Towards Self-Aware Dependability Management in Virtualized Service Infrastructures

Samuel Kounev

www.descartes-research.net
Agenda

- Descartes Research Group @ KIT
- Challenges Posed by Cloud Computing
- Resource Management in the Cloud
- Vision and Research Roadmap
- Initial Steps and Preliminary Proof-of-Concept

The Descartes Research Group @ KIT

- Named after the french philosopher René Descartes
- Funding: DFG, KIT, EU, Industry
- Focus: Engineering of *Self-Aware* Software Systems
- www.descartes-research.net
Traditional Data Center Infrastructures

- Applications running on dedicated hardware
- Over-provisioned system resources
- Poor resource utilization and energy efficiency
- Increasing number of servers → rising operating costs
Cloud Computing Infrastructures

Applications running in a virtualized environment
Shared physical infrastructure
Flexible mapping of logical to physical resources
Higher resource utilization & energy efficiency
Lower operating costs
Cloud Computing Infrastructures (2)

Load Spike
Cloud Computing Infrastructures (3)

Network Attack / Intrusion
Cloud Computing Infrastructures (4)
Cloud Computing Infrastructures (5)

Service Provider (SP)

Infrastructure Provider (IP)
Challenges Posed by Cloud Computing

- Increased system complexity and dynamics
- Lack of direct control over underlying hardware
- New threats and vulnerabilities due to resource sharing
- Separation of service providers and infrastructure providers

- Inability to provide QoS and dependability guarantees
- Lack of trust
Run-time Performance Management

Server Utilization

<table>
<thead>
<tr>
<th>Service</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>max</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

SLAs for response times (sec)

Server Utilization

<table>
<thead>
<tr>
<th>Service</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>max</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Modern Service-Oriented System
Modern Service-Oriented System

System workload and usage profile
- Number and type of clients
- Input parameters and input data
- Data formats used
- Service workflow

Software architecture
- Connections between components
- Flow of control and data
- Component resource demands
- Component usage profiles

Execution environment
- Number of component instances
- Server execution threads
- Amount of Java heap memory
- Size of database connection pools

Virtualization layer
- Physical resources allocated to VMs
  - number of physical CPUs
  - amount of physical memory
  - secondary storage devices

Network bandwidth between system nodes
1. Performance prediction at design & deployment time

- Descriptive architecture-level performance meta-models
  - E.g., PCM, SPE-MM, CSM, CBML, KLAPER, UML SPT, UML MARTE

- Automated transformation to predictive models
  - E.g., layered queueing networks, stochastic Petri nets

Main issues:

- Overhead in building and analyzing models
- Models assume static system architecture
- Maintaining models during operation is prohibitively expensive

[M. Woodside et al], [D. Petriu et al], [R. Reussner et al], [C. Smith et al], [R. Mirandola et al], [K. Trivedi et al], [V. Cortellessa et al], [I. Gorton et al], [J. Merseguer et al], [D. Menasce et al], [E. Eskenazi et al], [J. Murphy et al],…
2. Performance and resource management at run-time

- Simple models used that abstract the system at very high level
- Services modeled as black boxes
- Restrictive assumptions often imposed, e.g.:
  - Single workload class
  - Homogeneous servers
  - Single-threaded components
  - Exponential request interarrival times and service times
- Layers of the execution environment not modeled explicitly

[G. Pacifici et al], [A. D’Ambrogio et al], [G. Tesauro et al], [D. Menasce et al], [C. Adam et al], [Rashid A. Ali et al], [I. Foster et al], [S. Bleul et al], [A. Othman et al], [P. Shivam et al], [R. Berbner et al], [H. Song et al], …
The Past

System

Models
The Future

System Models

© Samuel Kounev
The Future

“I think, therefore I am …”
-- René Descartes
Next generation **self-aware** software systems:

1. Aware of their architecture and the environment they are running in

2. Aware of internal and external changes and able to predict their effect
   a) External changes, e.g., evolving service workloads
   b) Internal changes, e.g., dynamically undertaken reconfiguration actions

   “thought (cogitatio) is what happens in me such that I am immediately conscious of it…“ -- Rene Descartes

3. Proactively adapting to enforce QoS and resource efficiency

   “For it is not enough to have a good mind: one must use it well”
   -- Rene Descartes

4. Based on integrated **dynamic** QoS prediction models

   Dualism: “the mind controls the body, but that the body can also influence the mind” -- Rene Descartes
Dynamic Model Composition
Run-time Performance Management

Server Utilization

<table>
<thead>
<tr>
<th>Service</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>85%</td>
<td>55%</td>
<td>60%</td>
<td>15%</td>
</tr>
</tbody>
</table>

Service A
Service C
Service D
Service E
Service F

1

SLAs for response times (sec)

<table>
<thead>
<tr>
<th>Service</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>max</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Shutdown

Server Utilization

<table>
<thead>
<tr>
<th>Service</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>85%</td>
<td>?</td>
<td>?</td>
<td>0%</td>
</tr>
</tbody>
</table>

Service A
Service C
Service D
Service F

Shutdown

Server Utilization

<table>
<thead>
<tr>
<th>Service</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>85%</td>
<td>0%</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Service A
Service B
Service D
Service E
Service F

Service C

Shutdown
Performance Prediction On-The-Fly

Model Composition

Dynamic Service Models

Model-to-Model Transformation

Architecture-level Performance Model

Model Analysis

Predictive Performance Model

Performance Predictions

Service A
Service B
Service C
Service D
Service E
Service F

Shutdown

Service A
Service B
Service C
Service E
Service D
Service F
### Performance Prediction On-The-Fly (2)

