</td>
<td>0.583</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.836485 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>1.007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.837482 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.996</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.837784 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.302</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.837989 [0003] m postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.208</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.837993 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>10.493</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.848847 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.504</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.848991 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.495</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.849487 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.260</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.849748 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.164</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.849912 [0003] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.851477 [0001] s postgres[25082]</td>
<td>0.000</td>
<td>0.000</td>
<td>1.866</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.851481 [0002] s postgres[25080]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.851778 [0003] s postgres[12935]</td>
<td>15.017</td>
<td>0.000</td>
<td>1.866</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.852259 [0003] m postgres[12935]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.484</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.852263 [0003] s postgres[12935]</td>
<td>0.000</td>
<td>0.000</td>
<td>0.214</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.852477 [0003] s postgres[25083]</td>
<td>0.000</td>
<td>0.000</td>
<td>1.001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4227.852478 [0001] s postgres[25082]</td>
<td>0.000</td>
<td>0.000</td>
<td>1.133</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

migrated: :25077 cpu 1 -> 3
migrated: postgres[25083] cpu 1 -> 3
Tunables

- `/proc/sys/kernel/sched_migration_cost_ns`: 500000
- `/proc/sys/kernel/sched_wakeup_granularity_ns`: 2000000
- `/proc/sys/kernel/sched_min_granularity_ns`: 1500000
- `/proc/sys/kernel/sched_latency_ns`: 12000000
Tunables

→ /proc/sys/kernel/sched_migration_cost_ns

→ /proc/sys/kernel/sched_wakeup_granularity_ns

→ /proc/sys/kernel/sched_min_granularity_ns

→ /proc/sys/kernel/sched_latency_ns
Tunables

→ /proc/sys/kernel/sched_migration_cost_ns
→ 500000
Tunables

→ /proc/sys/kernel/sched_migration_cost_ns
→ 500000
→ /proc/sys/kernel/sched_wakeup_granularity_ns
→ 2000000
→ /proc/sys/kernel/sched_min_granularity_ns
→ 1500000
→ /proc/sys/kernel/sched_latency_ns
→ 12000000
Tunables

- /proc/sys/kernel/sched_migration_cost_ns
  - 500000
- /proc/sys/kernel/sched_wakeup_granularity_ns
  - 2000000
- /proc/sys/kernel/sched_min_granularity_ns
  - 1500000
- /proc/sys/kernel/sched_latency_ns
  - 12000000
Tunables

- `/proc/sys/kernel/sched_migration_cost_ns` → 500000
- `/proc/sys/kernel/sched_wakeup_granularity_ns` → 2000000
- `/proc/sys/kernel/sched_min_granularity_ns`
Tunables

- /proc/sys/kernel/sched_migration_cost_ns:
  - Value: 500000

- /proc/sys/kernel/sched_wakeup_granularity_ns:
  - Value: 2000000

- /proc/sys/kernel/sched_min_granularity_ns:
  - Value: 1500000

- /proc/sys/kernel/sched_latency_ns:
  - Value: 12000000
Tunables

→ /proc/sys/kernel/sched_migration_cost_ns
→ 500000
→ /proc/sys/kernel/sched_wakeup_granularity_ns
→ 2000000
→ /proc/sys/kernel/sched_min_granularity_ns
→ 1500000
→ /proc/sys/kernel/sched latency_ns
Tunables

→ /proc/sys/kernel/sched_migration_cost_ns
→ 500000
→ /proc/sys/kernel/sched_wakeup_granularity_ns
→ 2000000
→ /proc/sys/kernel/sched_min_granularity_ns
→ 1500000
→ /proc/sys/kernel/sched_latency_ns
→ 12000000
pgbench and pg_dump

real 1m38.990s
user 1m9.127s
sys 0m2.066s

<table>
<thead>
<tr>
<th>usecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>4604</td>
<td>**</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>6812</td>
<td>****</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>14888</td>
<td>**********</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>19267</td>
<td>**********</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>65795</td>
<td>***************************</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>50454</td>
<td>*****************</td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>16393</td>
<td>*********</td>
</tr>
<tr>
<td>256 -&gt; 511</td>
<td>5981</td>
<td>***</td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td>12300</td>
<td>*********</td>
</tr>
<tr>
<td>1024 -&gt; 2047</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>2048 -&gt; 4095</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### pgbench and pg_dump

