Enriching Software Architecture Models with Statistical Models for Performance Prediction in Modern Storage Environments

Qais Noorshams, Roland Reeb, Andreas Rentschler,
Samuel Kounev, Ralf Reussner
Chair for Software Design and Quality
Karlsruhe Institute of Technology
Karlsruhe, Germany
[firstname.lastname]@kit.edu

ABSTRACT

Model-based performance prediction approaches on the software architecture-level provide a powerful tool for capacity planning due to their high abstraction level. To process the increasing amount of data produced by today's applications, modern storage systems are becoming increasingly complex having multiple tiers and intricate optimization strategies. Current software architecture-level modeling approaches, however, struggle to account for this development and are not well-suited in complex storage environments due to overly simplistic storage assumptions, which consequently leads to inaccurate performance predictions. To address this problem, in this paper we present a novel approach to combine software architecture-level performance models with statistical models that capture the complex behavior of modern storage systems. More specifically, we first propose a general methodology for enriching software architecture modeling approaches with statistical I/O performance models. Then, we present how we realize the modeling concepts as well as model solving to obtain performance results. Finally, we evaluate our approach extensively in the context of three case studies with two state-of-the-art environments based on Sun Fire and IBM System z server hardware. Using our approach, we are able to successfully predict the application performance within 20 % prediction error in almost all cases.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques, Performance attributes; D.2.11 [**Software Engineering**]: Software Architectures

Keywords

I/O, Statistical Model, Storage, Software Architecture, Performance, Prediction

CBSE'14, June 30–July 4, 2014, Marcq-en-Baroeul, France.

ACM 978-1-4503-2577-6/14/06 ...$15.00.
http://dx.doi.org/10.1145/2602458.2602475.

1. INTRODUCTION

Over the past couple of decades, the I/O resource demands of modern IT systems have grown exponentially [28]. Until the year 2020, the amount of digital data is expected to double every two years [9]. As a consequence of this development, storage systems have evolved significantly from simple disks to sophisticated tiered systems employing intricate caching and optimization algorithms.

With the increasing amount of I/O-intensive applications, however, performance modeling and evaluation techniques are required for capacity planning at deployment time as well as to guarantee the continuous compliance to Service-Level Agreements (SLAs) during operation as application workloads evolve. Component-based software architecture-level performance modeling techniques are popular approaches to address capacity planning issues at design time and during operation. It is clear and well-understood that the execution environment has a significant impact on the performance of a software system [15]. Current software architecture-level performance modeling approaches, however, struggle to account for the increasingly complex storage infrastructures and only provide rudimentary support for I/O performance prediction.

Existing approaches model the storage performance of I/O-intensive applications on a low abstraction level, e.g., [10, 16], however, it is difficult to include such models into software architecture models, because the required information between the two modeling abstraction levels needs to be synchronized. Such a synchronization is not straightforward and multiple questions arise, such as, for instance: i) What system information is required to enable fine-granular performance analysis? ii) What information needs to be part of the component interfaces? iii) How can the information be analyzed to reason about the end-to-end system performance?

To address this issue, in this paper we propose a novel approach for performance modeling of modern storage systems combining high-level component-based software architecture models with low-level statistical I/O models. More specifically, we first propose a general methodology for enriching software architecture modeling approaches with statistical I/O performance models. Statistical models are a powerful approach to capture the complex behavior of storage systems and they can be typically obtained in a fully automated manner. Then, we present how we realize the modeling concepts as well as model solving to obtain performance results. We

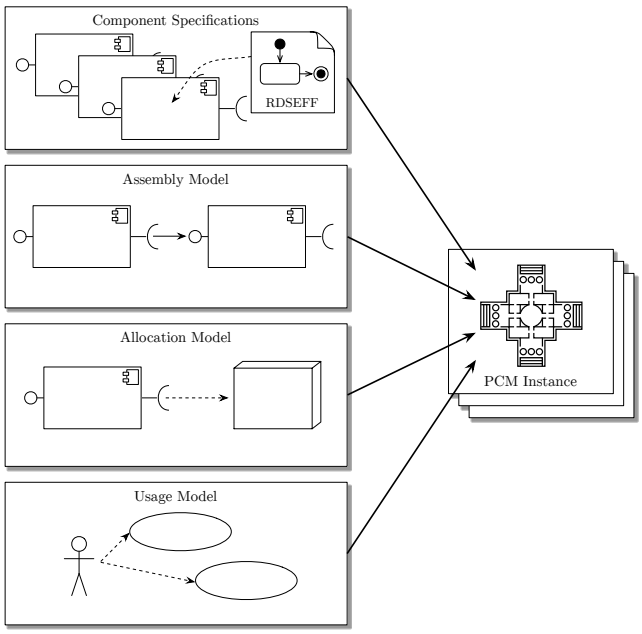


Figure 1: PCM Model Instance (Source: [3])

show how we model I/O requests on the architecture-level and how we map the requests to the required information on the storage level. Finally, we extensively evaluate our approach in the context of three case studies with two state-of-the-art environments based on Sun Fire and IBM System z server hardware. Using our approach, we are able to successfully predict the application performance within 20 % prediction error for both end-to-end response time as well as I/O response time in almost all cases.

In summary, the contribution of this paper is two-fold: i) We present a novel approach to integrate statistical I/O models into software architecture-level performance models allowing to predict the performance in modern storage environments. ii) We validate our approach in two state-of-the-art real-world environments based on Sun Fire and IBM System z server hardware.

The remainder of this paper is organized as follows: Next, we introduce the background of our approach in Section 2. In Section 3, we present our general methodology. Section 4 presents the realization of our approach. We evaluate our approach in Section 5. Finally, Section 6 reviews related work and Section 7 summarizes and concludes the paper.

