A Methodology and a Tool for QoS-Oriented Design of Multi-Cloud Applications

Giovanni Paolo Gibilisco
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano

Supervisor: Prof. Danilo Ardagna
Co-supervisor: PhD. Michele Ciavotta
Tutor: Prof. Carlo Ghezzi

A thesis submitted for the degree of

Doctor of Philosophy

05 Feb. 2015
Abstract

This work focuses on the support of the development of multi-cloud enabled applications with Quality of Service (QoS) guarantees. It embraces the model driven engineering principles and aims at providing development teams with methodologies and tools to assess the expected QoS of their applications early in the design stages. To do so we adopt and enrich different component based and UML-like modeling technologies like the Palladio Component Model and MODACloudML extending them in order to determine an optimized deployment in multi-cloud environments by introducing a new cloud specific meta-model. The integration of the new meta-model into state of the art modeling tools like Palladio Bench or Modelio allows software architects to use well known modeling approaches and specify a cloud specific deployment for their applications. In order to ease the portability of both the model and the application the meta-model uses three abstraction levels. The Cloud enabled Computation Independent Model (CCIM) allows to describe the application without any reference to specific cloud technologies or providers; the Cloud Provider Independent Model (CPIM) adds the specificity of some cloud technologies introducing concepts like Infrastructure and Platform as a Service (IaaS/PaaS) but still abstracts away the specificity of each particular provider; the Cloud Provider Specific Model (CPSM) adds all the details related to a particular cloud provider and the services offered allowing to automatize the deployment of the application and generate performance models that can be analyzed to assess the expected QoS of the application. High level architectural models of the application are then transformed into a Layered Queuing Network performance model that is analyzed with state of the art solvers like LQNS or LINE in order to derive performance metrics. The result of the evaluation can be used by
software architects to refine their design choices. Furthermore, the approach automates the exploration of deployment configurations in order to minimize operational costs of the cloud infrastructure and guarantee application QoS, in terms of availability and response time. In the IaaS context, as an example, the deployment choices analyzed by the tool are the size of instances (e.g. Amazon EC2 m3.xlarge) used to host each application tier and the number of replicas for each hour of the day. The problem of finding the optimal deployment configuration has been analyzed from a mathematical point of view and has been shown to be NP-hard. For this reason a heuristic approach has been proposed to effectively explore the space of possible deployment configurations. The heuristic approach uses a relaxed formulation based on M/G/1 queuing models to derive a promising initial solution that is then refined by means of a two level hybrid heuristic and validated against the LQN performance model. The proposed methodology has been validated by two industrial case study in the context of the MODAClouds project. A scalability and robustness analysis has also been performed and shows that our heuristic approach allows reductions in the cost of the solution ranging from 39% to 78% with respect to current best practice policies implemented by cloud vendors. The scalability analysis shows that the approach is applicable also to complex scenarios, with the optimized solution of the most complex instance analyzed being found in 16 minutes for a single cloud deployment and in 40 minutes for a multi-cloud scenario.
# Contents

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>Nomenclature</td>
<td>vii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 State of the art</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Modeling Approaches</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Other approaches for Designing Applications with QoS Guarantees</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Deployment Selection Approach Classification</td>
<td>20</td>
</tr>
<tr>
<td>3 The Model Driven Approach</td>
<td>21</td>
</tr>
<tr>
<td>3.1 Introduction to Model Driven Engineering</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Modeling the Application</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Modeling the Cloud</td>
<td>31</td>
</tr>
<tr>
<td>3.4 Modelling Cloud Specific Concepts</td>
<td>33</td>
</tr>
<tr>
<td>3.4.1 Cloud Provider Independent Model</td>
<td>34</td>
</tr>
<tr>
<td>3.4.2 CPIM for Private Cloud specification</td>
<td>40</td>
</tr>
<tr>
<td>3.4.3 Cloud Provider Specific Model: The Windows Azure use case</td>
<td>41</td>
</tr>
<tr>
<td>3.5 Modeling the Quality</td>
<td>52</td>
</tr>
<tr>
<td>4 Design Methodology and Optimization Approach Overview</td>
<td>54</td>
</tr>
<tr>
<td>4.1 QoS-Oriented Design Methodology</td>
<td>55</td>
</tr>
<tr>
<td>4.1.1 Definition of the characteristics of the candidate cloud services</td>
<td>56</td>
</tr>
</tbody>
</table>
CONTENTS

