Introduction to the Descartes Meta-Model (DMM)

Samuel Kounev
Charles University Prague, June 4, 2012
References

- Papers can be downloaded from http://www.descartes-research.net

- Vision of Self-Aware IT Systems, Infrastructures and Services

- Descartes Meta-Model (DMM) / Online Models for Self-Awareness
References (2)

Automatic Model Extraction (Model Inference) based on Online Monitoring


Modeling Virtualization Platforms


Miscellaneous


Agenda

- Motivation
- Approach & Methodology
- Exemplary Results
- Vision
- Conclusion
## Research Areas

<table>
<thead>
<tr>
<th>Dependability</th>
<th>Quality-of-Service</th>
<th>Elasticity</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmarking</td>
<td>Meta-models for dynamic software systems</td>
<td>Dynamic resource provisioning &amp; capacity mgmt</td>
<td>• SaaS, PaaS, IaaS</td>
</tr>
<tr>
<td>Workload characterization</td>
<td>Analytical and simulation-based prediction models</td>
<td>Application quality-of-service management</td>
<td>Service-oriented Computing</td>
</tr>
<tr>
<td>Instrumentation &amp; profiling</td>
<td>Model extraction, calibration &amp; maintenance</td>
<td>Elastic scalability</td>
<td>• Web Services, SOA, ESB</td>
</tr>
<tr>
<td>Experimental analysis</td>
<td>Predictability at design-time</td>
<td>Cost and efficiency management</td>
<td>Distributed Component-based Systems</td>
</tr>
<tr>
<td>Online monitoring</td>
<td>Predictability at run-time</td>
<td>Power/energy management</td>
<td>• Java EE, MS .NET</td>
</tr>
</tbody>
</table>

### Research Areas

#### Technology Domains
- Event-based Systems
  - • EDA, MOM, distributed pub/sub
- Grid/Cluster Computing
  - • Service-oriented Grids
Motivation

- Increasing data center operating costs
  - System management costs
  - Power consumption costs
  - Cooling costs

- Gartner (2009)
  - Power consumption doubled from 2000 to 2005
  - By 2025, an increase by 1600% expected!

- Increasing carbon footprint of ICT
  - 2% to 4% of global CO2 emissions
  - Projected to rise to 10% in 10 years
Conventional Data Centers

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

Application 1

Application 2

Application N

Distributed virtualized data centers
Challenges

Load Spike
Challenges (2)

Network Attack / Intrusion
Challenges (3)
Challenges (4)

Load Fluctuations

Expand/shrink resources?
- At what rate?
- At what granularity?
- Reserve for how long?
Challenges

- 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 dependability and QoS guarantees
- Lack of trust
High-Level Research Questions

- How to *automatically predict* vulnerabilities arising from varying workloads, network attacks or system failures?

- How to *proactively adapt* the system to avoid SLA violations or inefficient resource usage?

- How to *provide dependability / QoS guarantees* while ensuring high resource utilization and energy efficiency?

How to engineer **trustworthy** and **efficient** systems?

Further details in:

Agenda

- Motivation
- **Approach & Methodology**
- Exemplary Results
- Vision
- Conclusion
Proactive Self-Adaptive Systems Management

Online QoS prediction for problem anticipation

Online QoS prediction for reconfiguration impact analysis

Autonomic system adaptation

Online reconfiguration impact prediction for trade-off analysis

Service A
VM replication/cloning

Service A
Scaling up/Improving dependability

Service C
Dynamic server consolidation

SLA OK

SLA

Service B
LiveVM migration

Efficiency OK

Cost/Energy efficiency metric

Time [mins, hours, days, weeks, months]

Time [mins, hours, days, weeks, months]

Cost/Energy efficiency metric

Time [mins, hours, days, weeks, months]
Proactive Self-Adaptive Systems Management

System needs to be explicitly aware of its

Online QoS prediction for problem anticipation

Goals & objectives
Dynamic changes
Effect of changes

System architecture
Resource landscape
Possible adaptation actions
Impact of adaptation (beforehand)

