Using Performance Models for Planning the Redeployment to Infrastructure-as-a-Service Environments: A Case Study*

Sebastian Lehrig & Steffen Becker
s-lab – Software Quality Lab, Paderborn University, Germany (software-engineering@cloudscale-project.eu)

***Abstract*—*Context:* Performance models allow software architects to conduct what-if analyses, e.g., to assess deployment scenarios regarding performance. While a typical scenario is the redeployment to Infrastructure-as-a-Service (IaaS) environments, there is currently no empirical evidence that architects can apply performance models in such scenarios for accurate performance analyses and how much effort is required. *Objectives:* Therefore, we explore the applicability of software performance engineering for planning the redeployment of existing software applications to IaaS environments. *Methods:* We conduct a case study in which we apply performance engineering to redeploy a realistic existing application to IaaS environments. We select an online book shop implementation (CloudStore) as existing application and engineer a corresponding Palladio performance model. Subsequently, we compare analysis results with measurements gathered from operating CloudStore within (I) a classical on-premise setup, (II) OpenStack, and (III) Amazon EC2. *Results:* Our case study shows that performance models have a relative accuracy error of less than 12% even for IaaS environments (scenarios (II) and (III)). For scenarios (II) and (III), we saved up to 98% model creation effort by reusing the model from scenario (I); we just re-calibrated processing rates of CPUs within our deployment model. *Conclusions:* Software architects can plan redeployments by reusing performance models of their existing system, thus, with only minor effort. This is particularly possible for virtualized third-party environments like for Amazon EC2.**

***Keywords*-case study; performance engineering; redeployment;**

I. Introduction

In software performance engineering (SPE) [28], software architects analyze performance characteristics of software systems by using performance models like Markov chains, layered queuing networks, stochastic process algebras, and annotated UML models. With such performance models, architects can conduct what-if analyses, e.g., allowing them to plan deployment scenarios regarding performance.

A. Problem Statement

A nowadays typical scenario is the redeployment of existing applications to virtualized third-party environments, so-called Infrastructure-as-a-Service (IaaS) environments [5]. Redeployments to IaaS environments are challenging for software architects because of (1) the suspected performance unpredictability of IaaS environments [5] and (2) potentially high efforts to adapt their existing performance models.

*The research leading to these results has received funding from the European Seventh Framework Programme (FP7/2007-2013) under grant no 317704 (CloudScale) and was supported by AWS in Education Grant award.

The problem is that there are currently no empirical studies of the extend to which SPE can cope with these issues [23]. Initial case studies indicate that SPE provides accurate predictions for virtualized environments [18], [25], [12], [9]. However, these studies neither consider less controlled public, third-party (i.e., IaaS) environments nor the effort to adapt existing performance models in redeployment scenarios. Additional empirical evidence is therefore needed with particular focus on (1) IaaS environments and (2) model adaptation effort and the degree to which models can be reused. Such evidence would help software architects and project managers to assess SPE for similar scenarios.

B. Research Objectives

To cope with the lack of evidence, we conduct a case study where we follow SPE processes to redeploy a realistic, existing system to IaaS environments. Using the Goal-Question-Metric (GQM) template [31], typically used to describe goals of empirical studies, our goal is to:

Analyze: the software performance engineering method
For the purpose of: planning redeployments
With respect to: applicability
From the viewpoint of: software architects
In the context of: realistic, existing web applications to be redeployed to IaaS environments.

C. Context

In this section, we detail the context of our study, i.e., the factors that potentially affect generality and utility of our conclusions. These details allow readers to easily determine whether our case study also fits to their specific situation [15].

Application type & domain; type of companies. We select the CPU-bound online book shop "CloudStore" [2] as existing web application. Typical small- and medium-sized companies may realize such an application.

In our previous efforts, we started to establish CloudStore as a common migration scenario, e.g., regarding new deployments [2]. We therefore provide CloudStore's documentation, source code, deployment scripts, performance models, and raw measurement data online [1]. Currently, CloudStore is the only scenario we are aware of with this amount of open material.

Tools. We create and analyze performance models with the Palladio-Bench, a modeling and analysis tool that comes with Palladio [7]. Palladio is a concrete SPE approach that is

component-based. For reuse scenarios, previous studies [19] show that component-based SPE approaches save model creation effort compared to monolithic approaches like classical SPE [28]. Because we expect to reuse existing performance models for deployment planning, we accordingly chose to use the Palladio-Bench. All steps we conducted with the Palladio-Bench are documented in online screencasts [3].

Process. We proceeded according to software performance engineering (SPE) [28]. For assessing applicability, we particularly evaluate the accuracy of our performance models by comparing analysis results with measurements gathered from operating CloudStore within (I) our on-premise setup, (II) OpenStack, and (III) Amazon EC2. We additionally measure the effort for each conducted action, thus, serving as a first baseline for estimating the effort for deployment planning.

Scenarios. Scenario (I) represents the starting point of our investigations, i.e., the initial on-premise environment that shall be redeployed to an IaaS environment. Scenario (II) focuses on OpenStack because it is a popular open source IaaS environment. In scenario (III), we decided to focus on Amazon EC2 because it is a popular public IaaS environment.

Experience of the participants & time constraints. We, the authors of this paper, conducted this case study. We represent professional software architects and performance engineers with an average of ten years related practical experience. We had no time constraints for creating and evaluating the model for scenario (I); however, we conducted scenarios (II) and (III) with a time restriction of two month. In terms of person month, we spend about one person month to engineer the model for scenario (I) (loosely distributed over three month) and two additional person month for scenarios (II) and (III) as well as for writing this paper.

D. Results and Contribution

Our results indicate that performance models have a relative accuracy error of less than 12% even for IaaS setups. For scenarios (II) and (III), we saved up to 98% model creation effort by reusing the model from scenario (I). Therefore, architects can efficiently plan redeployments based on performance models of their existing system, i.e., with negligible effort and high accuracy.

The contribution of this research paper is the report of our case study, including:

- description, execution, analysis, and interpretation of the CloudStore case with above main results,
- novel Palladio performance models for CloudStore that we evaluate within our case study, and
- an evaluation of potential threats to validity of our results.

E. Outline

This paper is strictly organized according to the reporting guidelines for case studies by Runeson and Höst [26]. In Sec. II, we discuss related empirical studies on performance engineering for IaaS environments. Section III describes the design of our case study and Sec. IV provides its results. We finally conclude the paper and highlight future work in Sec. V.

II. Related Work

Case Studies. Smith and Williams [28] report the first successfully executed case studies on SPE. Their case studies cover domains such as business information systems for web applications and the financial sector, computer-aided design, distributed systems, and embedded real-time systems. They therefore show that SPE is generally applicable to a variety of domains. However, they neither provide empirical evidence for deployment scenarios where performance models of the original system already exist and can potentially be reused, nor for virtualized environments like IaaS environments.

