The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications

Sören Frey and Wilhelm Hasselbring
Software Engineering Group
University of Kiel
24118 Kiel, Germany
{*sfr, wha*}*@informatik.uni-kiel.de*

***Abstract*—Cloud computing provides means for reducing over- and under-provisioning through enabling a highly flexible resource allocation. Running an existing software system on a cloud computing basis can involve extensive reengineering activities during the migration. To reduce the correspondent effort, it is often possible to deploy an existing system widely unmodified in IaaS VM instances. However, this simplistic migration approach does not solve the challenge of over- and under-provisioning or scalability issues per se, as our experiments using Eucalyptus and the popular open source system Apache OFBiz show. Moreover, current migration approaches suffer from several further shortcomings. For example, they are often limited to specific cloud environments or do not provide automated support for the alignment with a cloud environment. We present our model-based approach CloudMIG which addresses these shortcomings. It aims at supporting SaaS providers to semi-automatically migrate existing enterprise software systems to scalable and resource-efficient PaaS and IaaS-based applications. To facilitate reasoning about the suitability of certain cloud environments for a given system and the degree of alignment during the reengineering process, we introduce the Cloud Suitability and Alignment (CSA) hierarchy. For example, Apache OFBiz used in our experiments is initially categorized "cloud compatible" but not "cloud optimized" as it does not exploit the cloud's advantages.**

Keywords-Approach CloudMIG, Cloud Computing, Model-based software migration to cloud-based applications, Resource-efficient cloud-based applications, Eucalyptus, CSA hierarchy.

I. Introduction

Most enterprise applications' workload underlies substantial variations over time. For example, user behavior tends to be daytime-dependent or media coverage can lead to rapidly increasing popularity of provided services. These variations often result in over- or under-provisioning of data center resources (e.g., #CPUs or storage capacity). Cloud computing provides means for reducing over- and under-provisioning through supplying elastic services. Thereby, the conformance with contractually agreed Service Level Agreements (SLAs) has to be ensured. Considering legacy software systems, is there a way established enterprise applications can benefit from present cloud computing technologies? For reasoning about this issue, it is useful to clarify the main participants in providing and consuming cloud computing services. Three different roles can be distinguished. Software as a Service (SaaS) providers (cloud users) offer software services, which are being utilized by SaaS users. For this purpose, the SaaS providers may build upon services offered by cloud providers (cloud vendors). In the following, we will employ the terms SaaS user, SaaS provider, and cloud provider.

Newly developed enterprise software may easily be designed for utilizing cloud computing technologies in a greenfield project. Though, SaaS providers may also consider to grant responsibility of operation and maintenance tasks to a cloud provider for an already existing software system. Running established enterprise software on a cloud computing basis may involve extensive reengineering activities during the migration. Nevertheless, instead of recreating the functionalities of an established software system from scratch for being compatible with a selected cloud provider's environment, a migration enables the SaaS provider to reuse substantial parts of a system. The number of system parts which might be migrated is dependent on the weighting of several parameters in a specific migration project. For example, implications concerning the performance or structural quality metrics regarding the resulting software architecture can be taken into account. Furthermore, aligning a software system to a cloud environment's special properties during the migration process has the potential to increase the software system's efficiency. For example, a reengineer could decide to prefer utilization of certain resources according to their pricing. Considering such kinds of favorable resource utilization and a cloud environment's specific scalability mechanisms can improve overall resource efficiency (e.g., according to the aforementioned prioritization) and scalability. However, there are several major obstacles which can impede such migration projects. Current approaches are often limited to specific cloud environments or do not provide automated support for the alignment with a cloud environment, for instance. In this work, we propose our model-based approach CloudMIG, which addresses these shortcomings and focuses on the SaaS provider perspective. The semi-automated approach aims at assisting reengineers in migrating existing enterprise software systems to scalable and resource-efficient Platform as a Service (PaaS) and Infrastructure as a Service

(IaaS) based applications. This paper is an updated and extended version of [1]. It mainly adds two contributions to the original version. First, experiments were conducted utilizing the IaaS cloud environment Eucalyptus [2] and the open source system Apache OFBiz [3] that illustrate the limitations of simplistic migration strategies and advocate profound evaluation and reengineering measures during a migration. Second, we introduce the Cloud Suitability and Alignment (CSA) hierarchy that enables a classification of existing software systems regarding their suitability for specific cloud environments and their level of alignment after initial migration steps.

The remainder of the paper is structured as follows: The related work is described in Section II. Section III presents the experiments utilizing Eucalyptus and Apache OFBiz. These constitute an example scenario for demonstrating the shortcomings of the prevalent simplistic migration approaches that are described in Section IV. The CSA hierarchy forms a basis for reasoning about migration alternatives and is introduced in Section V. Our approach CloudMIG is then presented in the following Section VI, before Section VII draws the conclusions and outlines future work.

II. Related Work

CloudMIG supports reengineers to migrate existing enterprise software systems to the cloud and to reduce complexity aligning their system with the targeted cloud environment. The general complexity of legacy system migration as well as potential measures to cope with the complexity are described in [4]. The authors in [5] sketch a research agenda in cloud technologies and summarize the currently published cloud computing literature. Issues and challenges regarding the cloud computing technology are investigated in [6]. Here, the integration of existing legacy systems in the cloud is regarded a challenging subject, as currently the cloud landscape is diversified and there is a lack of common practices and general interoperability. With CloudMIG we address the prevalent heterogeneity concerning cloud environments and strive towards a more generic migration approach.

A case study of migrating an enterprise IT system to an IaaS cloud environment is presented in [7]. The case study shows achievable costs savings in the cloud environment. However, the authors recommend to consider overall organizational implications as well. A large science database was migrated to the cloud in [8]. The main hurdles the authors faced lay in the transfer of huge amounts of data and performance degradations when trying to avoid changes to the schema and settings. The design and an evaluation of the tool CloudAnalyst is presented in [9]. It is a visual modeller that utilizes the cloud simulation framework CloudSim [10] for analyzing cloud computing environments and applications. Different user bases, as well as various regions, data centers, applications, and workloads can be modeled. A simulation can be used to estimate the operational costs. However, different application architecture candidates have to be modeled manually, no information concerning the structure of an existing software system can be applied automatically, and the cloud's utilized resources cannot be varied dynamically during a simulation run.

