Affinity-aware Dynamic Pinning Scheduling for Virtual Machines

Zhi Li
lizhi@cse.buaa.edu.cn
School of Computer Science, BeiHang University, Beijing, China
Outline

- Motivation
- CPU Affinity-aware Method
- Dynamic Pinning Scheduling
- Performance Evaluation
Motivation

L2 Cache misses on both non-virtualization and virtualization

Virtualization leads to worse cache miss
Analysis of the Issue

- VCPUs from same Domain take turns to run in a same CPU runq
- Frequent migrations from this kind of VCPU
How to bridge the semantic gap between Guest OS and VMM

- Affinity-aware DP-Scheduling:
  - Affinity-aware method: providing the task affinity information to VMM.
  - DP-Scheduling: implementing that VCPU can be pinned or unpinned dynamically
CPU Affinity-aware Method

- Timing Control
  - When CR3 is changing

- Methodology for Capture
  - Affinity Coefficient (AC)

- API
  - Provide AC for the scheduler
Dynamic Pinning Scheduling

- Driver Domain
- Guest OS 1
- Guest OS 2
- ... Guest OS n

- PI Manager
- Affinity-aware Detector
- API
- Scheduler
- VCPU Monitor
- DP-Scheduling

- Disk
- Memory
- Core
- Core
- Core
- Core
- ... Core
- L2 cache
- L2 cache
- L2 cache
Dynamic Pinning Scheduling

Common VCPU Run Queue

Pinned VCPU Run Queue
Dynamic Pinning Scheduling

- strategies:
  - a. pin VCPU to the CPU with no pinned VCPU at this time.
  - b. pin VCPU to the CPU with lower workload.
  - c. pin VCPU to the local CPU if both (a) and (b) do not happen.
  - d. do not migrate the VCPU actively when it is unpinned.
  - e. unpin the VCPU with the lowest value of AC when the number of CPU equals.
  - f. unpin the VCPU pinned before if it goes to the state of OVER or IDLE.
DP-Scheduling Algorithm

Let $\Phi$ stand for VCPU set from all domains, that is, $\Phi = \{VCPU_1, VCPU_2, ..., VCPU_{|\Phi|}\}$. Let $\phi_p$ be the set of VCPUs that were pinned, whereas $\phi_{up}$ is the set of unpinning VCPUs. Hence, $\Phi = \phi_p \cup \phi_{up}$, and $|\Phi| = |\phi_p| + |\phi_{up}|$. Besides, let $\gamma = \{\gamma_1, \gamma_2, ..., \gamma_{|\Phi|}\}$, where $\gamma_i$ is the value of AC for VCPU$_i$.

For each VCPU$_i$ $\in \Phi$

- CPU$_x$ = VCPU$_i$ $\rightarrow$ processor;
  /*VCPU$_i$ has been pinned by its high value of AC*/
- If ($\gamma_i \geq$ THRESHOLD and VCPU$_i$ $\in \phi_p$) Return;
  /*VCPU$_i$ with its high value of AC was not pinned*/
- Else If ($\gamma_i \geq$ THRESHOLD and VCPU$_i$ $\in \phi_{up}$ and VCPU$_i$ $\rightarrow$ pri > OVER) {
  peer_cpu = select_peer(VCPU$_i$);
  set_unmigrateable(peer_cpu, vcpu);
  del(\phi_{up}, VCPU$_i$);
  add(\phi_p, VCPU$_i$);
}

/*Pinned VCPU$_i$ contains low value of AC currently
* according to strategy (d)*/

Else If ($\gamma_i <$ THRESHOLD and VCPU$_i$ $\in \phi_p$) {
  set_migrateable(CPU$_x$, VCPU$_i$);
  del(\phi_p, VCPU$_i$);
  add(\phi_{up}, VCPU$_i$);
}

/*The total number of VCPUs should not over
*CPU number -1*/

If ($|\phi_p| == |CPU|$)
  set_migrateable_by_ac(\phi_p, \phi_{up});
Performance Evaluation

Platform
- Xeon 5405(two quad-core)
- L2 cache: 6M
- RAM: 4G
- Xen: 3.4.3

Benchmark

<table>
<thead>
<tr>
<th>Benchmark Category</th>
<th>Code Name</th>
<th>Variable</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPCC</td>
<td>STREAM</td>
<td>Array size</td>
<td>Memory Bandwidth</td>
</tr>
<tr>
<td>EPCC</td>
<td>OpenMP Micro-benchmark suite</td>
<td>Thread Number</td>
<td>Time</td>
</tr>
<tr>
<td>IMB</td>
<td>Sendrecv</td>
<td>Message Size</td>
<td>Transfer Speed</td>
</tr>
<tr>
<td></td>
<td>Exchange</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance Evaluation
• Conclusion

DP-Scheduling outperforms Credit scheduling for kinds of CPU-bound tasks, without interfering the load balance
Thank You!