Koziolek's survey on component-based SPE approaches reveals that no case study fills this gap before 2010 [16]. We therefore searched for a "case study" starting from 2010 on "component-based" "software performance engineering" approaches that fills this gap (we used quoted terms as search phrases in the meta search engine Google Scholar[1] and found four relevant studies after manual relevance assessment [18], [25], [12], [9]). We restrict our search to component-based SPE approaches because these promise the lowest effort for planning redeployments of existing systems (cf., Sec. I-C).

We conducted a case study with Palladio and within our virtualized on-premise infrastructure [18]. For the four cases we consider, our performance analysis provides accurate results (e.g., our prediction error regarding throughput was below 14% on average), indicating that SPE is indeed applicable to virtualized environments. However, we did not investigate whether our models can be reused for planning further deployments.

Rathfelder et al. [25] conduct a case study with Palladio but within a realistic, industrial context (a commercial e-mail system). Their case study shows that Palladio is applicable in such realistic contexts while maintaining high accuracy (their prediction error regarding resource utilization was "mostly less than 10%" [25]). Unfortunately, this case study neither covers virtualization nor planning different deployments.

Within their case study on the IBM System z9 mainframe for storage virtualization, Huber et al. [12] provide further evidence for Palladio's applicability in industry. Their results indicate that Palladio accurately analyzes such virtualized environments (prediction error regarding throughput $\sim$20%). Again, only a single deployment is considered in this study.

De Gooijer et al. [9] use Palladio to analyze an industrial distributed system from ABB that is deployed in a virtualized environment. They provide further evidence that SPE is applicable to such environments because their predictions regarding resource utilization deviate at most by 30%. Still, they do not investigate whether their models can be reused for planning redeplyoments to third-party hosted IaaS environments.

Both Rathfelder et al. [25] and Huber et al. [12] argue that software architects have to trade-off between analysis accuracy and modeling effort. While all above authors describe means to lower such effort, only Huber et al. [12] and Smith and Williams [28] quantify effort. Huber et al. [12] estimate that non-Palladio experts can create accurate performance models

[1] http://scholar.google.com

for similar systems like IBM System z9 with an effort of 4 person months. Smith and Williams [28, p. 458] estimate that SPE efforts may cause up to 10% of total project costs. In a later work [32], they confirm this estimate by conducting a case study with an artificial business case where they estimate that one full-time performance engineer is needed in addition to a team of 15 developers. While such estimates can serve as good base lines for effort estimations of new projects, they are unsuited for planning redeployments and their expected lower effort due to model reuse.

Controlled Experiments. Above case studies indicate that SPE is applicable to our domain of web applications. This insight, however, should be considered with care because case study results are not necessarily generalizable [27]; they are more explanatory. For more generalizable theories, controlled experiments are favorable [27].

According to our recent survey [17], only Martens et al. [19] conducted such experiments on the accuracy and effort of SPE. For reuse scenarios of performance models, their results indicate that component-based SPE approaches like Palladio save effort without sacrificing accuracy—compared to monolithic performance modeling approaches like classical SPE [28], UML Performance Simulator [20], and Capacity Planning [22] for which no performance model reuse is intended. For this reason, we chose Palladio as SPE approach for planning redeployments. Moreover, Martens et al. provide evidence that effort metrics "time for model creation" and "size of model" positively correlate. This correlation allows to triangulate results regarding effort over multiple metrics. In our case study, we evaluate whether such hypotheses also hold in realistic contexts and for planning redeployments.

III. Case Study Design

We plan the complete case study design in this section. At first, we refine the goal of our case study (cf., Sec. I-B) into research questions (Sec. III-A). Based on these questions, we select and describe the CloudStore case and involved subjects (Sec. III-B). We subsequently formulate CloudStore-specific metrics to collect the empirical data for answering our research questions (Sec. III-C). We particularly specify how we analyze this data and which answers we expect to our questions (Sec. III-D). A description of our validity procedures concludes our case study design (Sec. III-E).

A. Research Questions

Our goal (Sec. I-B) targets the applicability of SPE for planning redeployments. Two crucial criteria for applicability are the achieved prediction accuracy and the needed effort [17]. Following the GQM method [31], we can accordingly derive two research questions from our goal: $Q_{\textbf{Accuracy}}$ ("What is the accuracy of SPE in our context?") and $Q_{\textbf{Effort}}$ ("What is the effort for applying SPE in our context?") where "our context" refers to planning redeployments of realistic business information systems to IaaS environments. Table I summarizes these questions, along with the metrics and hypotheses we associate to them in the following sections.

B. Case and Subjects Selection

Case. We select CloudStore [2], a CPU-bound 3-tier online bookstore where customers can search and order books, as a suitable case for answering our research questions. CloudStore's implementation is based on an implementation of the TPC-W benchmark [14] and is envisioned to be redeployed from on-premise to IaaS environments. Accordingly, CloudStore has two advantages: (1) it refers to the well-specified TPC-W benchmark [29] that is popular both in academia and industry and (2) it represents a typical business information system for which redeployment needs to be planned, thus, fitting the context of our goal.

CloudStore's "BrowsingMix" workload [29] resembles its typical usage: browsing through the book catalogue and ordering some books. We reuse this workload with a load of 300 concurrent users during our SPE efforts.

Moreover, we investigate the scenarios we previously motivated in Sec. I-C, i.e., (I) a classical on-premise setup for which we engineer the initial CloudStore performance model as well as (II) OpenStack and (III) Amazon EC2 for which we plan to reuse the performance model of the on-premise scenario. Table I lists these scenarios in variable $V_{\textbf{Scenarios}}$.

Subjects. The author of this paper and his colleague act as subjects because we conducted the necessary SPE steps on our own. Being the first case study that applies SPE for planning redeployments to IaaS environments, this subject selection is a good first baseline for effort estimation.

C. Data Collection Procedure(s)

Data collection procedures specify how the empirical data for answering research questions is gained [26]. Fitting to the GQM method [31], we specify these procedures via metrics aligned to our research questions (cf. Table I).

1) Accuracy Question Metrics: The first accuracy metric, $M_{\textbf{Prediction error}}$, measures the relative error between analysis results and measurements in terms of median response times of CloudStores "Browsing Mix" workload. Measuring the prediction error is a common practice in SPE [28], [11] and the median is often less misleading than the mean [8].

The second accuracy metric, $M_{\textbf{Performance isolation}}$, measures the standard deviation in response times for one-user closed workload requests to CloudStore's homepage over 30 minutes, i.e., during this time, a single user repeats requesting CloudStore's homepage just after a former request was processed. Performance isolation indicates how controlled a scenario of $V_{\textbf{Scenarios}}$ can be analyzed. For example, high values within a multi-tenant IaaS environment like Amazon EC2 indicate that other tenants could negatively influence performance of this environment. ($M_{\textbf{Performance isolation}}$ only serves as an indicator, i.e., figuring out the concrete reasons is a subsequent task.)