The authors in [11] contribute a performance and cost assessment of real cloud infrastructures. Here, the authors modeled a Service-Oriented Architecture (SOA) e-business application and two different workloads. Different pricing plans and hosting scenarios were modeled and implications on performance were evaluated as well. Moreover, platform limitations were explored on a coarse grained level. Our Cloud Environment Constraint (CEC) model (see Section IV) allows a more detailed view and automatic detection capabilities on a source code level compared to the system level used in that work. The authors determine a considerable potential for cutting costs running the modeled application in the cloud. However, they point out that "optimizing applications for very specific cost models may result in vendor lock in and a lack of flexibility and maintainability." In [12], the authors propose a conceptual cloud adoption toolkit that addresses the challenges of cloud adoption in enterprises. The toolkit provides five tools/techniques. Among those, the cost modeling tool utilizes Unified Modeling Language (UML) deployment diagrams to model an intended architecture for running existing software systems in a cloud environment. The deployment model is then augmented with price information that enables automated cost estimation for a specific cloud environment. In comparison, CloudMIG is intended to generate target architecture candidates for arbitrary cloud environments and to calculate the estimated costs for each deployment and in dependence of the observed or expected workload.

A profit-driven service request scheduling approach for clouds is described in [13]. Here, a particular focus is on a service provider and consumer perspective. Service requests have to be scheduled to satisfy the concerns of the service providers as well as the consumers. In this context, a pricing model and two profit-driven service request scheduling algorithms are presented. Linear programming is applied by the authors in [14] for addressing a task throughput maximization problem in a budget-constrained scenario.

Our CSA hierarchy evaluates the suitability and alignment of an existing software system with respect to a specific cloud environment. It focuses on technical opportunities and limitations and incorporates an automated detection of CEC violations (see Section IV). In contrast to that, the suitability index for the adoption of cloud computing technologies presented in [15] includes rather non-technical characteristics like the sensitivity of the system's data or its criticality. Nevertheless, it also considers the scale of existing IT resources and observed resource utilization patterns as well.

Table I. EUCALYPTUS HARDWARE CONFIGURATION

Component	Variant
CPU type	2x AMD Opteron 2384 2.7GHz (4 cores)
RAM	16 GB DDR2-667
Network	1 Gbit/s

Table II. VM INSTANCE TYPES

Name	#CPU cores	RAM (MB)
Standard.M	1	512
Standard.L	2	1,024
Memory.M	2	2,048
Memory.L	2	3,584
Compute.M	4	2,048
Compute.L	6	2,048

Table III. VM INSTANCE TYPE PRICE MODEL

VM instance type	Costs/hour ($)
Standard.M	0.3
Standard.L	0.4
Memory.M	0.5
Memory.L	0.75
Compute.M	0.6
Compute.L	1.15

III. EXAMPLE SCENARIO

The experiment setup of our example scenario is described in Section III-A, the results are then presented in Section III-B.

A. Experiment Setup

We investigated the deployment of Apache OFBiz 9.04 into an installation of Eucalyptus. Apache OFBiz is a Java-based open source E-Commerce/ Enterprise Resource Planning (ERP) system. For instance, it provides several modules for accounting, order processing, and human resource management that are accessible via a web-based Graphical User Interface (GUI).

Eucalyptus is a cloud software for building private, hybrid, or public IaaS clouds. Its Application Programming Interface (API) is compatible with the popular Amazon EC2 and S3 services and it is also available in an open source version. Therefore, Eucalyptus is ideally suited for building cloud computing research test beds. The hardware listed in Table I was utilized for Eucalyptus' cluster and node controllers responsible for allocating and controlling the cluster of Virtual Machines (VMs). The superordinate cloud controller node was installed on an identically equipped machine. However, that second machine did not provide dedicated resources for VM allocation. In typical IaaS offerings as well as with Eucalyptus, a cloud user can choose between different VM instance types as basic building blocks. A VM instance type determines the hardware configuration that is available for running the user's virtual machine. With every start of a VM an appropriate instance type can be assigned according to the user's current needs.

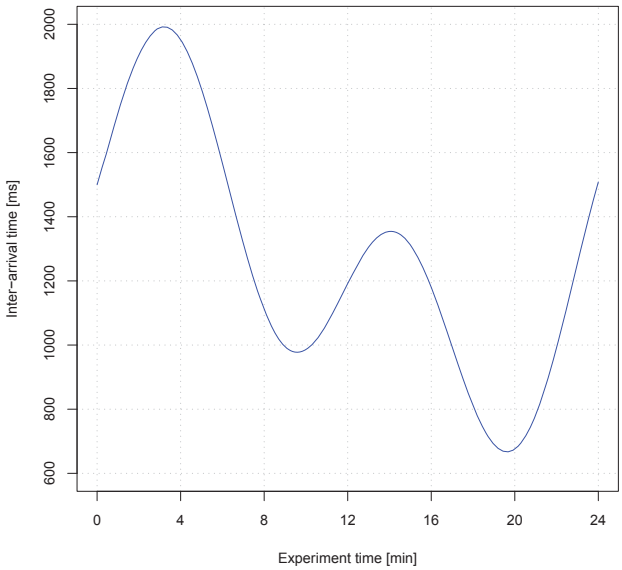

Figure 1. Inter-arrival time function.

To evaluate the implications of VM instance type selection we configured the six different VM instance types that are listed in Table II. In many cloud offerings, the VM instance types are priced on a pay-per-use basis and proportional to supplied resources (e.g, see [16], [17]). The selection of proper VM instance types may therefore have a substantial impact on overall operational costs. Since in real cloud offerings the amply equipped VM instance types are more costly by tendency, we used the price model shown in Table III that roughly follows this principle. The inter-arrival time function illustrated in Fig. 1 was applied to simulate a typical day night cycle usage pattern where the experiment minutes map to the hours of a day. Our employed user behavior emulated customers visiting the web store and browsing a product category. The number of user requests exhibits two peaks, one in the morning and one in the evening hours.

