Towards Dynamic Application Distribution Support for Performance Optimization in the Cloud

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

Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
http://www.iaas.uni-stuttgart.de

in: Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014).
See also BibTeX entry below.

BibTeX:

```
@inproceedings {INPROC-2014-27,
    author = {Santiago G{\'o}mez S{\'a}ez and Vasilios Andrikopoulos and Frank Leymann and Steve Strauch},
    title = {{Towards Dynamic Application Distribution Support for Performance Optimization in the Cloud}},
    booktitle = {Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014)},
    publisher = {IEEE Computer Society},
    institution = {Universit{\"a}t Stuttgart, Fakult{\"a}t Informatik,
    Elektrotechnik und Informationstechnik, Germany},
    pages = {248--255},
    month = {June},
    year = {2014},
    keywords = {Synthetic Workload; Benchmark; Application Distribution; Application Deployment;
    Relational Database; TPC; Database-as-a-Service (DBaaS)},
    contact = {gomez-saez@iaas.uni-stuttgart.de}
 }
```

Towards Dynamic Application Distribution Support for Performance Optimization in the Cloud

Santiago Gómez Sáez, Vasilios Andrikopoulos, Frank Leymann, Steve Strauch
Institute of Architecture of Application Systems (IAAS),
University of Stuttgart, Stuttgart, Germany
{gomez-saez, andrikopoulos, leymann, strauch}@iaas.uni-stuttgart.de

***Abstract*—The Cloud computing paradigm emerged by establishing new resources provisioning and consumption models. Together with the improvement of resource management techniques, these models have contributed to an increase in the number of application developers that are strong supporters of partially or completely migrating their application to a highly scalable and pay-per-use infrastructure. In this paper we derive a set of functional and non-functional requirements and propose a process-based approach to support the optimal distribution of an application in the Cloud in order to handle fluctuating over time workloads. Using the TPC-H workload as the basis, and by means of empirical workload analysis and characterization, we evaluate the application persistence layer's performance under different deployment scenarios using generated workloads with particular behavior characteristics.**

Keywords*—Synthetic Workload; Benchmark; Application Distribution; Application Deployment; Relational Database; TPC; Database-as-a-Service (DBaaS)

I. Introduction

An increasing number of available Cloud services which allow to partially or completely deploy the application in the Cloud is available to application developers these days. Furthermore, the number of non VM-oriented services has also been increasing with the successful introduction of offerings like Database as a Service (DBaaS) from major Cloud providers. It is therefore possible to host only some of the application components off-premise (in the Cloud), e.g. its database, while the remaining of the application remains on-premise [1]. Standards like TOSCA[1] 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.

In this work, we aim to leverage the 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 performance 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 it is beneficial or detrimental. Secondly, the workload of a realistic application fluctuates over time for different reasons, and its topology may have to be adapted to address these oscillations.

For the scope of this paper we 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. The contributions of this work can therefore be summarized as follows:

1) a workload characterization focusing on the application persistence layer by using a well known benchmark (TPC-H),
2) the generation and performance evaluation of generated synthetic workloads for different deployment topologies of the application persistence layer,
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.

The remainder of this paper is structured as follows: Section II summarizes relevant concepts and motivates further investigations. Section III presents our experiments and discusses the most important findings. A process to support the dynamic distribution of the application in the Cloud is proposed in Section IV. Finally, Section V summarizes related work and Section VI concludes with some future work.

II. Background

The deployment of an application in the Cloud regularly requires the realization of preliminary compliance tasks. 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

[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

978-1-4799-5063-8/14 $31.00 © 2014 IEEE
DOI 10.1109/CLOUD.2014.42

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]–[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[2] as the basis to generate application workloads with different characteristics.

III. Experiments

A. 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[2] is then used as the application load driver to emulate the application business logic layer, using the generated set of 23 TPC-H 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[3] 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) *m1.xlarge* instance hosted in the EU zone of the Amazon EC2 offering[4] running Ubuntu 10.04 Linux OS and MySQL 5.1.67,
- and an off-premise Amazon RDS[5] DBaaS *db.m1.xlarge* database instance 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.

B. TPC-H 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 table for the initial 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 I). 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 analyze the relationship between the complexity of the workload queries and the performance of different application persistence deployment topologies. 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 I.

