Institute of Architecture of Application Systems

Design Support for Performance Aware Dynamic Application (Re-)Distribution in the Cloud

Santiago Gómez Sáez, Vasilios Andrikopoulos, Frank Leymann, Steve Strauch

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{gomez-saez, andrikopoulos, leymann, strauch}@iaas.uni-stuttgart.de

BibTeX:

```
@article {ART-2014-12,
  author = {Santiago G{\'o}mez S{\'a}ez and Vasilios Andrikopoulos and Frank Leymann and Steve
Strauch},
  title = {{Design Support for Performance Aware Dynamic Application (Re-)Distribution in the
Cloud}},
  journal = {IEEE Transactions on Service Computing},
  publisher = {IEEE Computer Society},
  pages = {1--14},
  type = {Article in Journal},
  month = {December},
  year = {2014},
  language = {English},
  cr-category = {D.2.11 Software Engineering Software Architectures,  C.2.4 Distributed
Systems,  D.2.8 Software Engineering Metrics},
  contact = {Santiago G{\'o}mez S{\'a}ez: gomez-saez@iaas.uni-stuttgart.de},
  department = {University of Stuttgart, Institute of Architecture of Application Systems},
}
```


Universität Stuttgart
Germany

Design Support for Performance Aware Dynamic Application (Re-)Distribution in the Cloud

Santiago Gómez Sáez, Vasilios Andrikopoulos, Frank Leymann, Steve Strauch

Abstract—The wide adoption of the Cloud computing paradigm by many different domains has increased both the number and type of available offerings as a service, e.g. Database-as-a-service (DBaaS) or Platform-as-a-Service (PaaS), and with them the possibilities in deploying and operating an application partially or completely in the Cloud. The need for providing design support to application developers in this environment is the focus of this work. Toward this goal, in the following we first scope the discussion on the persistence layer of applications and we investigate the effect of different deployment scenarios on the performance of the application over time. Based on the results of this analyses we then propose an application (re-)distribution design support process, which we evaluate empirically by means of a well-known application. The results of this evaluation both highlight the strengths of our proposal, and at the same time, provide a clear path for the following steps in our work.

Index Terms—Synthetic Workload; Benchmark; Cloud Application Distribution; Application Deployment; Relational Database; Cloud Services Selection; TPC; MediaWiki

1 Introduction

In the last years the Cloud computing paradigm has been widely adopted by different domains, both in industry and research. This has led to an increase in the number of applications that are partially or completely running in the Cloud. The number of non hardware virtualization-oriented services (essentially, Virtual Machine-as-a-Service, or VMaaS, offerings) has also increased with the successful introduction of offerings like Database-as-a-Service (DBaaS) or Platform-as-a-Service (PaaS) from major Cloud providers. It has become thus possible to host only some of the application components off-premise (in the Cloud), e.g. its database, while the remaining application remains on-premise [1]. With such a wide space of possible deployment and runtime viable combinations, application developers can distribute or replace application components within or with a wide variety of Cloud offerings. Standards like TOSCA[1], for example, allow for the modeling and management of application topology models in an interoperable and dynamic manner, further supporting the application distribution capabilities, potentially even in a multi-Cloud environment. However, such technological approaches lack support for assisting the application developer towards an efficient selection of Cloud offerings for a partial or complete deployment of the application components.

This investigation leverages the vast amount of opportunities provided by such a technological landscape towards developing the means that allow for the dynamic deployment and re-deployment of application components across service providers and solutions in order to cope with evolving performance and resource demands. There are two fundamental observations in this effort that are going to be discussed in more length during the rest of the paper. Firstly, the distribution of the application topology in the Cloud has a severe effect on the performance of the application — however it is not always obvious whether this effect is beneficial or detrimental. Secondly, a real application workload typically fluctuates over time, leading to an evolution of its resources demands, which may or not be fully available in the underlying virtualized infrastructure of the utilized Cloud offering. Hence, the application topology may have to be adapted in order to cope with such resource and performance fluctuation demands.

For the scope of this paper we first focus on the *persistence layer* of applications [2] and study the effect on performance of different application topology distributions of a sample application for different generated workloads. A presentation and analysis of our experimental results is discussed, based on which we design a dynamic application distribution support process aimed at performance optimization, which we then evaluate using a well-known application. The contributions of this work can therefore be summarized as follows:

1) a workload characterization and performance variation analysis focusing on the application persistence layer by using an established benchmark (TPC-H[2]) and well-known Cloud offerings,
2) the generation and performance evaluation of generated synthetic workloads for different deployment topologies of the application persistence layer based on this analysis,
3) the design of a process which supports application designers in optimizing the distribution of their application across Cloud and non-Cloud solutions in a dynamic manner, and,
4) the evaluation of our approach using the MediaWiki[3] application, with real workload and data.

• *Santiago Gómez Sáez, Vasilios Andrikopoulos, Frank Leymann, and Steve Strauch are with the Institute of Architecture of Application Systems, University of Stuttgart, Germany.*
E-mail: {gomez-saez, andrikopoulos, leymann, strauch}@iaas.uni-stuttgart.de

Manuscript received October 31, 2014; revised -.

1. Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html

2. TPC-H Benchmark: http://www.tpc.org/tpch/

3. MediaWiki: http://www.mediawiki.org/wiki/MediaWiki

The remaining of this paper is structured as follows: Section 2 summarizes relevant concepts and motivates further investigations. Section 3 presents our experiments with respect to the persistence application layer, and discusses the most important findings. A process to support the dynamic distribution of the application in the Cloud is proposed in Section 4. Section 5 presents an empirical and conceptual evaluation of our process-based approach. Finally, Section 6 summarizes related work and Section 7 concludes with some future work.

2 Background

The deployment of an application in the Cloud regularly requires the realization of preliminary compliance tasks. These often involve specifying the required underlying resources, cost calculations, or even architectural or realization adaptations. Towards achieving the desired performance, such tasks should incorporate performance awareness. The migration of the different layers of an application to the Cloud is analyzed in [1], where multiple migration types are categorized and their corresponding application adaptation requirements are identified. In [3] a migration method and tool chain based on application model enrichment for optimally distributing the application across one or multiple Cloud providers is presented. This work targets the challenge of optimizing the application layers' distribution in the Cloud based on its workload and expected performance. Moreover, internal or external parameters produce variations of the application workload. For example, an online Web store workload is increased at certain time periods, e.g. before the Christmas season, which may generate unnecessary monetary costs and/or performance degradation. An analysis of this problem can be conducted from two different perspectives, based on the *Cloud Consumer* and the *Cloud Provider* interests. On the one hand, the Cloud consumer aims to maximize the resource usage while minimizing the incurred monetary costs. On the other hand, the Cloud provider's goals are associated with the utilization of virtualization and multi-tenancy techniques to minimize operational costs while ensuring isolation between the loads generated by each Cloud consumer. In this context, our goal is to provide the necessary methodology and artifacts to *analyze workload fluctuations over time and dynamically (re-)distribute the application layers* towards bridging the gap produced by the existing conflict of interests between Cloud consumers and Cloud providers.

According to several investigations [4], [5], [6], [7], [8], two approaches for analyzing the application workload behavior and evolution can be identified: *top-down* and *bottom-up*. In the former, the application workload is characterized, and the application behavior model is derived before or during the deployment of the application. As discussed in [6], the understanding of the application workload is mandatory in order to achieve efficiency. Workload and statistical analysis are often combined to derive the application workload behavior model. However, the top-down analysis approach is restricted to handle the workload evolution over time. Bottom-up approaches address this deficiency with the help of resource consumption monitoring techniques and performance metrics. The analysis of the cyclical aspect of multiple workloads can ease the application workload characterization, prediction, and placement operations [5]. Furthermore, the analysis and generation of application performance models for application workloads in the Cloud can be used to ease capacity management operations and predict the workload behavior to determine the most cost-effective resource configuration [7].

In this work we therefore propose to consolidate the top-down and bottom-up application workload analysis approaches over time in order to proactively satisfy application demands by dynamically (re-)adapting its topology. Toward this goal, in this paper we focus on the application persistence layer. For this purpose, we analyze the application performance under different deployment topologies, using the TPC-H benchmark as the basis to generate application workloads with different characteristics.

3 Experiments

3.1 Experimental Setup

The experiments discussed in the following emulate the behavior of an application which is built using the three layers pattern (*presentation*, *business logic*, and *data*, i.e. persistence) proposed in [2]. We first generate 1GB of representative application data using the TPC-H Benchmark. Apache JMeter 2.9[4] is then used as the application load driver to emulate the application business logic layer, using the set of 23 TPC-H generated SQL queries as the load. The following infrastructures are used for distributing the application business logic and persistence layers:

- an on-premise virtualized server on 4 CPUs Intel Xeon 2.53 GHz with 8192KB cache, 4GB RAM, running Ubuntu 10.04 Linux OS and MySQL 5.1.72,
- an off-premise virtualized server (IaaS) hosted in the Flexiscale service[5] consuming 8GB RAM, 4 CPUs AMD Opteron 2GHz with 512KB cache, and running Ubuntu 10.04 Linux OS and MySQL 5.1.67,
- an off-premise virtualized server (IaaS) Amazon EC2[6] *m1.xlarge* instance hosted in the EU (Ireland) zone and running Ubuntu 10.04 Linux OS and MySQL 5.1.67,
- and an off-premise Amazon RDS[7] DBaaS *db.m1.xlarge* database instance hosted in the EU (Ireland) zone and running MySQL 5.1.69.

