Adaptive, Model-driven Autoscaling for Cloud Applications

Anshul Gandhi
IBM T. J. Watson Research Center
Stony Brook University

Parijat Dube, Alexei Karve, Andrew Kochut, Li Zhang
IBM T. J. Watson Research Center
Motivation

- Businesses have started moving to the cloud for their IT needs
  - reduces capital cost of buying servers
  - allows for elastic resizing of applications that have dynamic workload demand

- Cloud Service Providers (CSPs) offer monitoring and rule-based triggers to enable dynamic scaling of applications

![Amazon auto scaling](image1)
![Microsoft Azure Watch](image2)

![Diagram](image3)
Motivation

• The values have to be determined by the user
  – requires expert knowledge of application (CPU, memory, n/w thresholds)
  – requires performance modeling expertise (when and how to scale)

How to set these values ??

Amazon auto scaling

Microsoft Azure Watch
• The values have to be determined by the user
  – requires expert knowledge of application (CPU, memory, n/w thresholds)
  – requires performance modeling expertise (when and how to scale)

Not possible for CSPs!
View from user’s perspective
View from CSP’s perspective
How to scale an unobservable cloud application to provide performance guarantees?
DC2: High-level idea

Service requirements of requests at each tier

Network delay

Background utilization (overhead)

End-to-end response time

Request rate

VM utilization
DC2: High-level idea

Kalman filtering

- Service requirements of requests at each tier
- Network delay
- Background utilization (overhead)

End-to-end response time

Request rate

VM utilization
DC2: High-level idea

Kalman filtering

- Service requirements of requests at each tier
- Network delay
- Background utilization (overhead)

End-to-end response time

- Request rate
- VM utilization
DC2: Modeling

multi-tier queueing network model
Parameters:
- $\lambda_i$ – Request rate for class $i$
- $T_i$ – Response time for class $i$
- $S_{ij}$ – Service requirement for class $i$ at tier $j$
- $d_i$ – Network latency for class $i$
- $U_{0j}$ – Background utilization on tier $j$
- $U_j$ – Utilization of tier $j$

24 parameters

\[
T_i = d_i + \sum_j \frac{S_{ij}}{1 - U_j}
\]

6 equations
Parameters:

- $\lambda_i$ – Request rate for class $i$
- $T_i$ – Response time for class $i$
- $S_{ij}$ – Service requirement for class $i$ at tier $j$
- $d_i$ – Network latency for class $i$
- $U_{0j}$ – Background utilization on tier $j$
- $U_j$ – Utilization of tier $j$

24 parameters
9 known + 15 unknown

- Underdetermined system
- Need to “infer” unknowns
- Can leverage monitored values

$$T_i = d_i + \sum_j \frac{S_{ij}}{1 - U_j}$$

$$U_j = U_{0j} + \sum_i \lambda_i S_{ij}$$

6 equations

“Guess” unknowns

Evaluate functions using guesses

Compare with monitored values

Improve guess
Kalman filtering + Queueing: Evaluation

Change in workload triggered

- Time to converge: ~1 min (6 intervals)
- Good accuracy

- Time to converge: ~3 min (18 intervals)
- Good accuracy
• RUBiS is an open source benchmark inspired by ebay.com
• Hosted on SoftLayer hypervisors via OpenStack
• We focus on scaling Tomcat app tier

SLA: \( T_{\text{browse}} < 40\text{ms} \) for every 10 s monitoring interval
DC2: All traces

- Bursty trace [WITS]
- Hill trace [ITA]
- Rampdown trace [WITS]
Bursty trace: All policies

Bursty trace [WITS]

DC2

THRES(30,60)

<table>
<thead>
<tr>
<th>V</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>2.50</td>
</tr>
<tr>
<td>0%</td>
<td>2.50</td>
</tr>
</tbody>
</table>
All traces: All policies

<table>
<thead>
<tr>
<th>DC2</th>
<th>V=0%</th>
<th>K=2.50</th>
<th>V=0%</th>
<th>K=2.44</th>
<th>V=0%</th>
<th>K=4.76</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRES(30,60)</td>
<td>V=0%</td>
<td>K=2.50</td>
<td>V=6.66%</td>
<td>K=2.56</td>
<td>V=0%</td>
<td>K=6.00</td>
</tr>
</tbody>
</table>
Limitations and future work

• Evaluation limited to dynamic web applications
  – Currently investigating Hadoop-type applications

• Only applies to stateless tiers
  – DB scaling would be challenging

• Scaling algorithm can be modified

• Kalman Filtering can be replaced by other black-box approaches
  – Machine Learning approaches?