4.1.2 Modelling ................................................. 57
4.1.3 Reiteration ................................................ 58
4.2 Hybrid optimization architecture ............................. 59

5 Optimization Problem Formulation ......................... 64
  5.1 Problem definition ......................................... 65
  5.2 Analytic Formulation ...................................... 66
  5.3 Bi-level formulation and NP-Hardness .................... 72

6 Determining an Initial solution ......................... 75
  6.1 Generation of an initial solution ........................... 75

7 Meta-heuristic Approach ...................................... 83
  7.1 Main Algorithm ............................................ 84
    7.1.1 Solution Evaluation .................................. 91
  7.2 MakeFeasible .............................................. 91
  7.3 ScaleLS ................................................... 94
  7.4 TSMove .................................................. 99
  7.5 Restart ................................................ 102
  7.6 OptimizeWorkload ........................................ 104

8 Approach Evaluation ...................................... 107
  8.1 Industrial Case Studies ................................... 109
    8.1.1 The Constellation Platform ........................ 109
      8.1.1.1 Initial analysis .................................. 109
      8.1.1.2 Reiteration ...................................... 112
    8.1.2 The ADOxx platform ................................ 115
  8.2 Scalability Analysis ....................................... 123
    8.2.1 Design of Experiment ................................. 123
    8.2.2 Scalability Results ................................ 125
  8.3 Comparison with best practice heuristics .................. 134
  8.4 Initial solution Evaluation ................................ 137
    8.4.1 Quality Evaluation .................................. 138
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9  Conclusions</td>
<td>142</td>
</tr>
<tr>
<td>Bibliography</td>
<td>145</td>
</tr>
</tbody>
</table>
List of Figures

3.1 An instance of a CCIM model .................................. 26
3.2 Service Assembly diagram of the SVN Architecture of Constellation . 27
3.3 Orchestration model ............................................. 28
3.4 Usage Model .................................................... 29
3.5 An instance of a CPIM model ................................. 30
3.6 Deployment Model at CPIM level .............................. 32
3.7 Deployment Model at CPSM level .............................. 33
3.8 Cloud Meta-Model - General Concepts. ....................... 35
3.9 Cloud Meta-Model - Locations ................................ 37
3.10 Cloud Meta-Model - IaaS Cloud Resource .................... 38
3.11 Cloud Meta-Model - Cloud Platform Service ................. 39
3.12 meta-model for the specification of private clouds ............. 42
3.13 Azure CPSM - General overview. ............................ 43
3.14 Azure CPSM - Blob and Drive Storage overview. ............. 46
3.15 Azure CPSM - SQL Database and Table overview. ............. 47
3.16 Azure CPSM - Web and Worker Roles overview. ............. 48
3.17 Azure CPSM - Virtual Machine overview. ..................... 49
3.18 Azure CPSM - Virtual Machine details. ....................... 50
3.19 Azure CPSM - Virtual Machine Instance details. .............. 51
3.20 Azure CPSM - Virtual Machine Medium Instance example. ... 51
3.21 QoS Constraints ................................................. 53

4.1 Main modeling steps in the SPACE4Cloud approach. .......... 55
4.2 SPACE4Cloud architecture. .................................... 60
# LIST OF FIGURES

6.1 Call chain of functionalities ........................................ 80
6.2 DTMC derived by Figure 3.3 ................................. 81
7.1 Tabu Search behavior in presence of local optima ............. 85
8.1 Base workload for the Case Study analysis.................. 110
8.2 Optimization Result............................................. 111
8.3 Analysis of the case study *Constellation* and *Conference* architectures 112
8.4 Conference Architecture of Constellation .................. 113
8.5 Conference Architecture usage model ....................... 114
8.6 Cost analysis of the two architectures under evaluation (daily costs for VMs usage). ................................. 115
8.7 ADOxx Architecture ............................................ 117
8.8 Orchestration models .......................................... 118
8.9 ADOxx Usage Model ........................................... 119
8.10 ADOxx Deployment CPIM ...................................... 120
8.11 ADOxx Deployment CPSM ....................................... 121
8.12 Scalability Analysis Case Study, distribution of calls within the system components. ................................. 124
8.13 Distribution of time spent during the optimization in the main phases . 126
8.14 Scalability Analysis with a single candidate provider, time breakdown ................................. 131
8.15 Scalability Analysis with a two candidate providers, time breakdown . 132
8.16 Scalability Analysis with a three candidate providers, time breakdown ................................. 133
8.17 Comparison with heuristics Heur$_{60}$ and Heur$_{80}$ .............. 136
8.18 MILP optimization time varying the number of tiers and classes of requests ........................................ 138
8.19 Comparison of solution costs found by SPACE4Cloud in the optimization process starting from the MILP initial solution and the Heur$_{60}$ solution. ................................. 139
Chapter 1

