Applying Statistical Learning, Optimization, and Control to Application Performance Management in the Cloud

Xiaoyun Zhu

October 17, 2014
Rapidly growing public cloud market

Public Cloud Services Market and Annual Growth Rate, 2010-2016

Billions of Dollars

250

200

150

100

50

0


77 93 110 131 155 181 210

Percent

25

20

15

10

5

0

Source: Gartner (February 2013)
How about hosting critical applications?

Economics of IT and Cloud Computing

- 70% of Enterprise IT Budget
- 30% of Enterprise IT Budget
- Public Cloud Computing Sweet Spot

Source: Gartner (October 2012)
Application performance – a real concern

What are your biggest concerns about managing Cloud services?

- Poor end-user experience due to performance bottlenecks: 64%
- Impact of poor performance on brand reputation and/or customer loyalty: 51%
- Loss of revenue due to availability, performance, or troubleshooting cloud services: 44%

Application performance management is hard

Service Level Objective: 95% of all transactions should be completed within 500ms

SLO violation!

Performance troubleshooting & remediation

Cloud hosting provider

Many tenant applications
Challenges in managing application performance

- On average, **46.2 hours** spend in “war-room” scenarios each month

Source: Improving the usability of APM data: Essential capabilities and benefits. TRAC Research, June 2012, based on survey data from 400 IT organizations worldwide
Challenges in usability of performance data

- Time spent correlating performance data: 63%
- Amount of performance data that is not relevant: 61%
- Number or "false positives": 42%
- UI is difficult to use: 38%
- Number of false alerts: 32%

Source: Improving the usability of APM data: Essential capabilities and benefits. TRAC Research, June 2012, based on survey data from 400 IT organizations worldwide
APM goal: achieve service-level-objective (SLO)
Technical challenges

• Enterprise applications are distributed or multi-tiered
• App-level performance depends on access to many resources
  – HW: CPU, memory, cache, network, storage
  – SW: threads, connection pool, locks
• Time-varying application behavior
• Time-varying hosting condition
• Dynamic and bursty workload demands
• Performance interference among co-hosted applications
Better IT analytics for APM automation
Three-pronged approach
Why learning?

• Deals with APM-generated big data problem

• Fills the semantic gap with learned models

• Answers key modeling questions
APM-generated Big Data

• “APM tools were part of the huge explosion in metric collection, generating thousands of KPIs per application.”

• “83% of respondents agreed that metric data collection has grown >300% in the last 4 years alone.”

• 10 years ago data was mostly collected every 15 minutes; now typically every 5 minutes; 23% every 1 minute or less

• “88% of companies are only able to analyze less than half of the metric data they collect… 45% analyze less than a quarter of the data.”

• “77% of respondents cannot effectively correlate business, customer experience, and IT metrics.”

What performance data are collected?

Infrastructure-level

Physical host metrics

- System-level stats collected by the hypervisor
  - e.g., `esxtop` – CPU, memory, disk, network, interrupt
- CPU stats
  - `%USED, %RUN, %RDY, %SYS, %OVRLP, %CSTP, %WAIT, %IDLE, %SWPWT`
- ~100s-1000s metrics per host!

VM metrics

- Resource usage stats collected by the guest OS
  - e.g., `dstat, iostat`
- ~10s metrics per VM

- Widely available on most platforms
- Available at a time scale of seconds to minutes
What performance data are collected?
Application-level

Types of metrics
- End-user experience (response times, throughput)
- Application architecture discovery
- Transaction tracing
- Component monitoring

VMware Hyperic monitoring tool
- Agents deployed in VMs
- Auto-discovers types of applications running
- Plugins to extract application-related performance stats
- Stats available at a time scale of minutes
- Stats aggregated in Hyperic server
- Supports over 80 different application components
- Extensible framework to allow customized plugins
The **Semantic Gap** challenge

Correlating performance data from different sources
Learning helps answer key modeling questions

• **Q1:** Which metrics go into the model?
  • **Thousands** of metrics from each ESX host and their VMs
  • Which system **resources** or **parameters** affect application performance the most?