#### Server Utilization

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>max</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

#### SLAs for Resp. Times (sec)

<table>
<thead>
<tr>
<th>Service</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>max</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

#### Server Utilization

<table>
<thead>
<tr>
<th></th>
<th>85%</th>
<th>70%</th>
<th>80%</th>
<th>0%</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>55%</td>
<td>55%</td>
<td>55%</td>
<td>55%</td>
</tr>
<tr>
<td>max</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
</tr>
<tr>
<td></td>
<td>30%</td>
<td>30%</td>
<td>30%</td>
<td>30%</td>
</tr>
</tbody>
</table>

#### SLAs for Resp. Times (sec)

<table>
<thead>
<tr>
<th>Service</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>max</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Generalized Online Prediction Process

Online QoS Query

- Component QoS Models
  - Model Composition
    - Architecture-level System QoS Model

Query Results

- QoS Predictions
  - Model Analysis
    - Predictive System QoS Model

© Samuel Kounev
System Control Loop

Refine/Calibrate Model(s) → Collect
- Monitor System and Workload
  - SLA Violations
  - Inefficient Resource Usage

Forecast Workload → Analyze
- Anticipate/Detect Problem

Collect → Act
- Reconfigure System

Act → Decide
- Predict Reconfiguration Effect(s)
- Generate Reconfiguration Scenario
- Generate Query
- Online QoS Prediction

 Decide
- Analyze Query Results

Problem resolved

Problem persists
Input from Multiple Communities

- Software architecture meta-models
- Predictive performance models
- Dynamic virtualized service infrastructures
- Autonomic resource management techniques
- Software Engineering
- Performance Modeling
- Cluster / Grid / Cloud Computing
- Autonomic Computing

SELF-AWARE
Initial Steps: Meta-Model for Dynamic Systems
Initial Steps: Application Level

Scaling Number of Co-Located VMs

Core Affinity Memory Benchmarks on SunFire X4440

Memory Mark Rating

1 VM, Affinity  1 VM, no Affinity  23 VMs, Affinity  24 VMs, Affinity  24 VMs, no Affinity
Case Study: Automated Model Extraction

Extracting architecture-level performance models from online monitoring data

Case Study: Automated Model Extraction (2)

- Model A: Resource demands approximated with measured response times
- Model B: Resource demands estimated based on utilization and throughput data

Model A: $U_{WLS\_CPU} = 0.12$, Model B: $U_{WLS\_CPU} = 0.81$, Steady State Time: 1020 sec
Case Study: Online QoS Control

Case Study: Online QoS Control (cont.)

Client \( t_1 \) Server 1

Server 1 Thread Pool

Service Queue

Server \( N \) Thread Pool

QoS-Aware Resource Manager

Grid Server 1

Grid Server \( N \)
Case Study: Online QoS Control (cont.)

- 8-way Pentium Xeon 2.60 GHz, 9 GB, 64 bit, Xen hypervisor

- 4-way Pentium Xeon 3.16 GHz, 10 GB, 64 bit, Xen hypervisor
Case Study: Online QoS Control (cont.)

- 99 session requests executed over period of 2 hours
- Run until all sessions complete
- Average session duration 18 minutes (92 requests)
- Will compare two configurations
  - **Without QoS Control**
    - Incoming requests simply load-balanced
    - Reject session requests when servers saturated
  - **With QoS Control**
    - QoS-aware admission control enforced
Case Study: Online QoS Control (cont.)

![Graph showing response time](image-url)

- Response Time SLA
- Response Time with Overload Control
- Response Time with QoS Control
Case Study: Online QoS Control (cont.)

- **Config 1: Without QoS Control**
  - 96% of sessions admitted, SLAs observed by only 22% of them

- **Config 2: QoS Control / workload model available**
  - 54% of sessions accepted

- **Config 3: QoS Control / workload characterization on-the-fly**
  - Rejects only 14 sessions (16%) more compared to Config 2

<table>
<thead>
<tr>
<th>Config</th>
<th>SLA fulfilled</th>
<th>SLA violated</th>
<th>Sessions rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>63</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>2</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>34</td>
<td>0</td>
<td>51</td>
</tr>
</tbody>
</table>
Case Study: Online QoS Control (cont.)

- Config 1: All servers online / without QoS control.
- Config 2: All servers online / with QoS control.
- Config 3: Servers added on demand / without QoS control.
- Config 4: Servers added on demand / with QoS control.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SLA fulfilled</th>
<th>SLA violated</th>
<th>Sessions rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>63</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>2</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>61</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
<td>7</td>
<td>33</td>
</tr>
</tbody>
</table>
Case Study: Online QoS Control (cont.)

- Up to five server failures emulated during the run
- Points of server failures chosen randomly during the 2 hours
- Sessions reconfigured after each server failure

<table>
<thead>
<tr>
<th>Failures emulated</th>
<th>Without QoS Control</th>
<th>With QoS Control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SLA fulfilled</td>
<td>SLA violated</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>62</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>57</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>58</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>45</td>
</tr>
</tbody>
</table>
Summary

- CC promises to revolutionize the way software is built and run
- QoS issues and lack of trust → major show stoppers
- **Self-Aware Software Systems**
  - Systems with integrated dynamic prediction models
  - Models composed dynamically at run-time
  - Used for autonomic QoS management
- **Major challenges**
  - Platform-independent meta-model for dynamic software systems
  - Trade-off accuracy vs. management overhead
  - Bridging the gap between service provider and infrastructure provider
Thank You!

www.descartes-research.net