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

[2] Apache Jmeter: http://jmeter.apache.org

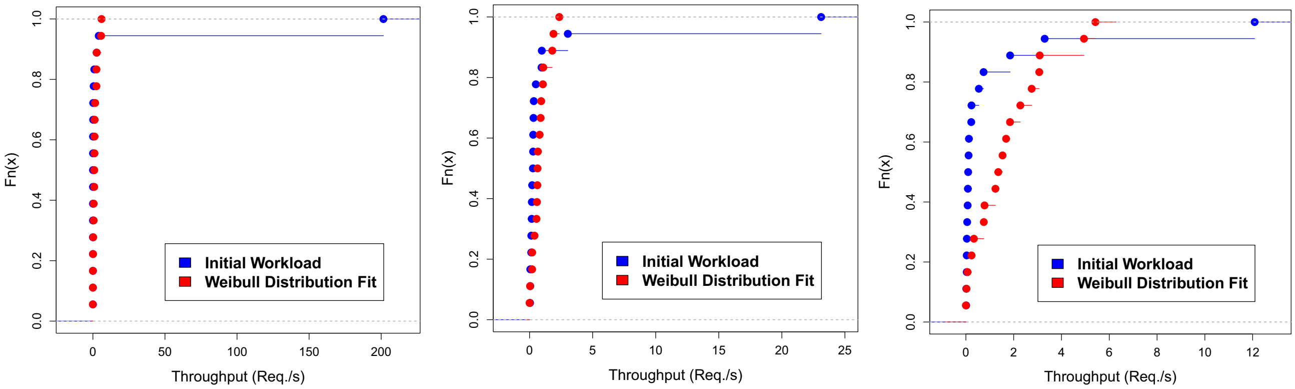
[3] Flexiscale: http://www.flexiscale.com

[4] Amazon EC2: http://aws.amazon.com/ec2/

[5] Amazon RDS: http://aws.amazon.com/rds/

Table I: 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



(a) On-premise

(b) DBaaS

(c) IaaS

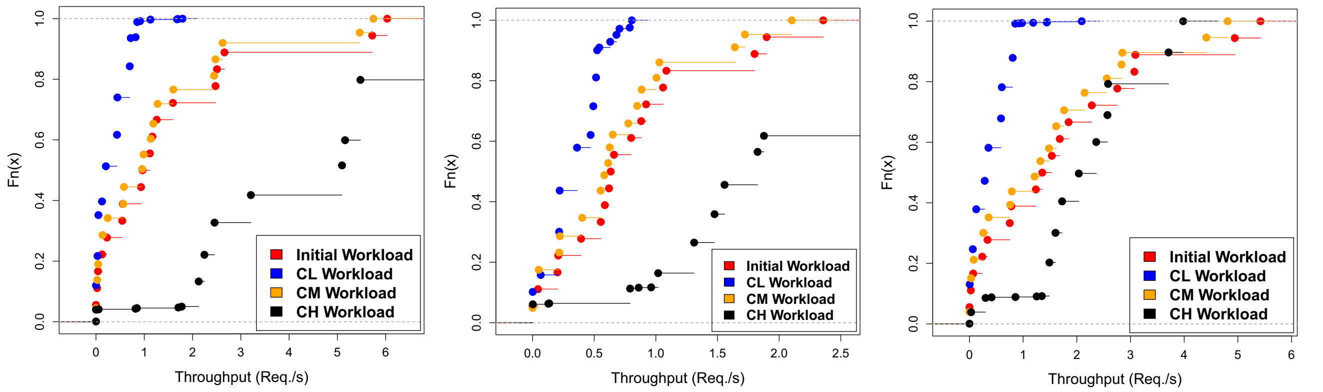
Figure 1: Initial Workload Behavior Distribution Analysis — Cumulative Distribution Fit.

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[6] 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. Figures 1a, 1b, and 1c depict the empirical and fitted Weibull cumulative distributions for the on-premise, DBaaS, and IaaS scenarios, respectively.

C. Generation and Evaluation of Synthetic Workloads

The previous analysis consisted of analyzing the initial workload behavior, evaluating its performance under different deployment topologies of the application persistence layer, and establishing the experimental baseline for interpreting the application performance under fluctuating over

[6] R Project: http://www.r-project.org



(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

Figure 2: Generated Workload Behavior Distribution Analysis.

Table II: 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

time workloads. Subsequent to characterizing and aligning the application workload into the previously presented categories, synthetic workload generation and evaluation phases can take place. For this purpose, probabilistic workload generation techniques are supported, for example, by Malgen[7] and 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 consider the distribution of the application layers.

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 II (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 I). The variation of the previously selected Weibull distributions of the generated workloads with respect to the initial workload are depicted in Figures 2a, 2b, and 2c, for the on-premise, DBaaS, and IaaS scenarios, respectively. Table II 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.