• **Q2:** What kind of model should we use?
  • **White-box** vs. **empirical** models
  • **Linear** vs. **nonlinear** models
  • **Offline** vs. **online** models

• **Q3:** Does our model capture current behavior?
  • Applications workloads and environments are **constantly changing**
An example multi-tier application

<table>
<thead>
<tr>
<th>Type of Metrics</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>app performance</td>
<td>8</td>
</tr>
<tr>
<td>raw host metric</td>
<td>7226</td>
</tr>
<tr>
<td>raw VM metrics</td>
<td>266</td>
</tr>
</tbody>
</table>
Q1: Which metrics go into the model?
Phase-1: Correlation-based metrics filtering

<table>
<thead>
<tr>
<th>Category</th>
<th>Type of Metrics</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>given app metric</td>
<td>1</td>
</tr>
<tr>
<td>Input</td>
<td>raw system metric</td>
<td>$N_{\text{raw}}$</td>
</tr>
<tr>
<td>Output</td>
<td>candidate metrics</td>
<td>$N_{\text{can}}$</td>
</tr>
</tbody>
</table>

- $|\text{corrcoef}| \geq 0.8$
- p-value $\leq 0.1$

- 132 candidate metrics
- 98% not highly correlated!

Total 7226 raw metrics from 4 ESX hosts (esxtop)

App metric: Throughput
App metric: Mean RT
App metric: 95-p RT
Q1: Which metrics go into the model?
Phase 2: Model-based metrics selection

**Input:** $N_{can}$ candidate metrics and a performance model $F$

**Output:** $N_{pred}$ top predictor metrics that provide a good fit for the data

**Tunable Parameter:** minimum incremental improvement in R2

![Graph showing steps of model selection with incremental improvements | Imp < 0.01, Imp = 0.063, Imp = 0.074, Imp = 0.668]
Q2: What kind of models should we use?
White-box performance models

• **Pros**
  • Solid theoretical foundation
  • Application-aware, easier to interpret
  • Closed-form solution in some special cases

• **Cons**
  • Detailed knowledge of system, application, workload, deployment
  • More appropriate for aggregate behavior or offline analysis
  • Harder to automate, scale, or adapt
Q2: What kind of models should we use?
Black-box empirical models

• **Pros**
  - **Generic**: No *a priori* assumptions
  - **Tools**: Many learning algorithms available
  - **Automation**: Easier to do partially or fully
  - **Scalable**: Easier to codify analysis in algorithms

• **Challenges**
  - **Efficiency**: Real-time data processing and analytics
  - **Accuracy**: Reduces *false positives* and *false-negatives*
  - **Adaptivity**: Handles changing workloads and environments
Q2: What kind of models should we use?
Linear vs. nonlinear models

- Linear regression

- Regression tree

- k nearest neighbors

- Boosting approach
Q2: What kind of models should we use?
Tradeoff between linear and nonlinear models

- **Nonlinear models** have better accuracy than linear regression model
- **Linear regression** model has the least computation cost
- **Boosting algorithm** has the best accuracy and highest cost
- **Regression tree** maybe a good tradeoff between accuracy and cost
Q2: What kind of models should we use?
Offline vs. online models

• **Offline** modeling
  • More appropriate for nonlinear models
  • More suitable for capacity planning and initial sizing
  • Cannot adapt to runtime changes in app, workload, or system

• **Online** modeling
  • Should be cheap to compute and update
  • Linear models more appropriate
  • Can adapt to changes in app, workload, and system
  • Suitable for runtime adaptation and reconfiguration
Q3: Does our model capture current behavior?

Online change-point detection

- **Hypothesis**: The distribution of prediction errors (residuals) is stationary if there are no changes in the application/environment.

- **Detection**: Use a hypothesis test to compare error distributions from adjacent time windows.
vPerfGuard: Learning-based troubleshooting

**Sensor Module**
- Application performance metrics (THP, MRT, RT_{95p})
- Host metrics (1000’s)
- Guest VM metrics (10’s)

**Online Change-Point Detection Module**
- Re-train model? Yes/No
- Online hypothesis testing

**Metric Filtering & Model Building Module**
- **Phase 1:** Correlation-based filtering
- **Phase 2:** Model-based filtering

**Model and top metrics**

**Online Change-Point Detection Module**
- Model and top metrics

**Yes**

**Metric Filtering & Model Building Module**
- raw metrics

**Sensor Module**
- new samples

**Remediation**

# Case study: CPU contention with co-located VMs

## Model retraining

Note: All models during the contention period show CPU on ESX1 as the top metric affecting application latency!