It should be noted that the demo installation of Apache OFBiz 9.04 was used that applies the rather slow embedded Java database Derby to deliver the demo catalog products. However, as the focus of our experiments was to compare the implications resulting from different VM instance types, this does not affect the results' validity. We were particularly interested in the resulting variations concerning the response times and the observed CPU utilizations. Regarding the response times we defined a limit of 1.5s that should not be exceeded for our test user sequence and which can be seen as a part of a virtual SLA [18]. As illustrated in Fig. 2, the usage of one single instance of a VM instance type was not always sufficient to fulfill the SLA. Here, one Standard.L instance provokes an SLA violation in the evening hours (Fig. 2 b). The single Standard.M instance (Fig. 2 a) exhibits an even more distinctive under-provisioning, as the CPU was

often used up to the full. As a consequence, Apache OFBiz repeatedly just returned error messages after experiencing a massive increase in response times up to minute 19, and therefore, caused the test to stop. Hence, in the following, we also investigated the minimum number of instances concerning each VM instance type that were necessary to satisfy the SLA. Here, we always maximized the Java Virtual Machine (JVM) heap size that could be configured according to the VM instance type specifications and that was available to Apache OFBiz.

B. Results

An overview regarding the measured response times and CPU utilizations following the varying load for each applied VM instance type is presented in Fig. 3. To stay below the 1.5s SLA response time limit, two instances of the Standard.M and Standard.L VM instance types were required in each case. The according Fig. 3a) and Fig. 3b) therefore show the average response times and CPU utilizations for both instances. The response times and CPU utilizations generally followed the usage pattern with a rise during peak times and exhibiting lower phases otherwise. Nevertheless, considering the response times this effect manifests more blurred for the aggregated measurements of the two Standard.M and Standard.L instances. Regarding the Standard.M instances the overall CPU utilization was still rather high. An interesting detail can be noticed in the Fig. 3c) - 3f) as there are short bursts around the 14th minute when the number of user requests leaves behind a local minimum.

Besides for the Standard.M VM instance type, the CPU utilizations fluctuate at a rather low level. Fig. 4 underlines this observation by showing the average CPU utilization for each experiment. Incorporating the Standard.M VM instance type, the avg. CPU utilizations range from 16%-59%, which translates to an avg. CPU over-provisioning ranging from 41%-84% at the same time. As mentioned before, we assume presence of a pay-per-use billing model. Considering our defined VM instance type price model (see Table III) the resulting operational costs being extrapolated for one month are presented in Fig. 5. Here, we simplifying presume that the usage pattern repeats each day and therefore the number of the minimally required instances remains stable. The cost minimum is reached by utilizing one Memory.M instance.

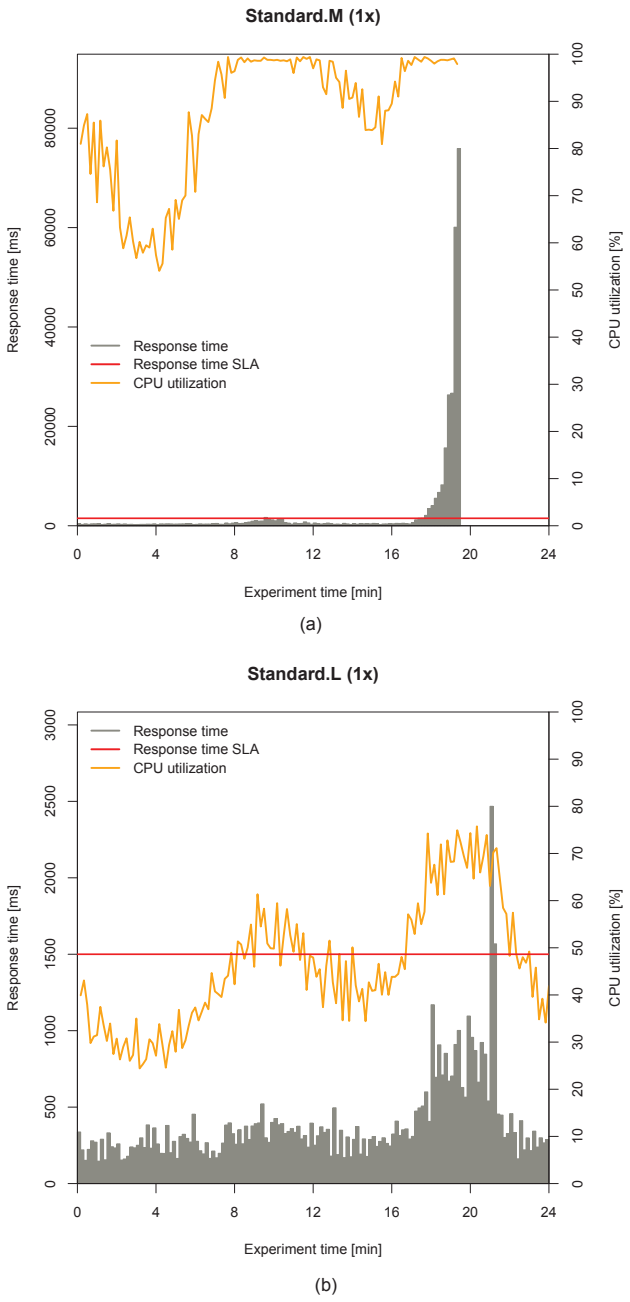

Figure 2. SLA violation when using a single instance of the Standard.M (a) or Standard.L (b) VM instance type.

IV. Current Shortcomings

The example scenario described in Section III reveals several general challenges considering the migration of software systems to a cloud environment. These shortcomings of the prevalent simplistic migration approaches form basic technical difficulties of cloud migration projects that need to be addressed by reengineers when migrating existing systems to the cloud and reworking them for optimized alignment. The example scenario emulates a common approach to minimize the migration effort and to obtain working results in a short period. It deploys the regarding software system to coarse grained IaaS building blocks (VMs). After altering the persistency layer the existing system can be used in a cloud environment.