<table>
<thead>
<tr>
<th></th>
<th>count distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>1</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>8</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>25</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>46</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>189</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>119</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>96</td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>93</td>
</tr>
<tr>
<td>256 -&gt; 511</td>
<td>238</td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td>323</td>
</tr>
<tr>
<td>1024 -&gt; 2047</td>
<td>1012</td>
</tr>
<tr>
<td>2048 -&gt; 4095</td>
<td>47</td>
</tr>
</tbody>
</table>
Wakeup granularity, microsec

![Graph showing latency over time with two lines, one labeled "Finer" and the other labeled "Coarser." The y-axis represents latency in microseconds, ranging from 94 to 102, while the x-axis represents time, ranging from 0.2 to 1.2 × 10^5.]
CPU hotplug and HyperThreading

Intel® 64 and IA-32 Architectures Optimization Reference Manual
CPU hotplug and HyperThreading

→ Share execution state and cache
CPU hotplug and HyperThreading

→ Share execution state and cache
→ Spin locks have significant impact
CPU hotplug and HyperThreading

- Share execution state and cache
- Spin locks have significant impact
- PAUSE instruction (skylake latency 140 cycles)
CPU hotplug and HyperThreading

- Share execution state and cache
- Spin locks have significant impact
- PAUSE instruction (skylake latency 140 cycles)
- More deviation for latency

Intel® 64 and IA-32 Architectures Optimization Reference Manual
Latency rolling standard deviation, r/w
Latency rolling standard deviation, readonly
Memory management
Dirty pages

bgw

linux

chkp

OS Cache

Storage
Dirty pages

bgw

linux

chkp

OS Cache

Storage
Dirty pages, r/w

- `vm.dirty_ratio` 20
- `vm.dirty_background_ratio` 10
- `vm.dirty_bytes` 0
- `vm.dirty_background_bytes` 0
Dirty pages

Latency, micro sec

Time

Dirty ratio
Dirty byte

zalando
Storage IO
WAL

client

storage
WAL

client

W

storage
WAL

client

storage

client

W

W


WAL

- Bufferer IO
- fdatasync
- Writeback error propagation
NVMe

→ better for resource sharing (PCI express) under the virtualization
→ /sys/block/sda/queue/scheduler [noop|none]
→ DSM operations
NVMe DSM

→ Expected lifetime
→ Prepare for some workload (read/write)
→ Access frequency
DSM support

→ Command DWORD 11 in ioctl
→ fcntl SET_FILE_RW_HINT
→ nvme-cli (ioctl)
→ Specify a start block and a range length
# get a start block
hdparm --fibmap data_file
data_file:
    filesystem blocksize 4096, begins at LBA 0;
    assuming 512 byte sectors.
    byte_offset begin_LBA end_LBA sectors
            0      55041560   55041567    8

# set dsm for sequential read optimized
nvme dsm /dev/nvme1n01 --slbs=55041560 --blocks=1 --idr
Virtualization
Timekeeping
Timekeeping

→ Statistical sampling
  (occasional incorrect charging)

Timekeeping in VMware Virtual: Information Guide
Timekeeping

→ Statistical sampling
  (occasional incorrect charging)
→ Exact measurement (TSC time drift)
Timekeeping

→ Statistical sampling (occasional incorrect charging)
→ Exact measurement (TSC time drift)
→ /sys/devices/system/clocksource/clocksource0/