Introduction

One of the most pervasive changes happened in recent years in the IT world is the appearance on the scene of cloud computing. The main feature of this new computing paradigm is its ability to offer IT resources and services in the same way fresh water or electric power is offered, as a utility. In a traditional environment in order to make use of a software system to address some business needs a company would need to acquire and manage the hardware infrastructure, different software stacks and, in many situations, develop their own software. Cloud computing changes this paradigm by offering all these elements as services that the user can acquire and release with high flexibility.

Cloud providers offer a variety of IT resources using the “as a Service” paradigm. Complex software systems that require multiple application stacks and different hardware resources can now be acquired, entirely or in parts, in matter of minutes. Hardware resources like computation nodes, storage space or network capacity are offered as Infrastructure as a Service (IaaS), software stacks that allows application developers to run their own code are offered as Platform as a Service (PaaS), finally, entire software system that can be directly used to provide some business value without the need of developing a new system are offered as Software as a Service (SaaS).

Since the early appearance of this technology in the market, many companies have decided to evolve their own infrastructure in order to embrace this new paradigm and the offering of cloud services, as well as the number of cloud providers, has grown quickly [38].

There are many advantages introduced by the adoption of the cloud technology,
among all one of the most important is the flexibility in the acquisition and decommission of software systems and the underlying infrastructure. This advantage is due to the ability of cloud providers to offer an almost unlimited amount of resources and a pricing policy that allow application developers to avoid long term commitments and pay only for resources they actually use. Another key advantage brought by the adoption of the cloud paradigm is the shift of responsibility in the management of the portion of the software system that is acquired from the cloud provider. If, for example, a company decides to decommission its own physical infrastructure in favor of a new infrastructure offered by a cloud provider all the maintenance operations required by the hardware and some software systems, like OS acquisition and update, are delegated to the cloud provider allowing the internal IT team of the company to focus on tasks that provide more value for the company.

Delegating management responsibility of part of the infrastructure to a third party, in this case a cloud provider, comes inevitably with a loss of control on the entire infrastructure. This change creates some new challenges for teams that were used to build entire software systems from the ground up. When faced with the selection of a cloud service the application developer has to take into consideration many new characteristics that he/she was probably not considering before. The wide variety of similar services, the lack of interoperability between APIs of services offered by different cloud providers and the lack of specific training for developers are just a few of the new challenges that the IT staff of a company has to face when considering a migration to a cloud infrastructure. Furthermore, the loss of control on the infrastructure exposes users to the variability of QoS offered by cloud providers. Usually providers address this issue by providing generic Service Level Agreements (SLAs) specifying their expected QoS level and providing discounts on future usage of their services in case the specified QoS is not met. Amazon EC2 SLA, for instances provides an availability of 99.95% of up-time in a month for VMs and in case this availability level is not met users are granted a discount on 10% on service cost. In many situations such a discount is not comparable to the possible loss generated by the downtime of the application. Many examples of cloud outages like the ones happened recently to Google Cloud ¹ or Microsoft Azure ² shows that availability concerns play a major factors in

¹https://status.cloud.google.com/incident/compute/15045
moving a critical application to the cloud environment.

A solution to this problem comes from the wide variety of similar cloud services offered by other providers. If, as an example, the software architect thinks that the application under development is critical and requires an availability of 99.999% (also called 5-nines availability) he/she could replicate the application on two cloud providers, say Amazon AWS and Microsoft Azure, obtaining the required availability. Moreover, the use of multiple providers might allow the application developer also to redistribute the incoming workload in order to exploit differences in pricing in order to reduce the operational costs.

The contributions of this thesis are a methodology and a tool, to help software developers to build applications capable of exploiting the cloud infrastructure. In particular the work presented in this thesis tries to simplify the development process by providing software architects with a *meta-model* in order to describe possible deployments of their application over the cloud infrastructure. We then automate the QoS and cost analysis of such deployments in order to provide software architects with feedback on their deployment choices. Finally, we automate the generation of possible deployment configurations, possibly on multiple cloud providers, in order to minimize infrastructural costs and guarantee a desired QoS level.