However, the experiments presented in Section III highlight many open issues. Running an existing application in the cloud does not imply relief of under- and over-provisioning concerns as such. Instead of supplying inappropriate physical on premise hardware configurations, the

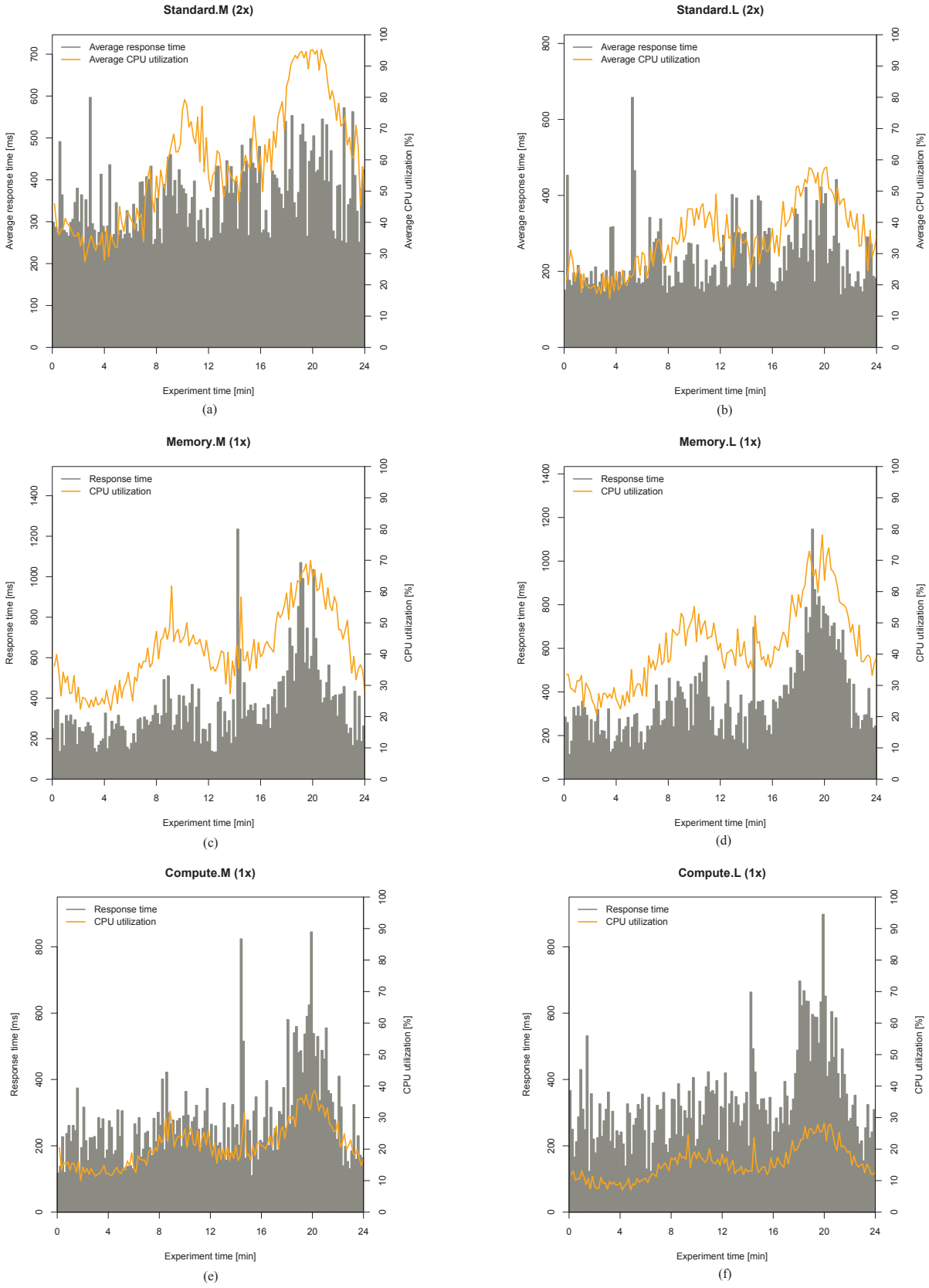

Figure 3. Response times and CPU utilizations for each VM instance type. Two instances were used for Standard.M (a) and Standard.L (b).

under- and over-provisioning of resources can easily be migrated to a cloud environment itself. For example, an inappropriate number of VM instances or unsuitable VM instance types could be employed. Fig. 2 demonstrates the resource under-provisioning in our example scenario. The hardware configuration of Standard.M and Standard.L VM instance types is too restricted for utilizing just a single instance. In this case the response times exceed the defined limit and cause a violation of the SLA. Moreover, this scenario shows the constrained scalability of an application running in a cloud environment. The operation in a cloud does not solve scalability issues per se. For example, an IaaS-based application often needs to have built-in self-adaptive capabilities for leveraging a cloud environment's elasticity. In contrast to the former example, Fig. 4 gives evidence for over-provisioning of cloud resources. Our experiments resulted in a maximum of average 84% over-provisioning of CPU resources for the Compute.L VM instance type implicating more than doubled operational costs compared to the possible minimum (see Fig. 5).

Nevertheless, the effects on additional expenditures cannot simply be evaluated according to the over-provisioning of resources. They depend on other factors as for example the selected VM instance type and do not necessarily scale linearly, as can be seen in Figs. 4 and 5, considering the Compute.M VM instance type in contrast. Comparing different cloud vendors would additionally complicate a cost estimation, as the different price models and VM instance type configurations impede assessment of real world usage scenarios as well. Hence, a better support for anticipating the operational costs without limiting the modeling capabilities to, for example, a set of specific cloud environments, a set of particular configurations, or resource types as VMs is needed. This is especially the case when incorporating PaaS cloud environments, which follow other design paradigms and offer basic building blocks that differ from the VMs used in IaaS-based clouds. Furthermore, our example scenario utilizes a repeating usage pattern as well as homogeneous VM instance types and a constant number of VM instances during an experiment run. This is likely to change in real world scenarios and adds additional complexity in evaluating migration alternatives and estimating the related costs. Further difficulties may arise considering architectural limitations of an existing system. For example, if distribution and parallelization is omitted in the present system design, there may emerge data inconsistency issues when scaling up horizontally while joining the VM instances to an existing data persistency layer. Moreover, exhibiting a reproducible short burst in response times after leaving behind a local minimum in the number of requests (see Section III-B), the experiments revealed an unexpected behavior of the application running in the cloud. In that regard, some effects may generally be hard to predict and therefore require profound evaluation.

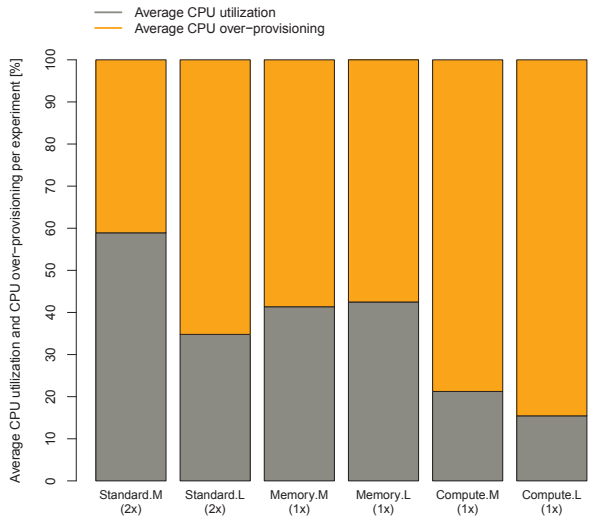

Figure 4. Average CPU utilization per conducted experiment. The statements in parentheses indicate the nr. of instances used for each VM instance type to satisfy the SLA.