Online QoS prediction for reconfiguration impact analysis

Online prediction
SLA violation
Workload change

Dependability/Responsiveness
Efficiency

Dynamic adaptation strategies
Impact of adaptation (afterwards)

Online prediction
VM replication/cloning
Dynamic server consolidation

Cost/EFM
service usage
Time [mins, hours, days, weeks, months]

Cost/EFM
service usage
Time [mins, hours, days, weeks, months]

Online prediction
Dependability/Responsiveness
Efficiency

PART 1

PART 2

PART 3

PART 1

PART 2

PART 3

© Samuel Kounev

Descartes Research Group
http://www.descartes-research.net
Self-Reflective
Aware of their software architecture, execution environment and hardware infrastructure, as well as of their operational goals

Self-Predictive
Able to anticipate and predict the effect of dynamic changes in the environment, as well as the effect of possible adaptation actions

Self-Adaptive
Proactively adapting as the environment evolves to ensure that their operational goals are continuously met

Details in:
Examples of Performance-Influencing Factors

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
State-of-the-Art: Model-based Approaches

1. Models for QoS prediction at run-time
   • Simple models used that abstract the system at very high level
   • Many restrictive assumptions imposed
   • Most of the mentioned aspects are 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 er al], [S. Bleul et al], [A. Othman et al], [P. Shivam et al], …

2. Models for QoS prediction at design & deployment time
   • Overhead in building and analyzing models
   • Models assume static system architecture
   • Maintaining models at run-time 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], [D. Menasce et al], [E. Eskenazi et al], …
Descartes Meta-Model (DMM)

- Architecture-level modeling language for **self-aware** run-time systems management of modern IT systems, infrastructures and services

- Main Goal: Provide Quality-of-Service (QoS) guarantees
  - Performance (current focus)
    - Response time, throughput, scalability and efficiency
  - Or more generally, dependability
    - Including also availability, reliability and security aspects
Descartes Meta-Model (DMM)

Collection of several meta-models each focusing on different system aspects
Tailored Model-to-Model Transformations

Usage Sub-model

Soft. Arch. Sub-model

Middleware Sub-model

Virtualization Sub-model

Infrastructure Sub-model

Dynamically Composed Model Instance

Operational Analysis

Queueing Network Models

Queueing Petri Nets

Stochastic Process Alg.

Full-Blown Simulation

Analytical Sol.

Analytical Sol.

Analytical Sol.

Simulation
Example 1: Simple Bounds Analysis

\[ R \geq \max \left[ N \times \max \{ D_i \}, \sum_{i=1}^{K} D_i \right] \]

\[ X_0 \leq \min \left[ \frac{1}{\max \{ D_i \}}, \frac{N}{\sum_{i=1}^{K} D_i} \right] \]

\[ \frac{N}{\max \{ D_i \} [K + N - 1]} \leq X_0 \leq \frac{N}{\text{avg} \{ D_i \} [K + N - 1]} \]
Example 2: Product-Form Queueing Network

Production Line Stations

Client

Application Server Cluster

Database Server

© Samuel Kounev
Example 3: Layered Queueing Network
Example 3a: Layered Queueing Network
Example 4: Queueing Petri Net
Example 4a: Queueing Petri Net

Further details in:

System Control Loop

Part 1
- Refine/Calibrate Model(s)
- Forecast Workload
- Collect
  - Monitor System and Workload
- Analyze
  - Anticipate/Detect Problem

Part 2
- Decide
  - Predict Reconfiguration Effect(s)
  - Generate Reconfiguration Scenario
  - Analyze Query Results
  - Online QoS Prediction
  - Generate Query

Part 3
- Act
  - Reconfigure System
  - Problem resolved
  - Problem persists
The Descartes Meta-Model

Samuel Kounев, Fabian Hrusaik, Nikolaus Huber
Descartes Research Group
Chair for Software Design & Quality (SDQ)
Karlsruhe Institute of Technology (KIT), Germany
kounev@kit.edu, brosia@kit.de, huber@kit.de