Timekeeping in VMware Virtual: Information Guide
Scheduling
Scheduling

Hypervisor

VM1

VM2

Hypervisor
Scheduling
vDSO

→ gettimeofday
→ clock_gettime
→ XEN doesn’t support vDSO for them
→ unnecessary context switches to a kernel

Two frequently used system calls are 77% slower on AWS EC2
Latency m4.xlarge XEN/TSC, r/w
Latency m5.xlarge KVM/TSC, r/w
Locks

Intel® 64 and IA-32 Architectures Software Developer’s Manual, Vol. 3
Locks

→ Lock holder preemption problem
Locks

- Lock holder preemption problem
- Lock waiter preemption problem

Intel® 64 and IA-32 Architectures Software Developer’s Manual, Vol. 3
Locks

→ Lock holder preemption problem
→ Lock waiter preemption problem
→ Intel PLE (pause loop exiting)
Locks

- Lock holder preemption problem
- Lock waiter preemption problem
- Intel PLE (pause loop exiting)
- PLE_Gap, PLE_Window
vCPU

vC1  vC2  vC3  vC4

Hypervisor
vCPU

vC1  vC2  vC3  vC4

Hypervisor
vCPU

vC1  vC2  vC3  vC4

Hypervisor
Containerization
### cgroups controllers

- cpu,cpuacct
- cpuset
- memory
- devices
- freezer
- net_cls
- rdma
- blkio
- perf_event
- net_prio
- hugetlb
- pids
- rdma
blk_throtl_bio+0x1
dm_make_request+0x80
generic_make_request+0xf6
submit_bio+0x7d
blkdev_issue_flush+0x68
ext4_sync_file+0x310
vfs_fsync_range+0x4b
do_fsync+0x3d
sys_fdatasync+0x13
fdatasync+0x10
XLogBackgroundFlush+0x17e
WalWriterMain+0x1cb
PostmasterMain+0xfea
bklio controller

- CFQ & throttling policy (generic block layer)
- No weight related options will work without CFQ
- Advisable io scheduler for SSD is noop/none
- Block layer do sampling to enforce throttling
throttle_sample_time

This is the time window that blk-throttle samples data, in millisecond. blk-throttle makes decision based on the samplings. Lower time means cgroups have more smooth throughput, but higher CPU overhead. This exists only when CONFIG_BLK_DEV_THROTTLING_LOW is enabled.
blkio

On traditional cgroup hierarchies, relationships between different controllers cannot be established making it impossible for writeback to operate accounting for cgroup resource restrictions and all writeback IOs are attributed to the root cgroup.

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
Bad neighbour

→ memory fragmentation
→ buddy allocator can fail
to find a page of proper size
→ kernel will start a compaction process
### Host, normal

Zone: Normal

Free KiB in zone: **807232.00**

<table>
<thead>
<tr>
<th>Fragment size</th>
<th>Free fragments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>29612</td>
</tr>
<tr>
<td>8192</td>
<td>23308</td>
</tr>
<tr>
<td>16384</td>
<td>13495</td>
</tr>
</tbody>
</table>

### Host with a container

Zone: Normal

Free KiB in zone: **109700.00**

<table>
<thead>
<tr>
<th>Fragment size</th>
<th>Free fragments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>3405</td>
</tr>
<tr>
<td>8192</td>
<td>7082</td>
</tr>
<tr>
<td>16384</td>
<td>1954</td>
</tr>
</tbody>
</table>
Bad neighbour

- PGSemaphore* functions make use of futex
- Per-cpu hash table for futex with hash buckets
Bad neighbour

→ WAL segment/heap file creation
→ inode lock contention

Understanding Manycore Scalability of File Systems
Questions?

GitHub: github.com/erthalion
GitHub: github.com/erthalion/ansible-ycsb
Twitter: @erthalion
Email: dmitrii.dolgov at zalando dot de
Email: 9erthalion6 at gmail dot com