Engineering Virtualized Services

Einar Broch Johnsen

University of Oslo, Norway
einarj@ifi.uio.no

IFIP WG2.2 Meeting
Lisbon, September 26, 2013

http://www.envisage-project.eu

Berndnaut Smilde: Nimbus II, 2012

Cloud Computing & Virtualization

Cloud Computing

- Execution environment with elastic resource provisioning, several stakeholders, and a metered service at multiple granularities for a specified level of quality of service (QoS)
- A host offers services to clients, including infrastructure and platform functionalities and software services to virtualize resource deployment

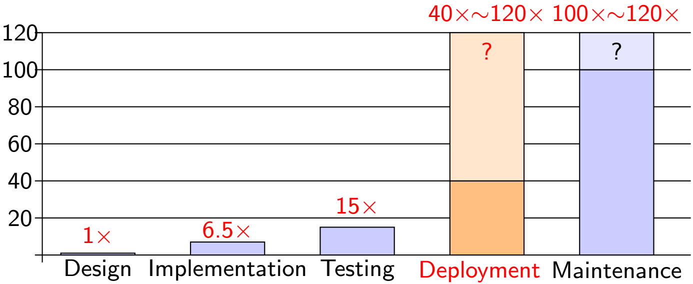
Virtualization

- Virtualization provides an elastic amount of resources to application-level services, e.g., by allocating a changing processing capacity to a service depending on demand
- We say that application-level services are virtualized if they can adapt to the elasticity of cloud computing

Goal: Model-based approach to evaluate and compare resource-management strategies and SLA-compliance

Relative Costs to Fix Software Defects

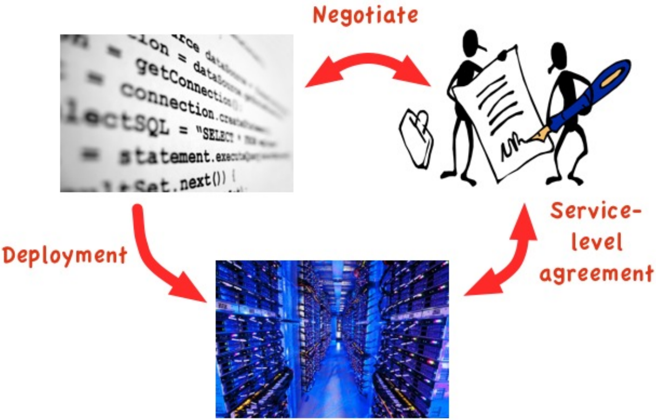
Virtualized systems with dynamic infrastructure



The columns indicate the phase/stage of the software development at which the defect is found and fixed.

(extending figure from IBM Systems Sciences Institute)

Virtualization & SLA



Can we make these pieces fit together?

Why Deployment Modeling?

Questions

1. How will the response time and cost of running my system change if I double the number of servers?
2. How do fluctuations in client traffic influence the performance of my system on a given deployment architecture?
3. Can I better control the performance of my system by means of application-specific load balancing?

Why ABS?

- **Abstraction:** Deployment decisions are expressed at the abstraction level of the modeling language (avoid "model drift")
- **Incrementality:** deployment decisions can be added at any stage in the model development
- Models reflect the **execution and data flow** of target programs
- **Formal** language specification with operational semantics

Modeling Virtualized Services in ABS

Models as Abstract Executable Designs

- Models follow the execution flow of distributed OO systems, but abstract from implementation details using ADTs
- Functional layer: user-defined types and functions, pattern matching
- Imperative layer: objects communicate by asynchronous method calls
- Flexible synchronization: blocking or suspending activities
- Java-like syntax: intuitive to the programmer

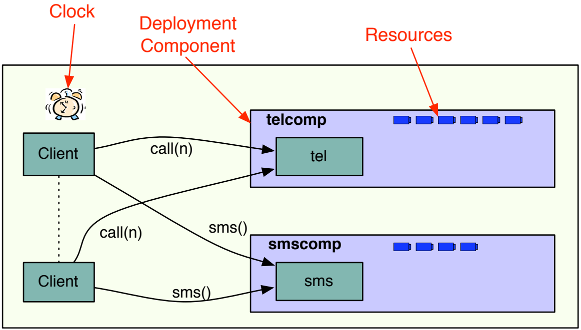
Separation of Concerns: Cost and Capacity