D. Discussion

In the following we discuss the most important findings from the previous results. Table I 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, the average performance of 70% of the queries improves between 11% and 3725%, and
- there are queries whose performance is degradated when being executed off-premise, such as $Q^{(1)}$, $Q^{(15)}$, and $Q^{(16)}$ (for the Flexiscale scenario), $Q^{(2)}$, $Q^{(3)}$, $Q^{(8)}$,

[7] Malgen: http://code.google.com/p/malgen/

$Q^{(9)}$, $Q^{(11)}$, $Q^{(15)}$, $Q^{(20)}$, 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 deployment topologies of the application persistence layer, we first analyze from Fig. 2 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 II). With respect to the overall performance under the different application persistence layer deployment topology, we can observe from the obtained results depicted in Table II 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. Providing support for (re-)adapting the application topology, i.e. (re-)distributing its layers to optimally 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 optimally adaptated to satisfy different workload demands.

IV. 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 support the dynamic (re-)distribution of the application and optimize its performance.

A. 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 performance optimization:

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.

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*: As we optimize for performance which is determined by the hardware, software, and the application itself [6], 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].

B. 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[7] notation (Figure 3). Two main parties are identified when dynamically (re-)distributing the application to optimize its performance in the Cloud: the *Application Developer* and the *Application Distribution Support System*. The application developer tasks are not only related to the application design and realization, but also incorporate responsibilities related to specifying the application dependencies on the underlying resources, e.g. middleware, OS, and hardware, which are commonly specified using topology languages such as TOSCA. On the other side, the application distribution support system aims to facilitate an optimal (re-)distribution of the application to proactively react to fluctuating workloads. As defined by the previous section, the combination of the top-down and bottom-up approaches must be supported. Hence, we identify in each process step the tasks associated with each approach.

Subsequently to specifying 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 must bear, 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. The application distribution support system interprets 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. In a next step, from the initial workload multiple artificial workloads with different characteristics are generated, 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 *Topologies** in Figure 3. Each of these topologies* represent a performance-aware application distribution alternative of the different potential migration types discussed in [1]. These alternatives consist of an ordered list of topologies according to the initial developer preferences. Once the most suitable topology candidate is selected, the application deployment phase (*Distribute Application* task) occurs, and the *Collaborative Loop* is initiated. We propose the collaborative loop as an approach to support the (re-)distribution of the application over time to proactively react to fluctuating workloads.

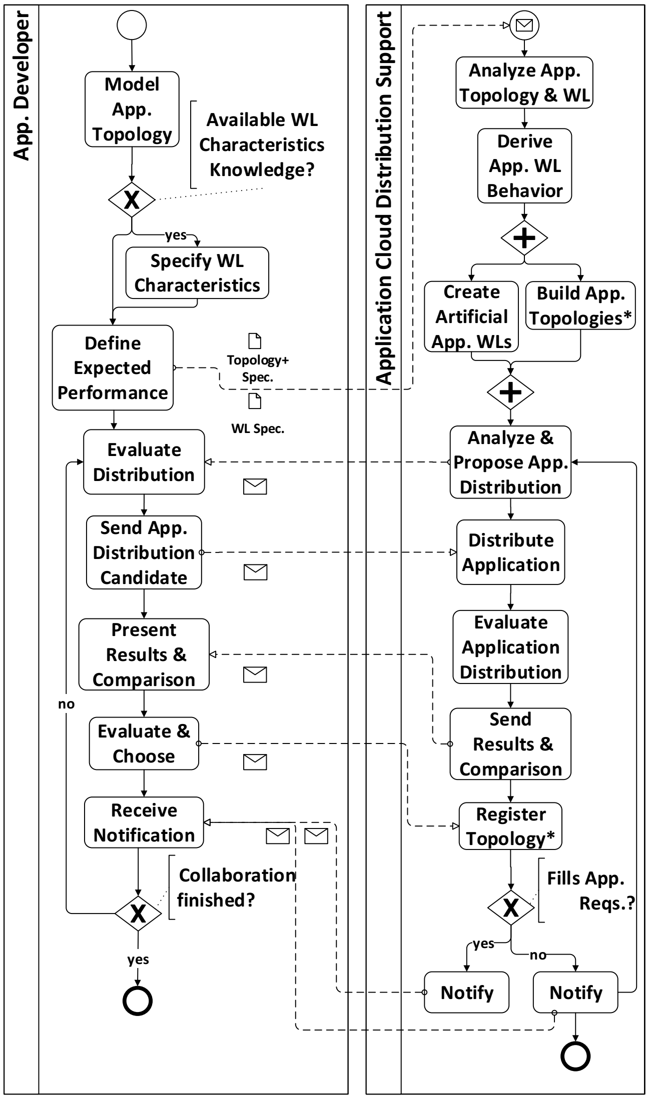

App. Developer
Application Cloud Distribution Support
Model App. Topology
Available WL Characteristics Knowledge?
X
yes
Specify WL Characteristics
Define Expected Performance
Topology+ Spec.
WL Spec.
Evaluate Distribution
Send App. Distribution Candidate
Present Results & Comparison
no
Evaluate & Choose
Receive Notification
Collaboration finished?
X
yes
Analyze App. Topology & WL
Derive App. WL Behavior
+
Create Artificial App. WLs
Build App. Topologies*
+
Analyze & Propose App. Distribution
Distribute Application
Evaluate Application Distribution
Send Results & Comparison
Register Topology*
Fills App. Reqs.?
X
yes
no
Notify
Notify