We create three distribution scenarios, with the application data 1) in the MySQL on-premise, 2) on the DBaaS solution, and 3) in the MySQL on the IaaS solutions. The load driver remains in all cases on-premise. The application persistence layer performance is measured by normalizing the throughput (Req./s) across 10 rounds on average per day for a period of three weeks in the last quarter of 2013.

3.2 TPC-H Workload Characterization & Distribution Analysis

The combination of top-down and bottom-up techniques can benefit the evaluation and analysis of the application workload and behavior over time. For this purpose, using the first distribution scenario (application data on-premise) we analyze the relationship between the database schema and the access count on each database table for the initial

4. Apache JMeter: http://jmeter.apache.org
5. Flexiscale: http://www.flexiscale.com
6. Amazon EC2: http://aws.amazon.com/ec2/
7. Amazon RDS: http://aws.amazon.com/rds/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2014.2381237, IEEE Transactions on Services Computing

TABLE 1: TPC-H Workload Analysis.

Query	Accessed Tables	Subqueries	Total Logical Evaluations	Throughput (Req./s)				Retrieved Data (B)	Category ID
				On-Premise	IaaS		DBaaS		
					Flexiscale	AWS EC2			
$Q^{(1)}$	1	0	1	0.03425	0.03396	0.04115	0.03817	538	CH
$Q^{(2)}$	5	1	13	0.07927	0.14884	0.07413	3.03260	15857	CH
$Q^{(3)}$	3	0	5	0.08687	0.11733	0.08446	0.31185	376	CH
$Q^{(4)}$	2	1	5	0.53950	0.73922	0.54244	0.94903	105	CL
$Q^{(5)}$	6	0	9	0.01148	0.02014	0.01377	0.33484	130	CH
$Q^{(6)}$	1	0	4	0.20583	0.21355	0.22450	0.28261	23	CL
$Q^{(7)}$	5	1	11	0.03123	0.04782	0.03477	0.20792	163	CH
$Q^{(8)}$	7	1	11	0.97156	1.45380	0.74072	0.18196	49	CM
$Q^{(9)}$	6	1	8	0.05947	0.09123	0.05470	0.05548	4764	CH
$Q^{(10)}$	4	0	6	0.09168	0.11970	0.09584	0.49834	3454	CH
$Q^{(11)}$	3	1	6	2.59998	4.07134	1.85092	0.26802	16069	CL
$Q^{(12)}$	2	0	7	0.21147	0.22465	0.23487	0.13981	71	CL
$Q^{(13)}$	2	1	2	0.12771	5.32350	-	-	16	CL
$Q^{(14)}$	2	0	3	0.03373	0.06017	0.03444	0.29052	28	CH
$Q^{(15)}$	1	0	2	201.53365	22.25911	12.0840	23.11528	9	CL
$Q^{(16)}$	2	1	2	0.11346	0.11219	0.12755	0.13471	120	CM
$Q^{(17)}$	3	1	6	0.10931	0.19021	0.11319	0.97148	648259	CL
$Q^{(18)}$	2	1	5	0.98213	1.81212	-	-	25	CL
$Q^{(19)}$	3	1	3	-	-	-	-	-	-
$Q^{(20)}$	2	0	25	4.05648	4.90228	3.29667	0.17083	21	CM
$Q^{(21)}$	5	2	8	3.02705	5.32847	2.36784	-	8989	CM
$Q^{(22)}$	4	2	13	0.01070	0.01734	0.01610	0.06065	8944	CH
$Q^{(23)}$	2	2	6	2.72083	3.30785	2.35940	-	137	CL
Average	3.17	0.73	7	9.89262	2.29976	1.21958	1.72467	32188.5	
Median	3	1	6	0.12058	0.20188	0.12037	0.27531	133.5	
σ	1.74	0.68	5.23	41.83568	4.73645	2.67162	5.23164	137693.78	

workload. A secondary analysis consists of dissecting the set of items which constitute the initial workload, and quantitatively analyzing their logical complexity, table joints, subqueries, etc. Table 1 presents our findings. Throughput and retrieved data size measurements are considered as performance metrics, and therefore are a part of the bottom-up analysis approach. We combined both analysis approaches to 1) analyze the relationship between the complexity of the workload queries, 2) to evaluate the performance of different application persistence deployment topologies, and 3) to analyze the performance variation of the initial and generated workload over time perceived by the application's end user. Towards this goal, queries are categorized by trimming the mid-range of the initial workload measured throughput and by comparing the total number of logical evaluations with respect to the remaining set of queries in the workload. Given the strong connection between the measured throughput and the resource consumption of the database engine in the TPC-H benchmark, the categories *compute high* (CH), *compute medium* (CM), and *compute low* (CL) are defined, and each query is associated with its corresponding category as shown in Table 1.

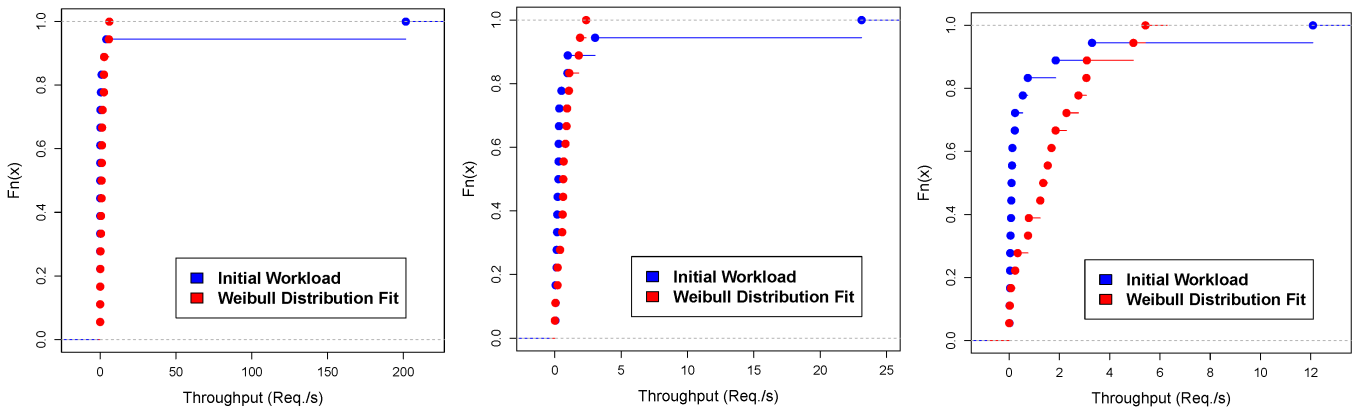
The initial workload empirical distribution was analyzed in terms of its cumulative distribution function. In order to derive the theoretical statistical model associated with the initial workload behavior distribution for the three distribution scenarios, the probability distribution fitting functionalities provided by the *MASS* and *Stats* libraries of R 3.0.2[8] were used. We selected the Kolmogorov-Smirnov (KS) goodness of fit tests, as these work well with small data samples. The KS goodness of fit tests showed a minimal distance between the empirical distribution and the estimated theoretical Weibull distribution with a *p-value* (confidence) greater than 0.05. Therefore, we can accept the KS goodness of fit null hypothesis, which determines the Weibull distribution as the theoretical distribution that best fits the initial empirical distribution representing the application workload behavior. Figs. 1a, 1b, and 1c depict the empirical and fitted Weibull cumulative distributions for the on-premise, DBaaS, and IaaS scenarios, respectively. The analysis above, however, does not take into account the variability in the performance of the different deployment scenarios over time. In the following we focus on its effect to our measurements.

3.3 Performance Variation Analysis

The derivation of the statistical model for the initial workload, and the characterization of each operation constituting this workload provide a representative view on how the underlying infrastructure is capable of handling a workload with concrete characteristics. However, such analysis does not explicitly provide a fined grained analytical view on the performance variation perceived by the end user when executing each workload operation during a time interval. Therefore, we

8. R Project: http://www.r-project.org

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2014.2381237, IEEE Transactions on Services Computing



(a) On-premise (b) DBaaS (c) IaaS

Fig. 1: Initial Workload Behavior Distribution Analysis — Cumulative Distribution Fit.

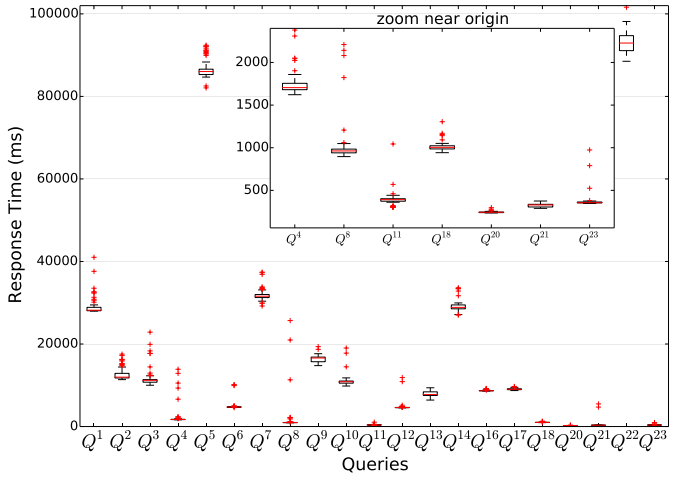


Fig. 2: Performance variation - Initial Workload & On-premise scenario.

focus in this third analytical step on analyzing the degree of performance variation among the experimental time period for the different persistency layer deployment scenarios. There are two fundamental aspects that we must take into account when analyzing the performance variation in the scope of this work. In the first place, the performance variation for each workload operation has an impact in the overall workload performance variation depending on the virtualized or non-virtualized infrastructure where it is executed. Secondly, the workload performance in highly virtualized environments, e.g. AWS or Flexiscale, mostly depends on the current demands of the underlying shared infrastructure. The observed performance variation, however, can be beneficial or detrimental, and directly depends on the concrete day and time when the experimental rounds are driven.