2) Effort Question Metrics: Martens et al.'s evidence that effort metrics "time for model creation" and "size of model" positively correlate [19] allows us to triangulate results regarding effort over such metrics (cf. Sec. II).

The first effort metric, $M_{\textbf{Time on task}}$, accordingly measures the time spend on the typical SPE tasks [28] "model construction", "model calibration", and "model evaluation".

TABLE I
SUMMARY OF QUESTIONS Q_{ACCURACY} AND Q_{EFFORT} INCLUDING THEIR METRICS, VARIABLES, AND HYPOTHESES

$Q_{\textbf{Accuracy}}$	**What is the accuracy of software performance engineering (SPE) in our context?**
$M_{\textbf{Prediction error}}$	Average, relative prediction error (see [11, p. 37]) between analysis results and actual measurements in terms of median response times of CloudStore's "Browsing Mix" workload.
$M_{\textbf{Performance isolation}}$	Standard deviation (response times) for one-user, 30 minute closed workload requests to CloudStore homepage.
$V_{\textbf{Scenarios}}$	Independent variable: different scenarios with levels $\{on\text{-}premise, OpenStack, EC2\}$.
$H_{\textbf{Sufficient accuracy}}$	When following SPE, analysis results are sufficiently accurate for every scenario of $V_{\textbf{Scenarios}}$. *Rejection:* $M_{\textbf{Prediction error}}$ gives an average prediction error over 30% [21] for any scenario of $V_{\textbf{Scenarios}}$.
$H_{\textbf{Stable environments}}$	All environments—even IaaS environments—provide stable measurements. *Rejection:* $M_{\textbf{Performance isolation}}$ gives a standard deviation over 0.1 seconds [6], [10] for any scenario of $V_{\textbf{Scenarios}}$.
$Q_{\textbf{Effort}}$	**What is the effort for applying software performance engineering (SPE) in our context?**
$M_{\textbf{Time on task}}$	Exclusive time on task, i.e., time from start to finish of a task minus time not spend on task during this interval.
$M_{\textbf{Size: Components}}$	Final number of Components (i.e., types of components) within Palladio Repository model.
$M_{\textbf{Size: Assembly Ctx.}}$	Final number of Assembly Contexts (i.e., instances of components) within Palladio System model.
$M_{\textbf{Size: Operations}}$	Final number of Operations as sum over all Interfaces (provided by components) in Palladio Repository model.
$H_{\textbf{Effort-size correlation}}$	Size metrics $M_{\textbf{Size: Components}}$, $M_{\textbf{Size: Assembly Contexts}}$, and $M_{\textbf{Size: Operations}}$ positively correlate with the time metric $M_{\textbf{Time on task}}$ for the $on\text{-}premise$ scenario. *Rejection:* No positive correlation is identified, i.e., the correlation coefficient is *not* greater than 0.0.
$H_{\textbf{Effort can be lowered}}$	Reusing the $on\text{-}premise$ model is accurate while lowering the effort for scenarios $OpenStack$ and $EC2$. *Rejection:* $H_{\textbf{Sufficient accuracy}}$ falsified and/or $M_{\text{Time on task}}$ lower for $on\text{-}premise$ than for $OpenStack$ and/or $EC2$.

The remaining three effort metrics ($M_{\textbf{Size: Components}}$, $M_{\textbf{Size: Assembly Contexts}}$, and $M_{\textbf{Size: Operations}}$) measure effort in terms of model size. The three size metrics differ in the Palladio construct they count (Components and Assembly Contexts are Palladio elements to model software component types and their run-time instances; Operations are provided by such components as specified within their interfaces).

D. Analysis Procedure(s)

Analysis procedures allow to interpret empirical data gained from metric measurements [26]. To specify these procedures, we follow again the GQM method [31]: for each question, we formulate our expected answers as hypotheses. Along with hypotheses, we state appropriate rejection criteria (cf., Table I).

1) Accuracy Question Hypotheses: According to our hypothesis $H_{\textbf{Sufficient accuracy}}$, we expect to get sufficiently accurate results for each scenario—based on our previous success with SPE in virtualized environments [18]. For defining "sufficiently accurate", we use an upper prediction error threshold of 30% as Menascé and Almeida suggest for analysis models of web applications like CloudStore [21].

Hypothesis $H_{\textbf{Stable environments}}$ expects a deviation below 0.1 seconds. Such a deviation was observed by Barham et al. [6] for XEN virtualization (like in the $OpenStack$ scenario) and is in line with our experience [18]. We expect $EC2$ to behave similarly, i.e., other tenants have little impact, as evidenced by Dejun et al. [10]. Being isolated and non-virtualized, the $on\text{-}premise$ scenario should have an even lower deviation.

2) Effort Question Hypotheses: As motivated along with size metrics for effort, hypothesis $H_{\textbf{Effort-size correlation}}$ expects these metrics to positively correlate with $M_{\textbf{Time on task}}$. This correlation would allow future studies to estimate effort based on size, i.e., without depending on human interactions.

In hypothesis $H_{\textbf{Effort can be lowered}}$ we expect to reuse parts of the performance model we created for the $on\text{-}premise$ scenario for the two other scenarios and without becoming inaccurate. Successful reuse indicates that redeployment planning has a lower SPE effort.

E. Validity Procedure(s)

Especially measuring effort comes with high validity threats because effort is often measured based on human interactions. Human-based measurements can, e.g., result in poor reproducibility of case studies. To cope with such threats, we triangulate (cf. Sec. II) results from human-based measurements regarding effort ($M_{\textbf{Time on task}}$) and size-based measurements ($M_{\textbf{Size: Components}}$, $M_{\textbf{Size: Assembly Contexts}}$, and $M_{\textbf{Size: Operations}}$).

Another aspect is the quality of the case study itself. We tried to maximize its quality by rigorously following case study reporting guidelines [26] and letting colleagues review our case study protocol two times regarding reproducibility.

IV. RESULTS

We report the results of our case study in this section. Our report includes creation of performance models and measurement of metrics (Sec. IV-A), analysis of these measurements (Sec. IV-B), and interpretation of analysis results to answer our research questions for goal attainment (Sec. IV-C). We also discuss potential threats to validity of our study (Sec. IV-D).

A. Execution

In this section, we describe how we create our performance models for CloudStore. We consider models for on-premise, OpenStack, and Amazon EC2 as target environments.