- The cost depends on the program/model
- The capacity depends on the deployment

Deployment Scenarios Capture Virtualized Architectures

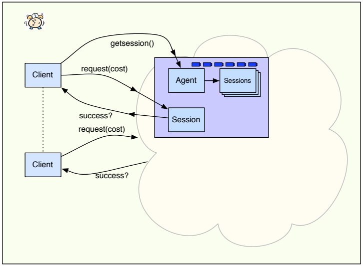
Express how artefacts are deployed at (virtualized) locations with given resource capacities

Deployment Components with Parametric Resources



- Example: Processing resources
- Parametric bound on abstract processing capacity
- Resources reflect the execution capacity of the deployment component in a time interval
- Resources abstract from the number and speed of the (physical) processors available to the component

Deployment Configuration and Resource Reallocation



Let components and resources be first-class citizens in Real-Time ABS

```
data Resource = InfCPU | CPU(Int capacity) ;

interface DC {
  Int total();
  Int load(Int n);
  Int transfer(DC target, Int amount);
}
```

Johnsen, Owe, Schlatte, Tapia Tarifa. *Dynamic Resource Reallocation Between Deployment Components*. Proc. ICFEM 2010, LNCS 6447, Springer 2010

Example: Phone Services - Abstract Behavioral Model

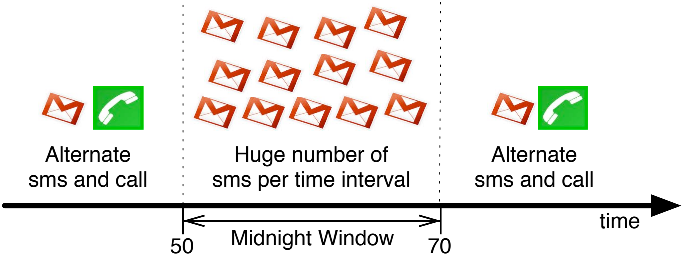
Telephone Service

```
interface TelephoneService {
    Unit call(Int calltime);
}
class TelephoneServer implements TelephoneService {
  Int callcount = 0;
  Unit call(Int calltime){
    while (calltime > 0) { [Cost: 1] calltime = calltime − 1;
                           await duration(1, 1); }
    callcount = callcount + 1;
  }
}
```

SMS Service

```
interface SMSService {
    Unit sendSMS();
}
class SMSServer implements SMSService {
  Int smscount = 0;
  Unit sendSMS() {[Cost: 1] smscount = smscount + 1;}
}
```

Example: The New Year's Eve Client Behavior




```
class NYEclient(Int frequency,TelephoneService ts,SMSService smss){
  Time created=now(); Bool call=false;
  Unit normalBehavior(){ ... }
  Unit midnightWindow(){ ... } // Switch at appropriate time...
}
{// Main block:
   DC smscomp = new cog DeploymentComponent("smscomp", CPU(50));
   DC telcomp = new cog DeploymentComponent("telcomp", CPU(50));
   [DC : smscomp] SMSService sms = new cog SMSServer();
   [DC : telcomp] TelephoneService tel = new cog TelephoneServer();
   Client c = new cog NYEbehavior(1,tel,sms); ... // Clients
}
```

Formal Semantics of Resource-Restricted Execution

- SOS style operational semantics
- Embed the ABS semantics in rules for annotations and time
- Let $[\![e]\!]^t_\sigma$, $\rightarrow$ denote the "untimed" reduction system of ABS

Runtime syntax

$$
\begin{array}{rcl}
cn & ::= & \varepsilon \mid obj \mid msg \mid fut \mid cmp \mid cn\ \ cn \\
obj & ::= & o(a, p, q) \\
cmp & ::= & dc(n, u, k, \overline{h}, \overline{z})
\end{array}
\qquad
\begin{array}{rcl}
tcn & ::= & cn\ \ cl(t) \mid \{cn\ \ cl(t)\} \\
msg & ::= & m(o, \overline{v}, f, d) \\
fut & ::= & f \mid f(v)
\end{array}
$$

