Monitoring and Analyzing Virtual Machines – Resource Overcommitment Detection and Virtual Machine Classification

Hani Nemati

May 5, 2016

Polytechnique Montréal
Laboratoire DORSAL
Agenda

Motivation

- Why detecting resource overcommitment is important?
- Why we need a virtual machine classification model?

Investigations

- Detecting Resource overcommitment for Infrastructure providers
  - Virtual CPU State Detection
    - CPU Overcommitment
    - Memory Overcommitment
  - Virtual Machine Dissection
- Virtual Machine Classification

Conclusion and in-progress

References
Motivation

Why detecting resource overcommitment is important?

- **Resource Overcommitment** lets the administrator allocate more resources to virtual machines than the physical host has available.
- Identifying Over-committed and Under-committed hosts
- Impact of Resource Overcommitment:
  - **Cloud User:**
    - Latency in response time of programs inside VMs
  - **Infrastructure Provider:**
    - Increase resource utilization
    - Maximizing profit
Motivation

Why we need a virtual machine classification model?

• Classifying virtual machines based on different resource consuming patterns [1][3]

• Categorizing virtual machines into:
  • CPU-intensive
  • Memory-intensive
  • I/O-intensive
  • Network-intensive
  • Compound

• Advantages:
  • Better capacity planning
  • Increase infrastructure revenue by utilizing host resources
  • Better decision for migration of VMs
Investigations

Qemu threads in Control Flow View

- Virtual machine executed into ioctl()
- Block I/O? Network? CPU processing?
In the virtualization process with Intel VT-x, we focus on the VMX (Virtual Machine Extension) transition [5], which happens between the Guest and Virtual Machine Manager (VMM). This includes operations such as `vm_entry` and `vm_exit`. The `vm_entry` involves saving the VMCS area for the Host, loading the VMCS area for the Guest, and handling the exit reason. Conversely, `vm_exit` includes saving the VMCS area for the Guest, loading the VMCS area for the Host, and checking the exit reason.

The Virtual Machine Control Structure (VMCS) plays a crucial role in managing these transitions, as it contains information about the state of the virtual machine, including the current context and resources needed for different operations. This structure is accessed during entry and exit transitions to ensure the seamless switching between the Guest and Host environments.
Investigations

vCPU Thread

ioctl
...
Investigations

Resource View for Virtual Machine
Investigations

Our Architecture and tracepoints used for vCPU state detection:

- **sched_wakeup**: To detect wait state and VMX Root Network and Disk
- **kvm_exit**: To detect VMX Root state
- **kvm_entry**: To detect VMX Non-root state
- **sched_switch**: To detect vCPU thread, Preemption state, and IDLE state
Investigations

vCPU State Detection

• Virtual CPU State Detection:
  • VMX Root and Non-Root State
  • IDLE state
  • Wait state
  • Preempted State

• VMX Root
  • Block I/O
  • Network
  • Memory
  • Other
Investigations

VM Resource view and Thread view

- During VMX Root with exit reason 30 (I/O Request)
- Wake up a thread from qemu thread pool ==> Disk request
- Wake up a thread from vhost threads ==> Network request
Investigations

For more information about resource usage:

- **Disk I/O:**
  - Using Qemu trace points
    - `qemu:thread_pool_submit`, `qemu:thread_pool_complete`, `qemu:bdrv_co_io_em`

- **Network:**
  - Using Host Kernel Trace
    - `net_if_rx`, `net_dev_xmit`, `sched_switch` and `sched_wakup` for `vhost-$\text{(VM-main-thread)}`

- **Memory:**
  - Written Module that investigate accessed paged for a period of time
Investigations

VM statistics for vCPU
Investigations

VM statistics for Disk and Network

<table>
<thead>
<tr>
<th>DiskQemu</th>
<th>3651</th>
<th>16:49:41.0</th>
<th>16:49:5.0</th>
<th>DiskQemu</th>
</tr>
</thead>
<tbody>
<tr>
<td>23842</td>
<td>3652</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>DiskQemu/23842</td>
</tr>
<tr>
<td>vmName</td>
<td>3653</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>DiskQemu/23842/vmName</td>
</tr>
<tr>
<td>read</td>
<td>4055</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>DiskQemu/23842/read</td>
</tr>
<tr>
<td>numberOfSubmitted</td>
<td>4056</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>DiskQemu/23842/read/numberOfSubmitted</td>
</tr>
<tr>
<td>transfer</td>
<td>4057</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>DiskQemu/23842/read/transfer</td>
</tr>
<tr>
<td>STATUS</td>
<td>4058</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>DiskQemu/23842/read/STATUS</td>
</tr>
<tr>
<td>latency</td>
<td>4067</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>DiskQemu/23842/read/latency</td>
</tr>
<tr>
<td>totalTransfer</td>
<td>4068</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>DiskQemu/23842/read/totalTransfer</td>
</tr>
</tbody>
</table>

| write    | 5329 | 16:49:41.0 | 16:49:5.0 | DiskQemu/23842/write |
| numberOfSubmitted | 5330 | 16:49:41.0 | 16:49:5.0 | DiskQemu/23842/write/numberOfSubmitted |
| transfer | 5331 | 16:49:41.0 | 16:49:5.0 | DiskQemu/23842/write/transfer |
| STATUS   | 5332 | 16:49:41.0 | 16:49:5.0 | DiskQemu/23842/write/STATUS |
| latency  | 5340 | 16:49:41.0 | 16:49:5.0 | DiskQemu/23842/write/latency |
| totalTransfer | 5341 | 16:49:41.0 | 16:49:5.0 | DiskQemu/23842/write/totalTransfer |

