Introduction to Cache Quality of service in Linux Kernel

Vikas Shivappa
(vikas.shivappa@linux.intel.com)
Agenda

- Problem definition
- Existing techniques
- Why use Kernel QOS framework
- Intel Cache qos support
- Kernel implementation
- Challenges
- Performance improvement
- Future Work
Without Cache QoS

- **Noisy neighbour** => Degrade/inconsistency in response => QoS difficulties
- Cache Contention with multi threading
Agenda

• Problem definition

• **Existing techniques**
  • Why use Kernel QOS framework
  • Intel Cache qos support
  • Kernel implementation

• Challenges

• Performance improvement

• Future Work
Existing techniques

• Mostly heuristics on real systems
• **No methodology to identify cache lines** belonging to a particular thread
• Lacks configurability by OS
Agenda

• Problem definition
• Existing techniques

• Why use Kernel QOS framework
• Intel Cache qos support
• Kernel implementation
• Challenges
• Performance improvement
• Future Work
Why use the QOS framework?

- Lightweight
  powerful tool to manage cache
- Without a lot of architectural details
With Cache QoS

- Help maximize performance and meet QoS requirements
  - In Cloud or Server Clusters
  - Mitigate jitter/inconsistent response times due to ‘Noisy neighbour’
Agenda

• Problem definition
• Existing techniques
• Why use Kernel QoS framework
• **Intel Cache QoS support**
• Kernel implementation
• Challenges
• Performance improvement
• Future Work
What is Cache QoS?

• Cache Monitoring
  – cache occupancy per thread
  – `perf` interface

• Cache Allocation
  – user can allocate overlapping subsets of cache to applications
  – `cgroup` interface
Cache lines $\Leftrightarrow$ Thread ID (Identification)

- Cache Monitoring
  - RMID (Resource Monitoring ID)
- Cache Allocation
  - CLOSid (Class of service ID)
Representing cache capacity in Cache Allocation (example)

- Cache capacity represented using ‘Cache bitmask’
- However mappings are hardware implementation specific
Bitmask ↔ Class of service IDs (CLOS)

**Default Bitmask – All CLOS ids have all cache**

<table>
<thead>
<tr>
<th></th>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOS0</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>CLOS1</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>CLOS2</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>CLOS3</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

**Overlapping Bitmask (only contiguous bits)**

<table>
<thead>
<tr>
<th></th>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOS0</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>CLOS1</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>CLOS2</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>CLOS3</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>
Agenda

• Problem definition
• Existing techniques
• Why use Kernel QOS framework
• Intel Cache qos support

• **Kernel implementation**
• Challenges
• Performance improvement
• Future Work
Kernel Implementation

- Threads
- User interface
  - /sys/fs/cgroup
  - perf
- User Space
- Kernel Space
- Allocation configuration
  - Configure bitmask per CLOS
- During ctx switch
  - Set CLOS/RMID for thread
- Read Monitored data
  - Read Event counter
- MSR
- Kernel QoS support
  - Cache alloc
  - Cgroup fs
  - cache monitoring
- Hardware
- Intel Xeon QOS support
- Shared L3 Cache
Usage

Monitoring per thread cache occupancy in bytes

```
./tools/perf/perf stat -e intel_cqm/llc_occupancy/ <cmd or tid>
```

Performance counter stats for thread id '5236':

```
638976.00 Bytes intel_cqm/llc_occupancy/
16.140199267 seconds time elapsed
```

Allocating Cache per thread through cache bitmask

```
/bin/echo 4938 > /sys/fs/cgroup/rdt/group1/tasks
```

```
/bin/echo 0xf > /sys/fs/cgroup/rdt/group1/intel_rdt.cache_mask
```

Exposed to user land

- Cgroup
  - Clos : Parent.Clos
  - bitmask : Parent.bitmask
  - Tasks : Empty
Scenarios

• Units that can be allocated cache
  – Process/tasks
  – Virtual machines (transfer all PIDs of VM to one cgroup)
  – Containers (put the entire container into one cgroup)

• Restrict the noisy neighbour

• Fair cache allocation to resolve cache contention
Agenda

• Problem definition
• Existing techniques
• Why use Kernel QOS framework
• Intel Cache qos support
• Kernel implementation

• Challenges
• Performance improvement
• Future Work
Challenges

• Openstack usage
• What if we run out of IDs?
• What about Scheduling overhead
• Doing monitoring and allocation together
Work beginning, not stable yet to add changes to Ceilometer (With Qiaowei qiaowei.ren@intel.com)
What if we run out of IDs?

• Group tasks together (by process?)
• Group cgroups together with same mask
• return –ENOSPC
• Postpone
Scheduling performance

• msrread/write costs 250-300 cycles
• Keep a cache. Grouping helps!
• Don’t use till user actually creates a new cache mask
Monitor and Allocate

- RMID(Monitoring) CLOSid(allocation) different
- Monitoring and allocate same set of tasks easily
  - perf cannot monitor the cache alloc cgroup(?)
Agenda

• Problem definition
• Existing techniques
• Why use Kernel QOS framework
• Intel Cache qos support
• Kernel implementation
• Challenges

• Performance improvement and Future Work
Performance Measurement

• Intel Xeon based server, 16GB RAM
• 30MB L3, 24 LPs
• RHEL 6.3
• With and without cache allocation comparison
• Controlled experiment
  – PCIe generating MSI interrupt and measure time for response
  – Also run memory traffic generating workloads (noisy neighbour)
• *Experiment Not using current cache alloc patch*
Performance Measurement

- Minimum latency: 1.3x improvement, Max latency: 1.5x improvement, Avg latency: 2.8x improvement
- Better consistency in response times and less jitter and latency with the noisy neighbour
## Patch status

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Monitoring</td>
<td>Upstream 4.1 (Matt Fleming, <a href="mailto:matt.fleming@intel.com">matt.fleming@intel.com</a>)</td>
</tr>
<tr>
<td>Cache Allocation</td>
<td>Under review. (Vikas Shivappa, <a href="mailto:vikas.shivappa@intel.com">vikas.shivappa@intel.com</a>)</td>
</tr>
<tr>
<td>Code Data prioritization</td>
<td>Under review. (Vikas Shivappa, <a href="mailto:vikas.shivappa@intel.com">vikas.shivappa@intel.com</a>)</td>
</tr>
<tr>
<td>Open stack integration (libvirt update)</td>
<td>Work started (Qiaowei <a href="mailto:qiaowei.ren@intel.com">qiaowei.ren@intel.com</a>)</td>
</tr>
</tbody>
</table>
Future Work

• Performance improvement measurement
• Code and data allocation separately
  – First patches shared on lkml
• Monitor and allocate same unit
• Openstack integration
• Container usage
Acknowledgements

• Matt Fleming (cache monitoring support, Intel SSG)
• Will Auld (Architect and Principal engineer, Intel SSG)
• CSIG, Intel
References

Questions ?