For the on-premise scenario, we execute the complete SPE process [28] illustrated in Fig. 1: (1) construct basic performance model, (2) calibrate model with target environment

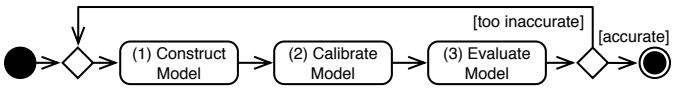

Fig. 1. Performance model creation in software performance engineering [28].

data, and (3) evaluate model accuracy by comparing analysis results with target environment measurements for a representative workload. We repeat this process as long as action (3) indicates that the performance analysis was too inaccurate.

For the remaining scenarios, we reuse most parts of the so-created model; we only recalibrate CPU processing rates within our deployment model to reflect different environments and measure the scenario-dependent metrics (Sec. IV-A4).

1) Execution: (1) Construct Model: Based on analyzing the source code of the CloudStore implementation [14], we created the Palladio performance model illustrated in Fig. 2. Figure 2 combines UML component, deployment, and use-case diagrams: it shows (1) the static system structure of the CloudStore system via connected component instances, (2) the deployment of these components to system resources (Web & Application Server, Image Server, Database Server), and (3) the system entry point for customers (interfaces for accessing web pages for books, home page, shopping carts, orders). We specified each of these views in a dedicated Palladio model with a behavior as described next.

Customers enter the system via the web pages provided by the BookPages, HomePage, ShoppingCartPages, and OrderPages components deployed on the Web & Application Server. BookPages provides operations regarding books (e.g., to query book details or search for books). The HomePage component shows CloudStore's home page, which welcomes its customers and displays possible book categories for browsing. ShoppingCartPages allows customers to register, add books to a shopping cart, and to check-out the shopping cart. Afterwards, OrderPages allows to follow-up on the order. BookPages, HomePage, and ShoppingCartPages additionally require the PromotionalProcessing component to receive an advertisement area for related books. The before mentioned components require operations of the ImageLoading and Database components as allocated on the Image and Database Server, respectively. ImageLoading provides access to image files, e.g., needed for book covers, and CloudStore's Database stores entries for books, customers, shopping carts, and orders.

Calls to the Database are intercepted by a DatabaseAccess component that manages database connections. DatabaseAccess receives [returns] such connections from [to] the DBConnectionPool component. Also Web & Application, Image, and Database Server use pools for handling customer requests (WebServerConnection, ImageServerConnection, DBServerConnection). These pools (gray-colored components in Fig. 2) are typical performance factors as their pool-size limits the amount of requests that can be processed in parallel.

Palladio supports acquiring and releasing connections from these resource pools within so-called *service effect specifications* (SEFFs). SEFFs specify the behavior (control and data flow) of component operations similar to UML activity diagrams. In our model, every interaction requires the acquisition of connections and a release once the interaction ends.

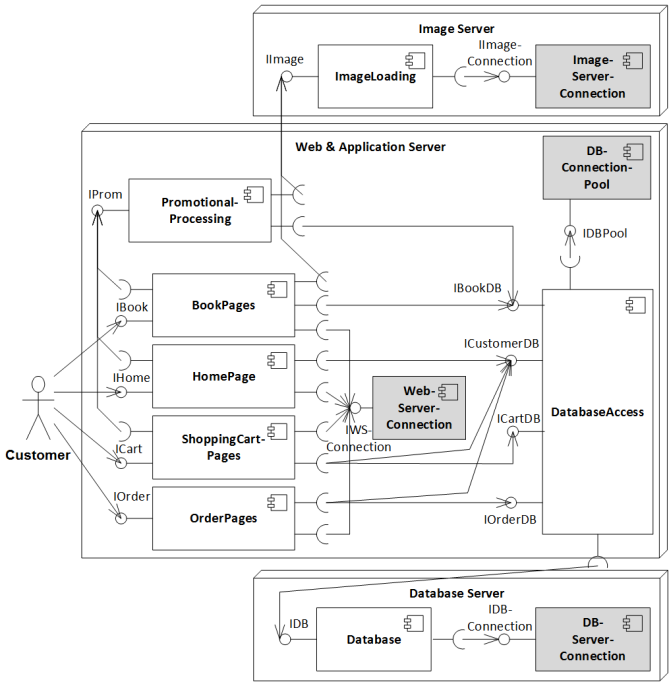

Fig. 2. System overview of constructed CloudStore model (simplified).

Figure 3 illustrates this pattern for SEFFs of front-end component operations that interact with database and image components. Actions (1) to (3) model the performance impact of creating an HTML page for customers while action (4) models the performance impact of subsequently resolving image references. These two phases—receiving an HTML page and subsequently its references—reflect the typical behavior of web browsers [13].

We also used the concept of connection pools to realize transactions; modeled as connection pools with pool-size one. Our transactions include write operations for creating new customers, books, and orders. Such transactions typically have a major performance impact.

Besides modeling the static structure and the behavior of CloudStore, we also modeled the "Browsing Mix" customer

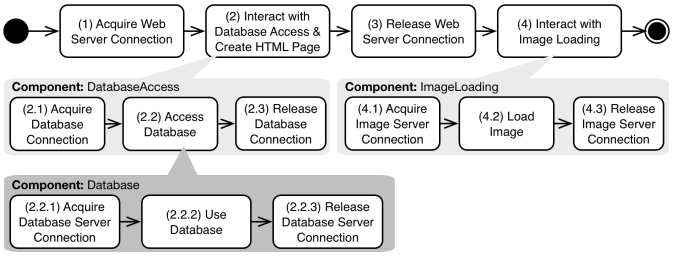

Fig. 3. Behavior of front-end components interacting with database and image components.

workload according to the TPC-W specification [29]. "Browsing Mix" is a workload where customers browse through the book catalogue and occasionally order books. In Palladio's usage scenario model, we specified the probabilities with which customers call the 14 CloudStore operations[2]. For example, the home page is requested in 29.00% while orders are placed in only 0.69% of all cases [29].

2) Execution: (2) Calibrate Model: We have to calibrate the initial Palladio model after its construction. Such a calibration enriches the model with concrete resource demands to active resources, e.g., to a CPU. Resource demands can either be estimated by experts or measured in the target environment (based on prototype or actual implementations). As we have the actual implementation available, we follow the latter approach and use it to measure demands directly in the target environment; this is generally more accurate [28]. For our calibration, we accordingly executed the following actions:

Setup on-premise environment. We used three equivalent computers for each of CloudStore's tiers. Each computer operates with Ubuntu 14.04, 2.67 GHz 2-core CPUs, and 2x 4.00 GiB main memory. We synchronized these values with our performance model for CloudStore.

Deploy CloudStore servlets into environment.
For the front-end, we deployed CloudStore's servlets on a TomCat 7.0.30 (Web & Application Server). For the back-end, we used an Apache HTTP 2.2.29 server (Image Server) and MySQL 5.5.27 (Database Server).
We limited the size of pools to 250 server connections (for servers) respectively to 100 database connections (for the Database Access component). We synchronized these values with our performance model.