March 21, 2012
v0.50

www.descartes-research.net
Agenda

- Motivation
- Approach & Methodology
- Exemplary Results
- Vision
- Conclusion
Further details in:

Resource Landscape: Example

© Samuel Kounev
Resource Landscape: Container Templates and Runtime Environment Classes

ContainerRepository
  1 \rightarrow templates
    0..*

ContainerTemplate
  0..1 \rightarrow templateConfig
  0..1

ConfigurationSpecification

RuntimeEnvironmentClasses

«enumeration»

HYPERVERSOR
OS
OS_VM
PROCESS_VM
MIDDLEWARE
OTHER

© Samuel Kounev
Resource Landscape:
Configuration Specification

- PassiveResourceCapacity
  - capacity : EBigInteger

- PassiveResourceSpecification
  - passiveResourceSpecification

- ActiveResourceSpecification
  - parentResourceSpecification
  - processingResources

- ProcessingResourceSpecification
  - schedulingPolicy : SchedulingPolicy
  - processingRate : EDouble

- NumberofParallelProcessingUnits
  - number : EBigInteger

- LinkingResourceSpecification
  - communicationLinkResourceType

- CustomConfigurationSpecification
  - parentResourceSpecification
  - linkingResources

- ConfigurationSpecification
  - non-functionalProperties

- EObject
  - 0..1

- «enumeration»
  - SchedulingPolicy
    - DELAY
    - FCFS
    - PROCESSOR_SHARING
    - RANDOM
    - N/A

© Samuel Kounev
Example: Virtualization Layer

- Virtualization Platform
  - Virtualization Type
    - Full Virtualization
    - Para-Virtualization
      - Binary Translation
  - VMM Architecture
    - Dom0
    - Monolithic
  - Resource Management
    - Configuration
      - CPU Scheduling
      - CPU Allocation
        - e.g. vcpu=4
        - e.g. mask=1,2
      - CPU Priority
      - Core Affinity
      - Resource Overcommitment
    - Number of VMs
    - Memory Allocation
    - Memory
    - Network
    - Disk
    - I/O

Legend
- exclusive OR
- inclusive OR

© Samuel Kounev

Descartes Research Group
http://www.descartes-research.net
**DMM: Adaptation Points**

Adaptation Points: Examples

- Scaling Resources

- Replicating VMs, Migrating VMs
Adaptation Points

- Specification of valid system configurations
- “Decorator” model of static view

VariabilityPoint

VariabilityPointDescriptions

VariationType

VariationType 1

VariableModelEntity

PropertyRange

minValueConstraint : EString
maxValueConstraint : EString

ControllableVariableValue

minValue : EDouble
maxValue : EDouble

UnorderedSet

valueConstraint : EString

EObject

0..*

variabilityPoints

value

1

entity

1

variants

0..*

VariabilityPoint

value

1

[HBK12]

© Samuel Kounev

Descartes Research Group
http://www.descartes-research.net
DMM: Application Architecture

Further details in:

Modeling the Application Level

- Service Behavior Abstractions for Different Levels of Granularity
- Probabilistic Parameter Dependencies
- Deployment-Specific Resource Demands / Response Times
Online Performance Prediction Scenario

Customer A → Workload A → Cluster A → Application Server Cluster → Database Server

Customer B → Workload B → Cluster B

Benchmark Driver Agents

20 Compute Nodes
each node has 2 x 4-core Intel CPUs, 32GB main memory

Cluster A

Cluster B

SPECjEnterprise2010 deployed in a clustered environment

Dell PowerEdge R904
4 x 6-core AMD CPUs, 128 GB main memory

GBit LAN
Service Behavior Abstractions

- BlackBoxBehavior
  - No information about resources, resource demands, control flow, call frequencies,…

- CoarseGrainedBehavior
  - Information at component boundary level (external services, resource consumption,…)

