Runtime Vertical Scaling of Virtualized Applications via Online Model Estimation

Simon Spinner, Samuel Kounev
Dept. of Computer Science, University of Würzburg

Xiaoyun Zhu, Lei Lu, Mustafa Uysal, Anne Holler, Rean Griffith
VMware, Inc.

SASO, September 11th 2014, London, UK
Motivation

**Service-level objective (SLO)**
- End-to-end latency
- Throughput

**Dynamic workload**
- Seasonal patterns
- Trends
- Bursts

**Goal:** Scale resource allocations at runtime depending on current workload

---

**Example virtualized application**

- Webserver (VM 1)
- Application Server (VM 2)
- Database Server (VM 3)

---

**Introduction**

**Approach**

**Evaluation**

**Related Work**

**Conclusions**
Horizontal vs. Vertical Scaling

**Horizontal Scaling**

- VM 1
- VM 2
- VM 3

**Issues:**
- Architectural complexity
  - Load balancing
  - State replication
- Overhead

**Vertical Scaling**

- VM

**Issues:**
- Scale up limited by
  - Physical host
  - Application bottlenecks

**Introduction**

**Approach**

**Evaluation**

**Related Work**

**Conclusions**
- Adaptation of scheduler settings
  - E.g. CPU time: VM1, VM2, VM3
  - Limited by the initially configured VM size

- Add and remove resources during system runtime
  - Hot-add/-remove of virtual CPUs (vCPUs) or memory
  - No service interruption during reconfiguration
State of the Art in Industry

- Metric $M$ is monitored (e.g., CPU Utilization)

- Trigger-based approaches (e.g., Amazon EC2):

  - If $M > M_{up}$ then add resources
  - If $M < M_{down}$ then remove resources

- Complex relationship between application performance and resource allocation

  $\rightarrow$ Determination of thresholds is challenging
Challenges for Vertical Scaling

Complicated application architectures
- Multiple tiers
- Distributed control flow
- Synchronous/asynchronous communication

Heterogeneous resource access
- CPU, memory, hard disk, etc.
- Changing bottleneck resources

Resource contention
- Over-commitment
- Scheduling of VMs on physical resources
- Performance model \( p = f(\lambda, a) \)
  - Performance metric \( p \)
  - Arrival rate \( \lambda \)
  - Resource allocation \( a \)

- Questions answered with the model:
  - Are the allocated resources sufficient to fulfill the application-level performance target?
  - Which resource is currently the bottleneck?
Idea: Apply resource demand estimation techniques

- Using monitoring data from the system
- Continuously updated during system runtime

Definition Resource Demand:
Average time a request obtains service at a resource excluding any waiting time.
1. Layered performance model capturing hypervisor scheduling delays

2. Learning-based approach to automatically estimate the performance model at runtime

3. Feedback controller to dynamically allocate vCPUs to VMs

4. Evaluation with a real-world application (Zimbra Collaboration Server)
Approach Overview

Desired resource allocation \((a_{t+1})\)

Current resource usage \((u_t)\)

Observed app performance \((p_t)\)

Model: \(p = f(\lambda, a)\)

Application Controller

Model Builder

vApp Manager

New VM resource settings
(number of vCPUs, configured memory size)

Introduction  Approach  Evaluation  Related Work  Conclusions
Layered Performance Model

Hierarchical modeling approach (Method of Layers [1]):
Service time at layer $i$ is equal to response time of an underlying closed queueing network at layer $i - 1$
Model Estimation

**Application demand**

\[ R_{v,a} = D_{v,a}^{app} (1 + \frac{Q_{v,a}}{a} B_{v,a}) \]

- \( R_{v,a} \): residence time
- \( Q_{v,a} \): mean queue length
- \( a \): number of vCPUs

\[ R_{v,a} \neq \text{if waiting for other resources} \]

**Virtual resource demand**

\[ D_{v,cpu}^{virt} = D_{v,cpu}^{phys} + \frac{c_{v}^{\text{ready}} + c_{v}^{\text{costop}}}{L \cdot X} \]

- \( c_{v}^{\text{ready}} \): time in ready state
- \( c_{v}^{\text{costop}} \): time in costop state

\[ D_{v,cpu}^{virt} \neq \text{if contention at hypervisor level} \]

**Physical resource demand**

\[ D_{v,cpu}^{phys} = \frac{c_{v}^{\text{run}}}{L \cdot X} \]

- \( c_{v}^{\text{run}} \): time in run state
- \( X_{v} \): throughput

\( v \) denotes the VM

- \( L \): length of observation period
- \( B_{v,a} \): probability that a new job has to wait in queue (assumed 1)

