Mitigating Excessive Pause-Loop-Exiting in VM-Agnostic KVM

Kenta Ishiguro / Keio University
About Me

• 3rd year PhD student at Keio University, Japan

• Research interests:
  • performance and security of hypervisors
  • particularly KVM
Oversubscribing **virtual** CPUs

- Enables better hardware utilization
- Requires multiplexing of virtual CPUs (vCPUs) on physical CPUs (pCPUs)
Critical section on pCPU

- Operating System assumption:
  - pCPUs are always active
Excessive vCPU spinning

• Oversubscription incurs **excessive spinning**
• **vCPU pre-emption** by Hypervisor violates OS assumption
• Lock-holder pre-emption problem

![Diagram of vCPU0 acquiring spinlock, then preempted by Hypervisor, and vCPU1 spinning (pause loop)]
Improving guest performance by solving excessive vCPU spinning

• Problem
  • vCPU wastes its execution time by excessive spinning

• Ideal case
  • Hypervisor knows which vCPU should be scheduled right now

• But, hard to solve excessive vCPU spinning due to semantic gap
  • between KVM and Linux scheduler
  • between KVM and guest VMs
Improving guest performance by solving excessive vCPU spinning

• Problem
  • vCPU wastes its execution time by excessive spinning

• Ideal case
  • Hypervisor knows which vCPU should be scheduled right now

• But, hard to solve excessive vCPU spinning due to semantic gap
  • between KVM and Linux scheduler
  • between KVM and guest VMs

Boosting a target vCPU can be impeded by the Linux scheduler to keep fairness between vCPUs
Improving guest performance by solving excessive vCPU spinning

- **Problem**
  - vCPU *wastes* its execution time by *excessive spinning*

- **Ideal case**
  - Hypervisor knows which vCPU should be scheduled right now

- But, hard to solve excessive vCPU spinning due to *semantic gap*
  - between KVM and Linux scheduler
  - between KVM and guest VMs

Hard to build comprehensive vCPU candidate set for boosting
KVM approach with hardware feature

• KVM leverages Pause Loop Exiting (PLE) on Intel x86
  • detects excessive vCPU spinning
  • gives hypervisors a chance to re-schedule vCPUs

PLE detects excessive spinning and transfers the control to the hypervisor for re-scheduling
KVM’s strategy to suppress PLE events

• Reschedule PLE-ed vCPU to another preempted vCPU
  • in kvm_vcpu_on_spin

• Co-operative rescheduling with Linux scheduler
  • leverages yield_to provided by CFS scheduler
  • makes a request to Linux scheduler to yield and boost vCPUs

• Selects a vCPU to boost in round-robin from candidates
  • resource-waiter, lock-waiter, and IPI-receiver vCPUs are candidates
What happens in the worst case?

KVM trace with running two 8-vCPU VMs simultaneously

<table>
<thead>
<tr>
<th>CPU 2/KVM</th>
<th>2679 [006]</th>
<th>276.731784:</th>
<th>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.731823:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.731849:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.731875:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.731901:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.731927:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.731954:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.731980:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.732006:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.732032:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM</td>
<td>2679 [006]</td>
<td>276.732058:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
</tbody>
</table>

PLE events occur continuously (> 100 times)
What happens in the worst case?

Continuous PLE occur in the short period

<table>
<thead>
<tr>
<th>CPU 2/KVM 2679 [806]</th>
<th>276.731784</th>
<th>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731819</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731823</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731845</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731849</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731871</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731875</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731897</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731901</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731923</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731927</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731949</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731954</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731975</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.731980</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.732001</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.732006</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.732027</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.732032</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.732053</td>
<td>kvm:kvm_entry: vcpu 2</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [806]</td>
<td>276.732058</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffffb72e29e3 info 0 0</td>
</tr>
</tbody>
</table>

KVM trace with running two 8-vCPU VMs simultaneously
What happens in the worst case?

KVM trace with running two 8-vCPU VMs simultaneously

<table>
<thead>
<tr>
<th>CPU 2/KVM 2679 [606]</th>
<th>276.731784:</th>
<th>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.731823:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.731849:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.731875:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.731901:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.731927:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.731954:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.731980:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.732006:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.732032:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
<tr>
<td>CPU 2/KVM 2679 [606]</td>
<td>276.732058:</td>
<td>kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xfffffffff72e29e3 info 0 0</td>
</tr>
</tbody>
</table>

At the same code location
What happens in the worst case?