- FineGrainedBehavior
  - Information about component-internals (control flow, resource demands, parametric dependencies,…)

© Samuel Kounev
Service Behavior Abstractions

- **ServiceBehaviorAbstraction**
- **Signature**
- **BlackBoxBehavior**
- **CoarseGrainedBehavior**
- **FineGrainedBehavior**
- **InterfaceProvidingEntity**
- **InterfaceRequiringEntity**
- **RepositoryComponent**
- **System**
- **CompositeComponent**
- **Subsystem**
- **BasicComponent**

© Samuel Kounev
Service Behavior Abstractions

BlackBoxBehavior

CoarseGrainedBehavior

ResponseTime

ExternalCallFrequency

ResourceDemandSpecification

ExternalCall

CallFrequency

ProcessingResourceType

ResourceDemand
Service Behavior Abstractions

- Control flow abstraction

FineGrainedBehavior

- AcquireAction
- ReleaseAction
- PassiveResource
- StartAction
- StopAction

ComponentInternalBehavior

- AbstractAction
  - pred
  - succ
  - forks

Control flow abstraction

- Signature
- ResourceDemandSpecification
- LoopIterationCount

- ForkAction
  - body

- BranchAction
  - BranchProbabilities

© Samuel Kounev
Probabilistic Parameter Dependencies

- Characterize dependencies statistically

- Influencing parameters
  - Service input parameters
  - Return parameters of external service calls

- Influenced quantities
  - Loop iteration numbers (FineGrainedBehavior)
  - Branch probabilities (FineGrainedBehavior)
  - Call frequencies (CoarseGrainedBehavior)
  - Resource demands (FineGrainedBehavior, CoarseGrainedBehavior)
  - Response times (BlackBoxBehavior)
  - Input parameter of ext. service call (FineGrainedBehavior, CoarseGrainedBehavior)
Model Variables - Metamodel

**ModelVariable**

- **InfluencingParameter**
  - name : string
  - description : string

- **InfluencedVariable**
  - * 1
  - * 1

- **ServiceBehaviorAbstraction**

- **CallParameter**

- **ControlFlowVariable**
  - desc : RandomVariable
  - 0..1

- **ResourceDemand**
  - desc : RandomVariable
  - 0..1

- **ResponseTime**
  - desc : RandomVariable
  - 0..1

- **BranchProbabilities**

- **CallFrequency**

- **LoopIterationCount**

© Samuel Kounev
Probabilistic Parameter Dependencies

Required service: sendRequisitionToBuyer(List demands)

Provided service: scheduleWorkOrder(String assemblyId, int quantity)

b) Derived Probability Mass Function (PMF)
Scope - Motivation

- Customer-specific application server cluster
- Scenario: Replicate Server Instance

© Samuel Kounev
Scope - Motivation

Customer-specific application server cluster

© Samuel Kounev

Descartes Research Group
http://www.descartes-research.net
Relationship Characterization - Metamodel

ExplicitFunction

ExplicitCharacterization

DependencyRelationship

EmpiricalCharacterization

CorrelationRelationship

EmpiricalFunction

+compute(ein influencingParam : Literal, ein scope : AssemblyContext) : ProbabilityFunction
Deployment-Specific Resource Demands

Component Repository

Resource Demand: 15 ms CPU

Component Repository

Resource Demand: 12 ms CPU

System

Deployment

Resource Landscape

CompB

<<Assembly Context>>

<<Resource Container>>

Application Server 1

<<Resource Container>>

Application Server 2

CompB

<<Assembly Context>>

<<ResourceContainer>>

CompB

<<Assembly Context>>

<<ResourceContainer>>

CompB

<<Assembly Context>>

<<ResourceContainer>>

CompB

<<Assembly Context>>

<<ResourceContainer>>

CompB

<<Assembly Context>>

<<ResourceContainer>>

CompB

<<Assembly Context>>

<<ResourceContainer>>

CompB

<<Assembly Context>>

<<ResourceContainer>>

CompB

<<Assembly Context>>

<<ResourceContainer>>
DMM: Modeling Adaptation Processes

Details in:

Motivation

- Rapid growth of autonomic computing and self-adaptive systems engineering

Open challenges
- System-specific reconfiguration techniques typically hard-coded in the system’s implementation
- How to separate software design and implementation from system reconfiguration logic?