## Intervals and MRT Models

<table>
<thead>
<tr>
<th>Intervals</th>
<th>MRT Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>27 – 45</td>
<td>MRT = 1.13 H_ESX1_CPU_Util + 1.97 H_ESX4_Mem_Active – 89.7</td>
</tr>
<tr>
<td>46 – 74</td>
<td>MRT = 752.8 H_ESX1_CPULoad_1MinAvg – 562.9</td>
</tr>
<tr>
<td>75 – 89</td>
<td>MRT = 12.5 H_ESX1_Web_vCPU_Ready – 25.0</td>
</tr>
<tr>
<td>90 – 102</td>
<td>MRT = -7.70 H_ESX1_vCPU_Idle + 410.3</td>
</tr>
</tbody>
</table>
Why control and optimization?

• **Control** (or **dynamic adaptation**) takes advantage of newly exposed performance tuning knobs

• **Feedback** allows tolerance of model imperfection and uncertainties

• **Optimization** handles tradeoffs between competing goals
  – performance vs. power
  – responsiveness vs. stability
Auto-Scaling to maintain application SLO
A feedback-control approach
Auto-Scaling to maintain application SLO
A feedback-control approach

Horizontal scaling
Auto-Scaling to maintain application SLO
A feedback-control approach

Front Tier

DB Tier

Application Latency

End User

Horizontal scaling

Vertical scaling
Existing solutions to horizontal scaling
Threshold-based approach

- User-defined threshold on a specific metric
  - Spin up new instances when threshold is violated
  - e.g. AWS Auto Scaling: http://aws.amazon.com/autoscaling/

**Challenges**
- How to handle multiple application tiers?
- How to handle multiple resources?
- How to determine the threshold value?
Our Solution: Learning-based auto scaling

- User only needs to provide end-to-end performance goal
- Uses reinforcement learning to capture application’s scaling behavior and inform future actions
- Uses heuristics to seed the learning process
- Handles multiple resources and tiers
- Fully automated without human intervention
Vertical scaling of resource containers
Automatic tuning of resource control settings

• Available on various virtualization platforms

• For shared CPU, memory, disk I/O*, network I/O*:
  – Reservation (R)* – minimum guaranteed amount of resources
  – Limit (L) – upper bound on resource consumption (non-work-conserving)
  – Shares (S) – relative priority during resource contention

• VM’s CPU/memory demand (D): estimated by hypervisor, critical to actual allocation

\[
\text{Actual-allocation} = f(R, L, S, D, \text{Cap})
\]

Available capacity
DRS (Distributed Resource Scheduler)
Resource pool hierarchy

- Capacity of an RP divvied hierarchically based on resource settings
- Sibling RPs share capacity of the VDC
- Sibling VMs share capacity of the parent RP

Powerful knobs, hard to use

• How do VM-level settings impact application performance?
• How to set RP-level settings to protect high priority applications within the RP?
• Fully reserved \((R=L=C)\) for critical applications
  – Leads to lower consolidation ratio due to admission control
• Others left at default \((R=0, L=C)\) until performance problem arises
  – Increases reservation for the bottleneck resource (which one? by how much?)
Performance model learned for each vApp

Maps VM-level resource allocations to app-level performance

- Captures multiple tiers and multiple resource types
- Choose a linear low-order model (easy to compute)
- Workload indirectly captured in model parameters
- Model parameters updated online in each interval (tracks nonlinearity)

\[
\text{VM memory usage } u^k_m(t) \\
\text{VM CPU usage } u^k_c(t) \\
\text{VM I/O usage } u^k_{io}(t)
\]

\[
p(t) = f(p(t-1), u(t))
\]
Use optimization to handle design tradeoff

• An example cost function

\[ J(u(t+1)) = (p(t+1) - p_{SLO})^2 + \beta \| u(t+1) - u(t) \|^2 \]