Our work embraces the Model Driven Engineering (MDE) paradigm and makes use of models of the application specified at different levels of abstractions. Allowing the software architect to start by building simple component based models in a UML-like notation and then refine them adding information related to the desired cloud environment allows the development team to keep the focus on the functionality of the system their are building and delegate some of the architectural choices to the tool we have developed and integrated in the modeling environment.

Our approach targets the development team and in particular software architects, application developers and domain experts. The modeling paradigm that we embraced allow separation of concerns between these figures involved in the software development process. In the reminder of this thesis we will refer to this actors as *users* of our tool. In contrast the users of the cloud applications developed by using our approach and deployed on a cloud environment are mentioned as *final users* or *end users*. When the use of these terms might generate ambiguity we will speak directly of software architects or application developers.
The use of state of the art performance models and tools allows to provide to the development team estimation on the expected QoS of the system under development, in order to take informed decisions and adapt the design of the system early in the design stages avoiding complex and expensive re-factoring of the application.

The proposed approach allows designers to annotate theirs models with requirements related to the QoS of the application, like the expected response time of certain functionality or the minimum availability of the system and delegates to the tool the task of finding a deployment plan capable of fulfilling such constraints.

The deployment optimization strategy proposed in this thesis explores a huge and complex space of possible configurations by assigning to each component described in the model of the application a particular cloud service and analyze the behavior of the entire application in order to see if a particular choice of cloud services is capable of fulfilling the constraints. This search process takes also in to consideration the cost of the deployment solution and tries to minimize it.

The scientific literature shows some similar approaches that try to automate deployment decisions on component based systems but, to the best of our knowledge, this is the first approach that targets directly multi-cloud environments. In [49], Koziolek shows that due to the increasing size and complexity of software systems, architects have to choose from a combinatorially growing number of design and deployment alternatives. Different approaches have been developed to help architects explore this space with the help of automated tools like [10, 25] or [23, 49]. These approaches are presented in Chapter 2 help developers to analyze different design choices but do not address directly the cloud environment.

We argue that the problem becomes significantly more complex when considering the cloud services that can be employed for the execution of the application components. Traditionally the allocation problem has been considered independently from the design of the application but the possibility of exploiting different cloud services for different parts of the application has an impact on how the entire system works and makes the deployment problem even more relevant in the design phase.

If we consider even the simple example of a web application deployed on a single tier, we need to decide if we want to use a PaaS service to host our application code or to directly manage the platform used to run our code, say a Tomcat server. This choice directly affects the design and the development of the application. If we choose
to manage directly a Tomcat instance and deploy it on a Virtual Machine (VM) offered by Amazon, we still need to decide which type of VM we need to use and how many number of replicas of this machine according to the expected number of end user of our system. Using a high number of cheap VMs, like the m3.large in order to cope with a variable workload might seem a good strategy but the software stack needed to run our application might required more resources and, in this case using a smaller number of more powerful instances, like the c4.3xlarge, might be more convenient.

In a multi-cloud scenario this problem becomes even more complex because, beside making these decisions for both providers, we also have to define how the incoming workload is split among the providers. Since the performance and the price of resources offered by cloud providers might change with the time of the day, this problem is very dynamic and requires some automation to help the designer to generate possible configurations.

When we deal with a more realistic and complex application the development team is faced with deployment decisions and analyzing all the possible alternatives is a daunting tasks that calls for automation. The tool developed during this work, called SPACE4Cloud (System Performance and Cost Evaluation for Cloud), allows to automate the exploration of these design alternatives. SPACE4Cloud has been developed in the context of the MODAClouds FP7 IP European project and constitutes one of the core tools of the MODAClouds IDE design time environment. Our approach takes into consideration QoS constraints that predicate on the response time, both on average and percentiles, constraints on the availability of the application and service allocation constraints. By service allocation constraints we mean those constraints that are related to the type of technology chosen to build the application and include minimum requirements on some characteristics of the cloud services required to host specific components, e.g., minimum amount of memory or cores, or limitations on the scalability of some service. Our approach differs from those already available in the literature, since it targets directly the cloud environment taking into consideration some peculiar features. Cloud environments are naturally shared by multiple users, the use of a shared infrastructure might lead to contention problems. To address this kind of behaviors we make use of a performance analysis tool called LINE that take into consideration variability in the characteristics of the processing resources by using

\[^{1}\text{www.modaclouds.eu}\]
a statistical characterization (via Random Environment [29]). Web applications, like those developed in a cloud environment are also dynamic and the number of end users and the price of the cloud resources changes during the day. In many applications, the incoming workload shows a daily pattern, for this reason we introduce a time-dependent workload profile over a 24-hour time horizon, which leads to the solutions of 24 intertwined capacity allocation problems.