Figure 3: Application Analysis and Distribution Process using BPMN2 Notation

The bottom-up analysis tasks evaluate the distributed application performance for the generated artificial workloads. The application performance is initially evaluated using benchmark and monitoring techniques to measure the

[7] BPMN2 specification: http://www.omg.org/spec/BPMN/2.0/

most relevant to the user KPIs, e.g. response time, resource consumption, monetary costs, etc. The performance evaluation results are then presented to the application developer, who registers the topology in the system as a suitable or non suitable application distribution with respect to the application performance requirements. If the application distribution does not fit the performance requirements, the system proposes further application distributions.

During the production phase of the application, monitoring techniques ease the analysis of the application workload evolution in order to derive workload and performance patterns, e.g. using periodogram and workload frequency analysis techniques [5]. The representation 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 optimally cope with future workload demands.

V. 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 [14]. An analysis on the problems when designing and deploying a Cloud application in [15] motivates the definition of a methodological approach to create structured 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[8] 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 [16] 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[9], TPC-C[10], or SPECjbb2013[11]. 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 optimize performance. 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[12] 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[13] provides database workload replay, industry-standard benchmark testing, and scalability testing. The Wisconsin Benchmark is for evaluation of performance of relational database systems [17]. 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 tool HammerDB[14] 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 [18] 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 [19]. In the scope of IaaS solutions, a family of truthful greedy mechanisms is proposed in [20] 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 [21]. VScaler [22] 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 [23]. 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 [24]. 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.

[8] Faban: http://faban.org

[9] TPC-W Benchmark: http://www.tpc.org/tpcw/

[10] TPC-C Benchmark: http://www.tpc.org/tpcc/

[11] SPECjbb2013: http://www.spec.org/jbb2013/

[12] SysBench: http://sysbench.sourceforge.net

[13] Benchmark Factory^{TM}: http://software.dell.com/products/benchmark-factory/

[14] HammerDB: http://hammerora.sourceforge.net

VI. Conclusions and Future Work

The previous sections motivated the need for a combination of top down and bottom up application workload analysis approaches in order to support the dynamic (re-)destribution of an application topology to cope with varying demand. The discussion was scoped on the database (persistence) layer of the application, and the TPC-H benchmark was used as the basis of our experimentation. In particular, the load generated by the benchmark was characterized according to its computation and network consumption and the resulting categorization of the benchmark queries were used to generate synthetic application workloads. These workloads were in turn used to emulate the behavior of an application with its database across different solutions (on-premises, on a DBaaS solution, on different IaaS solutions). The results show that depending on the distribution of the queries in the load, application performance can increase or decrease significantly with the application distribution topology used.

Based on these results we then proposed an application analysis and distribution process which can be used to enable application (re-)destribution based on dynamic analysis of the workload. Implementing the toolchain required as part of this process and creating a comprehensive framework for application distribution support is our main task in ongoing work. As discussed in the previous sections, a number of tools are already in place both for workload analysis, as well as application topology management. In this respect, our focus is on integrating them, rather than developing them from scratch, except from when deemed necessary, as for example in the case of defining a performance-aware deployment language and container for the Cloud. Furthermore, evaluating the performance of the overall process is indeed necessary when (re-)distributing, i.e. (re-)deploying the different application components, in a Cloud infrastructure. Utility-based analysis both at provider and consumer level to investigate the relationship between user preferences and application performance is also part of this effort, as well as identifying and integrating appropriate monitoring and analysis tools and approaches.

ACKNOWLEDGEMENTS

This work is funded by the FP7 EU-FET project 600792 ALLOW Ensembles and the BMBF project ECHO (01XZ13023G). Thanks to Johannes Wettinger for his support during the experiments.

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] 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.

[15] 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.

[16] 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.”

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

[18] 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.

[19] R. Mian and P. Martin, “Executing Data-Intensive Workloads in a Cloud,” in *Proceedings of CCGrid’12*, 2012, pp. 758–763.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

All links were last followed on April 24, 2014.