CDOSim: Simulating Cloud Deployment Options for Software Migration Support

Florian Fittkau, Sören Frey, and Wilhelm Hasselbring

Software Engineering Group, Kiel University, Germany

24.09.2012
Motivation

- Migration of enterprise software to the cloud
- Many different cloud deployment options
- Simulation helps to find the best trade-off between high performance and low costs
Cloud Deployment Option (CDO)

In the context of a deployment of software on a cloud platform, a cloud deployment option is a **combination of decisions** concerning the

- selection of a cloud provider,
- the deployment of components to virtual machine instances,
- the virtual machine instances’ configuration,
- and specific adaptation strategies.
CloudSim

- CloudSim \cite{CRDRB09}
  - Cloud computing system and application simulator
  - Cloud provider perspective
  - We extended it by cloud user perspective
CloudMIG

CDOSim – Simulation Architecture
Million Integer Plus Instructions Per Second (MIPIPS)

- Measure for the computing performance of a computer / virtual machine instance
- Idea: Measure the execution time and divide by instruction count
- Example: 10 seconds for 200 million instructions results in 20 MIPIPS
- Benchmark generated from meta-model with current support for Java, C, C++, C#, Ruby, Python
**CDOSim:**
Simulating Cloud Deployment Options for Software Migration Support

Fittkau, Frey, and Hasselbring

1. Introduction
2. Context
3. Simulation
4. Evaluation
5. Related Work
6. Conclusions and Future Work

---

**MIPIPS**

```
int x = 0;

long startTime = System.currentTimeMillis();

int i = -2147483647;
while (i < 2147483647) {
    x = x + 2;
    i += 1;
}

long endTime = System.currentTimeMillis();
long difftime = endTime - startTime;
System.out.println(difftime);
System.out.println(x);
```
## MIPIPS

```java
int x = 0;
int y = 0;

long startTime = System.currentTimeMillis();

int i = -2147483647;
while (i < 2147483647) {
    x = x + 2;
    y = y + 3;
    i += 1;
}

long endTime = System.currentTimeMillis();
long difftime = endTime - startTime;
System.out.println(difftime);
System.out.println(x);
System.out.println(y);
```
## Instruction Count Overview

<table>
<thead>
<tr>
<th>Approach</th>
<th>Preconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic approach</td>
<td>1. Part of source code</td>
</tr>
<tr>
<td></td>
<td>2. Response times</td>
</tr>
<tr>
<td></td>
<td>3. MIPIPS</td>
</tr>
<tr>
<td>Static approach</td>
<td>1. Full source code</td>
</tr>
<tr>
<td>Hybrid approach</td>
<td>1. Full source code</td>
</tr>
<tr>
<td></td>
<td>2. Response times</td>
</tr>
<tr>
<td></td>
<td>3. MIPIPS</td>
</tr>
</tbody>
</table>
Dynamic Approach

- Approach: MIPIPS divided by the response time reveals instruction count
- Example: 200 MIPIPS / 0.1 seconds = 20 million integer plus instructions
Static Approach

- **Approach**: Count each instruction and convert to integer plus instruction through weight
- **Example**: Convert a double times to an integer plus instruction
Static Approach - Example

Equation for loop instruction count derivation:

\[ ic_{for\_loop} = ic_{init} + (iter_{count} \cdot (ic_{cond} + ic_{iter} + ic_{loop})) \]

Example:

```c
for (int i = 0; i < 10; i++) {
    x = i + 3;
}
```

\[ ic_{for\_loop} = 1 + (10 \cdot (1 + 1 + 1)) = 31 \]
Hybrid Approach

- Dynamic approach: Most often no data from a fully-instrumentated system is available, but the monitored data is accurate
- Static approach: Detailed insight but imprecise
- Hybrid approach combines the advantages of both
- Idea: Use dynamic analysis results for correction of static analysis results
Hybrid Approach

```java
public void method3000() { // from static: 3000 IC
    for (int i = 0; i < 1000; i++) {
        x = i + 3;
    }
}

public void method50() { // from dynamic: 50 IC
    method3000();
}
```
Hybrid Approach

1. Introduction
2. Context
3. Simulation
4. Evaluation
5. Related Work
6. Conclusions and Future Work

```java
// from hybrid: 50 IC
public void method3000() { // from static: 3000 IC
    for (int i = 0; i < 1000; i++) {
        x = i + 3;
    }
}

public void method50() { // from dynamic: 50 IC
    method3000();
}
```
Weights Per Statement

- For example, a double divide instruction takes more time than an integer plus instruction on most platforms.
- Idea: Convert double divide instruction into integer plus instruction.
- Approach: Divide MIPIPS by million double divide instructions per seconds (MDDIPS) from adapted benchmark.
- Example: $400 \text{ MIPIPS} / 100 \text{ MDDIPS} = 4$
Simulation Output

- Costs
- Response times
- SLA violations
- Rating: Rate each output from 1 (best) to 5 (worst)
Evaluation Overview