We first introduce the modeling paradigm proposed to apply the MDE principles in the context of cloud application development in Chapter 3. We also present an industrial case study that is used later on in the evaluation of the approach and throughout the thesis to clarify both modeling concepts and the optimization approach.

We then introduce the general design methodology and optimization strategy used to tackle the problem along with the architecture of the tool in Chapter 4. We then formalize the problem in Chapter 5 and we show it is equivalent to a class of NP-hard problems. These initial part of the work has been submitted for publication in IEEE Transactions on Software Engineering. In Chapter 6 we use a simplified performance model and a relaxed formulation of the problem in order to quickly derive a promising initial solution for the heuristic algorithm and in Chapter 7 we describe in details the main optimization algorithm used to explore the design space and derive the optimized deployment configuration. Details on the impact of the initial solution on the entire optimization procedure have been published in [17].

To validate our approach we have used two industrial case studies that show how software architects can benefit from using the early QoS analysis and deployment optimization provided by our work. We have also inspected how the complexity of the application under development affects the cost of the solutions obtained and the time required to execute the optimization with a scalability analysis, the results of this study is reported in Chapter 8. We compared our heuristic approach against common used threshold based heuristic that keep the utilization of the system below 60% or 80%. Using our heuristic we found optimized solutions with cost reductions ranging from 39% to 78%. The analysis also shows that the algorithm is both scalable and robust, the optimized solution for the most complex case was found in 36 minutes in a single cloud scenario and 42 minutes in multi-clouds. Robustness has been analyzed by repeating several times the optimization procedure during the scalability analysis and in the worst case the standard deviation of the time spent in the optimization is 18%.
of the average execution time and the standard deviation of the cost of the solution is within 6% of the average solution cost. For what concerns the correctness of the QoS estimation with respect to the real model we relay on the extended literary work on performance prediction based on LQN starting from [36] that analyzes the accuracy of the QoS prediction for LQN models. With respect to the characterization of the parameters of the performance model used to evaluate application QoS we relay again on previous works like the one by Casale et al. [68] that presents several techniques to estimate application demands.

A discussion of the results achieved and an outline of future work are drawn in Chapter 9.
Chapter 9

Conclusions

“That’s all Folks!”

Porky Pig

In this work we presented an approach that tries to simplify the process of migrating an application to the cloud by providing a methodology and a tool to support development teams in building new applications capable of running in a multi-cloud environment. We proposed a meta-model that describes cloud services and integrated it with well established modeling tools like Palladio and Modelio in order to allow application architects to specify configurations in cloud environments. We then automated the process of evaluating the QoS of the deployment configuration specified by the software architect allowing her/him to gain valuable insights on how the design reacts to different working conditions (e.g., variable incoming workload). This ability empowers the application architects to follow MDE principles and perform QoS and cost analyses early in the design stage allowing prompt modification of the architecture to tailor it better to the runtime environment. This ability has been shown by the first industrial case study by Softeam in which an early analysis of an initial architecture model revealed its inability to gracefully scale and support higher workloads. This discovery led to a re-design of part of the architecture leading to a system that could exploit better the scalability feature offered by cloud environments. A second industrial case study used the tool to evaluate different application deployments in a multi-cloud scenario.

We then focused on helping the application architect not only in the discovery of potential issues in the architecture or in a particular deployment configuration, but also in deriving an optimized deployment that minimize the cost of using cloud services and provide QoS guarantees at the same time. To derive this configuration we designed an optimization heuristic that
effectively explores a wide space of possible configurations. We first formalized the problem from a mathematical point of view and showed it to be NP-hard. We then used M/G/1 queuing network models to derive a closed form formulation of the application response time and used this model to solve a relaxation of the original problem and derive a promising initial solution. This solution is then modified by our hybrid heuristic that makes use of a more accurate performance model, i.e. LQN models, to evaluate the feasibility of the application against user defined constraints.

We evaluated the applicability of the proposed approach to complex models by means of a scalability analysis that showed how the solution derived by means of our heuristic algorithm outperform those derive by policies currently used by practitioners providing an average reduction in the cost of the deployment around 55%. The scalability analysis also showed that the approach can be effectively applied to complex problems, since the optimized solution was obtained in around 40 minutes in for the most complex model considered.