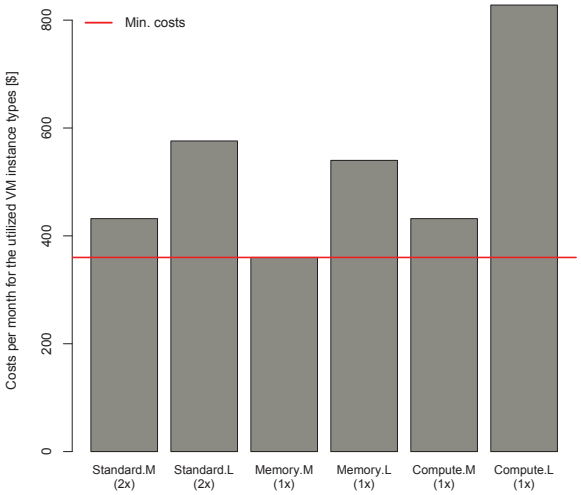

Figure 5. Extrapolated operational costs per month. The statements in parentheses indicate the nr. of instances used for each VM instance type to satisfy the SLA (included in calculation).

As mentioned before, the scalability issues as well as challenges regarding under- and over-provisioning are most often not solved by merely deploying an existing software system in a virtual machine and running it in an IaaS cloud environment. Therefore, we argue that migrating typical enterprise software to a cloud-based application usually implies an architectural restructuring step for aligning it with a cloud environment and exploit the cloud's offered advantages. However, knowledge about the internal structure

of an existing software system is often insufficient and therefore an architectural model has to be reconstructed first. The architectural model serves as a starting point for restructuring activities towards a cloud-optimized target architecture, which at the moment most often has to be created manually. This often is not an easy task, as construction of the advanced architecture usually presumes profound comprehension of the existing one. Furthermore, the target architecture must comply with the specific cloud environment's offered resources and imposed constraints, for example application frameworks and limitations of programming interfaces in PaaS cloud environments, respectively.

In this context, we introduced the notions of Cloud Environment Constraints (CECs), CEC violations, and CEC violation severities in [19] and [20]. For example, considering the cloud environment Google App Engine for Java a CEC would be the restriction of its sandbox environment that limits usage of Java Runtime Environment (JRE) types to only a subset of all types. A system that shall be migrated and that utilizes such an excluded type so far would raise a CEC violation. We defined the three CEC violation severities *Warning*, *Critical*, and *Breaking* that describe the likely effort for fixing a CEC violation, whereas the *Breaking* severity is most serious and causes the CloudMIG process to stop, for instance.

Besides the need for an automated detection of the CEC violations, a mapping model that describes the relationships between system parts of the status quo and a target architecture is required as well. Future workload in combination with the target architecture arrangement will determine resource utilization of the cloud environment during operation. As most cloud providers follow the paradigm of utility computing, and therefore, charge resource utilization on a pay-as-you-use basis, the arrangement of the target architecture has a direct impact on the operational costs.

To condense the difficulties and challenges described in this section, the shortcomings of today's simplistic migration approaches from typical enterprise software to cloud-based applications can be summarized as follows:

- ***S1*** **Applicability:** Solutions for migrating and aligning enterprise software to cloud-based applications are limited to particular cloud providers.
- ***S2*** **Level of automation:** To align existing systems with a cloud environment and to enable them to exploit the cloud's offered advantages, a reengineering step is required. Here, a target architecture and a mapping model currently often have to be built entirely manual. Additionally, the target architecture's violations against the cloud environment's constraints are not identified automatically at design time.
- ***S3*** **Resource efficiency:** Various migrated software systems are not designed to be resource-efficient and do not leverage the cloud environments' elasticity, because even transfering an established application to a new cloud environment can be a cumbersome task itself. Over- and under-provisioning of resources is a challenge in cloud environments, too. Furthermore, means for evaluating a target architecture's dynamic resource utilization at design time are most often inadequate. This even strengthens the general problem that estimating the future operational costs for arbitrary cloud environments is difficult.
- ***S4*** **Scalability**: Scalability remains a concern in cloud environments as well. Automated support for evaluating a target architecture's scalability at design time is rare in the cloud computing context.

V. CSA Hierarchy

To reason about the challenges emerging when migrating a specific system to a cloud environment and restructuring its architecture to facilitate a smooth integration into the cloud's service landscape, one has to judge the system's suitability upfront and the level of alignment with the cloud environment once the first steps are accomplished. To enable an evaluation and classification of software systems in this respect, we introduce the coarse grained Cloud Suitability and Alignment hierarchy (CSA hierarchy). As illustrated in Fig. 6, it comprises the five levels *cloud incompatible*, *cloud compatible*, *cloud ready*, *cloud aligned*, and *cloud optimized*. The levels are defined employing the notions of CECs, CEC violations, and associated CEC violation severities (see Section IV) and constitute revisited and modified applications of CloudMIG's workflow states explained in [19]. The five CSA hierarchy levels are being described in the following.

- ***L0*** **Cloud incompatible:** At least one CEC violation with severity *Breaking* exists.
- ***L1*** **Cloud compatible:** No CEC violations with severity *Breaking* exist.
- ***L2*** **Cloud ready:** No CEC violations exist.
- ***L3*** **Cloud aligned:** The execution context, utilized cloud services, or the migrated software system itself were configured to achieve an improved resource consumption (measurable in decreased costs that are to this effect charged by the cloud provider) or scalability without pervasively modifying the software system.
- ***L4*** **Cloud optimized:** The migrated software system was pervasively modified to enable automated exploitation of the cloud's elasticity. For example, it's architecture was restructured to increase the level of parallelization. An evaluation was conducted to identify system parts which would experience an overall benefit from substitution or supplement with offered cloud services. These substitutions and supplements were performed.