Main issue:
- How to abstract from system-specific details?
- How to enable the reuse of adaptation strategies?
Holistic Model-based Approach

- Describe system adaptation processes at the system architecture level
  - Distinguish high-level reconfiguration objectives from low-level implementation details
  - Explicitly separate technical from logical aspects
  - Capture reconfiguration logic in a generic, human-understandable, machine-processable and reusable way
- Provide intuitive modeling concepts that can be employed by system architects and software developers
- Facilitate maintenance and reuse
Autonomic Resource Management

Refine/Calibrate Model(s)  Forecast Workload

Collect

* Service Workloads  * SLA Violations
* Resource Utilization  * Inefficient Resource Usage
* SLAs

Monitor System and Workload

Act

Reconfigure System

Problem resolved

Problem persists

Decide

Predict Reconfiguration Effect(s)

Analyze

Anticipate/Detect Problem

Generate Reconfiguration Scenario

Generate Query

Analyze Query Results

Online QoS Prediction
Model-based System Reconfiguration

Reconfiguration Language

- Strategies
- Tactics
- Actions

Architecture-Level System QoS Model

System Architecture Sub-Model

Configuration Space Sub-Model

Managed System

Uses

Reconfigures

Parameterizes

Models

Strategies Tactics Actions

© Samuel Kounev
Motivation

- Detailed models of the environment
  - Technical: Structuring, Migration
  - QoS property influences
  - Resource layers
  - → Improved decision making

- Dynamic Reconfiguration
  - On the model level (end-to-end)
  - Decouple from technical, system-specific details
S/T/A Reconfiguration Language

Separate
- Logical view, high-level process
- Technical view, low-level operations

[HHK+12]
Separation of Concerns

Strategies
- High-level
- Independent of system specific details
- Describe process view
- Indeterminism

Tactics & Actions
- Low-level
- System specific
- Reconfiguration operations
- Deterministic
S/T/A Meta-Model

- Actions refer to adaptation points / DoF Model
- Tactics execute Actions in Reconfiguration Plans
- Strategies use weighted Tactics
Example Strategies

<<OverallGoal>>
Maintain SLAs using as little resources as possible

<<Objective>>
MaintainSLAs

<<Objective>>
OptimizeResourceEfficiency

<<Strategy>>
PUSH
(Increase Resources)

<<WeightedTactic>>
addResources

<<Event>>
SlaViolated

<<Strategy>>
PULL
(Decrease Resources)

<<Event>>
Scheduled Optimization

<<WeightedTactic>>
removeResources

<<WeightedTactic>>
undoPreviousAction
Example Tactics & Actions

<<Tactic>>
addResources

<<Loop>>
counter=#iterations

<<ActionReference>>
addVCPU

<<ActionReference>>
addAppServer

<<Tactic>>
removeResources

<<ActionReference>>
removeAppServer

<<ActionReference>>
removeVCPU

<<Tactic>>
undoPreviousAction

<<ActionReference>>
addAppServer

<<ActionReference>>
addVCPU

PrevActionAddServer

TRUE

FALSE

© Samuel Kounev
Evaluation

- S/T/A implemented in PerOpteryx
Ongoing and Future Work

- Efficient heuristics and optimization algorithms specifically tailored for use in S/T/A
- Graphical tools for modeling reconfiguration processes
- Model-based evaluation of reconfiguration processes/algorithms at system design time
  - Efficiency, Self-stabilization, Scalability, Elasticity, …
- Standard metrics and benchmarks
Self-Adaptive Resource Management

Online reconfiguration impact prediction for trade-off analysis

Service A
VM replication/cloning
Scaling up/Improving dependability
Online prediction
Dependability/Responsiveness OK