2. BACKGROUND

In general, our approach can be applied to any software architecture modeling approach supporting a model-based performance prediction. We apply our approach to the Palladio Component Model (PCM) [4], a modeling language for component-based software architectures, since it is a mature approach with a significant amount of validation case studies, e.g., [3, 4, 11, 12, 17, 19]. The PCM is aligned with the component-based software engineering (CBSE) development process and provides modeling constructs to describe the i) software components, ii) system architecture, iii) resource allocations, and iv) usage profile of a component-based software system in respective sub-models, cf. Figure 1:

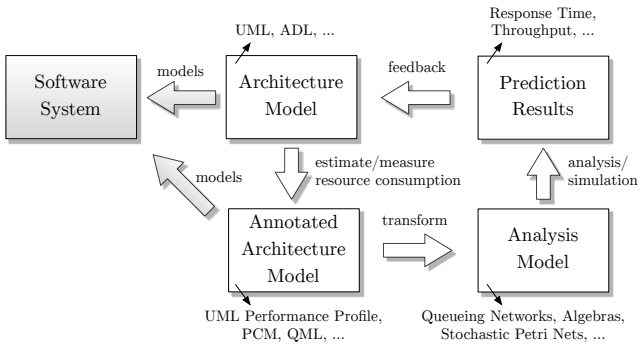


Figure 2: Model-based Performance Prediction (Source: [3])

- *Component specifications* contain an abstract, parametric description of components. Furthermore, the behavior of a component as well as its resource demands can be specified in *RDSEFFs* (short for *Resource Demanding Service EFFect specification*) in a UML activity diagram-like syntax.
- An *assembly model* defines the software architecture, i.e., which components are used and how they are connected.
- The resource environment and the allocation of components to resources are specified in an *allocation model*.
- The *usage model* specifies the usage of the system, i.e., a description of the sequence and frequency of users accessing the system operations.

Using model transformations, e.g., to simulation code or to queueing networks, a PCM model instance can be simulated or analytically solved to predict the system performance. The performance prediction serves as feedback and enables a model-based quality assessment of software systems. The model-based performance prediction process is illustrated in Figure 2.

In the PCM, I/O resources are represented as a single-/multi-server queue with FCFS (First-Come-First-Serve) or processor sharing service policy and I/O requests are represented merely by their demands, which need to be estimated by the modeler manually. While this has been shown to be a reasonable abstraction for basic hardware environments [3], modern storage systems are much more complex and have many influencing factors [26, 27]. Therefore, we focus on enhancing the PCM with statistical I/O models, especially since their creation can be fully automated and does not require additional expert knowledge [24, 25].

3. METHODOLOGY

In this section, we show how we extend the model-based performance prediction process to allow performance predictions in modern storage environments. An overview of our approach is given in Figure 3 (the upper elements are analog to the corresponding elements in Figure 2). First, we present an analysis of the factors that need to be considered when modeling I/O performance using a storage analysis model. Based on the factors, we design a storage interface that is integrated into the software architecture model. Finally, we propose a combination approach of the software architecture model with the storage analysis model.

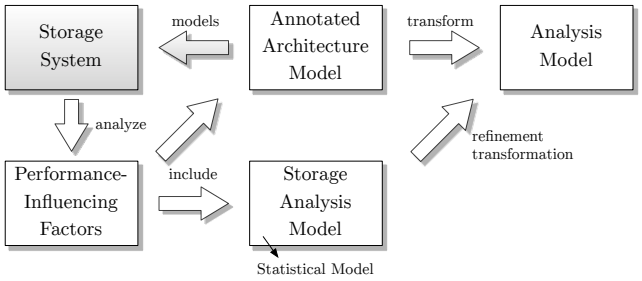


Figure 3: Proposed Approach

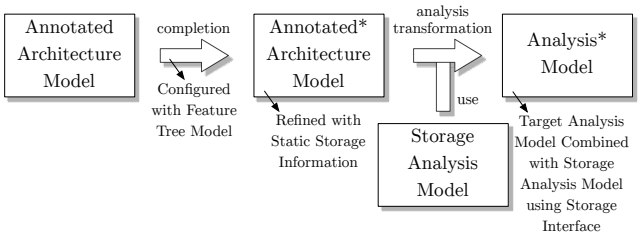


Figure 4: Refinement Transformation Approach

3.1 Performance-Influencing Factors

Starting point for creating a performance model of a storage system is analyzing the performance-influencing factors. In [26], we identified and evaluated the performance-influencing factors of applications deployed in a sophisticated environment with a complex storage system. We categorized the factors into workload-specific characteristics and system-specific configuration parameters.

It is important to find an appropriate abstraction level for the factors. On the one hand, too coarse-grained factors lead to inaccuracies in the performance models. On the other hand, too fine-grained factors cannot be specified by software developers and deployers. The performance-influencing factors are included into the architecture model and the storage analysis model. While the system-specific factors are specified once in the architecture model, the workload-specific factors need to be specified for every I/O request in the modeled application.

3.2 Storage Analysis Model

Based on the systematic evaluation of the performance-influencing factors, we create storage analysis models by applying multiple statistical regression techniques. Regression-based models are powerful techniques to capture and learn the influence of multiple *independent variables*, e.g., requests size and request type, on a *dependent variable*, e.g., request response time. The models are especially suited as they can be obtained in a fully automated manner, thus, lifting the burden for performance engineers to manually develop the models.

We have automated the storage model building process as part of our *Storage Performance Analyzer (SPA)* [23] framework. We specify the workload factors (i.e., the independent variables) that should be analyzed and the performance metric (i.e., the dependent variable) we want to characterize. Using the specified workload factors, a system profile is created by exploring the possible workload values with systematic measurements. The measurements are then used to optimally tune and build a *statistical I/O model*, i.e., a specific I/O model using a statistical regression technique, which in turn can be integrated into the software architecture model.

3.3 Storage Interface Design

The goal is to combine the low-level storage analysis model with the high-level software architecture model. A prerequisite for the combination of the two modeling abstractions is the design of a *storage interface* at the software architecture-level. The storage interface determines the well-defined use of the storage analysis model in the software architecture. The storage interface includes the required dynamic information (i.e., the workload-related performance-influencing factors) in the software architecture model that is passed as input to the storage analysis model (e.g., the distinction between read and write requests and their access patterns). Furthermore, the set of static storage-relevant information (i.e., the system-related performance-influencing factors) is used to configure or choose the appropriate storage analysis model. While the static information needs to be passed only once, e.g., when initializing the storage analysis model, the dynamic information needs to be passed with every I/O request.