<table>
<thead>
<tr>
<th>NetQemu</th>
<th>3922</th>
<th>16:49:41.0</th>
<th>16:49:5.0</th>
<th>NetQemu</th>
</tr>
</thead>
<tbody>
<tr>
<td>23870</td>
<td>3923</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>NetQemu/23870</td>
</tr>
<tr>
<td>STATUS</td>
<td>3924</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>NetQemu/23870/STATUS</td>
</tr>
<tr>
<td>Netif</td>
<td>3925</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>NetQemu/23870/Netif</td>
</tr>
<tr>
<td>Netdev</td>
<td>3926</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>NetQemu/23870/Netdev</td>
</tr>
<tr>
<td>tNetdev</td>
<td>3937</td>
<td>16:49:41.0</td>
<td>16:49:5.0</td>
<td>NetQemu/23870/tNetdev</td>
</tr>
<tr>
<td>tNetif</td>
<td>3933</td>
<td>1078579542</td>
<td>1078584864</td>
<td>NetQemu/23870/tNetif</td>
</tr>
</tbody>
</table>
Investigations

VM statistics for vCPU
Investigations


- **Process Identifier (PID)** and **Process Name** inside the VMs is not accessible from Host tracing.
- **CR3** points to the page directory of process inside the VM.
- **SP** points to the stack of the thread inside the virtual machine.
- **CR3** and **SP** are unique identifiers of threads, but to be more human readable we map it with the process info inside the guest.
Investigations

Virtual Machine Dissection

• New trace point:
  • vcpu_enter_guest to obtain unique tuple (CR3, SP, IP)
  • If you want more information:
    • lttng_statedump_stack to obtain thread stack range, thread ID, thread name, hostname

vcpu_enter_guest:
- Compare SP with the stack range of all threads retrieved from guest
- Save TID and Thread Name
Investigations

Virtual Machine Dissection

- First, fibo runs then after 1 sec, we run cpu_burn program
Investigations

Use Case – Preemptive Virtual Machine

30 runs of Sysbench to find primes < 10000

Average = 324ms  STD = 5ms

Average = 443ms  STD = 116ms
Investigations

Use Case – Frequent Transition – Memory Overcommitment

- VM1, VM2, and VM3 are Memory intensive
- VM4, and VM5 are CPU intensive
Investigations
Overload Analysis

- Compare overload of vCPU State Builder (VSB) with multi-level tracing approach by Mohamad Gebai [2]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Baseline</th>
<th>VSB</th>
<th>Multi-level</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VSB (%)</td>
</tr>
<tr>
<td>File I/O (ms)</td>
<td>233</td>
<td>328</td>
<td>361</td>
<td>28.72</td>
</tr>
<tr>
<td>Memory (ms)</td>
<td>319</td>
<td>331</td>
<td>344</td>
<td>3.67</td>
</tr>
<tr>
<td>CPU (ms)</td>
<td>339</td>
<td>352</td>
<td>361</td>
<td>3.72</td>
</tr>
</tbody>
</table>
Investigations

Demo
Investigations

One More Thing …
## Investigations

### Virtual Machine Classification

- **KMeans clustering** is a method for cluster analysis in data mining.
- Aim to partition n VMs into K clusters based on resource consuming patterns.

Metrics: disk_transfer, disk_request, CPU_avg, CPU_request

\[
X = [[30512.80, 503.4, 9447.93, 8845.87],
    [588.97, 6.827, 75537, 3641.5],
    [46206.73, 1383.75, 4298.9, 23756.9],
    [722.22, 8.78, 205141.1, 2716.7],
    [708.41, 8.78, 214826.4, 3373.6],
    [714.003, 7.67, 255922.2, 2811.57]]
\]

\[
Y = [[0.94222657, 0.36056743, -1.16197163, 0.17481621],
     [-0.69031714, -0.61500025, -0.51162465, -0.51364317],
     [1.7984347, 2.0901037, -1.21264046, 2.14731981],
     [-0.68304747, -0.61116339, 0.76373987, -0.63598022],
     [-0.6838009, -0.61116339, 0.8590478, -0.54908229],
     [-0.68349576, -0.6133441, 1.26344907, -0.62343035]]
\]

KMeans clustering

- **StandardScaler**

- **Result**
Conclusion and in-progress

Inferences

- vCPU state builder (VSB) lets the Infrastructure provider to detect Over-committed and under-committed hosts.

- Profiling threads inside the VMs.

- Kmeans could cluster VMs based on resource usage pattern

Going Further

- Using kmeans with more metrics for clustering VMs based on resource usage pattern

- Applying supervised machine learning algorithms like SVM to Cluster VMs
References


Questions?

Hani.nemati@polymtl.ca
https://github.com/Nemati