Generate workload and measure resource demands.
For each operation of interest, we generated dedicated workloads to measure the exclusive resource demands for that operation (with a 10-minute warm-up phase). We used Kieker 1.9 [30] as monitoring framework. Kieker sent response time measurements to a home-brew SPE tool.
For requests to Image and Database Server, we only measured their response times as externally observed by the Web & Application Server, i.e., without conducting more fine-grained measurements on these servers. Because our focus is on software applications, we wanted to focus on the behavior of the Web & Application Server itself and avoided to dig deeper in the file management of Image Server respectively database details.

Put measured demands into performance model.
Our home-brew SPE tool allowed us to live-monitor and store received measurements. Based on this data, we calculated probabilistic distribution functions for response times and enriched our performance model with these functions. Because CloudStore is CPU-bound, we enriched our model only with functions for CPU demands.

[2]The system interfaces IBook, IHome, ICart, and IOrder warrant each of the 14 CloudStore operations. Operations are realized within respective components (each operation is modeled as SEFF according to Fig. 3).

All of these steps are documented in online screencasts [3].

3) Execution: (3) Evaluate Model: We evaluated the performance model's accuracy with the "Browsing Mix" workload and a load of 300 users. In contrast to the calibration workloads (which are separated per operation), such a workload is considerably more complex and, thus, suited for evaluation.

Because the Palladio-Bench allowed us to generate JMeter workload scripts from our "Browsing Mix" usage scenario model, we used Apache JMeter 2.12 [4] as workload generator and for response time measurements. We had to iterate the model creation process two times in a 2-hour effort as we discovered modeling mistakes during evaluation (e.g., we forgot two times to return acquired connections, resulting in drastically increasing response times). After these two iterations, we achieved a prediction error ($M_{\text{Prediction error}}$) of 2.76%. As this value was below our 30% threshold, we successfully finished with our model creation process.

We additionally ran performance isolation measurements ($M_{\textbf{Performance isolation}}$) with a one-user closed workload to the CloudStore homepage. We used JMeter for workload creation.

4) Execution: Other Scenarios: In scenarios II and III (OpenStack and Amazon EC2), we reused most of the on-premise model. We only had to execute the following actions:

Scenario II: Setup OpenStack environment. In scenario II, we installed OpenStack nodes into the infrastructure we previously used for our on-premise setup; we deployed OpenStack management components (e.g., a network manager) to additional servers. We installed OpenStack according to the "Juno" guide for Ubuntu 14.04 [24]. After installation, we used three equivalent virtual machines (VMs) for each of CloudStore's tiers. Each VM operates with Ubuntu 14.04, 2x 2.67 GHz virtual CPUs, and 1x 4.00 GiB main memory. The only differences to scenario I are the OpenStack-layer and different computing resource characteristics. We synchronized the latter values with our scenario II performance model but ignored potential OpenStack overhead. Our evaluation shows that this overhead is indeed negligible.

Scenario III: Setup Amazon EC2 environment.
In scenario III, we rented three "t2.medium" virtual machines for each of CloudStore's tiers within Amazon EC2. Each virtual machine operates with Ubuntu 14.04, 2x 2.5 GHz virtual CPUs, 1x 4.00 GiB main memory, and is shared (multi-tenant hardware). We additionally rented a fourth "t2.medium" machine for deploying JMeter that allowed us to generate load from within Amazon EC2. This way we were able to resemble our setup for scenario I and II where we had direct access to our local infrastructure.
The only differences to scenario I are the EC2-layer and different computing resource characteristics. Again, we synchronized the latter values with our scenario III performance model but ignored potential EC2 overhead. In contrast to the overhead from OpenStack, we expected this overhead to be more crucial because of the uncontrolled nature of third-party hosting. For example, performance isolation against other tenants can be an issue. However, our evaluation shows that also Amazon EC2's overhead is negligible.

Deploy CloudStore servlet into environment.
We deployed scenarios II and III analogous to scenario I.

Re-calibrate CPU processing rate. After we deployed the CloudStore implementation in the target environment, we measured $M_{\textbf{Performance isolation}}$. We used these measurements to re-calibrate the CPU processing rate of scenario II and III models, respectively. This was the only change we applied compared to the on-premise model.

Evaluate Model. The resulting models allowed us to measure $M_{\textbf{Prediction error}}$ for scenarios II and III. We did not had to iterate the performance model creation process as our prediction error was below the 30% threshold.

B. Analysis

The empirical data collected during model creation (previous section) serves us as input to the analysis we conduct in this section. Table II provides an overview of this data; depicted measurements directly correspond to the metrics of our GQM plan (cf. Tab. I). The first column specifies the metric of interest while the remaining columns provide the measurements per scenario. We describe this Tab. II in the following subsections; we leave the interpretation of our descriptions to a dedicated interpretation section (Sec. IV-C).

1) Accuracy Measurements: The upper row in Tab. II depicts accuracy measurements. The performance model for the on-premise scenario allows to analyze the performance of CloudStore with high accuracy ($M_{\textbf{Prediction error}}$ is only $\sim$3.22%; the lowest prediction error among the scenarios). But also the other scenarios have a prediction error below 12%. Therefore, we are below our pre-defined threshold of 30% and consequently accept our hypothesis $H_{\textbf{Sufficient accuracy}}$.

The box plots in Fig. 4 support these results: for each scenario, the median response time values of measurement and analysis differ only slightly (the median is represented by the thick horizontal line within box plots). Interestingly, we observe a wide distribution of response times for the on-premise environment, as evidenced by the quartiles of the corresponding "measurement" box plot. The other scenarios have a narrower distribution in comparison; especially the Amazon EC2 scenario (discussion in Sec. IV-C).

TABLE II
METRIC MEASUREMENTS FOR ALL SCENARIOS

	Scenario I: On-Premise	Scenario II: OpenStack	Scenario III: Amazon EC2
$M_{\textbf{Prediction error; Median}}$	3.22%	5.41%	11.11%
$M_{\textbf{Prediction error; Mean}}$[1]	-67.50%	57.50%	88.23%
$M_{\textbf{Performance isolat.}}$	0.001 sec.	0.012 sec.	0.002 sec.
$M_{\textbf{Time on task}}$			
- (1) Construct Model	$\sim$ 83 h.	0 h.	0 h.
- (2) Calibrate Model	$\sim$ 121 h.	$\sim$ 52 h.	$\sim$ 3 h.
- (3) Evaluate Model	$\sim$ 10 h.	$\sim$ 2 h.	$\sim$ 2 h.
- Total	$\sim$ 214 h.	$\sim$ 54 h.	$\sim$ 5 h.
- Total (% Scenario I)	100%	25%	2%
$M_{\textbf{Size: Components}}$	12	12	12
$M_{\textbf{Size: Assembly Ctx.}}$	14	14	14
$M_{\textbf{Size: Operations}}$	55	55	55

[1] Mean measurements are discussed as threat to construct validity in Sec. IV-D.