3.4 Refinement Transformations

For the combination of the software architecture model with the statistical I/O model, we propose to use *refinement transformations*. As illustrated in Figure 4, the transformations may include both i) a *completion* [33] as well as ii) an *analysis transformation*.

The completion refines the architecture model with the static storage-relevant information and is configured, e.g., using feature tree models, cf. [13]. Depending on the context, the completion may be realized using a full-blown higher-order transformation (HOT) [13] or by simply adding the required static information where required. The analysis transformation uses the architecture model and creates the target analysis model to predict the system performance, cf. Figure 2. This transformation is extended to combine the target analysis model with the storage analysis model over the storage interface using a bridge or adapter. Finally, the combined analysis model is solved to obtain the performance results of the modeled software architecture.

4. REALIZATION

In this section, we present how we realize our approach presented in the previous section in the PCM and how we combine the software architecture model with the statistical I/O model. We start by presenting the modeling concept. Then, we detail how we obtain our I/O model and we show how the software model is solved to obtain the performance results. Finally, we outline the process how to use our approach for performance prediction.

4.1 Modeling Concept

For the models, we use the modeling concept of *layered execution environments* of the PCM introduced in [11] and illustrated in Figure 5.

We distinguish *business components* running at the application layer and *resources* running at the infrastructure layer. The interface of one or more business components may

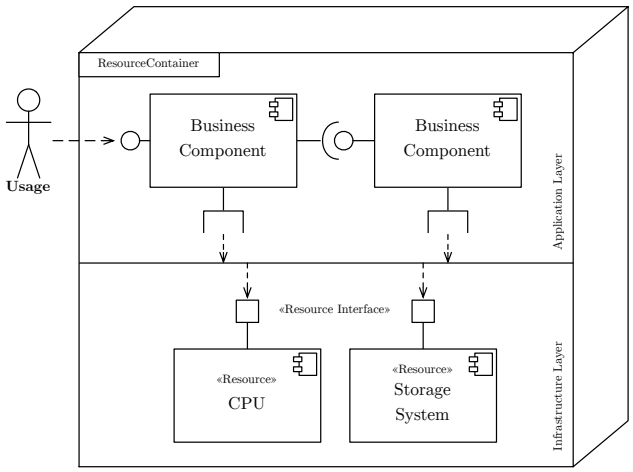


Figure 5: Modeling Concept Overview (derived from [11])

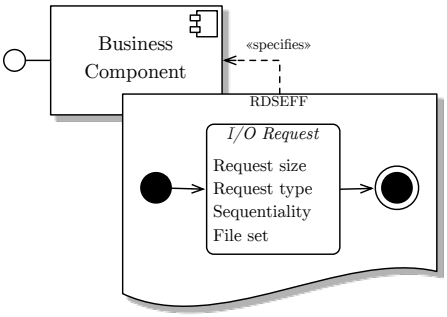


Figure 6: Realization I/O Request

be exposed to users. The business components access the CPU and I/O resources using so-called *resource interfaces*. We use this concept to model our storage system as an I/O resource enabling to realize our storage interface introduced in Section 3.3.

To use the storage interface, the I/O requests of the business components are modeled in the RDSEFF as illustrated in Figure 6. The storage interface is comprised of the following four parameters:

- *Request size*: specifies the I/O demand
- *Request type*: read or write request
- *Sequentiality*: percentage (or probability) of a sequential request
- *File set*: name and size of the file set the request is operating on

The need for former two parameters is apparent, e.g., to distinguish between small read requests and big write requests. The latter two parameters are required to estimate the impact on the caching hierarchy at the storage system, e.g., if the file set is too large to fit in the caches or if the requests are sufficiently sequential such that the requests can be reasonably anticipated and pre-fetched by the storage system.

The storage resource encapsulates the statistical I/O model. The exact model depends on a number of static parameters,

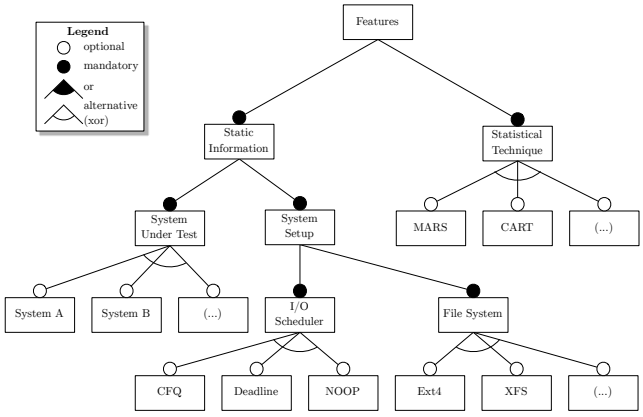


Figure 7: Feature Tree of Statistical I/O Model (cf. [26])

e.g., which target environment is used. The feature tree we employ leading to the fitting I/O model is shown in Figure 7. The *system under study* and its *setup*, i.e., *I/O scheduler* and *file system*, are captured by separate models. While it would be possible to use all the information in one big statistical model, we chose to integrate only the information that is necessarily required as part of the model for a clear conceptual separation and reduced model building effort (e.g., required measurements). Moreover, for given measurements, the model can be extracted with different *statistical techniques* with different strengths and weaknesses (cf. [24]).

4.2 Statistical I/O Model

In our previous work [24], we showed how to effectively create I/O performance models in complex environments with statistical regression techniques. Our approach is briefly summarized in the following for the sake of completeness. For more details, the reader can refer to [24]. As previously mentioned, the process is fully automated.