3.3.1 *Individual Deployment Alternatives Analysis*

Fig. 2 shows the dispersion of the measured response time among multiple experimental rounds, each one corresponding to the execution of the initial workload in a database deployed on-premise. The latency experienced by the end database user when executing each workload operation among multiple evaluation rounds is spread into three main intervals: (80000, 100000) ms., (20000, 40000) ms., and (0, 20000) ms. These intervals already show that there are workload operations, e.g. $Q^{(5)}$ and $Q^{(22)}$, which can substantially reduce the overall latency if executed off-premise. A similar performance pattern is observed when executing the same workload operations in a database deployed in an IaaS offering, such as AWS EC2 (see Fig. 3). However, when executing such operations in a database deployed in a DBaaS solution, e.g. AWS RDS, the observed performance degradation is overturned, as the latency is significantly reduced in approximately 90% in average (see Fig. 4). Such improvement can be attributed to the fact that the database's engine configuration in a DBaaS solution is highly optimized for complex (retrieval) query processing. However, such performance improvement pattern is not observed across all sampled workload operations.

The latency observed in most of the workload operations in the on-premise scenario shows a detrimental variation, as most of the latency measurements for each workload operation sample present outliers tending to a higher expected latency with respect to the experimental observations. Such performance fluctuation would require a further empirical analysis for requests $Q^{(1)}$, $Q^{(2)}$, $Q^{(3)}$, $Q^{(4)}$, $Q^{(8)}$, $Q^{(10)}$, and $Q^{(12)}$, potentially investigating individually the underlying resources, such as the storage system or the concrete database server configuration. The scenario comprising the execution of the workload in a database deployed in an IaaS solution presents a lower average performance variation with respect to the on-premise scenario (see Fig. 3). Most of the observations show a minimum amount of outliers, potentially caused by extraordinary events in the local network or load driver. However, the observed latency of most of the workload operations shows a slight performance deterioration with respect to the mean.

The latency variation analysis shown in Fig. 4 corresponds to the execution of the workload operations in a database deployed in a DBaaS solution. At a first glange, an overall performance improvement can be observed when executing the complete workload. However, we can also observe by driving an individual analysis of the operations that such overall performance improvement is not present in all workload operations. For example, there are workload operations which

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2014.2381237, IEEE Transactions on Services Computing

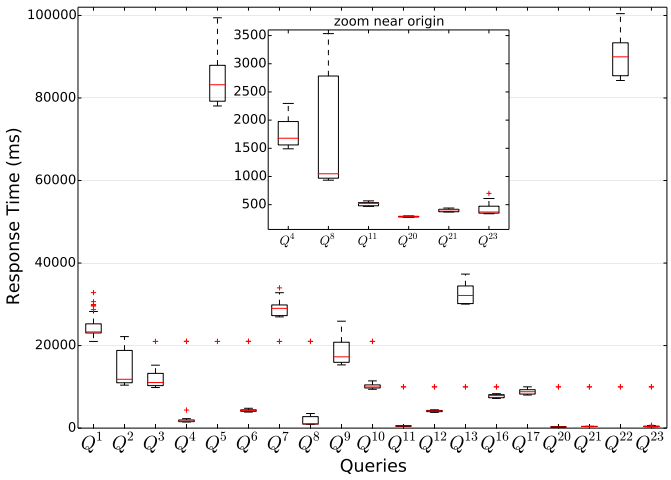


Fig. 3: Performance variation - Initial Workload & IaaS scenario.

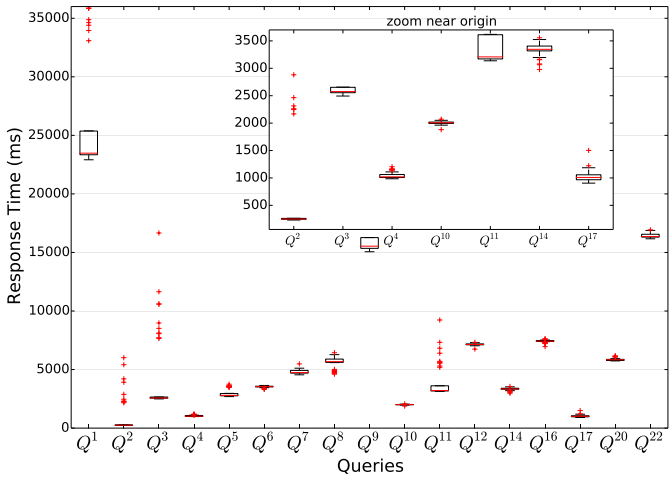


Fig. 4: Performance variation - Initial Workload & DBaaS scenario.

clearly show a performance deterioration ($Q^{(18)}$, $Q^{(11)}$, and $Q^{(20)}$) when comparing their execution to the on-premise or IaaS solution scenarios (see Fig. 2 and 3, respectively). Moreover, $Q^{(1)}$ shows a slight performance degradation with respect to executing it in a database deployed in an IaaS solution. Moreover, such workload operations show a trend towards increasing the latency with respect to the median. Focusing on the performance variation in the DBaaS offering, we can observe that more than 50% of the workload operations present a high amount of outliers. Such variations, however, do not always follow a trend towards a performance degradation, as observed in the on-premise scenario. For example, for the workload operations $Q^{(8)}$, $Q^{(12)}$, $Q^{(14)}$, and $Q^{(16)}$ there are some latency observations which improve the performance in approximately 10% in average with respect to the median.

The previous results show that there exists a divergent performance variation which fundamentally depends on the concrete workload operation and the infrastructure where it is executed. However, there are some scenarios where a certain set of workload operations converge to at least a range of performance measured values. With respect to the variation trend among the different database deployment scenarios, we can deduce that the majority of workload operations tend to show a degraded performance among the experimental rounds due to this variation.

3.3.2 *Overall Deployment Alternatives Weekly Analysis*

As discussed in Section 2, the application's workload fluctuation can highly vary according to different parameters, e.g. end users' demands, popularity of the data, year's season, etc. Moreover, virtualized environments typically share the underlying resources, e.g. hardware, middleware, etc. between multiple applications. The resources assigned to each application depend on the current and predicted demand of the other applications [5]. Resource management, monitoring, and scheduling techniques are typically used to perform analytical operations on current and predicted time intervals. For example, Gmach et al. [5] focus on deriving workload patterns to describe the resources demand's behavioral aspects over time. The driven experiments focused on performing a weekly monitoring, analysis, and patterns derivation of multiple workloads in an HP data center in order to derive solid prediction and resource scheduling models.

Following a similar approach in our experiments, we focus on analyzing the application's performance from the user's perspective, rather than investigating resources assignment optimization techniques in a virtualized environment. Therefore, we analyze the perceived by the user latency for the same workload on a weekly basis for different deployment scenarios of the application's persistency layer. The measurements were taken from 10 experimental rounds on average per day for a period of three weeks in the last quarter of 2013. Fig. 5 shows a comparison among the different deployment scenarios covered in these experiments focusing on the latency measured in different experimental rounds.

In a first step, we can observe that there exists an overall performance improvement when running the application's persistency layer off-premise. Furtermore, there exists a clear gain in performance when using database related Cloud offerings, such as AWS RDS. The observations in this this analysis, however, also show a performance variation when executing the initial workload under different deployment scenarios. The on-premise and DBaaS scenarios present the highest performance variation, while a steadier over time performance is observed for the IaaS scenarios. Focusing on the data trend for each scenario, off-premise scenarios present a trend towards improving the performance when reaching the weekend, while the on-premise scenario behaves in an inverse manner. Such performance degradation trend observed in the on-premise scenario relate to scheduled maintenance tasks running in parallel to our experiments in the on-premise infrastructure. However, the on-premise deployment scenario shows, for example, a better performance until Wednesday, when comparing it to the IaaS EC2. The comparison of trends based on empirical data assists the application developer to make an efficient decision towards which deployment scenario offers a better performance during a concrete time interval.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2014.2381237, IEEE Transactions on Services Computing

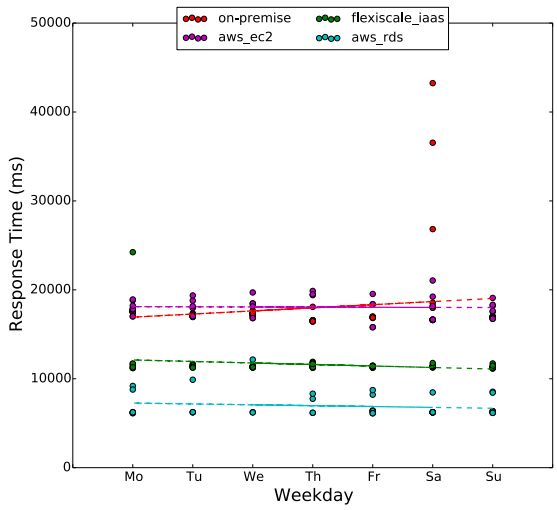


Fig. 5: Workload and Application Data Distribution Evaluation Results.

TABLE 2: Workload and Application Data Distribution Evaluation Results.