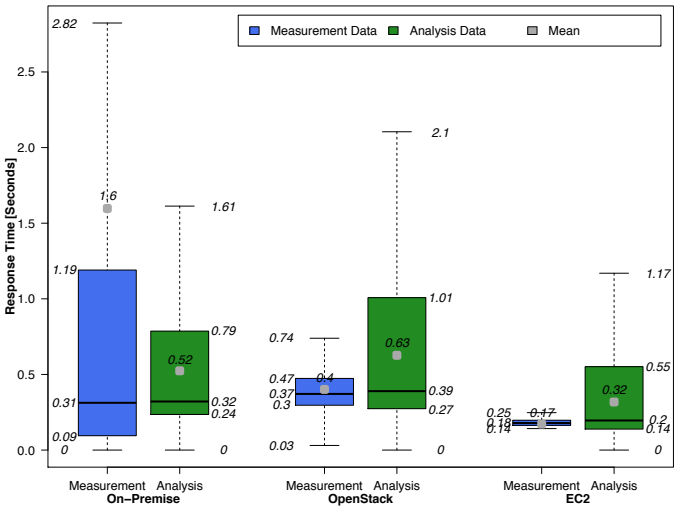

Fig. 4. Comparison of measurement and analysis response times.

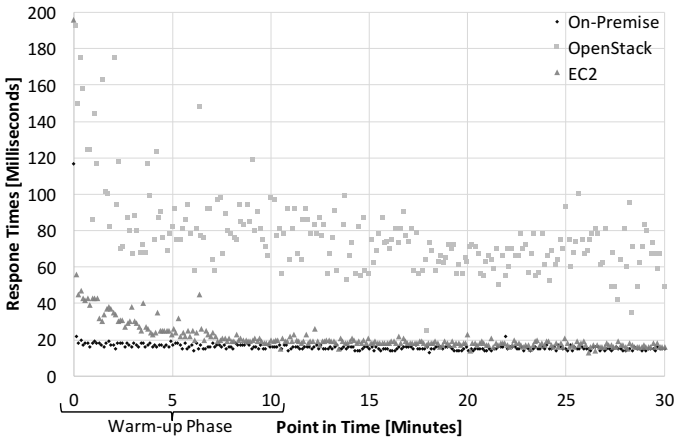

Fig. 5. Response times for one-user closed workload requests to CloudStore's homepage for all scenarios.

The response time measurements for the performance isolation metric ($M_{\textbf{Performance isolation}}$) show that each scenario was below our 0.5 seconds threshold for repeating one-user requests to CloudStore's homepage. The on-premise and Amazon EC2 scenarios have a standard deviation of only 0.001 and 0.002 seconds in response times, respectively; the OpenStack scenario has a standard deviation of 0.012 seconds. Being below our threshold, we also accept $H_{\textbf{Stable environments}}$.

The response time graph in Fig. 5 visualizes and confirms these results: the graph denotes the point in time (X-axis; relative to experiment start) when we received a measurement with a given response time (Y-axis). The graph shows that—after a warm-up phase of 10 minutes—response times became stable. For the on-premise and EC2 scenarios, response times stabilized around the 20 milliseconds response time mark. For the OpenStack scenario, we observe a wider distribution around the 70 milliseconds mark (fitting to a higher $M_{\textbf{Performance isolation}}$). We discuss this wider distribution later in the interpretation section (Sec. IV-C).

2) Effort Measurements: In the on-premise scenario, we had relatively high effort for model creation ($M_{\textbf{Time on task}}$ is $\sim$214 hours; this is considerably higher than measured in the other scenarios with $M_{\textbf{Time on task}}$ $\sim$54 and $\sim$5 hours, respectively). Due to model reuse, we did not had any effort to construct models for the OpenStack and Amazon EC2 scenario (action (1)). However, we needed $\sim$52 hours for calibrating the OpenStack model because we first had to setup the OpenStack environment for our calibration. This setup caused $\sim$51 hours of the calibration step (action (2)). In Amazon EC2, we had less effort ($\sim$2 hours) to setup the relevant virtual machines. The evaluation for the on-premise scenario (action (3)) consumed the most time among the scenarios as we had to iterate the performance model creation process two times (see Sec. IV-A3).

In summary, we required less effort for the OpenStack (75% less time) and Amazon EC2 (98% less time) scenarios compared to the on-premise scenario. Given that we already accepted $H_{\textbf{Sufficient accuracy}}$, we can therefore accept $H_{\textbf{Effort can be lowered}}$ as well: we indeed can save effort by reusing models without loosing too much accuracy.

Finally, the corresponding size metrics do not differ among the investigated scenarios at all. We were surprisingly able to reuse nearly the complete performance model of scenario I such that we did not had to alter any components, assembly contexts, and operations (we just had to re-calibrate CPU processing rates in our deployment models). Thus, we reject $H_{\textbf{Effort-size correlation}}$ as we cannot positively correlate size metric measurements with measurements for $M_{\textbf{Time on task}}$.

C. Interpretation

Based on the data analysis in the previous section, we can now proceed to answer the questions of our GQM plan.

1) What is the accuracy of software performance engineering in our context? ($Q_{Accuracy}$): The context of $Q_{\textbf{Accuracy}}$ targets redeployments to IaaS environments. Even for such environments, our analysis results indicate that SPE is applicable with sufficient accuracy: we both observe a low prediction error (below 12%) and stable base data for model calibration (maximum of 0.012 seconds standard deviation in response times). Therefore, we particularly accepted both hypotheses of $Q_{\textbf{Accuracy}}$, allowing us to answer it with "high".

This evidence is in contrast to the suspected performance unpredictability of IaaS environments [5] and, in fact, a sign that such environments are better predictable than is nowadays assumed. A typical factor raised when discussing the predictability of IaaS environments is the degree from which multiple tenants can be performance-isolated from each other. In our Amazon EC2 scenario, we deployed CloudStore in such an environment (see Sec. IV-A4), however, did not find evidence that would indicate that performance isolation would be a problem (at least for the time interval/availability zone we investigated: 2015/07/17 20:25 - 20:55 CET / Frankfurt). For example, the performance isolation measurements ($M_{\textbf{Performance isolation}}$) indicate that performance in such environments can be just as stable as in on-premise environments.

It is further interesting that response times within the on-premise environment are widely distributed (illustrated in Fig. 4; also see the "Documentation/Overview.pdf" at [1] for a response time graph). We suspect that contention effects, e.g., at the database server, caused these deviations. We did not further investigate this issue as our model evaluation (last step of SPE) already indicated that we were accurate enough.