We first identify the independent variables, we need to capture in the models. The space spanned by the independent variables is then explored to extract a system profile with systematic measurements. The independent variables we use for I/O performance modeling are:

- Number of clients (concurrent requests)
- Request type (read or write)
- Mean read request size
- Mean write request size
- Read access pattern (random or sequential)
- Write access pattern (random or sequential)
- Read/write ratio
- File set size

The dependent variables are read response time and write response time.

Since regression techniques usually have configuration parameters (e.g., the maximum number of modeling terms) that influence their effectiveness in a certain application scenario, we apply a heuristic search algorithm (introduced in [24])

Figure 8: Transformation of the PCM Model

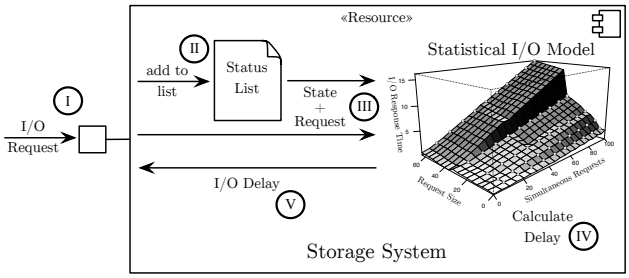


Figure 9: Storage Component Simulation

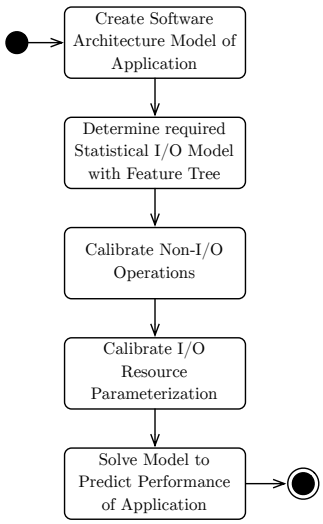


Figure 10: Performance Prediction Steps

to optimally tune the regression techniques. For given measurements, we iteratively search for best fitting regression parameters that minimize the average root mean square error of a 10-fold cross-validation.

In this paper, we use *Multivariate Adaptive Regressions Splines (MARS)* [8] for the statistical models. However, other popular methods, e.g., binary decision trees such as *Classification and Regression Trees (CART)* [5], could be used as well. MARS models consist of piecewise linear functions, so-called *hinge functions* h_i. Thus, MARS constructs a model f of the form $f(\vec{x}) = \beta_0 + \sum_{i=1}^{n} \beta_i \, h_i(\vec{x})$ with coefficients $\beta_0, \ldots, \beta_n$. Furthermore, we consider MARS models with *interaction terms*, which includes terms that are a product of one or more hinge functions.

4.3 Simulation for Model Solving

We use a simulation approach to solve the PCM model and to obtain the performance results. We transform the annotated PCM Model to the target analysis model, which is simulation code, as indicated in Figure 4 and realized as schematically shown in Figure 8. In the simulation, we integrated the statistical I/O model that is used over the storage interface. More specifically, we have extended the PCM simulator SimuCom [3] by a storage system scheduler as illustrated in Figure 9 to simulate the I/O delay at the storage system. The simulation is comprised of the following five steps:

I. Initially, the I/O requests of the business components, which are modeled as shown in Figure 6, are passed to the storage resource.

II. The arriving request is added to an internal *status list*, where the last n requests are stored in order to determine and derive the current state of the I/O workload at the storage system, e.g., the current read/write ratio of requests accessing the storage system, since such information is not given by the request itself. The choice of n determines the *memory length* and can be estimated from the workload using the average I/O delay and the number of I/O requests arriving per time $delay_{I/O}^{avg} \cdot requests_{I/O}^{Intensity}$. While this may not always be the best choice, the memory length n can also be calibrated for a given model.

III. The workload state information is calculated and passed together with the request information to the statistical I/O model. The exact information (i.e., the independent variables) required by the statistical model is given in Section 4.2.

IV. The statistical I/O model, which is encapsulated by the resource component, uses the independent variables to calculate the actual I/O delay. Since the model is a mathematical function, the calculation is fast and does not introduce significant simulation overhead.

V. By using the workload state information of the storage resource including the number of concurrent requests, the statistical model inherently captures the contention at the resource. Thus, the calculated response time is assigned to the arriving request such that the request is delayed by this calculated value.

4.4 Prediction Process

Figure 10 illustrates the process to obtain performance predictions of an application with our approach. The process is comprised of the following five steps:

1. The starting point is creating the PCM model instance of an application whose performance should be evaluated. The model instance is comprised of the submodels as before our extension (cf. Figure 1) with the added information on the I/O requests, which are the parameters of the storage interface (cf. Figure 6).

2. Using a feature configuration of the feature tree model shown in Figure 7, the required statistical I/O model is chosen. The feature tree is used by simply choosing the statistical technique and evaluating the appropriate system configurations. The statistical models can be stored in a repository or created with measurements when they are needed, e.g., for a new environment. If a new model is created, it can be created incrementally by

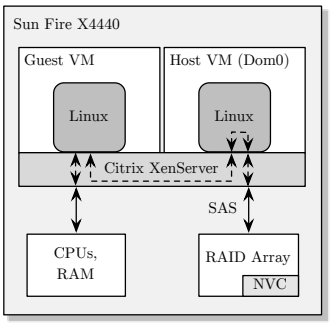


Figure 11: Schematic Illustration of Sun Fire X4440

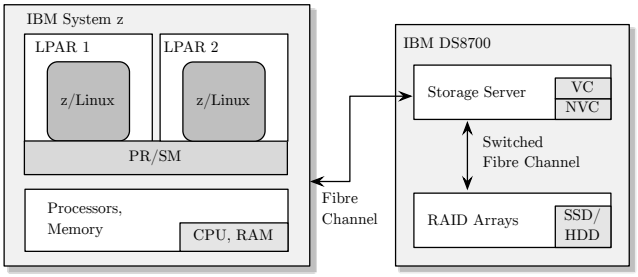


Figure 12: Schematic Illustration of IBM System z and DS8700

fixing some independent variables when measuring the environment, e.g., if an application is known to access a file set of a certain size, this independent variable is not needed to be fully evaluated and its exploration can be postponed to save time.