Deployment components

- n = available resources
- u = used resources
- k = resources available in the next interval
- $\bar{h}$ = resource usage over time intervals
- $\bar{z}$ = available resources over time intervals

Rules for Annotations

CPU = Gradual reduction

- Annotations are reduced until the statement can be executed
- With lower capacity, the reduction may take several time intervals

$$\text{(EMP-ANNOTATION)}\quad \begin{array}{c} o(a, \{l \mid [\varepsilon]\ s\}, q) \\ \rightarrow o(a, \{l \mid s\}, q) \end{array}$$

$$\text{(COST1)}\quad \frac{\begin{array}{c} a(thisDC) = dc \quad an = \texttt{Cost}: e, an' \\ [\![e]\!]^t_{aol} = c \quad c \leq n - u \\ o(a, \{l \mid [an']\ s\}, q)\ cl(t)\ cn \\ \rightarrow o(a', p', q')\ cl(t)\ cn' \end{array}}{\begin{array}{c} o(a, \{l \mid [an]\ s\}, q)\ dc(n, u, k, \overline{h}, \overline{z})\ cl(t)\ cn \\ \rightarrow o(a', p', q')\ dc(n, u + c, k, \overline{h}, \overline{z})\ cl(t)\ cn' \end{array}}$$

$$\text{(COST2)}\quad \frac{\begin{array}{c} a(thisDC) = dc \quad an = \texttt{Cost}: e', an' \\ [\![e']\!]^t_{aol} = c \quad c > n - u \quad n \neq u \\ c' = c - (n - u) \quad an'' = \texttt{Cost}: c', an' \end{array}}{\begin{array}{c} o(a, \{l \mid [an]\ s\}, q)\ dc(n, u, k, \overline{h}, \overline{z})\ cl(t)\ cn \\ \rightarrow o(a, \{l \mid [an'']\ s\}, q) \\ dc(n, n, k, \overline{h}, \overline{z})\ cl(t)\ cn \end{array}}$$

Rules for Time Advance

Maximal progress

- Time only advances when the execution is otherwise blocked
  - The objects have run out or resources
  - The objects are blocked (e.g., method replies)
  - The objects are idle

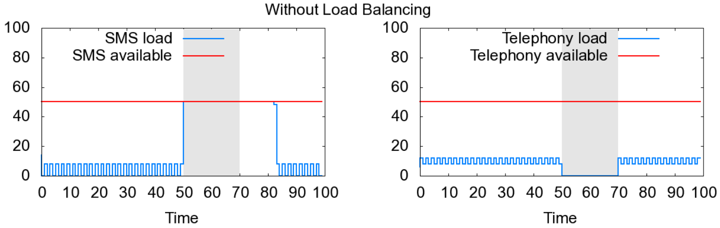
(Run-Inside-Interval)

$$\frac{cn\ cl(t) \xrightarrow{!} cn'\ cl(t) \qquad 0 < d \leq mte(cn', t) \quad \lfloor t \rfloor = \lfloor t + d \rfloor}{\{cn\ cl(t)\} \\ \rightarrow_t \{timeAdv(cn', d)\ cl(t + d)\}}$$

(Run-To-New-Interval)

$$\frac{cn\ cl(t) \xrightarrow{!} cn'\ cl(t) \qquad 0 < d \leq mte(cn', t) \quad \lceil t \rceil = t + d}{\{cn\ cl(t)\} \\ \rightarrow_t \{timeAdv(rescRefill(cn'), d)\ cl(t + d)\}}$$