Another interesting observation is the wide distribution of response times for the one-user closed workload to CloudStore's homepage for the OpenStack scenario (illustrated in Fig. 5). For this scenario, we installed OpenStack into the hardware we used for the on-premise scenario. OpenStack then added a virtualization layer on top of this hardware. Accordingly, we can explain the wider distribution by virtualization overhead. Such an explanation is in line with one of our previous results where we also observed wider distributions in virtualized environments [18]. We explain the nonetheless narrow distribution for the EC2 scenario by the higher processing rates of provisioned EC2 resources that potentially even-out virtualization overheads.

2) What is the effort for applying software performance engineering in our context? (Q_{Effort}): In $Q_{\textbf{Effort}}$, we ask for the effort of engineering a performance model from scratch compared to reusing such a model for deployments to different target environments; in our context IaaS environments. Our results indicate that effort is up to 98% lower when reusing models (in terms of engineering time spend). Thus, we accepted $H_{\textbf{Effort can be lowered}}$ and answer $Q_{\textbf{Effort}}$ with "redeployment planning requires significantly less SPE effort when a model of the original system already exists".

However, we had to reject $H_{\textbf{Effort-size correlation}}$ because we did not observe a positive correlation between time and size metrics. In consequence, software architects cannot apply the size metrics of Martens et al. [19] for effort evaluation of redeployment planning. Instead, we suggest to quantify the differences between different deployment models, thus, reflecting the actual model changes. In our case study, we did not adapt components, assembly contexts, or operations, therefore would quantify the difference for these models with "0". We only modified processing rates of the 3 modeled resources, e.g., allowing to quantify the "number of difference with respect to processing rates" with "3". Such quantifications better reflect the low effort we had for model adaptation.

With model differences, architects can triangulate effort evaluations with time-based metrics like $M_{\textbf{Time on task}}$. Triangulation can lower systematic errors, e.g., when measuring time of human interactions. In our case study, triangulation is interesting when comparing the OpenStack and EC2 scenarios: even though model differences provide the same values for both scenarios, we had 49 hours more effort for the OpenStack scenario. We derive two lessons learned:

- Model differences focus on particular effort-creating aspects. The size metrics of Martens et al. [19] focus on structural changes of the software model only. Other aspects like changed deployments and the needed effort for evaluating performance models are disregarded.

- Time metrics aligned to the model creation process (Fig. 1) cover effort for setting up environments. Given that we spend 94% of the time for the OpenStack scenario on environment setup, the "calibrate model" action appears to be too coarse-grained. Instead, we suggest to explicitly split this task into "Set Up Target Environment" and "Calibrate Model" tasks.

D. Evaluation of Validity

This section describes our main threats to validity, classified in [33]: (1) conclusion validity, (2) internal validity, (3) construct validity, and (4) external validity.

Conclusion validity is the degree to which conclusions relating measurements to the CloudStore case are correct. Being typical for case studies, our main threat to this validity is "low statistical power" [33] because we were the only subjects that executed our case. This threat is especially relevant for time-based effort metrics—other subjects may need significantly more or less time for the tasks we conducted. As we are architects with an average of ten years experience, we expect that our effort measurements can serve as first base line for architects with similar experience. Additional studies and controlled experiments are needed to confirm this hypothesis.

Internal validity is the degree to which conclusions relating measurements to the CloudStore case are causal (given an observed relationship). Such a threat is "maturation" [33], which describes effects on subjects as time passes, e.g., tiredness. During our case study, we performed repetitive tasks such as calibrating resource demands over 55 operations. We countered tiring effects for such repetitive tasks by limiting the amount of continuous work on these tasks to 3 hours; however, we acknowledge that such tasks are indeed tiring.

A similar threat is "history" [33]: after setting up the environment for the on-premise scenario (TomCat, Apache, and MySQL servers), the knowledge we gained in doing so lowered the effort for setting up the remaining environments (where we also configured such servers). Despite this threat, we expect our interpretation to remain valid because setting up these servers was a minor tasks during model calibration (~ 4 hours for the on-premise scenario).

Construct validity is the degree to which conclusions about derived theories are correct. Like most case studies, we face the "mono-operation bias" [33], i.e., the risk that sticking only to a single case may under-represent the derived theory. In our case study, we only investigated the CloudStore case as a redeployment scenario to IaaS environments. Based on our results, we derived that performance models can efficiently be reused in such scenarios. Whether the (CPU-bound) CloudStore case was under-representative to draw such conclusions needs to be clarified in further empirical studies.

Another threat that falls in this category is the "mono-method bias" [33], i.e., sticking to a single metric that potentially provides misleading results. For the prediction error, we only sticked to *median* values. However, the *mean* measurements of $M_{\textbf{Prediction error}}$ in Tab. II indicate higher errors (up to 88%). An inspection of the underlying response times (cf. "Documentation/Overview.pdf" at [1]) reveals that their distribution is left-skewed and, thus, mean values are misleading [8]. If only means were used, as a consequence, our results would have been biased.

External validity is the degree to which internally valid results can be generalized to industrial practice. An example of such a threat is the "interaction of setting and treatment" [33]. In our case study, such a setting refers to the investigated target environments (on-premise, OpenStack, Amazon EC2). While OpenStack and Amazon EC2 represent environments found in industrial practice, the hardware we used for the on-premise and OpenStack environments may differ in industrial practice. If hardware supports unconsidered performance-impacting features, e.g., CPUs with dynamic frequency scaling, calibration data may not be as reusable as in our case study.

V. Conclusions and Future Work

In this section, we conclude our case study with a summary (Sec. V-A), the relation of our study to existing empirical evidence (Sec. V-B), an overview of expected implications (Sec. V-C), limitations (Sec. V-D), and future work (Sec. V-E).

A. Summary of Conclusions

In this paper, we conduct a case study to explore the applicability of software performance engineering (SPE) to redeployment planning. We focus on redeploying existing software to IaaS environments. Our results indicate that adaptations to existing performance models cause low effort (up to 98% less effort compared to engineering a model from scratch) while providing accurate performance predictions (with a relative accuracy error below 12%).

Thus, existing performance models can become highly beneficial when evaluating different deployment options to IaaS environments. Existing models particularly allow to assess different IaaS environments before actually redeploying existing implementations. This kind of assessment reduces risks of discovering performance problems too late, e.g., during operation. When discovered too late, such problems potentially lead to expensive reimplementations and unsatisfied customers.

B. Relation to Existing Evidence

Our results confirm existing studies that apply SPE on virtualized environments [18], [12]. Even in such environments, performance models can provide accurate results.

Martens et al. [19] quantify the benefits of reusing existing performance models in terms of effort reduction. Our study both supports and extends their result: our effort comparison indicates that reuse can reduce effort by up to 98% for redeployment planning. We particularly followed Martens et al.'s suggestion to use Palladio as SPE approach, due to its good reuse features.