3. If the structural and behavioral model of the application is created, the resource demands of the non-I/O operations of the model need to be estimated and calibrated, e.g., for CPU resources.

4. Additionally, the parameters used for the storage component as illustrated in Figure 9 might be calibrated for the application model, e.g., the memory length can be adapted if required.

5. After the model creation and calibration steps have been completed, the solution of the model, which is fully automated, can be triggered and the performance results can be used to predict the performance of the application, e.g, to predict the performance for an increasing number of users.

5. EVALUATION

In this section, we present three case studies to evaluate our approach. We start by introducing our system environments and conclude with a summary and discussion of the results after the case studies.

5.1 System Environments

For the evaluation of our approach, we use two representative, state-of-the-art server environments.

5.1.1 Sun Fire

Our first environment is a *Sun Fire* X4440 x64 server system and is illustrated in Figure 11. It contains four 6-core processors and 128 GB of memory. The storage back end is a RAID array with 8 Serial Attached SCSI (SAS) hard disks. The array contains a battery-backed, non-volatile write cache (NVC).

The application runs in a guest Linux virtual machine (VM) virtualized using a Citrix XenServer hypervisor. The scheduler of the hypervisor manages the access of the guest VMs to the CPUs. For I/O requests, XenServer uses a privileged host VM (Dom0) able to access the physical devices.

The guest VM is equipped with 4 cores and 2 GB of memory and we focus the measurements on the storage performance using POSIX configuration. The file system is configured to the de facto standard *EXT4* and as I/O scheduler, we use the default scheduler in virtualized environments [22].

5.1.2 IBM System z

Our second environment is based on the IBM mainframe *System z* and the storage system *DS8700*. As illustrated in Figure 12, the System z provides processors and memory, whereas the DS8700 provides storage space. The resources are managed by the *Processor Resource/System Manager (PR/SM)*, which is basically a hypervisor creating logical partitions (*LPARs*) of the machine.

The System z supports special Linux ports for System z commonly denoted as *z/Linux*. The System z is connected to the DS8700 via fibre channel. In the DS8700, storage requests are handled by a storage server containing a volatile cache (VC) and a non-volatile cache (NVC). The storage server is connected via switched fibre channel to SSD- or HDD-based RAID arrays. Furthermore, the storage server applies several pre-fetching and destaging algorithms for optimal performance [7]. When possible, read requests are served from the volatile cache, otherwise, they are served from the RAID arrays and stored together with pre-fetched data in the volatile cache for future accesses. Write-requests are propagated both to the volatile and non-volatile cache and are destaged to the RAID arrays asynchronously.

In our experimental environment, the DS8700 contains 2 GB NVC and 50 GB VC with a RAID5 array containing seven HDDs and measurements are obtained in a z/Linux LPAR. The I/O-related operating system configuration is as in our previous environment.

5.2 Case Study I

In our first case study, we model a *file server* application in the Sun Fire environment. The file server application is emulated using the popular *Filebench*[1] framework. Filebench has a workload definition language used to describe and emulate typical I/O-intensive applications. The application workload is comprised of a sequence of file system operations that is repeatedly executed by a number of clients (threads), e.g., a file in a file set is opened, read, and then closed. Filebench uses the following operations to define the workload:

- *openfile:* Opens a randomly chosen file returning the file handle.
- *closefile:* Closes an opened file.

[1] `https://github.com/Filebench-Revise/Filebench-Revise` (version with fixes from `http://sourceforge.net/projects/filebench/`)

Listing 1: File Server Workload

```
File set:
  - number of files = 10000
  - mean file size = 128 KB
  - file preallocation = 80%
Threads:
  - 50 (default)
Operations:
  - createfile
  - writewholefile
  - closefile
  - openfile
  - appendfilerand, mean size = 16 KB
  - closefile
  - openfile
  - readwholefile
  - closefile
  - deletefile
  - statfile
```

- *createfile:* Creates an empty file.
- *deletefile:* Deletes a randomly chosen file.
- *statfile:* Requests the meta information of a file.
- *readwholefile/writewholefile:* Read or write a file in one request.
- *appendfilerand:* Appends a random amount of data (with specified mean size) to a file.

Using these operations, the *file server* application is defined as shown in Listing 1. During the application runtime, the operations are executed in a sequence repeatedly by the clients.

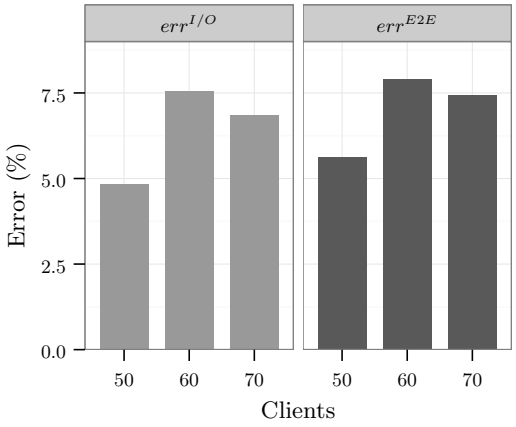
We run the file server application three times with one minute warm up time and five minutes measurement time and averaged the measurements across the three runs. We modeled the create and delete operations as requests to the CPU resource. We modeled the open, close and stat operations as requests to a so-called DELAY resource (infinite server resource). We modeled the I/O operations read, write, and append as requests to our new storage system resource.