• Non-zero convergence time
Conclusions

• Need for adaptive scaling services for (opaque) cloud applications
  – Application agnostic
  – Robust to arrival patterns
• Existing commercial offerings do not suffice: rule-based
• Existing auto-scaling research solutions do not apply due to lack of visibility and control of opaque cloud applications

• Our solution: Dependable Compute Cloud (DC2)
  – Does not require offline benchmarking or expert knowledge
  – Can adapt to dynamic changes in workload
• Well suited for cloud users who lack expertise in system modeling and application knowledge
Thank You!
Conclusions

• Need for adaptive scaling services for (opaque) cloud applications
  – Application agnostic
  – Robust to arrival patterns
• Existing commercial offerings do not suffice: rule-based
• Existing auto-scaling research solutions do not apply due to lack of visibility and control of opaque cloud applications

• Our solution: Dependable Compute Cloud (DC2)
  – Does not require offline benchmarking or expert knowledge
  – Can adapt to dynamic changes in workload
• Well suited for cloud users who lack expertise in system modeling and application knowledge
Backup
Existing CSP solutions

• Resource usage triggers
  – Amazon Auto Scaling, Microsoft Azure Watch, VMware AppInsight, CiRBA
• Request rate for specific software (ex: apache)
  – RightScale
• Latency/VM
  – Amazon Elastic Load balancing
• Web site response time
  – Scalr

User has to set values
All workloads: All policies (Bursty trace)

- Rule-based policies like THRES require tuning and are not robust
- Other auto-scaling policies require control of application

- **DC2 is superior to THRES and does not require application control**

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>MoreDB</th>
<th>MoreApp</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATIC-OPT</td>
<td>V=0% K=3.00</td>
<td>V=0% K=4.00</td>
<td>V=0% K=3.00</td>
</tr>
<tr>
<td>DC2</td>
<td>V=0% K=2.50</td>
<td>V=0% K=3.66</td>
<td>V=0% K=2.94</td>
</tr>
<tr>
<td>THRES(30,60)</td>
<td>V=0% K=2.50</td>
<td>V=3.06% K=3.40</td>
<td>V=2.04% K=2.98</td>
</tr>
</tbody>
</table>

Note: V and K are metrics related to application performance and resource allocation.
Kalman filtering

- KF is a reactive, feedback-based estimation approach that has only recently been employed for computer systems
- KF automatically learns the (possibly changing) system parameters, for any system, including combination of workloads
- We extend KF to a 3-tier 3-workload-class system
- Based on KF estimation, DC2 automatically, and proactively, detects which tier is the bottleneck, and how to resolve the bottleneck (scale VMs)
  - do not require any knowledge of application, except topology
Kalman filtering + Queueing

• KF can be integrated with system models (ex, queueing models) to improve accuracy and convergence
• Model *need not* be accurate
  – KF leverages (true) monitored values to account for model inaccuracies
  – Well suited for approximate system models such as queueing-theoretic models
  – Can use other models as well, ex: machine-learning based models
## All traces: All policies

<table>
<thead>
<tr>
<th>STATIC-OPT</th>
<th>DC2</th>
<th>THRES(30,60)</th>
<th>THRES(30,50)</th>
<th>THRES(40,60)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V=0%</td>
<td>V=0%</td>
<td>V=0%</td>
<td>V=2.02%</td>
<td>V=2.02%</td>
</tr>
<tr>
<td>K=3.00</td>
<td>K=4.00</td>
<td>K=2.50</td>
<td>K=2.79</td>
<td>K=2.19</td>
</tr>
<tr>
<td>V=0%</td>
<td>V=6.66%</td>
<td>V=0%</td>
<td>V=15.87%</td>
<td>V=0%</td>
</tr>
<tr>
<td>K=2.50</td>
<td>K=2.56</td>
<td>K=2.72</td>
<td>K=2.13</td>
<td>K=4.62</td>
</tr>
<tr>
<td>V=0%</td>
<td>V=0%</td>
<td>V=0%</td>
<td>V=0%</td>
<td>V=0%</td>
</tr>
<tr>
<td>K=6.00</td>
<td>K=4.76</td>
<td>K=6.00</td>
<td>K=6.00</td>
<td>K=4.62</td>
</tr>
</tbody>
</table>

### Diagrams

- **Bursty trace [WITS]**
- **Hill trace [ITA]**
- **Rampdown trace [WITS]**