KVM trace with running two 8-vCPU VMs simultaneously

| CPU 2/KVM | 2679 [006] | 276.731784: | kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0 |
| CPU 2/KVM | 2679 [006] | 276.731823: | kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0 |
| CPU 2/KVM | 2679 [006] | 276.731849: | kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0 |
| CPU 2/KVM | 2679 [006] | 276.731875: | kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0 |

- PLE events occur continuously (> 100 times)
- in the short period
- at the same code location
Continuous PLE events are NOT rare

- Most number of PLE events are from continuous PLE events
- Cause of PLE events: both spinlock and TLB shootdown
Continuous PLE events are **NOT** rare

- Most number of PLE events are from continuous PLE events
- Cause of PLE events: both spinlock and TLB shootdown

### Functionality

<table>
<thead>
<tr>
<th>Function causing PLE events</th>
</tr>
</thead>
<tbody>
<tr>
<td>native_queued_spin_lock_slowpath</td>
</tr>
<tr>
<td>queued_write_lock_slowpath</td>
</tr>
<tr>
<td>queued_read_lock_slowpath</td>
</tr>
<tr>
<td>try_to_wake_up</td>
</tr>
<tr>
<td>etc…</td>
</tr>
</tbody>
</table>

### TLB shootdown

<table>
<thead>
<tr>
<th>Function causing PLE events</th>
</tr>
</thead>
<tbody>
<tr>
<td>smp_call_function_many</td>
</tr>
<tr>
<td>smp_call_function_single</td>
</tr>
</tbody>
</table>
Why PLE events occur continuously?

• Lost opportunity and Overboost
  • Comprehensive approach to identify the root cause of PLE does not exist
  • Mitigation: **Strict Boost**

• Scheduler mismatch
  • Linux scheduler ignores hints from hypervisor
  • Mitigation: **Debooster**
How to select candidate vCPU

• PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
• Directed yield [2011]

• Less prioritize recently PLE-ed vCPUs [2012]
• Boost only preempted vCPUs [2013]
• Boost only vCPUs in kernel mode [2017]

Optimizations against spinlock

• Boost halted vCPUs where they have received an IPI [2019]
  • Optimization against TLB shootdown
How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]
- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]
- Boost halted vCPUs where they have received an IPI [2019]

Optimizations against spinlock

- Spinlock still incurs continuous PLE events
- vCPUs in user mode also need to ack when TLB shootdown happens
- No need to boost IPI-receivers when lock-holder preemption happens
How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

```
Optimizations against spinlock
```

- Boost halted vCPUs if they have received an IPI [2019]

```
Insights
```

- **Spinlock** still incurs continuous PLE events
  - vCPUs in user mode also need to ack when TLB shootdown happens
  - No need to boost IPI-receivers when lock-holder preemption happens
How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

- Boost halted vCPUs where they have received an IPI [2019]

Optimizations against spinlock

- Spinlock still incurs continuous PLE events
- vCPUs in user mode also need to ack when TLB shootdown happens
- No need to boost IPI-receivers when lock-holder preemption happens
How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]
- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]
- Boost halted vCPUs where they have received an IPI [2019]

Optimizations against spinlock
- Spinlock still incurs continuous PLE events
- vCPUs in user mode need to ack when TLB shootdown happens
- No need to boost IPI-receivers when lock-holder preemption happens
Mitigation: Strict Boost

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

- Boost halted vCPUs where they have received an IPI [2019]

- Introduce new strategies for candidate vCPU selection

- Boost IPI-receiver vCPUs in user mode
- Not boost halted vCPUs if the yielded vCPU has NOT sent an IPI to it

Optimizations against spinlock
Why PLE events occur continuously?