We calibrate the PCM model of the file server with 40 clients (a value below the default value to also predict the default number of clients), where we calibrate the operations using the CPU and Delay resource as well as the memory length for the statistical model and the passed parameters. We increase the number of clients and evaluate both the mean I/O response time prediction as well as the end-to-end response time prediction. The mean I/O response time prediction error $err^{I/O}$ and the end-to-end response time prediction error err^{E2E} are defined as

$$err^{I/O} = \frac{1}{n} \sum_{i}^{n} \left| \frac{opsIO_i^M - opsIO_i^P}{opsIO_i^M} \right| \quad (1)$$

and

$$err^{E2E} = \left| \frac{\sum_i ops_i^M - \sum_i ops_i^P}{\sum_i ops_i^M} \right|, \quad (2)$$



Clients	50	60	70
$err^{I/O}$	4.84 %	7.56 %	6.86 %
err^{E2E}	5.62 %	7.90 %	7.43 %

Figure 13: Prediction Error in Case Study I

respectively, where $opsIO_i^M$ and $opsIO_i^P$ is the measured and predicted response time of the i-th I/O operation (i.e., read, write, or append), respectively, and ops_i^M and ops_i^P is the measured and predicted response time of the i-th operation, respectively.

For the prediction, we increase the number of clients up to 70, while the end-to-end response time measurements increase from 103.09 ms (40 clients) to 188.51 ms (70 clients). Figure 13 summarizes the prediction error with the simulation model. Overall, the prediction accuracy in this case study is very high with a prediction error between 5 % and 8% for both the I/O response time and the end-to-end response time. The prediction accuracy is significantly high especially considering the fact that two different types of write operations (128 KB write and 16 KB append) are predicted using the same statistical model in the simulation.

5.3 Case Study II

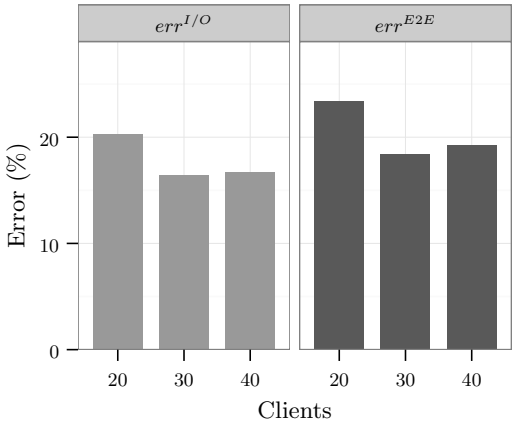
In this case study, we model a *mail server* application in the Sun Fire environment. The mail server application is emulated using Filebench similar as the file server application in the previous case study. The mail server application is defined as shown in Listing 2. Similar to the previous application, the operations are executed in a sequence repeatedly by the clients during the runtime of the application.

We also run the mail server application three times with one minute warm up time and five minutes measurement time and averaged the measurements across the three runs. The operations were modeled as in our previous case study.

We calibrate the PCM model of the mail server with 10 clients (a value below the default value to also predict a number of clients close to the default value), where we calibrate the operations using the CPU and Delay resource as well as the memory length for the statistical model and the passed parameters. We increase the number of clients and evaluate both the mean I/O response time prediction $err^{I/O}$ as well as the end-to-end response time prediction err^{E2E}.

Listing 2: Mail Server Workload

```
File set:
  - number of files = 50000
  - mean file size = 16 KB
  - file preallocation = 80%
Threads:
  - 16 (default)
Operations:
  - deletefile
  - createfile
  - appendfilerand, mean size = 16 KB
  - closefile
  - openfile
  - readwholefile
  - closefile
  - openfile
  - appendfilerand, mean size = 16 KB
  - closefile
  - openfile
  - readwholefile
  - closefile
```



Clients	20	30	40
$err^{I/O}$	20.31 %	16.43 %	16.68 %
err^{E2E}	23.44 %	18.36 %	19.21 %

Figure 14: Prediction Error in Case Study II

For the prediction, we increase the number of clients up to 40, while the end-to-end response time measurements increase from 24.98 ms (10 clients) to 56.40 ms (40 clients). Figure 14 summarizes the prediction error with the simulation model. Compared to our previous case study, the prediction error is slightly higher, however, the accuracy with an I/O prediction error between 16 % and 20 % and an end-to-end prediction error between 18 % and 23 % is still very good.

5.4 Case Study III

In our final case study, we model the file server application introduced in our first case study (cf. Listing 1) in the IBM System z environment. We again calibrate the PCM model of the file server with 40 clients and increase the number of clients to evaluate both the mean I/O response time prediction error $err^{I/O}$ and the end-to-end response time prediction error err^{E2E}.

For the prediction, we increase the number of clients up to 70, while the end-to-end response time measurements increase from 37.46 ms (40 clients) to 47.04 ms (70 clients). Figure 15 summarizes the prediction error with the simulation model. The prediction error in this case study is also very encouraging with a prediction error between 15 % and 21% for the I/O response time and a prediction error between 2 % and 10% for the end-to-end response time.

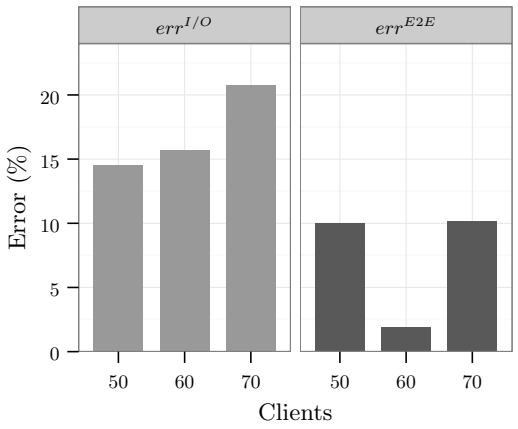
5.5 Summary and Discussion

In the three case studies, we modeled a file server and a mail server application in two sufficiently complex environments that are not easily modeled or simulated. We demonstrated that the combination approach can be used to predict the performance of the considered applications when the number of users increases with a mean end-to-end prediction error across the prediction scenarios of 6.98 %, 20.34 %, and 7.34 % in the three case studies, respectively. This increases the applicability of such software architecture performance models where the I/O-intensive operations can be captured by the statistical I/O model since, e.g., in our final case study, only the statistical I/O model needed to be exchanged when changing the environment and the structural and behavioral model of the application could be reused.