Scenario	Category	% Queries same Category	Distribution Parameters	Throughput (Req./s)
On-Premise	CL	79.4%	k= 0.35666 λ= 3.28983	0.27749
On-Premise	CM	18.9%	k= 0.36037 λ= 0.80655	0.05888
On-Premise	CH	95.0%	k= 0.53023 λ= 0.08990	0.02696
DBaaS	CL	66%	k= 0.54324 λ= 0.59264	0.45238
DBaaS	CM	21.6%	k= 0.57200 λ= 0.88472	0.19972
DBaaS	CH	88.3%	k= 0.74471 λ= 0.23991	0.10273
IaaS	CL	78.2%	k= 0.63816 λ= 1.64010	0.34477
IaaS	CM	20.0%	k= 0.52690 λ= 0.53472	0.06046
IaaS	CH	90.8%	k= 0.60906 λ= 0.11362	0.03378

3.4 Generation and Evaluation of Synthetic Workloads

The previous analyses consisted of analyzing the initial workload behavior, evaluating the persistency layer's performance and its variation under different deployment topologies, and establishing the experimental baseline for interpreting the application performance under fluctuating over time workloads. Subsequent to characterizing and aligning the application workload into the previously presented categories, synthetic workload generation and evaluation phases can take place. The workload generation consists of generating a set of representative workloads with different characteristics which take into account the different dimensions previously analyzed. With respect to the the workload generation, in this research work we specifically focus on the categorization and characterization aspects when probabilistically generating a significant workload. The incorporation of environment context aspects into the workload generation methodology, i.e. the observed performance variation in the environment, is planned to be incorporated in future investigations.

The probabilistic workload generation is typically supported by incorporating statistical methods into workload generation tools, such as Malgen[9] or Rain [9]. Malgen supports the creation of large distributed data sets for parallel processing benchmarking, e.g. Hadoop, while Rain provides a flexible and adaptable workload generator framework for Cloud computing applications based on associating workload items with a probability of occurrence. However, such approaches do not explicitly take into consideration the distribution of the application layers among multiple Cloud services. By using scripting techniques and the set of successfully executed TPC-H queries, we created multiple workloads for each of the categories we identified in the previous section with fixed size of 1000 SQL queries each. A generated synthetic workload is categorized based on the frequentist probability of the queries which constitute the workload. For example, the CL synthetic workload is generated by assigning a higher probability of occurrence to the CL queries, and consists of 79.4% of queries categorized as CL as depicted in Table 2 (in the on-premise scenario). However, for the generated CM synthetic workloads the number of CM queries decreases, as the amount of CM queries in the initial workload is lower with respect to the CL and CH queries (Table 1). The variation of the previously selected Weibull distributions of the generated workloads with respect to the initial workload are depicted in Figs. 6a, 6b, and 6c, for the on-premise, DBaaS, and IaaS scenarios, respectively. Table 2 provides the shape and scale parameters of the Weibull distribution for the multiple generated workloads. In the future, we plan to evaluate existing workload generation tools to incorporate support for generating multiple artificial workloads according to specified probability distributions considering the different deployment topologies of the application layers.

3.5 Discussion

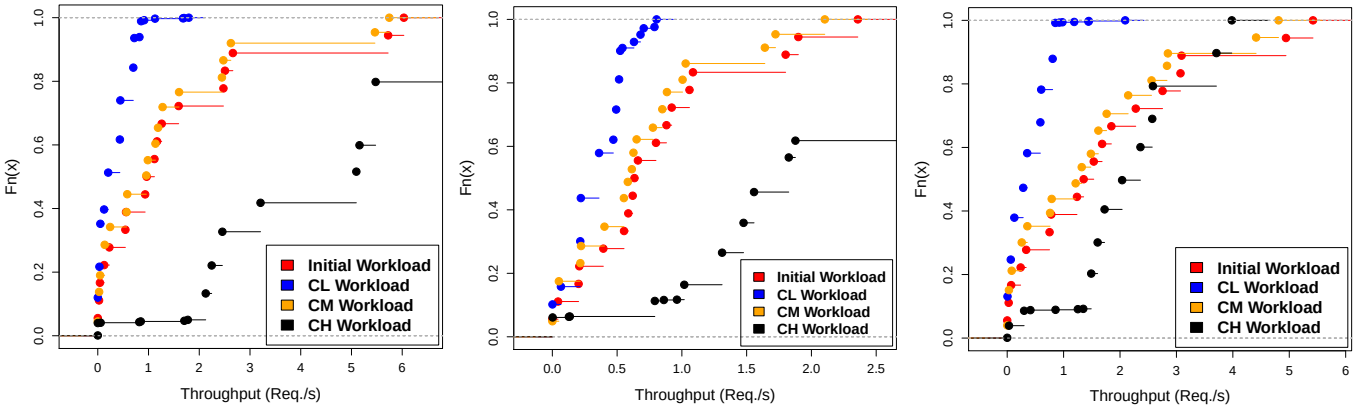
In the following we discuss the most important findings from the previous results. Table 1 drives the following conclusions with respect to the initial workload:

- When deploying the database in the Flexiscale IaaS solution, the average performance of 85% of the successfully executed queries improves between 3% and 4078%. However, when deploying the database in AWS EC2, a performance improvement is observed only in 61% of the successfully executed queries.
- When deploying the database in the DBaaS solution (RDS), the average performance of 70% of the queries improves between 11% and 3725%, but
- there are queries whose performance is degraded when being executed off-premise, such as $Q^{(1)}$, $Q^{(15)}$, and $Q^{(16)}$ (for the Flexiscale scenario), $Q^{(2)}$, $Q^{(3)}$, $Q^{(8)}$, $Q^{(9)}$, $Q^{(11)}$, $Q^{(15)}$, $Q^{(20)}$, and $Q^{(22)}$ (for the AWS EC2 scenario), and $Q^{(8)}$, $Q^{(9)}$, $Q^{(11)}$, $Q^{(12)}$, $Q^{(15)}$, and $Q^{(20)}$ (for the AWS RDS scenario).

In order to evaluate the performance improvement or degradation under different generated synthetic workloads and

9. Malgen: http://code.google.com/p/malgen/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2014.2381237, IEEE Transactions on Services Computing



(a) On-premise Generated WLs vs. Initial WL - Weibull Distribution Fit (b) DBaaS Generated WLs vs. Initial WL - Weibull Distribution Fit (c) IaaS Generated WLs vs. Initial WL 2 - Weibull Distribution Fit

Fig. 6: Generated Workload Behavior Distribution Analysis.

deployment topologies of the application persistence layer, we first analyze from Fig. 6 the workload behavior distribution variation with respect to the initial workload fitted Weibull distribution. It can be observed from the CL and CM fitted Weibull distributions across all scenarios that there exists a faster cumulative probability growth for queries with high throughput, while in the CH case, queries with high throughput are less likely to be included in the generated workload. Moreover, we can observe the impact that the workload fluctuation has on the distribution shape and scale parameters of the Weibull distribution (Table 2). With respect to the overall performance under the different application persistence layer deployment topology, we can observe from the obtained results depicted in Table 2 that:

- the compute demand is indeed increased among the three different workload categories, and the throughput is reduced by 78% to 90% when executing the workload on-premise, by 55% to 77% when executing the workload in a DBaaS solution, and by 80% to 90% when executing the workload in an IaaS solution, using the CL category as the baseline, and
- the overall performance is highly improved when executing the generated workloads off-premise. For the DBaaS solution an increase of 163%, 339%, and 381% is observed for the CL, CM, and CH workloads, respectively. In the IaaS AWS EC2 scenarios, the performance is improved in 124%, 102%, and 125% for the CL, CM, and CH workloads, respectively.

From the previous experiments we can conclude that 1) different workload distributions do not only perform in a different manner, but also that 2) adapting the application deployment topology with respect to the workload demands significantly and proactively improves the application performance. However, an efficient application distribution must 3) take into consideration the performance variability aspects of each Cloud service by means on identifying and predicting, e.g. the time period that offered an efficient performance in past observations. Providing support for (re-)adapting the application topology, i.e. (re-)distributing its layers to adequately consume the required resources to satisfy the workload demands fluctuations, is therefore necessary. With the help of workload characterization and generation techniques, probabilistic models, and prediction capabilities, the application can be proactively and efficiently adaptated to satisfy different workload demands.

4 Application Distribution Support

Based on the previous conclusions, we investigate the requirements and introduce a step-by-step process to analyze the application workload and its fluctuations over time in order to assist the application developer to dynamically and proactively (re-)distribute the application towards achieving an adequate performance.

4.1 Requirements

Functional Requirements

The following functional requirements must be fulfilled by any process based on the analysis of fluctuating application workload over time and the dynamic (re-)distribution of the application for achieving an efficient performance:

FR$_1$ *Support of Both Top-Down and Bottom-Up Analysis*: The process must support both analysis of the application workload and the derivation of its workload behavior model before the deployment of the application (top-down) and during runtime (bottom-up), respectively.

FR$_2$ *Performance-Aware Topology Specification*: The process has to support the definition of application topologies considering performance aspects in various formats such as TOSCA [10] or Blueprints [11].

FR$_3$ *Management and Configuration*: Any tool supporting such a process must provide management and configuration capabilities for Cloud services from different providers covering all Cloud Service Models and Cloud Delivery Models. Focusing on data storage as an example, this includes data stores, data services, and application deployment and provisioning artifacts bundling together different (re-)distribution actions.

FR$_4$ *Support of Different Migration Types*: in order to (re-)distribute an application the process has to support all migration types identified in [1]: replacement, partial and complete software stack migration, and cloudification.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2014.2381237, IEEE Transactions on Services Computing

FR$_5$ *Independence from Architectural Paradigm*: The process has to be independent from the architecture paradigm the application to be (re-)distributed is based on, e.g. SOA [12] or three-layered architecture [2].

