Linux QoS framework usage report for containers and cloud and challenges ahead
Agenda

• Problem definition
• Why use Kernel QOS framework
• Intel Cache/memory qos support
• Kernel implementation
• Openstack and Container support
• Performance improvement
• Future Work
Without Cache/Memory QoS framework (quality of service)

- **Noisy neighbour** => Degrade/inconsistency in response => QOS difficulties
- HPC
Agenda

• Problem definition
• **Why use Kernel QOS framework**
  • Intel cache/memory qos support
  • Kernel implementation
  • Openstack and Container support
  • Performance improvement
• Future Work
Why use the Cache/Memory QOS framework?

- User friendly interfaces: Perf/cgroup
- Abstracts a lot of architectural/System level details
With Cache QoS

- Help monitor and control shared resources => achieve consistent response => better QoS
  - Cloud or Server Clusters
  - Containers
  - HPC

Kernel Cache QOS framework

Intel QOS h/w support

Controls to allocate the appropriate cache to high pri apps

User space
Kernel space

Proc Cache

High Pri apps
Low Pri apps

h/w
Agenda

• Problem definition
• Why use Kernel QoS framework
• **Intel Cache/Memory QoS support**
• Kernel implementation
• Openstack support
• Container support
• Performance improvement
• Future Work
What is Cache/Mem QoS?

• Cache/Memory b/w Monitoring
  – cache occupancy/mem b/w per thread
  – `perf` interface

• Cache Allocation
  – user can allocate overlapping subsets of cache to applications
  – `cgroup` interface (out of tree only, new interface coming up)
Intel QoS Terminologies

• RDT – Resource director technology
  – is basically “Processor QoS” under which the cmt/cat/mbm etc are all sub-features
• CMT – Cache Monitoring Technology or also called CQM
• CAT – Cache Allocation Technology
• MBM – Memory b/w monitoring
Cache lines ⇔ Thread ID (Identification)

• Cache Monitoring
  – RMID (Resource Monitoring ID) ⇔ PID.
  – RMID tagged to cache lines allocated

• Cache Allocation
  – CLOSid (Class of service ID)
  – Restrict when Cache is filled

• Memory b/w
  – RMID <=> Total L3 external b/w
Agenda

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

• **Kernel implementation**
• Openstack and Container support
• Performance improvement
• Future Work
Kernel Implementation

Threads

User interface
/sys/fs/cgroup
perf

User Space

Allocation configuration
- Configure bitmask per CLOS
- Set CLOS/RMID for thread

During ctx switch

Read Monitored data
- Read Event counter

MSR

Kernel QOS support
- Cache alloc
- cache / mem b/w monitoring

Kernel Space

Intel Xeon QOS support

Shared L3 Cache

Hardware

Memory
Memory b/w Monitoring

- RMID1...RMIDn
- CLOSID1...RMIDn

Socket0
- Cores
- Shared L3
- Mem Ctrlr
- Memory

Socket1
- Cores
- Shared L3
- Mem Ctrlr
- Memory

Local mem b/w

Total mem b/w
MBM implementation continued

- Typically
  - sched_in
    - prev_count = read_hw_count();
  - sched_out
    - c = read_hw_count();
    - count += c - prev_count;

- Wont work for MBM as we have per package RMIDs
  - Doing the above on 2 core siblings for a PID with same RMID would result in duplicate count.
Sample cgrouphierarchy

- Other considerations
  - Movement of tasks between cgroups
  - MBM counters overflow

- e1 : should read 10MB
- e2 : should read 13MB
- e3 : should read 5MB
MBM hierarchy monitoring

• Implement using periodic updates of the ‘per-RMID count’ as well a ‘per event count’

• This helps take care of all the scenarios
  – Task movement between cgroups
  – RMID recycling
  – Events start counting the same cgroup at different times (they only need to read the current event count)
Usage

Basic monitoring per thread cache occupancy/ Mem b/w

- Basic usage example.
- Results display the total cache occupancy and total mem b/w for the thread.
Other Usage modes

• Monitor cgroup

• Per socket monitoring
  – --per-socket does not work as we are not cpu event
  – --per-cpu doesn’t work either
  – Use –C <cpu in the socketN>

• Systemwide
  – Fail if (–a && –t) option (system wide task mode)
Usage Scenarios

• Units that can be monitored for cache/memory b/w
  – Process/tasks
  – Virtual machines and cloud (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

• **OpenStack / Container support**
• Challenges
• Performance improvement
• Future Work
Openstack usage

Openstack dashboard

Integration

Shared L3 Cache

Applications

Open Stack Services

Standard hardware

Compute

Network

Storage

Openstack usage
Openstack usage ...