Moreover, our data suggests that Smith and Williams' [28], [32] estimate that SPE efforts may cause up to 10% of total project costs is an over-estimation for redeployment planning. Given that we could lower SPE effort up to 98% in our case study, it is likely that significantly less costs accrue for SPE than 10% of total project costs for redeployment planning.

C. Impact/Implications

Software architects can practically use our results to plan redeployments to IaaS environments. First, our metrics allow architects to assess model accuracy and effort. Such an assessments helps in assuring sufficient quality and in planning future SPE activities. Second, our results show that architects have only little effort for adapting existing performance models. This insight lowers barriers hindering architects to conduct SPE activities and enables them to benefit from early quality analyses. Third, we expect that our calibration data for EC2 is generic and can be reused by other architects.

Our findings also help project managers to estimate the efforts for SPE activities for redeployment planning. Our metrics particularly help managers to control such activities.

During our study, we highlighted observations that call for further empirical evidence. Researchers can use these observations to formulate hypotheses for their own studies. They can particularly use our publicly available material [1], [2], [3] (model, implementation, raw measurement data, screencasts, etc.) to reassess and to build-up on our findings.

D. Limitations

Our results are based on one existing business information system (CloudStore), two IaaS environments (OpenStack and Amazon EC2), and one SPE approach (Palladio). It is open whether our results can be generalized to other systems, environments, approaches, and to architects with different expertise as we have (cf. Sec. IV-D).

E. Future Work

Future work should conduct controlled experiments to cope with some of our threats to validity, especially to increase generalizability. For example, other IaaS environments, more and different metrics, and other SPE approaches can be considered. Such efforts strengthen our results and help building solid SPE theories. Particularly further case studies are needed that investigate the effort of software architects with SPE activities. This lack of significant conclusion makes it hard to convince projects managers that SPE is worth its effort.

Finally, we plan to extend our case study by investigating typical cloud computing characteristics. For example, cloud computing environments necessarily come with elasticity management systems that allow resources (e.g., virtual machines) to be replicated autonomously and rapidly, e.g., to cope with sudden workload peaks. Reflecting such a behavior requires effort-intensive additions to our performance model; architects need to specify (potentially complex) self-adaptation rules that model the replication behavior.

References

[1] Case Study Material: CloudStore documentation, source code, deployment scripts, analysis models, and raw measurement data. https://github.com/CloudScale-Project/CloudStore and https://github.com/CloudScale-Project/Examples/tree/master/CloudStore/analyser.

[2] CloudStore. http://www.cloudwatchhub.eu/cloudscales-cloudstore.

[3] Screencasts. http://www.cloudscale-project.eu/results/screencasts/.

[4] Apache Foundation. Apache JMeter. http://jmeter.apache.org/.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, and R. Katz et al. A view of cloud computing. *Comm. ACM*, 53(4):50–58, 2010.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[7] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *JSS*, 82(1), Jan. 2009.

[8] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev. Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. *TSE*, 41(2):157–175, Feb 2015.

[9] T. de Gooijer, A. Jansen, H. Koziolek, and A. Koziolek. An industrial case study of performance and cost design space exploration. ICPE '12, pages 205–216, New York, NY, USA, 2012. ACM.

[10] J. Dejun, G. Pierre, and C. Chi. Ec2 performance analysis for resource provisioning of service-oriented applications. In *ICSOC '09*, volume 6275 of *LNCS*, pages 197–207. Springer Berlin Heidelberg, 2010.

[11] N. J. Gunther. *Guerrilla Capacity Planning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[12] N. Huber, S. Becker, C. Rathfelder, J. Schweflinghaus, and R. Reussner. Performance modeling in industry: a case study on storage virtualization. In *SE '10*, volume 2, pages 1–10, May 2010.

[13] I. Jacobs and N. Walsh, editors. *Architecture of the World Wide Web, Volume One*. The World Wide Web Consortium (W3C), Dec. 2004.

[14] Java Middleware Open Benchmarking (JMOB). TPC-W benchmark: Java Servlets/MySQL implementation. http://jmob.ow2.org/tpcw.html.

[15] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. Reporting experiments in software engineering. In *Guide to Advanced Empirical Software Engineering*, pages 201–228. Springer London, 2008.

[16] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Perform. Eval.*, 67(8):634–658, Aug. 2010.

[17] S. Lehrig and S. Becker. Software architecture design assistants need controlled efficiency experiments: Lessons learned from a survey. FoSADA '15, pages 19–24, New York, NY, USA, 2015. ACM.

[18] S. Lehrig and T. Zolynski. Performance prototyping with protocom in a virtualised environment: A case study. In *Proceedings to Palladio Days 2011, 17-18 November 2011, Karlsruhe, Germany*, 2011.

[19] A. Martens, H. Koziolek, L. Prechelt, and R. Reussner. From monolithic to component-based performance evaluation of software architectures. *Empirical Softw. Eng.*, 16(5):587–622, Oct. 2011.

[20] M. Marzolla. *Simulation-Based Performance Modeling of UML Software Architectures*. PhD thesis, Università Ca' Foscari di Venezia, Feb. 2004.

[21] D. Menascé and V. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2002.

[22] D. Menascé, L. W. Dowdy, and V. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[23] J. Murphy. Performance Engineering for Cloud Computing. In N. Thomas, editor, *Computer Performance Engineering*, volume 6977 of *LNCS*, pages 1–9. Springer Berlin Heidelberg, 2011.

[24] OpenStack Foundation. OpenStack Installation Guide for Ubuntu 14.04. http://docs.openstack.org/juno/install-guide/install/apt/.

[25] C. Rathfelder, S. Becker, K. Krogmann, and R. Reussner. Workload-aware system monitoring using performance predictions applied to a large-scale e-mail system. In *WICSA/ECSA '12*, pages 31–40, Aug 2012.

[26] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Eng.*, 14(2):131–164, Apr. 2009.

[27] D. Sjøberg, T. Dybå, B. Anda, and J. Hannay. Building theories in software engineering. In *Guide to Advanced Empirical Software Engineering*, pages 312–336. Springer London, 2008.

[28] C. U. Smith and L. G. Williams. *Performance solutions: a practical guide to creating responsive, scalable software*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.

[29] Transaction Processing Performance Council (TPC). TPC-W benchmark (web commerce) specification version 1.8, Feb. 2002.

[30] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *ICPE 2012*, pages 247–248. ACM, Apr. 2012.

[31] R. van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. *Goal Question Metric (GQM) Approach*. John Wiley & Sons, Inc., 2002.

[32] L. G. Williams and C. U. Smith. Making the business case for software performance engineering. In *CMG '03, December 7-12, 2003, Dallas, Texas, USA*, pages 349–358, 2003.

[33] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.