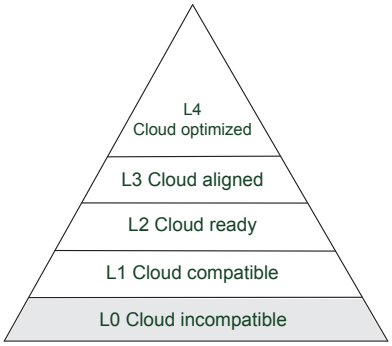

Figure 6. The CSA hierarchy.

The CSA hierarchy is "constructive" in its levels L1-L4. For example, for classifying a software system as *cloud aligned* it has to be *cloud compatible* and *cloud ready* as well. It should be noted that the CSA hierarchy solely considers technical concerns related to a migration to the cloud. In particular, it does not take organisational or economic restrictions into account, for example regarding governance issues, security policies, or a company's business model. Regarding the example scenario in Section III, there exist no CEC violations that would impede proper execution after Apache OFBiz's database is transfered to Eucalyptus' persistent block storage, for instance. Concerning Eucalyptus, this activity is sufficient to lift Apache OFBiz 9.04 from *cloud compatible* to *cloud ready*. However, only through selecting the Memory.M VM instance type the application would be *cloud aligned* (see Fig. 5).

The CSA hierarchy defines the relationship of a specific configuration of a software system (e.g., regarding the version of the system's software architecture) and a specific version of a cloud environment. A system being *cloud ready* concerning a specific cloud environment might be *cloud incompatible* regarding another one. Moreover, even for the same cloud environment this could change over time due to modifications of the incorporated cloud services offered by the cloud environment. Hence, the classification of a software system S regarding the CSA hierarchy depends on its configuration Θ and the cloud services Λ offered by a cloud environment. More specifically, the cloud environment provides n cloud services. A cloud service k is present in a particular version v: $\lambda_k^v \in \Lambda$. The classification of S regarding the CSA hierarchy level is then called Γ. Therefore, we can define a **CSA tuple** as follows:

$$\left(S, \Theta, \bigcup_{k}^{n} \lambda_k^v, \Gamma\right) \quad (1)$$

CSA tuples are utilized to compare cloud environment alternatives or competing software architectures when considering reengineering activities, for instance.

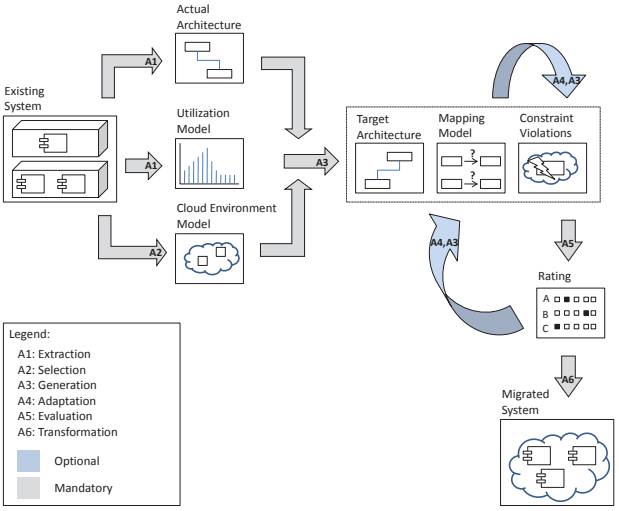

Figure 7. CloudMIG Overview.

VI. The Approach CloudMIG

CloudMIG is composed of six activities for migrating an enterprise system to a cloud environment while addressing the shortcomings described in Section IV. It provides model-driven generation of considerable parts of the system's target architecture. CEC violations are revealed automatically through analyzing an extracted system model. Furthermore, feedback loops allow for further alignment with the specific properties of the cloud environment and foster resource efficiency and scalability on an architectural level. Figure 7 outlines the approach. Its activities (A1-A6) are briefly described in the following, including the incorporated models.

A. Activity A1 - Extraction

CloudMIG aims at the migration of established enterprise applications. Usually, the architecture of software systems tends to erode over time. Therefore, initially envisioned architectures frequently diverge from actual implementations. The knowledge about the internal structure is often incomplete, erroneous, or even missing. As CloudMIG utilizes a model transformation during generation of its target architecture (cf. A3), a representation of the software system's actual architecture has to be available first. Concerning this issue, an appropriate model is extracted by means of a software architecture reconstruction methodology. We propose OMG's Knowledge Discovery Meta-Model (KDM) [21] for building a suitable meta-model.

For leveraging the commonly applied utility computing paradigm, the target architecture has to be laid out resource-efficient and elastic. Therefore, CloudMIG includes the extraction of an established software system's utilization model acting as a starting point. The utilization model (resp. its meta-model) includes statistical properties concerning user behavior like service invocation rates over time or average submitted datagram sizes per request. Relevant

information can be retrieved from various sources. For example, considering log files or instrumenting the given system with our tool Kieker [22] for setting up a monitoring step constitute possible techniques. Furthermore, the utilization model contains application-inherent information related to proportional resource consumption. Metrics of interest could be a method's cyclomatic complexity or memory footprint. We propose OMG's Structured Metrics Meta-Model (SMM) [23] as a foundation for building the related meta-model.

B. Activity A2 - Selection

Common properties of different cloud environments are described in a Cloud Environment Model (CEM) [19]. Selecting a cloud provider specific environment as a target platform for the migration activities therefore implies the selection of a specific instance of the CEM. For example, the CEM comprises entities like VM instances or worker threads for IaaS and PaaS-based cloud environments, respectively. As a result, for every cloud environment, which shall be targeted with CloudMIG, a corresponding instance of CEM has to be created once beforehand. Transformation rules define possible relationships to the architecture meta-model.

We plan to attach further information related to scalability issues to the included entities, which can be configured by the reengineer in activity A4. For example, VM instances could provide hooks for controlling their lifetime dependent on dynamic resource utilization during runtime. Furthermore, the CEM includes constraints imposed by cloud environments restricting the reengineering activities (CECs). For example, the opening of sockets or the access to the file system are often constrained.