In addition to the prediction results, the case studies also revealed some considerations. First, the prediction quality might also depend on the calibration of the I/O resource parameterization. For example, the produced I/O load on the system in our IBM System z environment was reduced due to a software bottleneck in the file server application during the measuring of response times because of very fast I/O request processing by the storage system. We accounted for this observation by reducing the calculated *concurrent request* parameter by 25% when passed to the statistical I/O model during simulation. Consequently, a more tailored calibration might increase prediction accuracy locally, however, with the possible sacrifice of extrapolation quality due to "overfitting" if the prediction configuration is more beyond the calibrated configuration. In our case studies, we reasonably limited the calibration process to obtain a sufficiently accurate prediction model. Another consideration is that the statistical I/O models employed in the case studies – as every model and modeling formalism – have an inherent potential for inaccuracies as they are created at a certain abstraction level with the goal to predict mean response times. To capture response time distributions, for example, the approach could be extended by further statistical I/O models, which are used during simulation, to capture important response time quantiles. Finally, to obtain a statistical I/O model, a potentially large number of measurements might be needed to cover all configuration combinations. As indicated in Section 4.4, we reduced the number of required measurements in the case studies by first testing the parameter ranges produced by the application and then explored the region around those ranges with measurements for the statistical I/O model.

6. RELATED WORK

Our approach is based on performance prediction of component-based software architectures [3, 4], modeling of layered



Clients	50	60	70
$err^{I/O}$	14.50 %	15.72 %	20.73 %
err^{E2E}	9.98 %	1.90 %	10.15 %

Figure 15: Prediction Error in Case Study III

component execution environments [11], and statistical I/O performance modeling of storage systems [24].

In general, Balsamo et al. [2] and Koziolek [15] present extensive surveys on approaches for performance modeling and evaluation of software systems without explicit focus on storage systems as in this paper.

More specifically, multiple performance modeling approaches of complex storage systems in native and virtualized environments have been proposed, e.g., [10, 20, 21, 30, 31] and [1, 6, 14, 16, 18], however, none of these approaches considers the system performance at the software architecture-level as presented in this paper. Closest to our work among those approaches, Kundu et al. [18] use artificial neural networks and support vector machines for dynamic capacity planning to answer which CPU and memory limits and I/O latency should be assigned to an application to meet performance limits. In [31], Wang et al. use CART models to predict disk device performance. They use an interesting set of workload parameters, however, it is unclear if the parameters can be used for more complex storage systems and whether the parameters can be used in software architecture-level approaches. Furthermore, Chiang et al. [6] use linear and second degree polynomials to model I/O performance interference of data-intensive applications. They use the models for scheduling algorithms to manage task assignments in XEN-based virtualized environments. As input to their model, they use read and write request arrival rates as well as local and global CPU utilization within the guest and the host VM, respectively. However, they do not distinguish between request sizes or sequential and random requests, for instance. Our measurements have shown that these and more factors have a significant impact on I/O performance.

The general approach closest to the work presented in this paper, Wert et al. [32] outline a general concept to combine software architecture models with domain-independent, measurement-based statistical models. They focus on the technical realization of the approach, but they show no evaluation of the combination. Thus, it is unclear what information needs to be exchanged when performing such a combination, whereas we propose a specific design of I/O-relevant parameters in the storage interface and in the statistical I/O models.

In [34], Woodside et al. present a workbench for statistical resource demand models of software components and subsystems. They outline that the resource models could be used in a performance model of the software environment without going into specific details how the combination could be realized.

Further, Shanthikumar et al. [29] present a classification scheme for hybrid analytic/simulation models and present several case studies including ones modeling simple disks as queues. As this was possible with the PCM before, our approach allows to model and predict the performance when using more complex storage systems.

7. CONCLUSION

In summary, we presented a novel approach for performance modeling and analysis of a) I/O-intensive applications in b) modern storage environments at the c) software architecture-level. To the best of our knowledge, there are no other approaches that combine these three aspects to our extent.

To realize our approach, we applied the model-based performance prediction process and extended the Palladio Component Model, a model-based performance prediction approach for component-based software architectures. We introduced a storage system scheduler that uses a statistical I/O model that was created in a fully automated manner to capture the behavior of the storage system. We used the concept of layered execution environments and encapsulated our storage system scheduler in a resource component that can be used over its resource interface. We described the required parameters at the resource interface and for the statistical I/O model. To solve the model and obtain performance data, we use a simulation approach where we calculate state information of the I/O workload (e.g., read/write mix) from the I/O requests. The state and I/O request information is then used to determine the delay at the storage resource with the statistical I/O model for a given I/O request.

The evaluation of our approach showed very promising results. We modeled a file server and a mail server application and used two modern environments based on Sun Fire and IBM System z server hardware. For the models, we evaluated the mean I/O response time error and the end-to-end response time error when predicting the application performance if the number of users increases. Modeling the file and mail server application in the Sun Fire environment, we obtained a mean I/O response time error of 6.42 % and 17.81 %, respectively, and a mean end-to-end response time error of 6.98 % and 20.34 %, respectively. Modeling the file server application in the IBM System z environment, we obtained a mean I/O response time error of 16.98 % and a mean end-to-end response time error of 7.34 %. Thus, the software architecture models were able to capture the performance characteristics in the respective environments.

The main lessons learned can be summarized as follows. i) We designed the component interface of a storage component specifying the required information for I/O modeling of storage systems with statistical regression techniques to use it in software architecture modeling approaches. ii) We modeled the I/O requests in a form that they can be easily expressed

by software architects, i.e., the I/O requests are specified without requiring to estimate their resource demands, and the required information for statistical I/O models can be derived. iii) We could easily transfer the existing software architecture model of an application from one environment to another by recalibrating after replacing the statistical I/O model. iv) Overall, we successfully predicted the performance in sufficiently complex storage environments with reduced manual effort due to the automated statistical I/O model creation, which is integrated in our approach.