Service A
VM replication/cloning
Scaling up/Improving dependability
Online prediction
Dependability/Responsiveness OK

Service A
LiveVM migration
Dynamic server consolidation
Efficiency OK

Service A
LiveVM migration
Dynamic server consolidation
Efficiency OK

Reconfiguration results

SLA OK

Further details in:

Self-Adaptive Resource Allocation

- **Analyze**
  - Collect
  - Monitor System and Workload

- **Act**
  - Reconfigure System

- **Decide**
  - Predict Reconfiguration Effect(s)
  - Generate Reconfiguration Scenario

- **Problem persists**
- **Problem resolved**

- **Anticipate/Detect**
  - Problem
- **Generate Reconfiguration Scenario**

- **Reconfigure System**

- **Customer Specification(s)**
- **Workload Forecasting**
Reconfiguration Algorithm

Decide

PUSH Phase
- Add resources
  - vCPUs (if available)
  - Application server nodes
  - until

\[
\overline{\text{cap}}(c, t) = \left[ \frac{\sum_{c \in C} c[\lambda] \cdot D(c[s])}{\sum_{c \in C} c[\lambda] \cdot D(c[s])} \right] \cdot \text{cap}(c, t)
\]

PULL Phase
- Remove underutilized resources as long as no SLAs are violated
Reconfiguration Algorithm

**PUSH**

\[
\text{while } \exists c \in \tilde{C} : \neg P_R(c) \text{ do} \\
\quad \text{for all } t \in V(c[s]) : \neg P_U(t) \text{ do} \\
\quad \quad \text{while } \text{cap}(c, t) \leq \overline{\text{cap}}(c, t) \text{ do} \\
\quad \quad \quad \text{if } \exists i \in F(c[s], t) : i[\kappa] < i[\overline{\kappa}] \text{ then} \\
\quad \quad \quad \quad i[\kappa] \leftarrow i[\kappa] + 1 \\
\quad \quad \quad \text{else} \\
\quad \quad \quad \quad F(c[s], t) \leftarrow F(c[s], t) \cup \{i\} \\
\quad \quad \text{end if} \\
\quad \quad \text{end while} \\
\quad \text{end for} \\
\text{end while}
\]

**PULL**

\[
\text{for all } c \in C \text{ do} \\
\quad \text{while } \exists t \in V(c[s]) : \overline{U}(t) - U(t) \geq \epsilon \text{ do} \\
\quad \quad \text{if } \exists i \in F(c[s], t) : i[\kappa] > 0 \text{ then} \\
\quad \quad \quad i[\kappa] \leftarrow i[\kappa] - 1 \\
\quad \quad \quad \text{if } \neg P_R(c) \text{ then} \\
\quad \quad \quad \quad i[\kappa] \leftarrow i[\kappa] + 1 \\
\quad \quad \text{end if} \\
\quad \quad \text{if } i[\kappa] = 0 \text{ then} \\
\quad \quad \quad F(c[s], t) \leftarrow F(c[s], t) \setminus \{i\} \\
\quad \quad \text{end if} \\
\quad \text{end while} \\
\text{end for}
\]
Case Study: SPECjEnterprise2010

System Under Test

Java Application Server

Orders Domain

JMS

LargeOrder

FulfillOrder

EJB / Web Services

CreateVehicleEJB, CreateVehicleWS

Servlets

Browse, Purchase, Manage

CreateVehicleEJB, CreateVehicleWS

JMS

Requisition

Delivery

Supplier Domain

JDBC

Database Server

Database

Benchmark Driver

Dealerships

Manufacturing Sites

Emulator

Suppliers

Web Services

PurchaseOrder

Delivery
Architecture-Level Performance Model
Semi-automatic extraction [Brosig11]
- 28 components, 63 behavior specs
- Example control flow specification
Experimental Setup

- Six blade servers
  - 2 Xeon E5430 4-core CPUs
  - 32 GB of main memory
- Citrix XenServer 5.5
- Oracle WebLogic 10.3.3
- Oracle Database 11g
What If: New Service Added?