C. Activity A3 - Generation

The generation activity produces three artefacts, namely a target architecture, a mapping model, and a model characterizing the target architecture's violations of the cloud environment constraints. The latter lists the CEC violations and results in the construction of an initial CSA tuple. These constraint violations explicitly highlight the target architecture's parts which have to be redesigned manually by the reengineer (cf. A6). The mapping model assigns elements from the actual architecture to those included in the target architecture. Finally, the target architecture constitutes a primary artefact. It is realized as an instance of the CEM, which embeds this model. We propose the three phases P1-P3 for the generation of the target architecture that are illustrated in Figure 8. The phases are constructed as follows.

P1 - Model transformation: The phase P1 produces an initial assignment from elements of the existing architecture to cloud-specific elements available in the CEM. The initial assignment is created applying a model-to-model transformation according to the transformation rules included in the cloud environment model (cf. activity A2).

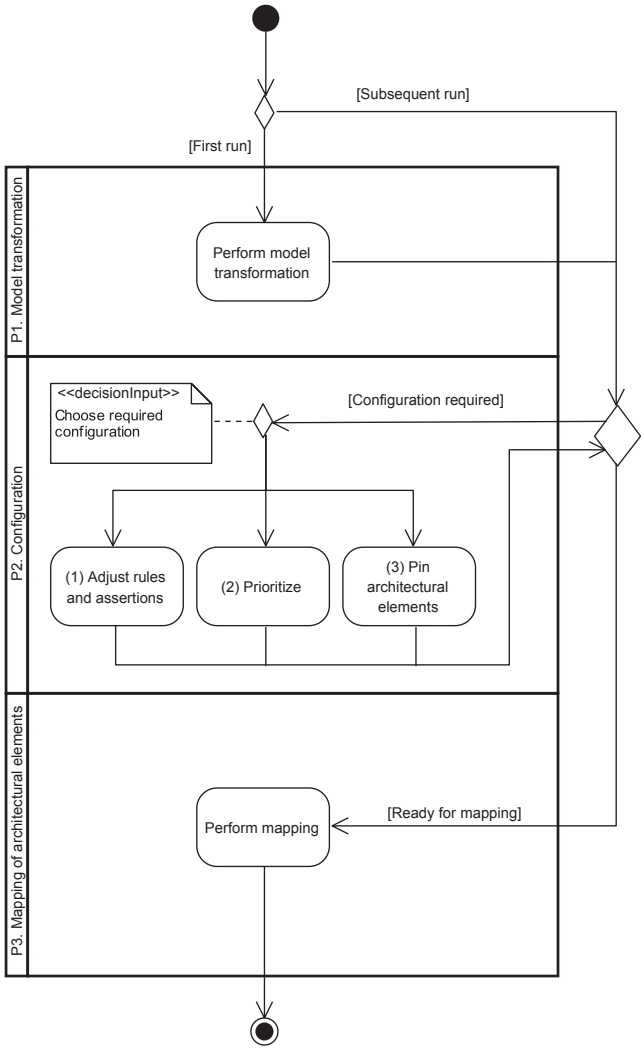

Figure 8. Target architecture generation process.

P2 - Configuration: The phase P2 serves as a configuration of the algorithm used for obtaining a mapping of architectural elements in the phase P3. During P2, a reengineer may adjust rules and assertions for heuristic computation (cf. P3). A rule could be formulated like the following examples: "Distribute the five most frequently used services to own virtual machines" or "The server methods responsible for at least 10% of overall consumption of the CPU time shall be moved to client side components if they do not need access to the database". An exemplary assertion could be: "An existing component must not be divided in more than 3 resulting components". It is intended to provide a set of default rules and assertions. In addition to that, the reengineer will be given the possibility to modify them either via altering

the regarding numerical values or applying a corresponding domain-specific language (DSL). In both cases, the rules and assertions have to be prioritized after their selection. Hereby, the reengineer determines their significance during execution of P3. This means that architectural elements which are related to higher-weighted rules will be considered priorly for assignment and therefore have a stronger impact on the further composition of the target architecture. Furthermore, a reengineer may pin architectural elements. This prevents the rearrangement of previously assigned architectural elements to other target architecture components in phase P3.

P3 - Mapping of architectural elements: The phase P3 improves the initial assignment of architectural elements generated in phase P1 referring to resource-efficiency. Therefore, the formulated rules are utilized and the compliance of the resulting architecture with the defined assertions is considered. There exists an enormous number of possible combinations for assigning architectural elements. Efficiency improvements for one resource can lead to degradation for other resources or impair some design quality attributes. For example, splitting a component's parts towards different virtual machines can improve relative CPU utilization, but may lead to increased network traffic for intra-component communication and a decreased cohesion. Additionally, those effects do not necessarily have to move on linearly and moreover, the interrelations are often ambiguous as well. Therefore, we propose application of a heuristic rule-based approach to achieve an overall improvement. A potential algorithm is sketched in Listing 1 and it works as follows.

The rules are considered successively according to their priority. Thus, rules with higher priorities are weighted higher and have a stronger impact on the generated target architecture. The selection criterion of a rule is defined to deliver a set of scalar architectural elements. All possible subsets of the set are rated respective to the quality of the target architecture that would result, if the elements in the subset would be assigned correspondingly. This aims at considering interdependencies at the level of a single rule. For regarding interdependencies on an inter-rule level, the formulated assertions are taken into account. A rule is only applied if the reengineer did not formulate an assertion with a higher priority that would be violated after the rule's execution. Furthermore, the rule is applied to all mentioned subsets in order of their score. However, the rule is only utilized if no rearrangement of elements is necessary whose subset was rated higher. The same applies to assignments that would lead to rearrangement of elements that were placed by rules of higher priority or formerly pinned elements.

```
1:  E_Pinned ← Pinned architectural elements
2:  R ← All rules
3:  A ← All assertions
4:  R_Sort ← Sort R descending by priority
5:  E_AllAffected ← E_Pinned
6:  for all r in R_Sort do
7:    E_r ← All architectural elements delivered by r's
      selection criterion
8:    P_r^E ← Power set of E_r
9:    Score ← New associative array
10:   for all p_r^E in P_r^E do
11:     Score[p_r^E] ← Rate p_r^E
12:   end for
13:   Score_Sort ← Sort Score descending by score
14:   Score_Sort^Keys ← Keys of Score_Sort
15:   for all p_r^E in Score_Sort^Keys do
16:     E_FormerlyAffected ← p_r^E ∩ E_AllAffected
17:     E_NeedReassignment ← Elements of
        E_FormerlyAffected that need reassignment
        conc. r
18:     if E_NeedReassignment == ∅ then
19:       A_HigherPrio ← All a ∈ A with higher priority
          than r
20:       if ∄a ∈ A_HigherPrio with r violates a then
21:         Apply rule r to all elements in p_r^E
22:         E_AllAffected = E_AllAffected ∪ p_r^E
23:       end if
24:     end if
25:   end for
26: end for
```

Listing 1. Rule-based heuristics for creating a mapping of architectural elements that improves resource efficiency.