Example: Simulation Results

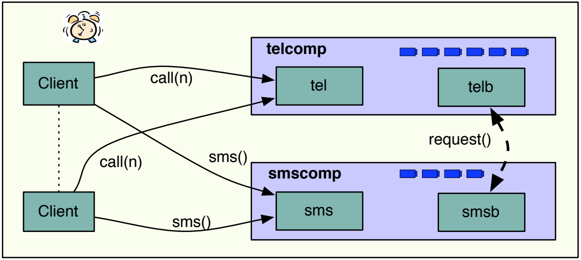


Load Balancing in Deployment Scenarios

Load Balancing: resource reallocation, object mobility, job distribution

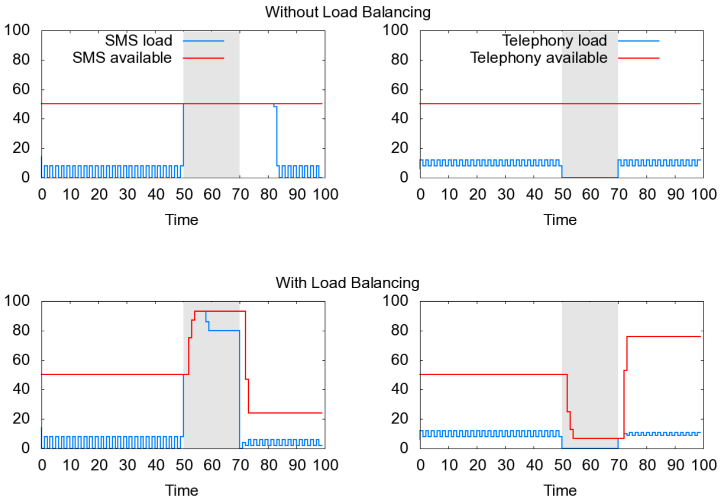
- dc.**load**(e): average load on dc during the last e time intervals
- dc.**total**(): currently allocated resources on dc
- dc.**transfer**(dc2, r): transfer r resources to dc2

Load Balancing in the Phone Services



Strategy: Reallocate 1/2×total resources upon request from partner

Example: Simulation Results



Virtual Resource Management & Cloud Provisioning

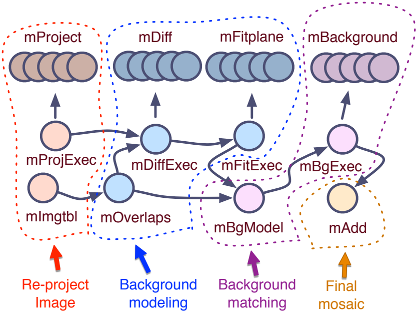
- DCs can be used to model virtual machines
- On the cloud, you pay almost exclusively for CPU resources
- Abstract model of a cloud provider
- Provides facilities for starting, and stopping virtual machines
- Implements a price policy "accumulated cost"

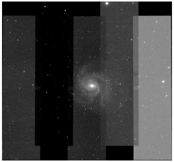
```
interface CloudProvider {
    DC createMachine(Int capacity);
    Unit acquireMachine(DC machine);
    Unit releaseMachine(DC machine);
    Int getAccumulatedCost();
}
```

- Allows the client to interact with resource provisioning on the cloud in an intuitive and fine-grained way
- Compare no. of clients, (missed) deadlines, and accumulated cost

Case Study: Montage (1)

Montage:
Creating a mosaic from a set of astronomical input images

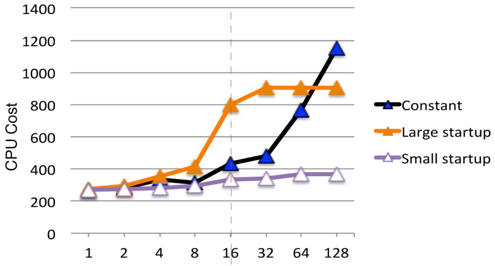


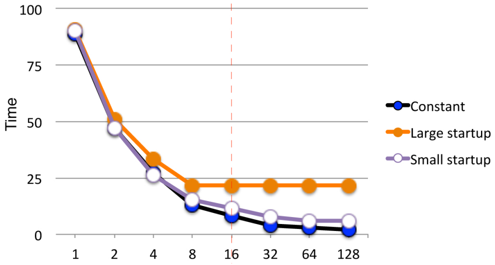


- Fairly complex workflow: partly ordered, highly parallelizable tasks
- Montage system supports both grid and cloud deployment
- The costs of running Montage on different deployment scenarios on the cloud have been studied in the GridSim simulation tool
- How would our results compare to results from the GridSim?