- Lost opportunity and Overboost  \textit{Strict Boost}

- Scheduler mismatch
Problem: Scheduler Mismatch

- Linux CFS does **not distinguish** between vCPUs and other threads
  - KVM makes request to Linux CFS for boosting vCPU
  - But Linux CFS always keeps fairness between vCPUs

```c
if (cfs_rq->next && wakeup_preempt_entity(cfs_rq->next, left) < 1) {
    /*
    * Someone really wants this to run. If it's not unfair, run it.
    */
    se = cfs_rq->next;
}
```

in kernel/sched/fair.c
Case Study: Scheduler Mismatch

1. **Picks the highest priority task:** vCPU 0
2. \( cputime_{vCPU1} - cputime_{vCPU0} > \text{threshold} \)
3. Decides not to yield vCPU 0 and not to boost vCPU 1
4. vCPU 0 triggers PLE again because vCPU 1 still has the lock
Case Study: Scheduler Mismatch

1. Picks the highest priority task: vCPU 0
2. $cputime_{vCPU1} - cputime_{vCPU0} > threshold$
3. Decides not to yield vCPU 0 and not to boost vCPU 1
4. vCPU 0 triggers PLE again because vCPU 1 still has the lock
Case Study: Scheduler Mismatch

1. Picks the highest priority task: vCPU 0
2. $\text{cputime}_{\text{vCPU}1} - \text{cputime}_{\text{vCPU}0} > \text{threshold}$
3. Decides not to yield vCPU 0 and not to boost vCPU 1
4. vCPU 0 triggers PLE again because vCPU 1 still has the lock

CFS considers scheduling vCPU 1 is too unfair

Boost request is ignored
Case Study: Scheduler Mismatch

1. Picks the highest priority task: vCPU 0
2. $\text{cputime}_{vCPU_1} - \text{cputime}_{vCPU_0} > \text{threshold}$
3. Decides not to yield vCPU 0 and not to boost vCPU 1
4. vCPU 0 triggers PLE again because vCPU 1 still has the lock
Case Study: Scheduler Mismatch

1. Picks the highest priority task: vCPU 0
2. vCPU 1 is boosted eventually because 
   \[ \text{cputime}_{vCPU1} - \text{cputime}_{vCPU0} < \text{threshold} \]
Mitigation: Debooster

- Debooster makes CFS not to hesitate to boost another vCPU instead of vCPU which exits due to PLE
  - by lowering PLE-ed vCPU priority
Implementation

• Modified Linux KVM 5.6.0

• Our mitigations require 41 LoC modification
  • Debooster: yield_to_task() interface in CFS
  • Strict Boost: vCPU candidate selection and IPI handler in KVM
Experimental Setup

• Testbed
  • 8-core (Intel Xeon)
  • 2 VMs with 8 vCPU for each

• Benchmarks
  • dedup and vips from parsec
  • psearchy from mosbench
  • ebizzy
  • swaptions: CPU-intensive workload co-runner in another VM
Evaluation: Reduction of PLE Occurrences

Normalized number of PLE events

<table>
<thead>
<tr>
<th>Baseline KVM</th>
<th>Debooster and Strict Boost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedup</td>
<td></td>
</tr>
<tr>
<td>Vips</td>
<td></td>
</tr>
<tr>
<td>Ebizzy</td>
<td></td>
</tr>
<tr>
<td>Psearchy</td>
<td></td>
</tr>
</tbody>
</table>

35
**Evaluation: Performance Improvement**

Execution time is **reduced by 40%**

Throughput is **improved by 75%**

<table>
<thead>
<tr>
<th>Normalized execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Normalized throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ebizzy</td>
</tr>
</tbody>
</table>

- **Baseline KVM**
- **Debooster and Strict Boost**
Evaluation: System Fairness

- Co-runner’s performance degradation cannot be seen
- Mitigations do not raise the priority of the boosted vCPU
Conclusion

• Oversubscribing vCPUs incurs excessive spinning
• Pause-Loop-Exiting (PLE) unfortunately does not fix it
• due to the semantic gap between
  • KVM and guest VMs
  • KVM and Linux scheduler

• Introduced mitigations against identified problems improve apps throughput by up to 75 %
  • Strict Boost against lost opportunity and overboost
  • Debooster against scheduler mismatch

• Problem investigated by the KVM community
  • https://lore.kernel.org/kvm/20210421150831.60133-1-kentaishiguro@sslab.ics.keio.ac.jp/
  • https://lore.kernel.org/kvm/1618542490-14756-1-git-send-email-wanpengli@tencent.com/

Thanks & QA

Mitigating Excessive Pause-Loop-Exiting in VM-Agnostic KVM

Contact: kentaishiguro@sslab.ics.keio.ac.jp