How to Tame your VM: an Automated Control System for Virtualized Services

Akkarit Sangpetch
asangpet@andrew.cmu.edu

Andrew Turner
andrewtu@andrew.cmu.edu

Hyong Kim
kim@ece.cmu.edu

Department of Electrical and Computer Engineering
Carnegie Mellon University

LISA 2010 - November 11, 2010
Virtualized Services

- Virtualized Infrastructure (vSphere, Hyper-V, ESX, KVM)
  - Easy to deploy
  - Easy to migrate
  - Easy to re-use virtual machines

- Multi-tier services are configured and deployed as virtual machines
Problem: What about performance?

- VMs share physical resource (CPU time / Disk / Network / Memory bandwidth)
- Need to maintain service quality
  - Response time / Latency
- Existing infrastructure provides mismatch interface between the service's performance goal (latency) and configurable parameters (resource shares)
  - Admins need to fully understand the services before able to tune the system
Existing Solutions – Managing Service Performance

• Resource Provisioning
  – Admins define resource shares / reservations / limits
  – The “correct” values depend on hardware / applications / workloads

• Load Balancing
  – Migrate VMs to balance host utilization
  – Free up capacity for each host (low utilization ~ better performance)
Automated Resource Control

- Dynamically adjust resource shares during run-time
- Input = required service response time
  - I want to serve pages from vm1 in less than 4 seconds
- The system calculates number of shares required to maintain service level
  - Adapt to changes in workload
  - Admins do not have to guess the number of shares
Control System

- Record service response time
- Calculate model parameters
- Adjust host CPU shares
- Calculate # of shares required
Sensor unit

- Record service response time
  - Starts when the server sees “request” packet
  - Until the server sends “response” packet
- Based on libpcap / tshark – need to recognize the application’s header

{service:blog, tier:web, vm: www1, response time 250 ms}, 
{service:blog, tier:db, vm:db1, response time 2.5 ms}
Actuator Unit

- Adjust number of shares based on controller’s allocation
- We use Linux/KVM as our hypervisor
  - Cgroup cpu.shares – control CFS scheduling shares
  - Similar mechanism exists on other platform (Xen weight / ESX shares)
• The modeler calculates the expected service response time based on shares allocated to each VM
• We use an intuitive model for our 2-tier services in the experiment

\[ T_{\text{web}} = f(T_{\text{DB}}) \]
\[ T_{\text{DB}} = f(S_{\text{cpu}}) \]
Resource Shares Effect

Host 1
- Web server VM (Wordpress)

Host 2
- SQL Server VM (mysql)
- Cruncher VM (100% cpu)

Test Clients

NAS

Clients 10
SQL Server CPU Allocation effect

Web Response

Database Response

# of DB CPU shares (out of 1000)

Web Response Time (ms)

Time step (5s)

Database Response Time (ms)

Time step (5s)
Database Response vs CPU

\[ T_{DB} = a_0 (S_{cpu})^{b_0} \]
Database Response vs CPU (controlled)
HTTP vs Database response

\[ T_{\text{web}} = a_1 (T_{\text{DB}}) + b_1 \]
Controllers

• Model uses readings to estimate the service parameters $<a_0, b_0, a_1, b_1>$

  - $T_{DB} = a_0 (S_{cpu})^{b_0}$
  - $T_{web} = a_1 (T_{DB}) + b_1$

• Controller finds the minimal $S_{cpu}$ such that $T_{web} < \text{specified response time}$

  - Long-term control: uses moving average to find model parameters & shares
  - Short-term control: uses the last-period reading to find model parameters

    • Avoid excessive response time violation while waiting for the long-term model to settle
System Evaluations

Test Clients

Host 1
- Web server 1 VM
- Sensor

Host 2
- Web server 2 VM
- Sensor
- DB Server 1 VM
- Actuator

Host 3
- Web server 2 VM
- Sensor

NAS
Dynamic Workload

Web1 Load

Service Response Time (ms)

- web1
- web1-target
- web2
- web2-target
- 50 per. Mov. Avg. (web1)
- 50 per. Mov. Avg. (web2)
## Deviation from Target

<table>
<thead>
<tr>
<th></th>
<th>Without Control</th>
<th></th>
<th>With Control</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean Deviation (ms)</td>
<td>Target Violation Period</td>
<td>Mean Deviation (ms)</td>
<td>Target Violation Period</td>
</tr>
<tr>
<td><strong>Static Load</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instance 1</td>
<td>540</td>
<td>0%</td>
<td>109 (-80%)</td>
<td>9%</td>
</tr>
<tr>
<td>Instance 2</td>
<td>1043</td>
<td>100%</td>
<td>282 (-73%)</td>
<td>26%</td>
</tr>
<tr>
<td><strong>Dynamic Load</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instance 1</td>
<td>276</td>
<td>7%</td>
<td>182 (-34%)</td>
<td>9%</td>
</tr>
<tr>
<td>Instance 2</td>
<td>354</td>
<td>27%</td>
<td>336 (-5%)</td>
<td>12%</td>
</tr>
</tbody>
</table>

Our control system helps track the target service response time.
Further Improvements

• Enhance Service Model
  – Service dependencies
    • Cache / Proxy / Load balancer effects
  – Non-cpu resource (disk / network / memory)

• Robust Control system
  – Still susceptible to temporal system effects
    (garbage collections / cron jobs)
  – Need to determine feasibility of the target performance
Conclusions

• Automated Resource Control system
  – Maintain target system’s response time
  – Allows admin to express allocation in term of performance target

• Managing service performance could be automate with sufficient domain knowledge
  – Need basic framework to describe performance model
  – Leave the details to the machines (parameter fitting / optimization / resource monitoring)
Sample Response & Share values