Johnsen, Schlatte, Tapia Tarifa. *Modeling Resource-Aware Virtualized Applications for the Cloud in Real-Time ABS*. Proc. ICFEM 2012, LNCS 7635, Springer 2012

Case Study: Montage (2)

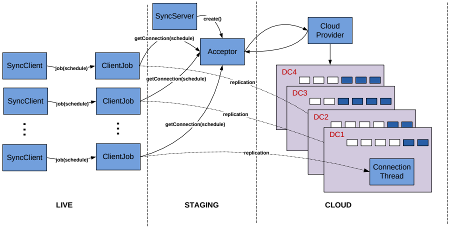




- Deployment scenarios range from 1 to 128 machines
- We got lower accumulated cost than the results from GridSim
- Explanation: The GridSim simulations did not shut down idle servers
- Modified our model to keep all created servers running ("constant")
- Comparison with more active resource management strategies

Case Study: Fredhopper Replication Server (1)

The Fredhopper Access Server (FAS) is a distributed, concurrent OO system providing search and merchandising services to e-Commerce companies. The Replication Server is one part of FAS.



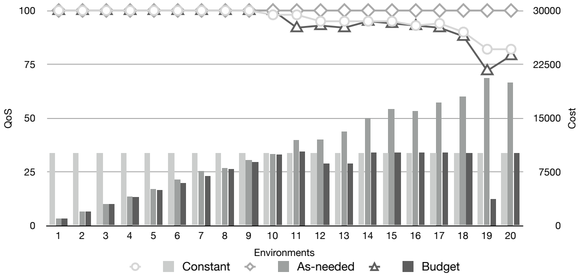
- Very detailed model: consists of 5000 lines of ABS
- Up to 20 environments are typically required to handle large query throughputs over a large number of product items

de Boer, Hähnle, Johnsen, Schlatte, Wong. *Formal Modeling of Resource Management for Cloud Architectures: An Industrial Case Study*. Proc. ESOCC 2012, LNCS 7592, Springer 2012

Case Study: Fredhopper Replication Server (2)

Parameters derived from measurements on the Java implementation

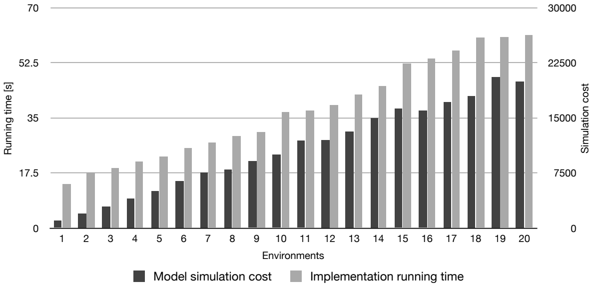
Schedule	Execution Time	Cost	Interval	Deadline
Search	34.0s	14	11	3
Business rules	2.5s	1	11	2
Data	274.9s	110	11	11



QoS as percentage of successful sessions (left scale)
and accumulated cost (right scale)

Case Study: Fredhopper Replication Server (3)

How does the accumulated cost in our model compare to the actual Java implementation?

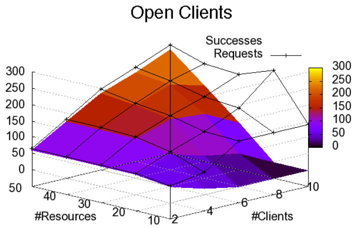


Measured execution time of the implementation (left scale)
Accumulated cost of the simulation for the as-needed policy (right scale)

The deviation roughly seems to correspond to the start-up time of JVM

Summary

- Need formal models and analysis methods for software which ranges over different deployment scenarios
- Express and compare interesting non-functional system properties
- Formal models seem fairly realistic



- Can be adapted to other resources: memory, bandwidth

Research Agenda

- Exploit the formal semantics to go beyond simulations
- SLA-aware Design by Contract: programming to interfaces
- Abstract executable models which integrate deployment, resource management, scalability, and SLA
- Tool support for analyzing SLA compliance based on scalable methods

Berndnaut Smilde: Nimbus II, 2012