FR$_6$ *Support & Reaction on Workload Evolution*: As the workload of an application is subject to fluctuations over time, the process must support the identification of these fluctuations, e.g. based on resource consumption monitoring techniques, and react by (re-)distributing the application accordingly.

FR$_7$ *Support of Multiple Workload Characteristics*: In the ideal case, implementation- or architecture-independent workload characteristics are used in order to create a generic application behavior model. As there are, for instance, an operating system influence on the application behavior, it is nearly impossible to obtain completely independent characteristics [6]. Thus, the process has to support both implementation dependent and independent workload characteristics.

FR$_8$ *Support of Hardware, Software, and Application Characteristics*: the performance optimization is determined by the hardware, software, and the application itself [6]. Hence, the process has to consider characteristics for all three.

FR$_9$ *Creation of Workload Behavior Model*: The process has to support workload behavior derivation and fitting capabilities in order to create the workload behavior model, e.g. based on probability [8].

Non-functional Requirements

In addition to the required functionalities, a process supporting the dynamic application (re-)distribution to cope with fluctuating over time workloads should also respect the following properties:

NFR$_1$ *Security*: (Re-)distribution and (re-)configuration of applications requires root access and administrative rights to the application. Any tool supporting the process should therefore enforce user-wide security policies, and incorporate necessary authorization, authentication, integrity, and confidentiality mechanisms.

NFR$_2$ *Extensibility*: The methodology should be extensible, e.g. to incorporate further provisioning and deployment approaches and technologies.

NFR$_3$ *Reusability*: The workload analysis, workload evolution observation, and application (re-)distribution mechanisms and underlying concepts should not be solution-specific and depend on specific technologies to be implemented. Components of a tool supporting such a process should therefore be extensible when required and reusable by other components and tools, e.g. to be integrated with a Decision Support System for application migration to the Cloud and application architecture refactoring as presented in [13].

4.2 Application Distribution Support Process

Towards fulfilling the functional and non-functional requirements previously identified, in this section a process-based realization approach is presented using the BPMN2 [10] notation (Fig. 7). Two main parties are identified when dynamically (re-)distributing the application to achieve an efficient performance in the Cloud: the *Application Developer* and the *Application Distribution Support System*.

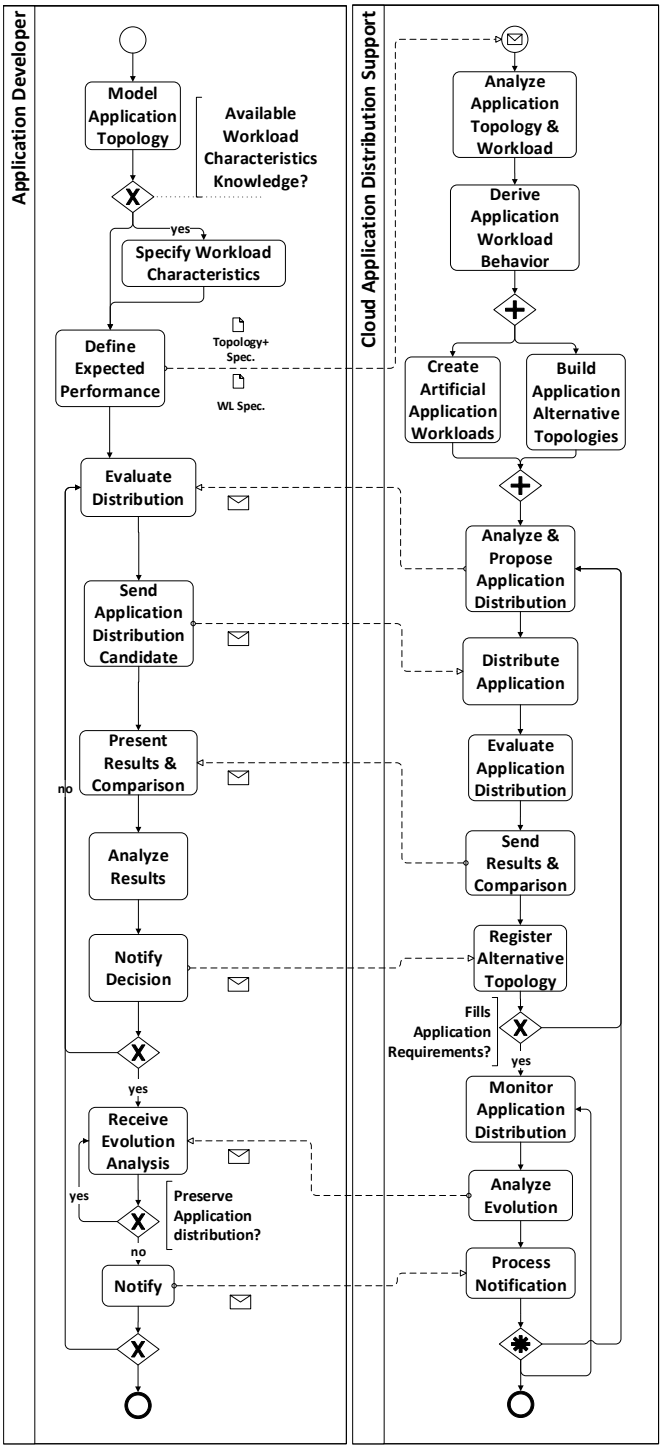


Fig. 7: Application Analysis and Distribution Process using BPMN2 Notation

The application developer tasks are not only related to the application design and realization, but also incorporate responsibilities associated with specifying the application dependencies on the underlying resources, e.g. middleware, OS, and hardware, which are commonly specified using

10. BPMN2 specification: http://www.omg.org/spec/BPMN/2.0/

application topology languages. Available Cloud application topology languages show nowadays a division based on the their degree of generalization among providers. For example, there exist topology languages that are technology specific, e.g. TOSCA [10], Blueprints [11], and AWS CloudFormation[11], while others provide a more generic view together with mapping techniques to enable its conversion to a more concrete technological language, e.g. GENTL [14] or CAMP[12]. There are two fundamental possibilities presented in [15] and supported in this approach for modeling an application topology:

1) A complete specification of the underlying application stack and their corresponding relations. For instance, explicitly specifying the desired middleware components and operating system.
2) A partial specification of the underlying resource, potentially only considering the application specific components (and their corresponding relations) in the topology. For instance, only depicting a front- and back-end tier of a two-tiered application, without any details on e.g. the operating system.

The modeled application topology may also contain information related to resource configuration aspects, e.g. elasticity rules in the form of policies. Subsequently to providing a *model of the application topology* in the top-down approach, the application developer has the possibility to enrich the topology with an initial set of *workload characteristics* that the application demonstrates, e.g. providing information related to the expected frequency of a query in the persistence layer, or defining a probability matrix for the operations in the presentation and business layers. The *expected performance* can be specified in a fine granular way, e.g. for each operation or application layer, or for the application as a whole, e.g. average response time for all application functionalities.

On the other side of Fig. 7, the application distribution support system aims to guide the application developer in the tasks related to efficiently (re-)distributing the application to proactively react to fluctuating and varying application workloads and performance demands. For this purpose, we proposed in the previous sections the combination of both top-down and bottom-up approaches over time. As a first step, the application distribution support system *analyzes the enriched topology and the workload specification.* The expected performance is expressed as set of preferences, which can be analyzed using utility based approaches. By using distribution fitting and goodness of fit statistical techniques, the system *derives an (initial) workload behavior* model. Following on, multiple *artificial application workloads* with different characteristics are generated from the initial workload, e.g. the CL, CM, and CH categories depicted in the previous section, and their behavior models are derived. In parallel, the system generates multiple *application distribution alternatives*, depicted as *alternative topologies* in Fig. 7. Each of these alternative topologies represent a performance-aware application distribution alternative of the different potential migration types discussed in [1]. The topology alternatives can be seen as a list of topologies which can be ordered (ranked) in subsequent phase of the process according to the initial developer preferences following a method like the one discussed in [15].

The remaining tasks in the application distribution support system initiate the *Collaborative Loop*. The collaborative loop is proposed in [16] as an approach to support the (re-)distribution of the application over time to proactively react to fluctuating workloads and resources demands. In the *analyze & propose application distribution* task, the application distribution support system prunes the application topology alternatives space using the previously created workloads by establishing an association between the workload behavior and the observed performance (taking into account its variation among time) of the same or similar applications in previous observations. The ranking of topologies alternatives is a result of using statistical similarity techniques on comparable applications. In case of not possessing such previous experience, the system presents the set of alternative topologies, which can be empirically evaluated prior to the application production phase, e.g. using benchmarking techniques. The application developer then selects a potential application distribution candidate and *sends an application distribution candidate*. Subsequently, the application distribution system *distributes the application* by provisioning the required resources 1) to deploy the different components of the application, e.g. in a public Cloud provider, or 2) to configure the simulation environment based on previous experience and infrastructure data.