Our approach enables multiple application scenarios. Generally, our approach is targeted at reducing the effort for the analysis and evaluation of data-intensive applications running on modern storage systems by allowing to analyze the application performance on a software architecture-level. Our approach is especially beneficial in cases that prohibit applying explicit, fine-grained performance models due to, e.g., time constraints for the manual performance model creation, calibration, and validation. Our automated performance modeling approach can aid (e.g., the system developer) to create a performance profile of the storage system once and tailor the configuration and resource allocation as needed using the software architecture model. This model can be used in different scenarios to assess the system capacity limits when the number of users increases and evaluate deployment and trade-off decisions when using different system environments.

Acknowledgements

This work was partially supported by the German Research Foundation (DFG) under grant No. KO 3445/6-1, and the German Federal Ministry of Economics and Energy (BMWI), grant No. 01MD11005 (PeerEnergyCloud). We especially thank the Informatics Innovation Center (IIC) – `http://www.iic.kit.edu/` – for providing the system environment of the IBM System z and the IBM DS8700.

8. REFERENCES

[1] I. Ahmad, J. Anderson, A. Holler, R. Kambo, and V. Makhija. An Analysis of Disk Performance in VMware ESX Server Virtual Machines. In *WWC-6*, 2003.

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE TSE*, 2004.

[3] S. Becker. *Coupled model transformations for QoS enabled component-based software design*. PhD thesis, Universität Oldenburg, 2008.

[4] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *J. of Systems and Software*, 82(1), 2009.

[5] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Chapman & Hall, 1984.

[6] R. C. Chiang and H. H. Huang. TRACON: Interference-aware Scheduling for Data-intensive Applications in Virtualized Environments. In *SC '11*.

[7] B. Dufrasne, W. Bauer, B. Careaga, J. Myyrrylainen, A. Rainero, and P. Usong. IBM System Storage DS8700 Architecture and Implementation. `http://www.redbooks.ibm.com/abstracts/sg248786.html`, 2010.

[8] J. H. Friedman. Multivariate Adaptive Regression Splines. *Annals of Statistics*, 19(1):1–141, 1991.

[9] J. Gantz and D. Reinsel (IDC). THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. `http://idcdocserv.com/1414`, 2012. Last accessed: Jan 2014.

[10] P. Harrison and S. Zertal. Queueing models of RAID systems with maxima of waiting times. *Performance Evaluation*, 64:664–689, 2007.

[11] M. Hauck, M. Kuperberg, K. Krogmann, and R. Reussner. Modelling Layered Component Execution Environments for Performance Prediction. In *CBSE '09*.

[12] N. Huber, S. Becker, C. Rathfelder, J. Schweflinghaus, and R. Reussner. Performance Modeling in Industry: A Case Study on Storage Virtualization. In *ICSE '10, Software Engineering in Practice Track*.

[13] L. Kapova and T. Goldschmidt. Automated feature model-based generation of refinement transformations. In *SEAA '09*.

[14] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An Analysis of Performance Interference Effects in Virtual Environments. In *ISPASS '07*.

[15] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Perform. Eval.*, 2010.

[16] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick. Performance Models of Storage Contention in Cloud Environments. *SoSyM*, 2012.

[17] K. Krogmann. *Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2010.

[18] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta. Modeling Virtualized Applications using Machine Learning Techniques. In *VEE '12*.

[19] M. Kuperberg, K. Krogmann, and R. Reussner. Performance Prediction for Black-Box Components using Reengineered Parametric Behaviour Models. In *CBSE '08*.

[20] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt. Analytical and Simulation Modelling of Zoned RAID Systems. *The Computer Journal*, 54:691–707, 2011.

[21] E. K. Lee and R. H. Katz. An analytic performance model of disk arrays. *SIGMETRICS Perform. Eval. Rev.*, 21(1), 1993.

[22] X. Ling, S. Ibrahim, H. Jin, S. Wu, and T. Songqiao. Exploiting Spatial Locality to Improve Disk Efficiency in Virtualized Environments. In *MASCOTS '13*.

[23] Q. Noorshams, D. Bruhn, A. Busch, S. Kounev, and R. Reussner. The Storage Performance Analyzer (SPA), 2014. `http://sdqweb.ipd.kit.edu/wiki/SPA`.

[24] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner. Predictive Performance Modeling of Virtualized Storage Systems using Optimized Statistical Regression Techniques. In *ICPE '13*.

[25] Q. Noorshams, A. Busch, A. Rentschler, D. Bruhn, S. Kounev, P. Tůma, and R. Reussner. Automated Modeling of I/O Performance and Interference Effects in Virtualized Storage Systems. In *DCPerf '14*.

[26] Q. Noorshams, S. Kounev, and R. Reussner. Experimental Evaluation of the Performance-Influencing Factors of Virtualized Storage Systems. In *EPEW '12*.

[27] Q. Noorshams, K. Rostami, S. Kounev, P. Tůma, and R. Reussner. I/O Performance Modeling of Virtualized Storage Systems. In *MASCOTS '13*.

[28] S. Oliveira, K. Furlinger, and D. Kranzlmuller. Trends in Computation, Communication and Storage and the Consequences for Data-intensive Science. In *HPCC-ICESS'12*.

[29] J. G. Shanthikumar and R. G. Sargent. A unifying view of hybrid simulation/analytic models and modeling. *Operations Research*, 31(6):pp. 1030–1052, 1983.

[30] E. Varki and S. X. Wang. A performance model of disk array storage systems. In *Int. CMG Conference*, 2000.

[31] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage Device Performance Prediction with CART Models. In *MASCOTS '04*.

[32] A. Wert, J. Happe, and D. Westermann. Integrating Software Performance Curves with the Palladio Component Model. In *ICPE '12*.

[33] M. Woodside, D. Petriu, and K. Siddiqui. Performance-related completions for software specifications. In *ICSE '02*.

[34] M. Woodside, V. Vetland, M. Courtois, and S. Bayarov. Resource Function Capture for Performance Aspects of Software Components and Sub-systems. In R. Dumke, C. Rautenstrauch, A. Scholz, and A. Schmietendorf, editors, *Performance Engineering, State of the Art and Current Trends*. 2001.