**Introduction**

**Approach**

**Evaluation**

**Related Work**

**Conclusions**
Resource Control Algorithm

- **Input**
  - Service level objective: end-to-end performance $p_{ref}$
  - For each VM $v$
    - Number of vCPUs $a_v$
    - Queue length $Q_v$
    - Estimated resource demands $D_{v,a}$, $D_{v,cpu}^{virt}$, and $D_{v,cpu}^{phys}$

- Run in each control interval (e.g., every 20 seconds)

- Hill-climbing optimization algorithm to determine a VM to
  - Remove 1 vCPU, or
  - Add 1 vCPU
Resource Control Algorithm

- For each VM $v$ analyse model with $a_v$ and $a_v - 1$ vCPUs and current queue length $Q_v$.

- Find VM $v_{down}$ minimizing the end-to-end performance $p_{down}$ for $a_{v_{down}} - 1$.

- If $p_{down} < \delta \cdot p_{ref}$
  - Check stability of system.
  - Remove 1 vCPU from $v_{down}$ and stop.

- If the current number of vCPUs $a_v$ is not sufficient
  - Find VM $v_{up}$ maximizing the speedup $s_{v_{up}}$ for $a_{v_{up}} + 1$.
  - If $s_{v_{up}} > 1$, add 1 vCPU to $v_{up}$ and stop.
Case Study: Zimbra Collaboration Server

- Open-source collaboration software
- Architecture:

```
Frontend: Mailbox Server
  - MySQL
  - File store
  - Jetty
  - OpenLDAP

Backend: Mail Transfer Agent (MTA)
  - Anti-spam
  - Anti-virus
  - Postfix

SOAP

Outgoing Mail
Incoming Mail
```

Introduction  Approach  Evaluation  Related Work  Conclusions

Simon Spinner - Runtime Vertical Scaling of Virtualized Applications  09/11/2014  15
**Experiment Setup**

*VMware ESX host 1*
- VCenter server
- Hyperic server
- Load driver
- Controller

*VMware ESX host 2*
- Zimbra mailbox server

*VMware ESX host 3*
- Zimbra mail transfer agent (MTA)

Each host: 2 x 4 core CPU, 32 GB RAM
Controllers

- Model-based controller
  - Control interval 20 seconds
  - Estimation interval 5 minutes

- Trigger-based controller
  - Thresholds
    - Scale-up if utilization > 90%
    - Scale-down if utilization < 40%
  - Control interval: 1 or 5 minutes

- Static allocation
Experiment 1: Dynamic Workload

- One week from FIFA 98 Worldcup access logs
- Scaled to 9 hours experiment duration
Experiment 1: End-to-end Latency

Zimbra MTA VM:

Both controllers successfully avoid SLA violations
## Experiment 1: vCPU Reconfigurations

### Zimbra MTA VM:

<table>
<thead>
<tr>
<th>Controller</th>
<th>Mean latency [s]</th>
<th>Reconfigurations</th>
<th>Mean vCPUs</th>
<th>Max vCPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-based</td>
<td>20.48</td>
<td>13</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>Trigger-based (1 min)</td>
<td>10.82</td>
<td>273</td>
<td>1.83</td>
<td>3</td>
</tr>
<tr>
<td>Trigger-based (5 min)</td>
<td>25.97</td>
<td>72</td>
<td>1.46</td>
<td>3</td>
</tr>
<tr>
<td>Static allocation</td>
<td>1385</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Model-based controller needs less reconfigurations and resources
Experiment 2: Limits of Scaling

Zimbra MTA with linearly increasing workload:

Estimated demands reflect contention at hypervisor and application level.
Experiment 3: Physical Host Contention

Zimbra MTA with constant workload:

Estimated application demand adapts to changes in the host contention level
Related Work

Horizontal Scaling (Bodik 09, Nguyen 13, Padala 13, …)
- Determining number of VMs per tier
  - Machine learning
  - Demand prediction techniques

Runtime control of scheduler settings
- Limits/Caps setting (Xu 2008, Padala 09)
- Shares/Weighs setting (Blagodurov 13)
- Limited by configured VM size

Adaptation of number of vCPUs
- CloudScale (Shen 11) requires manual thresholds
- (Yazdanov 12) does not consider application performance
- VScaler (Yazdanov 13) uses reinforcement learning
Conclusions

- Model-based approach to runtime vertical scaling using resource demand estimation techniques
  - Layered performance model
  - Online resource demand estimation
  - Feedback controller

- Benefits of model-based approach
  - No manual setting of thresholds required
  - Less oscillations compared to threshold-based approach
  - Makes bottleneck analysis feasible
Future Work

- Multiple workload classes
- Workload forecasting
- Control of additional resources: Memory, I/O
References