- Performance cost
- Control cost

Tradeoff between performance and stability

• Solve for optimal resource allocations

\[ u^*(t+1) = g(p(t), p_{SLO}, u(t), \lambda, \beta) \]
AppRM
SLO-driven auto-tuning of resource control settings

• For each application, **vApp Manager** translates its SLO into *desired* resource control settings at individual VM level
• For each resource pool, **RP Manager** computes the *actual* VM- and RP-level resource settings to satisfy all critical applications
vApp Manager overview

Desired resource allocations \((u_{t+1})\)

Current resource allocations \((u_t)\)

Observed app performance \((p_t)\)

Model: \(p = f(u)\)

App-level SLO \((p_{ref})\)

Desired VM resource settings \((s_{t+1})\)

Application Controller

Model Builder

Resource Controller

vApp Manager

RP Manager

vApp Manager

VM_1
VM_2
VM_n

App Sensor

System Sensor

Application Controller

Model: \(p = f(u)\)

Resource Controller

Desired resource allocations \((u_{t+1})\)

App-level SLO \((p_{ref})\)

Observed app performance \((p_t)\)

Current resource allocations \((u_t)\)

Desired VM resource settings \((s_{t+1})\)

vApp Manager

RP Manager

vApp Manager

VM_1
VM_2
VM_n

App Sensor

System Sensor

Application Controller

Model: \(p = f(u)\)

Resource Controller

Desired resource allocations \((u_{t+1})\)

App-level SLO \((p_{ref})\)

Observed app performance \((p_t)\)

Current resource allocations \((u_t)\)

Desired VM resource settings \((s_{t+1})\)

vApp Manager

RP Manager
Performance evaluation

- **Application**
  - **MongoDB** – distributed data processing application with sharding
  - **Rain** – workload generation tool to generate dynamic workload

- **Workload**
  - Number of clients
  - Read/write mix

- **Evaluation questions**
  - Can the vApp Manager meet individual application SLO?
  - Can the RP Manager meet SLOs of multiple vApps?
Result: Meeting mean response time target

- Under-provisioned initial settings: $R = 0$, Limit = 512 (MHz, MB)
- Over-provisioned initial settings: $R = 0$, $L = \text{unlimited}$ (cpu, mem)

Mean response time (target 300ms)
Resource utilization (under-provisioned case)

- Target response time = 300 ms
- Initial setting $R = 0$, $L = 512$ MHz/MB (under-provisioned)

CPU utilization

Memory utilization
Recap: APM automation requires better analytics

Online modeling of application performance

Tradeoff between competing goals

Model-driven online adaptation in face of uncertainty
Grand challenge


“Systems manage themselves according to an administrator’s goals. New components integrate as effortlessly as a new cell establishes itself in the human body. These ideas are not science fiction, but elements of the grand challenge to create self-managing computing systems.”

Enablers

- Widely deployed sensors and lots of (noisy) data
- New control knobs, resource fungibility and elasticity
- Increasing compute, storage, and network capacity
- Matured learning, control, and optimization techniques

Challenges

- Software complexity, nonlinearity, dependency, scalability
- Automated root-cause analysis, integrated diagnosis & control
- Need more collaborations between control and systems people
- How to teach control theory to CS students?
Thanks to collaborators

VMware
• Lei Lu, Rean Griffith, Mustafa Uysal, Anne Holler, Pradeep Padala, Aashish Parikh, Parth Shah

HP Labs
• Zhikui Wang, Sharad Singhal, Arif Merchant (now Google)

KIT
• Simon Spinner, Samuel Kounev

College of William & Mary
• Evgenia Smirni

Georgia Tech
• Pengcheng Xiong (now NEC Lab), Calton Pu

University of Michigan
• Kang Shin, Karen Hou
Related venues

- International Conference on Autonomic Computing
  https://www.usenix.org/conference/icac14

- Feedback Computing Workshop (formerly known as FeBID)
  http://feedbackcomputing.org/
  http://www.controlofsystems.org/
References