The *application distribution evaluation* consist of driving experiments or simulations using the generated artificial workloads and the application topology alternatives. The most relevant to the user measured KPIs, e.g. response time, resource consumption, monetary costs, etc, are then calculated and presented to the application developer in the *send results & comparison* and *present results & comparison* tasks, respectively. The application developer is then responsible for *analyzing the empirical results* and *notifying his decision* about the proposed application distribution alternative. Once the decision is received by the application distribution support system, such information is *registered in the system* for future analysis and prediction, e.g. for analyzing the performance variation as depicted in Section 3. There are two main remaining set of procedures in the collaborative loop. In case of fulfilling the application developer requirements, in the production phase of the application its performance and workload evolution are *monitored and analyzed*. The reiteration of such tasks enable the application distribution support system to analyze the application workload and performance demands evolution through the derivation of workload and performance demands patterns, e.g. using periodogram and workload frequency analysis techniques [5]. Such analyses are periodically reported to the application developer towards assisting in future decisions. The investigations of the workload behavior and evolution over time by means of using statistical techniques allows in this manner for the application to be proactively (re-)distributed to efficiently cope with future workload demands. On the other hand, if the proposed application distribution does not fulfill the application developer requirements, further application distribution alternatives are proposed and the collaborative loop iterates over the previously described tasks.

11. AWS CloudFormation: http://aws.amazon.com/cloudformation/

12. OASIS CAMP v1.1: http://docs.oasis-open.org/camp/camp-spec/v1.1/csprd03/camp-spec-v1.1-csprd03.pdf

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2014.2381237, IEEE Transactions on Services Computing

5 Evaluation

The previously proposed approach aims at assisting the application developer to (re-)distribute the application, i.e. to efficiently select the underlying application resources, to cope with the workload behavior and performance demands fluctuation. The experiments driven in Section 3 showed that different deployment approaches of the application persistence layer already benefit or deteriorate its performance depending on which Cloud offering is chosen. Moreover, different Cloud service delivery models and providers show a different performance variation. In the following we use these findings as part of evaluating the process discussed in the previous section.

5.1 Methodology & Setup

The evaluation discussed in this section aims at empirically and conceptually evaluating the previously presented application (re-)distribution support process using a real world application as a case study. For this purpose, we chose a well-known real world application currently used by a wide amount of internet users: MediaWiki. The two-tiered MediaWiki application is an open source PHP based implementation, which is well known due to its usage in the Wikipedia project and further projects of the Wikipedia foundation. Since the release of real access traces to several Wikipedia mirror servers, different research works have driven investigations on the level of benchmarking and workload analysis [17], [18], [19], [20]. In the experiments we discussed in Section 3 we focused on the computational intensity characteristic of modern applications using the TPC-H benchmark. In the following experiments we aim to move a step forward by considering further business application requirements, e.g. data transfer intensity.

We partially used and extended the MediaWiki benchmark Wikibench[13] for purposes of sampling and generating customized workloads in this work. We used the Wikipedia access traces and a Wikipedia database dump from the year 2008. The access traces were first sampled using the WikiBench sampling tools. We then randomly generated from this sample a workload consisting of 200K HTTP requests, targeting both the retrieval of server files and pages from the back-end database, and the storage and editing of Wikipedia pages stored in the back-end database. Apache JMeter version 2.9 was used for all experimental rounds as the load driver, creating 10 concurrent users and uniformly distributing the load of the application across them. Both the generated load and the load profile are publicly accessible in Bitbucket[14].

The application (re-)distribution process derived a wide set of possible application topologies alternatives for both on-premise and off-premise deployment scenarios. In all cases, the load driver responsible for generating the HTTP requests was maintained on-premise, as the latency introduced by remotely accessing must be taken into consideration when analyzing the latency experienced by the end user. Consequently to the derivation of the application topology alternatives, a subset of such alternatives was selected towards empirically evaluating the performance of the different deployment alternatives of the application. We measured the latency experienced by each user in milliseconds, like in Section 3.

13. Wikibench: http://www.wikibench.eu/

14. Load Driver Profile & Sample - Bitbucket Git Repository: http://bitbucket.org/sgomezsaez/mediawiki_load_driver

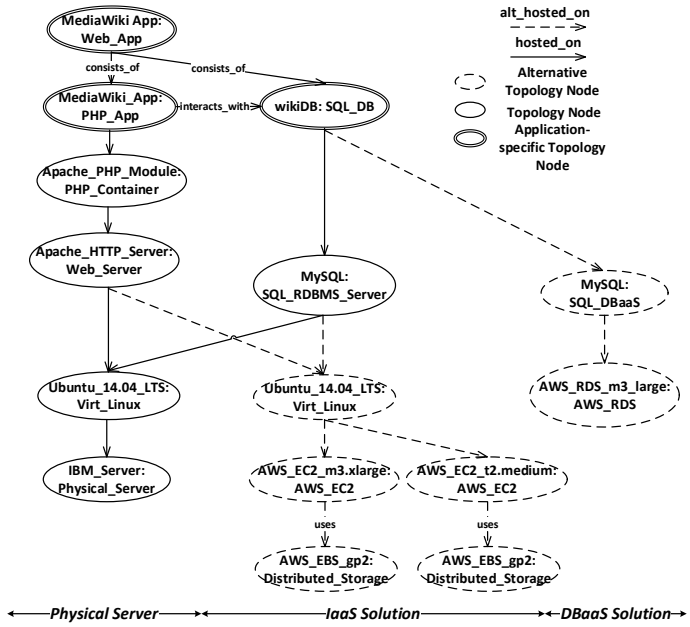


Fig. 8: MediaWiki Application Derived Topology Alternatives.

5.2 Evaluation Results

5.2.1 Topologies Alternatives Derivation

The first set of tasks in the process derived a set of viable topology alternatives for distributing the MediaWiki application, following [15]. From this topologies alternatives space, we then enforced an application developer requirement related to exclusively using AWS offerings, and extracted the subset of alternative topologies depicted in Fig. 8. As already explained, the MediaWiki application is a two-tiered application consisting of a front-end tier and a back-end database tier. The underlying components required by MediaWiki are represented as an acyclic graph in Fig. 8, where nodes depict the different components belonging to a concrete type, while the edges represent the relationships among them. There are two fundamental parts in such topology: the application specific and application independent sub-topologies (the $\alpha-$ and $\gamma-$topologies, respectively, following the terminology of [15]). The former describes the components required specifically by the application, while the latter depicts a subset of components which can be replaced by further sub-topologies, and potentially reused by similar applications. There are two modeling possibilities for the application developer. In a first approach, the application developer can only model the application specific topologies, while the process derives a viable set of alternative topologies. However, the modeling tasks of the process also support the scenario where the application developer models a complete application topology, e.g. the on-premise Physical Server topology depicted in Fig. 8.

Fig. 8 also shows the breaking down of the alternative topologies space to the ones exclusively using AWS offerings. The process derived a set of three application independent sub-topologies and 6 alternative topologies (represented with dashed lines in Fig. 8), consisting of:

1) deploying both tiers in an on-premise virtual machine,
2) migrating the backend database tier to the AWS Cloud and deploying it

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2014.2381237, IEEE Transactions on Services Computing

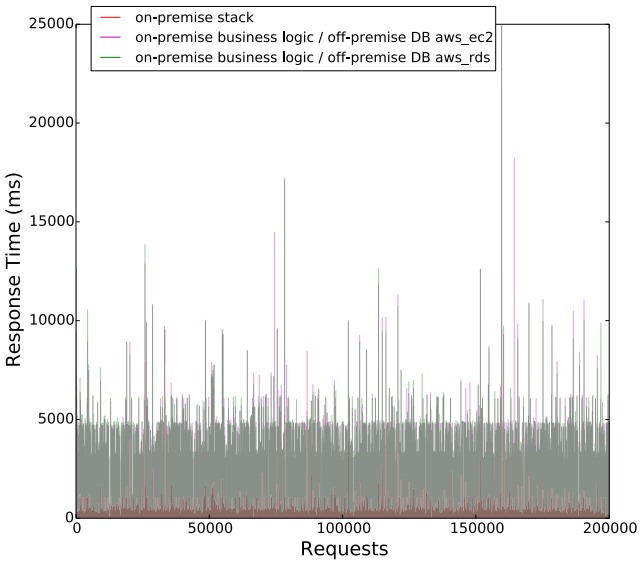


Fig. 9: On-premise deployment of MediaWiki font-end Tier. Back-end database tier deployed in AWS EC2 and AWS RDS solutions.

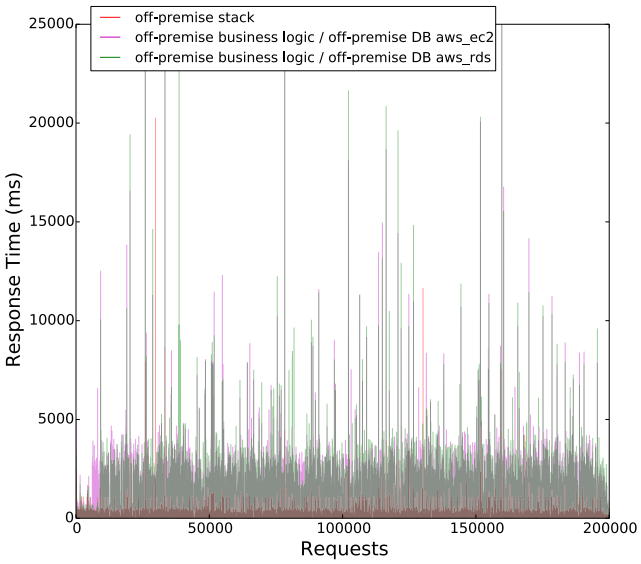


Fig. 10: Off-premise deployment of MediaWiki front-end Tier. Back-end database tier deployed in AWS EC2 and AWS RDS solutions.

- in an IaaS solution, such as AWS EC2
- and in a DBaaS solution, such as AWS RDS,

3) and migrating both front-end and back-end tiers of the application and deploying them in two IaaS EC2 virtual machines with different configurations.