The main threat to the validity of our approach is the lack of accurate data on the performance of cloud services. The optimization approaches uses a Resource Database to update the performance model with the characteristics of the cloud resource under analysis, as show in Section 4.2. While some of the information stored in this database are provided publicly by cloud providers (e.g. the cost of using such a resource or the number of cores of a particular VM type), other parameters are unknown and have to be estimated by benchmarking such resources. Furthermore the service offer of main cloud providers change very frequently both in terms of performance upgrades or cost reductions. Since a complete benchmark campaign was not feasible, we decided to integrate in our approach the results of the ARTIST\(^1\) European project which provide benchmarking information of many cloud resources.

From the optimization point of view, we allowed users to define constraint on the response time of the application. Current best practices use constraints on resource utilization since as long as the utilization is low the response time does not change significantly. Using directly the response time might be effective only if the constraint is so high that high utilization can be tolerated. To overcome this limitation we added the possibility to take into account resource utilization constraints as well so that if both type of constraints are defined, one will dominate the other. Finally the user can specify response time constraint on a functionality offered to the end user of the application and utilization constraints on part of the resources used to provide such a functionality. In such a scenario high utilization might be tolerated on some resourced involved in the processing of the user request while key components might be kept under a utilization constraint.

\(^1\)http://www.artist-project.eu/collaboration
Our approach has been particularly tailored to support the analysis and optimization of common web based applications but the modeling approach and the optimization techniques presented in this thesis can be extended to address different deployment problems.

An extension that we have considered takes into account the use of a mixed cloud environment in which a private infrastructure is used to process most of the workload and, when the private infrastructure is not capable of accepting more users, a public cloud is used to replicate the entire system.

In our work we considered application deployment on multiple clouds in which the entire system has been replicated on all the available providers. Another possible research line can consider a more flexible deployment solution where application tiers are deployed on different cloud providers. The analysis of such a deployment configuration should take into consideration the peculiarity of the technology used to interconnect the application tiers, which we expect will be enabled by progress in the area of network design. The deployment configuration we considered in this thesis allows to exploit high performance networks available in cloud providers datacenter and avoid the increased complexity in the analysis.

Recent years showed the appearance of Big Data frameworks and applications. These applications perform complex analysis of high volumes of data in order to extract insights with business value. The complexity of the frameworks used to process this kind of data and the size of the infrastructure required to perform such analysis make the cloud environment the default choice for many companies that want to exploit this new technology. The approach proposed in this thesis could be adapted to take into consideration the peculiarity of these applications and frameworks by the use of a more tailored performance model.

If we consider current technology trends, the raise of container based hosting services like Amazon EC2 Container Service (ECS)\textsuperscript{1} or Google Container Engine\textsuperscript{2} eases the use of component based approaches in the development of cloud applications. In this context an optimization technique that derive container characteristics like the container size could greatly help architects to speed up the development and testing of their application architectures.

Finally, adding the link between the runtime application execution and the models built at design time by the use of automated tools and an appropriate monitoring technology will enable a full DevOps approach. The main idea is to use real monitoring data to refine application performance parameters (e.g., service demands, number of users at peak, switch probabilities in end user behaviour, etc.). In this vision development decisions and operational aspects are considered together and the barrier between design time and runtime can be dissolved.

\textsuperscript{1}https://aws.amazon.com/it/blogs/aws/cloud-container-management/
\textsuperscript{2}https://cloud.google.com/container-engine/
Bibliography

[4] Omg model-driven architecture. 11


effective multi-objective software architecture optimization. In Proceedings of the 2Nd
International Workshop on the Quality of Service-Oriented Software Systems, QUASOSS
’10, pages 1:1–1:6, New York, NY, USA, 2010. ACM. 16


[64] OMG. A uml profile for marte: Modeling and analysis of real-time embedded systems,
2008. 9

[65] Mohamed Ouzineb, Mustapha Nourelfath, and Michel Gendreau. Tabu search for the
redundancy allocation problem of homogenous series-parallel multi-state systems. Rel.

[66] Trevor Parsons and John Murphy. Detecting performance antipatterns in component

Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th Inter-
national Symposium on, pages 409–416, Sept 2013. 22, 62

2015. 7

Press. 10

Model Driven Engineering Languages and Systems, pages 46–60, 2010. 19

[71] SOFTEAM. Modelio. The open source modeling environment. https://www.modelio.org,
2015. 9, 23

[72] T Stützle. Local search algorithms for combinatorial problems. Darmstadt University of

2002. 20