- Libvirt patches submitted (Qiaowei qiaowei.ren@intel.com) – based on kernel QOS framework
- CAT/CMT/MBM was demoed in openstack forums/conference
Containers support

- Dockers support patch was built to use the new CAT cgroup
- Was simpler change as dockers and systemd already have all the plumbing to use cgroups
Cyclic tests using docker

- With CAT (green curve) has a more consistent response latency range comparable to the no-noise scenario (0-16)
- Most of the samples falling the 1-9.
OSV adaption status

• RHEL : CQM – In RHEL 7.2, MBM – Ported to RHEL 7.3, testing in progress
• Baidu, alibaba: CQM,CAT ported to 3.10 kernel. Currently on test bed
Challenges

• Openstack, Container next steps
• What if we run out of IDs?
• What about Scheduling overhead
• Doing monitoring and allocation together
Openstack/container next steps for CAT

- kernel CAT *cgroup support will remain out of tree*
  - cgroup Pros
    - openstack/dockers other enterprise users like Google could use the feature on test bed and are ready to adapt
    - Was supported by much of community (Peterz/HPA/dockers/google) for quite sometime.
    - Issues like hierarchy/kernel thread issue was related to cgroup.
  - Cons
    - Thomas rejected cgroup interface eventually.
    - Quickly run out of CLOSIds with cgroup hierarchy, more in v2 – However reuse had mitigated some of the issues.
    - Per socket atomic update issue

- Openstack and Dockers *CAT support needs a rewrite to use the new CAT (resctl) interface.*
What if we run out of IDs?

• Group tasks together (by process?)
• Group cgroups together with same mask
• return –ENOSPC
• Postpone/ Recycle
RMID recycling

• Not really ‘virtual RMIDs’ currently as we don’t switch RMIDs at context switch.
• For cqm, cache occupancy is still tied to the RMID after we ‘free’ an RMID -> it goes to limbo list.
• However for MBM , the RMIDs can be used immediately without waiting for zero occupancy.
RMID recycling

F – Free state (f- free count)  
L – Limbo  
A - Allocated  
e – event (er- # of required RMIDs)
RMID recycling accuracy

• Current scheme eg:
• The counting time is proportional to the max RMID to required RMID ratio
• Ex: 80 RMIDs max, 100 required RMIDs
  – on average an event is counted for 80% of time and missed for 20% of the time
Scheduling performance

• msrread/write costs 250-300 cycles
• Keep a cache. Grouping helps!
Monitor and Allocate

- RMID(Monitoring) CLOSid(allocation) different
- Monitoring and allocate same set of tasks easily
  - perf cannot monitor the cache alloc cgroup/ now resctl(?)
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*
- 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 (CMT)</td>
<td>Upstream 4.1.</td>
</tr>
<tr>
<td>Cache Allocation(CAT)/CDP</td>
<td>Cgroup Interface rejected. New resctl built on the same kernel framework interface in progress (Fenghua, Yu)</td>
</tr>
<tr>
<td>Memory b/w Monitoring</td>
<td>Upstream 4.6 (Vikas, Shivappa).</td>
</tr>
<tr>
<td>Open stack integration (libvirt update)</td>
<td>Support built for CMT/MBM and CAT cgroup interface (Qiaowei <a href="mailto:qiaowei.ren@intel.com">qiaowei.ren@intel.com</a>)</td>
</tr>
<tr>
<td>Container support (Dockers)</td>
<td>Support built for CAT cgroup interface (Intel)</td>
</tr>
</tbody>
</table>
Future Work

• Perf overhead during CQM/MBM
• Support data per-process
• Improve and unify ID management for RMID/CLOSID
References

Questions ?
Backup
Representing cache capacity in Cache Allocation (example)

<table>
<thead>
<tr>
<th>Bn</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
</table>

Capacity Bitmask

<table>
<thead>
<tr>
<th>Wk</th>
<th>W(k-1)</th>
<th>W3</th>
<th>W2</th>
<th>W1</th>
<th>W0</th>
</tr>
</thead>
</table>

Cache Ways

- Cache capacity represented using ‘Cache bitmask’
- However mappings are hardware implementation specific
Bitmask $\leftrightarrow$ 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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOS3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
</tr>
</tbody>
</table>