5.2.2 *Performance Evaluation*

The performance evaluation of the previously derived alternative topologies consist in this section of measuring the latency experienced by the application end user. For presentation purposes, we defined two main scenarios for driving the experiments and presenting the results. Based on the location of the MediaWiki front-end tier (on-premise vs. off-premise), we present the evaluation analysis for the different deployment alternatives of the back-end database tier (see Fig. 8).

In the first set of results we compare the deployment of the MediaWiki complete application stack on-premise, against deploying the presentation and business logic on-premise and the application's database in an off-premise IaaS (AWS EC2), and DBaaS (AWS RDS) solution. The results presented in Fig. 9 show that there is a performance degradation of approximately 300% when deploying the MediaWiki database in an off-premise infrastructure, such as AWS EC2 and AWS RDS. Such degradation can be largely attributed to the network latency between the two application tiers. However, due to the fact that the database server is being executed off-premise, most of the resources previously hosting the whole MediaWiki stack were released.

The second set of results depicted in Fig. 10 corresponds to the migration of the complete MediaWiki application stack off-premise. The scenarios firstly consist of deploying the database in the same virtual machine as the presentation and business logic layers of the application, and secondly of deploying the database in independent AWS EC2 and RDS instances. The latency experienced by the application's end user is in average a 6.7% reduced with respect to the on-premise deployment. Such decrease is due to a lower network overall network latency existence in this scenario.

When comparing both scenarios, the experiments show that the usage of a DBaaS solution helps in improving the performance when deploying the application database off-premise. More specifically, for the first scenario (on-premise deployment), the performance improvement is approximately 1.79% while in the second scenario (off-premise) a 6.78%.

5.3 Discussion

From the previously derived case study evaluation in this section we discuss the most relevant findings with respect to the benefits and required improvements in the process based approach for (re-)distributing the application in the Cloud.

The previously driven experiments show significant findings when targeting a two-tiered application distribution evaluation and analysis from two perspectives: from the database tier (Section 3) and from the whole application point of view (Section 5). The former consisted of using a well-known database benchmark and load as the basis (TPC-H), while the latter consisted of empirically evaluating a real world application distribution using a real workload (based on the Wikipedia traces). The experiments reported two major process refinement points related to the performance analysis of the different Cloud services and the investigation of the characteristics of an application workload. More specifically, a performance degradation was observed when moving only the MediaWiki persistency layer to an off-premise DBaaS solution. In contrast, the TPC-H workload showed a significant performance improvement when executing its requests off-premise. The nature of both workload characteristics is however fundamentally different. On the one hand, TPC-H focus on the computational complexity and capacity of the database server, while on the other hand the Wikipedia workload groups a set

of actual requests which target the data retrieval and transfer aspects, putting an emphasis on data transfer. The process must therefore be able not only to provide the means to the application developer to partially or completely specify this characteristic during the application topology design time, but it must also be able to derive such behaviors during the evaluation and production phases through the usage of monitoring and analysis techniques.

With respect to the modeling and specification of performance-aware application requirements, we found that the process considers and adopts the most fundamental aspects of the application performance and workload behavior fluctuation. However, during the case study evaluation we encountered the necessity of not exclusively scoping the application topology enrichment with performance-aware information, but also enabling the specification of both application hard constraints and preliminary configuration tasks. For example, the deployment of the MediaWiki database tier in the AWS Cloud required the creation and configuration of *Security Groups*. Such configuration tasks must be performed prior to the empirical evaluation. Moreover, the scenario where the database tier was deployed on-premise while migrating the presentation and business logic to the Cloud raised a constraint related to accessing the database due to security aspects in our on-premise infrastructure. The specification of constraints must be supported during the application topology modeling phase and used during the alternative topologies space analysis.

Furthermore, vast majority of Cloud providers nowadays offer multiple configuration options for dealing with application workload spikes in line with optimizing the usage of the underlying resources and satisfying the application Service Level Objectives (SLO). Typical approaches are based on defining reactive and dynamic rules for replicating and horizontally scaling, e.g. VMs or database instances across multiple availability zones [21]. We investigated the adaptation options and incorporated the support for specifying the application adaptability options through policies in [22]. Such adaptation options have a direct impact on deciding upon a distribution and re-distribution of the application components. Therefore, we need to empirically evaluate in future experiments the short- and long-term cost of re-distributing (re-deploying) partially or completely the application vs. re-configuring and adapting the provisioned resources.

Container-based deployment approaches, such as Docker[15] are also on the rise, and most Cloud providers, e.g. AWS[16], Microsoft Azure[17], or Google App Engine[18], are incorporating support for provisioning and deploying light weight containers. In the scope of the proposed process based approach, a container-based application deployment is already taken into consideration, as it is considered by the process as a possible alternative topologies set. Moreover, the tasks related to the configuration of the required underlying resources are also taken into consideration during the specification of the application topology model and adaptability options.

15. Docker:https://www.docker.com/
16. AWS Beanstalk: http://aws.amazon.com/elasticbeanstalk/
17. Microsoft Azure:http://azure.microsoft.com/
18. Google App Engine - Containers: https://cloud.google.com/compute/docs/containers

6 Related Work

In the following we present our investigations on existing application architecture model optimization approaches, application workload generators, application and database benchmarks, as well as existing approaches for runtime performance evaluation of services and Cloud applications.

The evolution of software architecture models towards optimizing crucial quality properties is targeted in [23]. An analysis on the problems when designing and deploying a Cloud application in [24] motivates the definition of a methodological approach to create structured Cloud-native applications. Focusing on the application workload, existing application workload generators target the evaluation of the application as a whole, rather than evaluating the performance of each application layer or application component separately for different application topologies. For example, Faban Harness[19] is a free and open source performance workload creation and execution framework for running multi-tier benchmarks, e.g. Web server, cache, or database. Cloudstone [25] targets specifically Web 2.0 applications with a monolithic deployment implemented in Rails, PHP, and Java EE on Amazon EC2 and Sun's Niagara enterprise server. Rain [9] incorporates the possibility to determine the probability of occurrence of the different operations of a Web application, e.g. home page request, log in, or adding event items to the calendar. The language GT-CWSL [4] for specifying workload characteristics is used by the synthetic workload generator for generating Cloud computing application workloads. Existing application benchmarks focus either on evaluating a specific type and aspect of an application or are application implementation and technology specific, e.g. TPC-W[20], TPC-C[21], or SPECjbb2013[22]. All these tools and languages focus on the creation of HTTP-based workloads in order to evaluate the performance of monolithic Web and Cloud applications.

In this publication we focus on workload characterization and analysis of the database layer in order to achieve an efficient performance over time. There are several database benchmarks and data generators for distributed data benchmarking available. Malgen[7] provides a set of scripts that generate large distributed data sets based on probabilistic workload generation techniques, which are suitable for testing and benchmarking software designed to perform parallel processing of large data sets. SysBench[23] is a system performance benchmark for evaluating operating system parameters in order to improve database performance under intensive load. The proprietary database performance testing tool Benchmark Factory[24] provides database workload replay, industry-standard benchmark testing, and scalability testing. The Wisconsin Benchmark is for evaluation of performance of relational database systems [26]. The TPC-H Benchmark[1] illustrates decision support systems handling large volumes of data and using queries with high degree of complexity. The open source database load testing and benchmarking

19. Faban: http://faban.org
20. TPC-W Benchmark: http://www.tpc.org/tpcw/
21. TPC-C Benchmark: http://www.tpc.org/tpcc/
22. SPECjbb2013: http://www.spec.org/jbb2013/
23. SysBench: http://sysbench.sourceforge.net
24. Benchmark Factory™: http://software.dell.com/products/benchmark-factory/

tool HammerDB[25] comes with built-in workloads for TPC-C and TPC-H and supports various relational databases such as Oracle, PostgreSQL, and MySQL. We based our workload characterization and analysis on TPC-H, but we plan to broaden the scope by incorporating additional database benchmarks and performance testing tools.

Nowadays resource consumption monitoring techniques and performance metrics are used to support bottom-up analysis of the workload and in particular the workload evolution analysis. Van Hoorn et al. present the application performance monitoring and dynamic software analysis framework Kieker [27] for continuous monitoring of concurrent or distributed software systems. Efficient management of data-intensive workloads in the Cloud that are generated by data intensive applications, e.g. MapReduce of Apache Hadoop, require to minimize the number of computations and network bottlenecks. Therefore, Mian and Martin propose a framework for scheduling, resource allocation, and scaling capabilities in the Cloud [28]. In the scope of IaaS solutions, a family of truthful greedy mechanisms is proposed in [29] as an approach to optimally provision and allocate VMs in the Cloud. Further optimization techniques focusing on reducing resources reconfiguration costs and maximizing the resource utilization are investigated in [30]. VScaler [31] is proposed as an autonomic resource allocation framework for fine granular VM resource allocation. The systematic comparator of performance and cost of Cloud providers CloudCmp guides Cloud customers in selecting the best-performing provider for their applications [32]. Schad et al. analyze how the performance varies in EC2 over time and across multiple availability zones, using micro benchmarks to measure CPU, I/O, and network, and utilizing a MapReduce application in order to determine the impact of data intensive applications [33]. The above approaches use one of the analysis approaches (either top-down or bottom-up) and do not support the (re-)distribution of the application. In our work we propose to use a combination of these techniques in order to enable application (re-)distribution.