- E1: MIPIPS benchmark evaluation
- E2: Accuracy evaluation for single core instances
- E3: Inter-cloud accuracy evaluation

More evaluations in [Fit12]
Experiment Setup for E2 and E3

- Adapted JPetStore
- JMeter with Markov4JMeter
- Kieker [vHWH12] (monitoring framework)
  kieker-monitoring.net
- Eucalyptus\(^1\) and Amazon EC2
- Quantifying the relative error (RE) by comparing simulated values with measured values

\(^1\)2x AMD Opteron 2384 (8 cores), 24 GB DDR2-667 RAM
Workload

Workload Intensity

Method call rate [number of method calls/min]

Experiment time [day hour:minute]

0.0 589.1 1767.3 2945.5 4123.7 5301.9

1. Introduction
2. Context
3. Simulation
4. Evaluation
5. Related Work
6. Conclusions and Future Work
E1: MIPIPS Benchmark Evaluation – Reasonable to Other Measures

<table>
<thead>
<tr>
<th>Amazon EC2 instance type</th>
<th>MIPIPS</th>
<th>EC2 compute units per core</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1.micro</td>
<td>4.11</td>
<td>up to 2</td>
</tr>
<tr>
<td>m1.small</td>
<td>20.65</td>
<td>1</td>
</tr>
<tr>
<td>m1.large</td>
<td>142.13</td>
<td>2</td>
</tr>
<tr>
<td>c1.medium</td>
<td>148.81</td>
<td>2.5</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>235.57</td>
<td>3.25</td>
</tr>
</tbody>
</table>

Table 1: MIPIPS benchmark results for Amazon EC2
E2: Accuracy Evaluation for Single Core Instances

(a) Measured CPU utilization (b) Simulated CPU utilization

Figure 1: Average CPU utilization of allocated nodes using Eucalyptus

\[ RE_{CPU} = 29.18\% \quad RE_{InstanceCount} = 0.64\% \quad RE_{Costs} = 6.34\% \]
E2: Accuracy Evaluation for Single Core Instances

(a) Measured response times  (b) Simulated response times

Figure 2: Median of response times using Eucalyptus

\[ RE_{RT} = 24.85\% \]
E3: Inter-Cloud Accuracy Evaluation

Figure 3: Average CPU utilization of allocated nodes

(a) Amazon EC2 run  (b) Simulation for Eucalyptus  (c) Eucalyptus run

\[ RE_{CPU} = 21.60\% \quad RE_{InstanceCount} = 1.32\% \]
\[ RE_{Costs} = 1.53\% \quad RE_{RT} = 38.62\% \]
Related Work

- GroudSim \([\text{OPPF}10]\) (alternative to CloudSim)
- SLAStic.SIM \([\text{vMVHH}11]\) (performance simulator based on Palladio Component Model)
- iCanCloud \([\text{NCVP}+11]\) (cloud tool with manual application modelling)
- Cloudstone toolkit \([\text{SSS}^+08]\) (benchmark and measurement tools for Web 2.0)
- SMICloud \([\text{GVB}11]\) (framework for comparing different cloud providers)
Future Work

- Framework for parallelizing CDOSim’s simulations
- Extend elementary model for computing network costs
- Simulate further properties, e.g., memory consumption and I/O performance
- Use CDOSim for a simulation-based evolutionary optimization of CDOs
Conclusions

- CDOSim helps assessing CDO candidates and finding best suited CDO
- Three approaches for instruction count derivation
- MIPIPS and weights benchmark
- Simulation results can be used to appropriately predict costs, response times, and SLA violations of specific CDOs
- CDOSim is provided as part of our tool CloudMIG Xpress²

²http://www.cloudmig.org/
Methology

E1:
- Mean value and the standard deviation

E2 and E3:
- Quantifying the relative error (RE) by comparing simulated values with measured values

\[ re(t) = \frac{|m(t) - s(t)|}{m(t)}, \ m(t) \neq 0, \ t \in T \]

\[ RE = \frac{\sum_t re(t)}{|T|} \]

\[ OverallRE = \frac{RE_{CPU} + RE_{InstanceCount} + RE_{Costs} + RE_{RT}}{4} \]
References

Rodrigo N. Calheiros, Rajiv Ranjan, César A. F. De Rose, and Rajkumar Buyya.

Sören Frey and Wilhelm Hasselbring.
The cloudmig approach: Model-based migration of software systems to cloud-optimized applications.

Sören Frey, Wilhelm Hasselbring, and Benjamin Schnoor.
Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms.
doi: 10.1002/smr.582.

Florian Fittkau.
Simulating Cloud Deployment Options for Software Migration Support.


Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer.

Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, O Fox, and David Patterson.
Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0.

André van Hoorn, Jan Waller, and Wilhelm Hasselbring.
Kieker: A framework for application performance monitoring and dynamic software analysis.

Robert von Massow, André van Hoorn, and Wilhelm Hasselbring.
Performance simulation of runtime reconfigurable component-based software architectures.