![Chart showing mean response times for different services under different configurations.](chart.png)

- **Default configuration (c_o)**
- **New service (c_0)**
- **After reallocation (c_1)**
What If: Workload Changes?

a) increasing workload from 2x to 4x

b) increasing workload from 4x to 6x
Benefits in Cost Savings

Assigned Capacity (VCPU's)

Workload

day1 1x
day2 4x
day3 6x
day4 4x
day5 6x
day6 2x
day7 1x

static assignment

capacity
servers

Assigned Servers
0 1 2 3 4
Modeling Virtualization Platforms

Virtualization Platform

Virtualization Type
- Full Virtualization
- Para-Virtualization
- Binary Translation

VMM Architecture
- Dom0
- Monolithic

Resource Management Configuration
- CPU Scheduling
- Resource Overcommitment
- Memory Allocation
- CPU Priority
- Core Affinity

Resource Management
- CPU
- Memory
- Disk
- Network
- I/O

Workload Profile

Legend
- exclusive OR
- inclusive OR

Further details in:

Automated Experimental Analysis

Further details in:

Experiment Setup

- Virtualization Platforms
  - Citrix XenServer 5.5
  - VMware ESX 4.0

- Experimental environment
  - SunFire X4440 Server, AMD Opteron 24*2.4 GHz, 128 GB DDR2 RAM

- Different benchmark types
  - Passmark PerformanceTest v7 (CPU, Memory, HDD)
  - SPECcpu (CPU + Memory)
  - Iperf (Network)
Virtualization Overhead

XenServer 5.5
Throughput metric: higher values are better

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>native</th>
<th>virtualized</th>
<th>Delta (abs.)</th>
<th>Delta (rel.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passmark CPU, 1 core</td>
<td>639.3</td>
<td>634.0</td>
<td>5.3</td>
<td>0.83%</td>
</tr>
<tr>
<td>Passmark CPU, 2 cores</td>
<td>1232.0</td>
<td>1223.0</td>
<td>9.0</td>
<td>0.97%</td>
</tr>
<tr>
<td>SPECint(R)_base2006</td>
<td></td>
<td></td>
<td></td>
<td>3.61%</td>
</tr>
<tr>
<td>SPECfb(R)_base2006</td>
<td></td>
<td></td>
<td></td>
<td>3.15%</td>
</tr>
<tr>
<td>Passmark Memory, 1 core</td>
<td>492.9</td>
<td>297.0</td>
<td>195.9</td>
<td>39.74%</td>
</tr>
<tr>
<td>Passmark Memory, 2 cores</td>
<td>501.7</td>
<td>317.5</td>
<td>184.2</td>
<td>36.72%</td>
</tr>
<tr>
<td>Iperf (send)</td>
<td>527.0</td>
<td>393.0</td>
<td>134.0</td>
<td>25.43%</td>
</tr>
<tr>
<td>Iperf (receive)</td>
<td>528.0</td>
<td>370.0</td>
<td>158.0</td>
<td>29.92%</td>
</tr>
</tbody>
</table>
Virtualization Overhead (2)

Legend
- native (1 CPU core)
- virtualized (1 CPU core)
- native (2 CPU cores)
- virtualized (2 CPU cores)

<table>
<thead>
<tr>
<th>Category</th>
<th>Native 1 Core</th>
<th>Virtualized 1 Core</th>
<th>Native 2 Cores</th>
<th>Virtualized 2 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total (Memory Mark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocate Small Block</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Cached</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Uncached</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large RAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scalability

- Scaling CPU resource
- Performance impact of affinity

Affinity OFF
Mutual Performance Influences

- Citrix XenServer 5.5
- $\text{VM}_A$ and $\text{VM}_B$ pinned on the same core > core$_0$
- $r_A$, $r_B$ performance drop compared to isolation