D. Activity A4 - Adaptation

The activity A4 allows the reengineer to manually adjust the target architecture towards case-specific requirements that could not be fulfilled during generation activity A3. For example, the generation process might not have yielded an expected assignment of a critical component. Furthermore, for leveraging the elasticity of a cloud environment, the reengineer might configure a capacity management strategy by means of utilizing the hooks provided by entities contained in the CEM (cf. A2).

E. Activity A5 - Evaluation

For being able to judge about the produced target architecture and the configured capacity management strategy, A5 evaluates the outcomes of the activities A3 and A4. The evaluation involves static and dynamic analyses of the target architecture. The results can be aggregated in a CSA tuple. For example, metrics as LCOM or WMC can be utilized for static analyses. Considering the target architecture's expected runtime behavior, we propose to apply a simulation on the basis of CloudSim. Thus, we intend to contribute a transformation from CloudMIG's CEM to CloudSim's simulation model.

F. Activity A6 - Transformation

This activity comprises the actual transformation of the enterprise system from the generated and improved target architecture to the aimed cloud environment. No further support for actually accomplishing the implementation is planned at this time.

VII. Conclusion and Future Work

We presented an overview concerning our model-based approach CloudMIG for migrating legacy software systems to scalable and resource-efficient cloud-based applications. It concentrates on the SaaS provider perspective and facilitates the migration of enterprise software systems towards generic IaaS and PaaS-based cloud environments. We argued for explicit reengineering activities during the migration and motivated them based on experiments we conducted using the cloud software Eucalyptus and the e-commerce/ ERP system Apache OFBiz. Our example scenario demonstrated some of the limitations regarding the currently prevalent simplistic migration approaches. Considering the reengineering activities, CloudMIG is intended to generate considerable parts of a resource-efficient target architecture utilizing a rule-based heuristics. To classify the suitability of cloud environments for given systems and the degree of alignment during a reengineering process, we introduced the CSA hierarchy. The future work focuses on the realization, improvement, and evaluation of CloudMIG's target architecture generation and evaluation activities (A3 and A5).

References

[1] S. Frey and W. Hasselbring, "Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach," in *Proceedings of the First International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2010)*, Lisbon, Portugal, Nov. 2010, pp. 155–158.

[2] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID '09*, May 2009, pp. 124–131.

[3] The Apache Software Foundation, "The Apache Open For Business Project (Apache OFBiz)," http://ofbiz.apache.org/, (Accessed January 20, 2012).

[4] L. Wu, H. Sahraoui, and P. Valtchev, "Coping with legacy system migration complexity," in *Proceedings. 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. ICECCS 2005*, Jun. 2005, pp. 600–609.

[5] I. Sriram and A. Khajeh-Hosseini, "Research Agenda in Cloud Technologies," *CoRR*, vol. abs/1001.3259, 2010.

[6] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 2010, pp. 27–33.

[7] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS," *CoRR*, vol. abs/1002.3492, 2010.

[8] A. Thakar and A. Szalay, "Migrating a (Large) Science Database to the Cloud," in *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. New York, NY, USA: ACM, 2010, pp. 430–434.

[9] B. Wickremasinghe, R. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 2010, pp. 446–452.

[10] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," *CoRR*, vol. abs/0903.2525, 2009.

[11] P. Brebner and A. Liu, "Performance and Cost Assessment of Cloud Services," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, E. Maximilien, G. Rossi, S.-T. Yuan, H. Ludwig, and M. Fantinato, Eds. Springer Berlin/ Heidelberg, 2011, vol. 6568, pp. 39–50.

[12] D. Greenwood, A. Khajeh-Hosseini, J. W. Smith, and I. Sommerville, "The Cloud Adoption Toolkit: Addressing the Challenges of Cloud Adoption in Enterprise," *CoRR*, vol. abs/1003.3866, 2010.

[13] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-Driven Service Request Scheduling in Clouds," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, May 2010, pp. 15–24.

[14] W. Shi and B. Hong, "Resource Allocation with a Budget Constraint for Computing Independent Tasks in the Cloud," in *Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2010, pp. 327–334.

[15] S. C. Misra and A. Mondal, "Identification of a company's suitability for the adoption of cloud computing and modelling its corresponding Return on Investment," *Mathematical and Computer Modelling*, vol. 53, no. 3-4, pp. 504–521, 2011.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb. 2009.

[17] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2009.

[18] W. Iqbal, M. Dailey, and D. Carrera, "SLA-Driven Adaptive Resource Management for Web Applications on a Heterogeneous Compute Cloud," in *CloudCom*, ser. Lecture Notes in Computer Science, M. G. Jaatun, G. Zhao, and C. Rong, Eds., vol. 5931. Springer, 2009, pp. 243–253.

[19] S. Frey and W. Hasselbring, "An Extensible Architecture for Detecting Violations of a Cloud Environment's Constraints During Legacy Software System Migration," in *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*, T. Mens, Y. Kanellopoulos, and A. Winter, Eds. IEEE Computer Society, Mar. 2011, pp. 269–278.

[20] S. Frey, W. Hasselbring, and B. Schnoor, "Automatic conformance checking for migrating software systems to cloud infrastructures and platforms," *Journal of Software Maintenance and Evolution: Research and Practice*, doi: 10.1002/smr.582, 2012.

[21] Object Management Group, Inc., "Architecture-Driven Modernization (ADM): Knowledge Discovery Metamodel (KDM), V. 1.3," http://www.omg.org/spec/KDM/, (Accessed January 20, 2012).

[22] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst, "Continuous monitoring of software services: Design and application of the Kieker framework," Department of Computer Science, University of Kiel, Germany, Tech. Rep. TR-0921, Nov. 2009.

[23] Object Management Group, Inc., "Architecture-Driven Modernization (ADM): Structured Metrics Meta-Model (SMM), V. 1.0 Beta 3," http://www.omg.org/spec/SMM/, (Accessed January 20, 2012).