7 Conclusions and Future Work

In the previous sections we identified the need to combine both top-down and bottom-up application workload analysis approaches in order to proactively enable the (re-)distribution of the application components to cope with fluctuating resources demands. The first rounds of experiments positioned this work on the application database (persistence) layer, and used the TPC-H benchmark as the basis. More specifically, we characterized the TPC-H workload according to its computational demands. This characterization was then used as the basis to generate representative workloads with different behavioral characteristics, which emulated the business logic of an application. We evaluated different deployment approaches of the application's database (on-premise, on a DBaaS solution, on different IaaS solutions) and analyzed the perceived performance and its variation on a daily basis. The results show that there is a dependency between the workload distribution, the concrete distribution of the application components, and the performance variability observed in virtualized environments. Such a performance variation mostly increases in off-premise virtualized environments.

25. HammerDB: http://hammerora.sourceforge.net

The experimental results motivated the need for an application distribution process which can be used to enable the application (re-)destribution based on a dynamic analysis of the workload and the resources demands evolution. The process introduced the concept of the *Collaborative Loop* as an approach to assist the application developer in efficiently selecting the distribution of the application components and the selection of Cloud services to cope with the evolution of the application's performance demands. We evaluated our approach using a realistic application and workload under different possible application distribution scenarios. The evaluation showed a degraded performance attributable to an introduced network latency when deploying the database tier off-premise. However, such degradation is overturned when distributing the whole application stack in an off-premise environment. A second step in the evaluation identified potential refinement and adaptation points in the process to exploit further capabilities offered by most Cloud offerings.

Implementing the tool chain for this process is our main task in ongoing work. A number of tools are already in place both for workload analysis, as well as application topology management. Our focus is on integrating, rather than developing them from scratch, except from when deemed necessary, e.g. in the case of defining a performance-aware deployment language and container for deploying Cloud based applications. Future work also includes evaluating the performance variation for the used realistic application and workload, and the overhead, short-, and long-term cost of re-deploying vs. reconfiguring the underlying resources during runtime. Utility-based analysis to investigate the relationship between user preferences and application performance is also part of this effort.

Acknowledgments

This research has received funding from the FP7 EU project ALLOW Ensembles (600792), and the German BMBF project ECHO (01XZ13023G).

References

[1] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to Adapt Applications for the Cloud Environment," *Computing*, vol. 95, no. 6, pp. 493–535, 2013.

[2] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[3] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, and S. Dustdar, "Moving Applications to the Cloud: An Approach based on Application Model Enrichment," *IJCIS*, vol. 20, no. 3, pp. 307–356, October 2011.

[4] A. Bahga and V. K. Madisetti, "Synthetic Workload Generation for Cloud Computing Applications," *Journal of Software Engineering and Applications*, vol. 4, pp. 396–410, 2011.

[5] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload Analysis and Demand Prediction of Enterprise Data Center Applications," in *Proceedings of IISWC'07*, 2007, pp. 171–180.

[6] L. K. John, P. Vasudevan, and J. Sabarinathan, "Workload Characterization: Motivation, Goals and Methodology," in *Proceedings of WWC'98*, 1998.

[7] R. Mian, P. Martin, and J. L. Vazquez-Poletti, "Provisioning Data Analytic Workloads in a Cloud," *FGCS*, vol. 29, pp. 1452–1458, 2013.

[8] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang, "Probabilistic Performance Modeling of Virtualized Resource Allocation," in *Proceedings of ICAC'10*, 2010.

[9] A. Beitch, B. Liu, T. Yung, R. Griffith, A. Fox, and D. A. Patterson, "Rain: A Workload Generation Toolkit for Cloud Computing Applications," University of California, Tech. Rep. UCB/EECS-2010-14, 2010.

[10] T. Binz, G. Breiter, F. Leymann, and T. Spatzier, "Portable Cloud Services Using TOSCA," *Internet Computing, IEEE*, vol. 16, no. 3, pp. 80–85, 2012.

[11] M. Papazoglou and W. van den Heuvel, "Blueprinting the Cloud," *Internet Computing, IEEE*, vol. 15, no. 6, pp. 74–79, 2011.

[12] F. Curbera, F. Leymann, T. Storey, D. Ferguson, and S. Weerawarana, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall International, 2005.

[13] S. Strauch, V. Andrikopoulos, B. Thomas, D. Karastoyanova, S. Passow, and K. Vukojevic-Haupt, "Decision Support for the Migration of the Application Database Layer to the Cloud," in *Proceedings of CloudCom'13*, 2013, pp. 639–646.

[14] V. Andrikopoulos, A. Reuter, S. G. Sáez, and F. Leymann, "A GENTL Approach for Cloud Application Topologies," in *Proceedings ESOCC'14*. Springer, September 2014, pp. 1–11.

[15] V. Andrikopoulos, S. G. Sáez, F. Leymann, and J. Wettinger, "Optimal distribution of applications in the cloud," in *Advanced Information Systems Engineering*. Springer, 2014, pp. 75–90.

[16] S. G. Sáez, V. Andrikopoulos, F. Leymann, and S. Strauch, "Towards Dynamic Application Distribution Support for Performance Optimization in the Cloud," in *Proceedings of CLOUD'14*, June 2014, pp. 248–255.

[17] G. Urdaneta, G. Pierre, and M. Van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[18] L. Petrazickis, "Deploying php applications on ibm db2 in the cloud: Mediawiki as a case study," in *Proceedings of CASCON'09*. IBM Corp., 2009, pp. 304–305.

[19] C. Curino, E. P. Jones, S. Madden, and H. Balakrishnan, "Workload-aware database monitoring and consolidation," in *Proceedings of ACM SIGMOD'11*. ACM, 2011, pp. 313–324.

[20] R. Almeida, B. Mozafari, and J. Cho, "On the evolution of wikipedia." in *ICWSM*, 2007.

[21] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, 2011.

[22] S. G. Sáez, V. Andrikopoulos, F. Wessling, and C. C. Marquezan, "Cloud Adaptation & Application (Re-)Distribution: Bridging the two Perspectives," in *Proceedings EnCASE'14*. IEEE Computer Society Press, September 2014, pp. 1–10.

[23] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms," in *Proceedings of WOSP/SIPEW'10*. ACM, 2010, pp. 105–116.

[24] C. Inzinger, S. Nastic, S. Sehic, M. Voegler, F. Li, and S. Dustdar, "MADCAT - A Methodology For Architecture And Deployment Of Cloud Application Topologies," in *Proceedings of SOSE'14*, 2014.

[25] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, Multi-language Benchmark and Measurement Tools for Web 2.0."

[26] D. Bitton, D. J. DeWitt, and C. Turbyfill, "Benchmarking Database Systems: A Systematic Approach," in *Proceeding of VLDB'83*, 1983, pp. 8–19.

[27] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis," in *Proceedings of ICPE'12*. ACM, 2012, pp. 247–248.

[28] R. Mian and P. Martin, "Executing Data-Intensive Workloads in a Cloud," in *Proceedings of CCGrid'12*, 2012, pp. 758–763.

[29] M. M. Nejad, L. Mashayekhy, and D. Grosu, "A Family of Truthful Greedy Mechanisms for Dynamic Virtual Machine Provisioning and Allocation in Clouds," in *Proceedings of CLOUD'13*, 2013, pp. 188–195.

[30] W. Chen, X. Qiao, J. Wei, and T. Huang, "A Profit-Aware Virtual Machine Deployment Optimization Framework for Cloud Platform Providers," in *Proceedings of CLOUD'13*, R. Chang, Ed. IEEE, 2012, pp. 17–24.

[31] L. Yazdanov and C. Fetzer, "VScaler: Autonomic Virtual Machine Scaling," in *Proceedings of CLOUD'13*. Washington, DC, USA: IEEE Computer Society, 2013, pp. 212–219.

[32] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proceedings of IMC'10*. ACM, 2010, pp. 1–14.

[33] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 460–471, 2010.

Santiago Gómez Sáez is currently a PhD student and research associate in the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart. His experience and research interests relate to Service Oriented Architecture and EAI frameworks, focusing on the aspects related to the performance of enterprise applications and Cloud offerings discovery and selection. Santiago has contributed to the ESB-MT project, and currently contributes to the European Union project ALLOW Ensembles.

Dr. Vasilios Andrikopoulos is a post-doc researcher at Institute of Architecture of Application Systems (IAAS), University of Stuttgart. His research is in the areas of cloud computing, services science and engineering, and software engineering with an emphasis on evolution and adaptation. He received his PhD from Tilburg University, the Netherlands, where he was also a member of the European Research Institute in Service Science (ERISS). He has experience in research and teaching Database Systems and Management, Software Modeling and Programming, Business Process Management and Integration, and Service Engineering. He has participated in a number of EU projects, including the Network of Excellence S-Cube, and he currently contributes to the European Union project ALLOW Ensembles.

Prof. Dr. Frank Leymann is a full professor of computer science and director of the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart, Germany. His research interests include service-oriented architectures and associated middleware, workflow- and business process management, cloud computing and associated systems management aspects, and patterns. The projects he is working on are funded by the European Union, the German Government, or directly by industry partners. Frank is co-author of about 300 peer-reviewed papers, more than 40 patents, and several industry standards (e.g. BPEL, BPMN, TOSCA). He is invited expert to consult the European Commission in the area of Cloud Computing. Before accepting the professor position at University of Stuttgart he worked for two decades as an IBM Distinguished Engineer where he was member of a small team that was in charge of the architecture of IBMs complete middleware stack.

Steve Strauch works as a research associate and PhD student at the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart since April 2008. His research interests are data migration, data hosting, as well as data security and privacy in the area of Cloud Computing, with an emphasis on their application architectural aspects. Steve has contributed to the European projects COMPAS, 4CaaSt, ALLOW Ensembles, and the German government funded BMBF project ECHO.