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Mem</th>
<th>CPU</th>
<th>Mem</th>
<th>Disk</th>
<th>CPU</th>
<th>Mem</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{VM}_A$</td>
<td>CPU</td>
<td>Mem</td>
<td>Mem</td>
<td>CPU</td>
<td>Mem</td>
<td>Disk</td>
<td>Mem</td>
<td>Disk</td>
</tr>
<tr>
<td>$\text{VM}_B$</td>
<td>CPU</td>
<td>Mem</td>
<td>Mem</td>
<td>Disk</td>
<td>Disk</td>
<td>Net</td>
<td>Net</td>
<td>Net</td>
</tr>
<tr>
<td>$r_A$</td>
<td>46.71%</td>
<td>50.64%</td>
<td>50.33%</td>
<td>23.35%</td>
<td>24.82%</td>
<td>31.16%</td>
<td>52.88%</td>
<td>52.85%</td>
</tr>
<tr>
<td>$r_B$</td>
<td>52.44%</td>
<td>45.93%</td>
<td>49.04%</td>
<td>1.49%</td>
<td>-0.99%</td>
<td>45.99%</td>
<td>40.46%</td>
<td>42.18%</td>
</tr>
</tbody>
</table>
Automated Model Extraction @ Run-Time

Details in:


• S. Kounev, K. Bender, F. Brosig, N. Huber, and R. Okamoto. **Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics.** In 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2011), Barcelona, Spain, 2011. **Best Paper Award.**
Automated Model Extraction @ Run-Time

```
Extract Effective Architecture

Component Connections

Component-Internal Performance-Relevant Control Flow

Acceptable accuracy?

No

Refine Performance Model

Yes

Extract Model Parameters

Control Flow Statistics

Resource Demands

Probabilistic Parametric Dependencies

© Samuel Kounev
```
Automated Model Extraction @ Run-Time

b) Response times of benchmark operations

Legend
- Browse
- Purchase
- Manage
- CreateVehicleEJB
- CreateVehicleWS

Measured Response Time [ms]

low  medium  high  very high

workload

c) Relative error of response time predictions

Legend
- Browse
- Purchase
- Manage
- CreateVehicleEJB
- CreateVehicleWS

Relative Resp. Time Prediction Error in %

low  medium  high  very high

workload
Automated Model Extraction @ Run-Time

(a) Utilization measurements and predictions

Legend
- WLS Utilization, measured
- WLS Utilization, predicted
- DBS Utilization, measured
- DBS Utilization, predicted

CPU Utilization

low  medium  high  very high

(b) Relative error of response time predictions

Relative Resp. Time Prediction Error in %

low  medium  high  very high

workload

Browse  Purchase  Manage  CreateVehicleEJB  CreateVehicleWS
Agenda

- Motivation
- Approach & Methodology
- Exemplary Results
- Vision
- Conclusion
Today

Systems

Models
The Future

Systems

Models
The Future

“I think, therefore I am …”
-- René Descartes
Self-Aware Software & Systems Engineering

- Service-oriented architectures & modeling techniques
- Stochastic models for QoS prediction

Software & Systems Engineering
Computer Systems Modeling

Cluster, Grid and Cloud Computing, Green IT
Distributed Systems & Autonomic Computing

- Dynamic virtualized data center infrastructures
- Control theory and self-adaptation techniques

SELF-AWARE SYSTEMS & SERVICES
SPEC Research Group (SPEC RG)  
http://research.spec.org

- Founded in March 2011
- Platform for collaborative research in the area of quantitative system evaluation and analysis
- Foster interaction between industry and academia

Wider scope
- Metrics and benchmarking methodologies
- Methods and tools for experimental system analysis
- Covering all stages of the system lifecycle
- Both existing and newly emerging technologies
- Evaluation of early prototypes and research results

Classical performance metrics: response time, throughput, scalability, resource/cost/energy efficiency, elasticity

Plus dependability in general: Availability, reliability and security
Thank You!

http://www.descartes-research.net
http://www.relate-itn.eu
http://research.spec.org