Delivering Elastic Containerized Cloud Applications to Enable DevOps

Cornel Barna, Hamzeh Khazaei, Marios Fokaefs, Marin Litoiu
York University
fokaefs@yorku.ca
Where are we now and what do we know?

➢ State-of-the-practice and new trends
➢ Infrastructure, software, performance management
Performance management... *ex post facto!*
### Boxes and...not so boxes!

#### Virtual Machines
- Servers!
- Pre-packaged computation boxes (flavours)
- Isolation between VMs, dependent on physical machine
- Comes with an OS
- One runtime environment per VM

#### Containers
- Not Servers! Processes!
- Flexible, ad-hoc resource slicing
- Isolation between containers, dependent on host
- No OS! (inherited from host)
- Multiple runtime environments per host.
Data, data, data...

- Multiple tiers to manage.
- Different utilization metrics to consider.
- Possibility for multi-layer deployment (software-container-VM-physical)
- Can we design a single AMS for a multi-tier, multi-layer application? (*RQ2*)
Where do we go from here?

➢ Performance model for containers
➢ Model-based autonomic manager
Performance Model

- K in MAPE-K
- Layered Queuing Networks
- Contribution during planning phase
- “Predict” the effect of scaling actions on performance
- Dynamic planning for scaling actions.

LQN for Containers
Model estimation
Model-based AMS
Scaling Heat Algorithm

• Allow violations to occur a few times before taking action.
• Current practice: account only for consecutive violations; if sequence is interrupted, counter resets.
• BUT! This can hide “lurking” violations.
LQN for Containers

• Function tiers and infrastructure layers introduce delays.
• Delays are captured as demands for resources in the hardware layer.
• Delays are summed up to determine the total utilization and responsiveness of the system.
• Uncertainties are captured through a virtual delay center.
Model estimator/corrector

• Model is constructed for given infrastructure and workload level.

• As these change, the model has to be adapted.

• Kalman filter corrects the *demands* (model coefficients) according to current metrics.
Model-based Autonomic Management System
Planning for Spark Scaling

• Every Spark job needs one *driver* process and at least one *executor* process.

• [limited resources, multiple jobs] => only drivers with no executors => *driver starvation*

• Add Spark containers for every new Tomcat => artificially increase the probability to bind with free executor processes
Model-based AMS => DevOps

Monitoring → Analysis → Planning → Execution

Load Balancer

Web Cluster

DB Server

DevOps = Knowledge (Integration)
How well did we perform?

➢ Evaluate AMS
➢ Evaluate model
AMS Experiments (simple workload)
AMS Experiments (complex workload)
# Model Experiments

<table>
<thead>
<tr>
<th>Calibration</th>
<th>1</th>
<th>2</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU(_W) Error</td>
<td>~0</td>
<td>4.8</td>
<td>7.8</td>
</tr>
<tr>
<td>CPU(_S) Error</td>
<td>~0</td>
<td>4.5</td>
<td>7.8</td>
</tr>
<tr>
<td>RT Error</td>
<td>~0</td>
<td>4.0</td>
<td>8.9</td>
</tr>
<tr>
<td>T/P Error</td>
<td>~0</td>
<td>4.5</td>
<td>5.9</td>
</tr>
<tr>
<td>P (CPU(_W))</td>
<td>99</td>
<td>86</td>
<td>65</td>
</tr>
<tr>
<td>R (CPU(_W))</td>
<td>99</td>
<td>75</td>
<td>67</td>
</tr>
<tr>
<td>F (CPU(_W))</td>
<td>99</td>
<td>80</td>
<td>66</td>
</tr>
<tr>
<td>P (CPU(_S))</td>
<td>100</td>
<td>83</td>
<td>65</td>
</tr>
<tr>
<td>R (CPU(_S))</td>
<td>98</td>
<td>84</td>
<td>80</td>
</tr>
<tr>
<td>F (CPU(_S))</td>
<td>99</td>
<td>84</td>
<td>72</td>
</tr>
</tbody>
</table>
Contributions/Conclusions

• **DevOps in practice**: Automation (AMS) and Integration (model) to achieve efficient management for software systems in production.

• One of the first works to achieve autoscaling in containerized topologies.

• **Extension** of performance model to handle more layers and more tiers assisting in complex scaling decisions.

• **Cutting-edge** technologies, including containers (Docker), big data analytics (Spark and Cassandra), validated on an actual cloud environment (OpenStack)