I/O Performance Modeling of Virtualized Storage Systems

Qais Noorshams*, Kiana Rostami*, Samuel Kounev*, Petr Tůma†, Ralf Reussner*
*Karlsruhe Institute of Technology, Germany
{noorshams, kounev, reussner}@kit.edu
kiana.rostami@student.kit.edu
†Charles University in Prague, Czech Republic
tuma@d3s.mff.cuni.cz

***Abstract*—Server virtualization is a key technology to share physical resources efficiently and flexibly. With the increasing popularity of I/O-intensive applications, however, the virtualized storage used in shared environments can easily become a bottleneck and cause performance and scalability issues. Performance modeling and evaluation techniques applied prior to system deployment help to avoid such issues. In current practice, however, virtualized storage and its effects on the overall system performance are often neglected or treated as a black-box. In this paper, we present a systematic I/O performance modeling approach for virtualized storage systems based on queueing theory. We first propose a general performance model building methodology. Then, we demonstrate our methodology creating I/O queueing models of a real-world representative environment based on IBM System z and IBM DS8700 server hardware. Finally, we present an in-depth evaluation of our models considering both interpolation and extrapolation scenarios as well as scenarios with multiple virtual machines. Overall, we effectively create performance models with less than 11% mean prediction error in the worst case and less than 5% prediction error on average.**

***Keywords*-I/O; Storage; Performance; Prediction; Modeling**

I. Introduction

Current technology trends like cloud and sky computing as well as the increasing demand for greener IT have contributed to the wide adoption of virtualization technology. In the next years, the server virtualization market is expected to grow annually by more than 31% until 2016 [1]. By sharing physical resources with multiple virtual machines, server virtualization offers both efficient data center operation and flexible, on-demand resource provisioning.

In parallel to this development, the I/O resource demands of modern IT systems have increased exponentially over the past decades [2]. Until the year 2020, the amount of digital data is expected to double annually, 40% of which is expected to be either processed or stored in the cloud [3]. Moreover, video streaming portals, online file storage services, social networking applications, and traditional mail servers are already frequently deployed in virtualized environments for maximum flexibility and efficiency. Such services require explicit performance modeling and evaluation techniques for capacity planning at deployment time as well as to avoid performance bottlenecks during operation due to workload changes.

In current practice, however, virtualized storage and its effects on the overall system performance are often neglected or treated as a black-box. A few explicit modeling approaches considering I/O-intensive applications in virtualized environments have been proposed (e.g., [4]), however, with a focus on consolidation scenarios only or with validation limited to basic environments. The main obstacle is the increasing complexity of modern virtualized storage systems posing significant challenges for explicit performance modeling approaches to create practically usable models with reasonable amount of effort and associated costs. The multiple logical layers between the running applications and the physical storage lead to complex performance effects and call for a fine-grained analysis coupled with a systematic modeling methodology.

To address these challenges, in this paper, we propose an iterative performance modeling approach and show how it can be used to capture the complex behaviour of a representative virtualized storage system. More specifically, we first propose a general performance model building methodology. Then, we apply our methodology and create I/O queueing models of a real-world representative environment based on IBM System z and IBM DS8700 server hardware. The queueing models are calibrated with response time measurements and do not require hardware-level monitoring data. Finally, we evaluate the models in different scenarios to assess their prediction accuracy. The scenarios comprise interpolation and extrapolation scenarios as well as scenarios where the workload is distributed on multiple virtual machines. Using our approach, we effectively create performance models with less than 11% mean prediction error in the worst case, and less than 5% prediction error on average.

In summary, the contribution of this paper is two-fold: i) We present a step-by-step I/O performance modeling approach for virtualized storage systems that is calibrated with response time measurements. ii) We present a comprehensive evaluation and validation of the proposed approach in a real-world environment based on the state-of-the-art server technology of the IBM System z and IBM DS8700.

The remainder of this paper is organized as follows: Section II introduces our system under study. In Section III, we present our performance model building methodology. Section IV applies our methodology to extract I/O performance models of the considered system environment. In Section V, we present the evaluation of our approach. Finally, Section VI reviews related work and Section VII concludes.

1526-7539/13 $26.00 © 2013 IEEE
DOI 10.1109/MASCOTS.2013.20

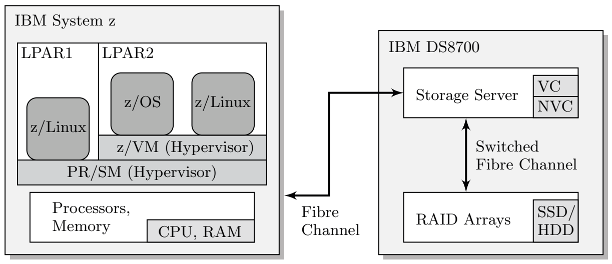


Fig. 1: IBM System z and IBM DS8700

II. System Under Study

In this paper, we consider a representative virtualized environment based on the IBM mainframe *System z* and the storage system *DS8700*. They are state-of-the-art high-performance virtualized systems with redundant and hot swappable resources for high availability. The System z combined with the DS8700 represents a typical virtualized environment that can be used as a building block of cloud computing infrastructures. It supports on-demand elasticity of pooled resources with a pay-per-use accounting system (cf. [5]). The System z provides processors and memory, whereas the DS8700 provides storage space. The structure of this environment is illustrated in Figure 1.

The *Processor Resource/System Manager (PR/SM)* is a hypervisor managing logical partitions (*LPARs*) of the machine (therefore also called *LPAR hypervisor*) and enabling CPU and storage virtualization. For memory virtualization and administration purposes, IBM introduces another hypervisor, *z/VM*. The System z supports the classical mainframe operating system *z/OS* and special Linux ports for System z commonly denoted as *z/Linux*. The System z is connected to the DS8700 via fibre channel. Storage requests are handled by the storage server, which is equipped with 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.

As explained in [6], the storage server applies several pre-fetching and destaging algorithms for optimal performance. 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.

III. Methodology

To extract I/O performance models based on queueing theory, we apply a systematic iterative approach as illustrated in Figure 2. Our approach follows the common generic steps in classical performance engineering including performance model *creation*, *calibration*, and *validation* [7], [8], [9] and is based on established approaches [8], [9], [10]. After a system environment analysis and a preparation phase, we create an initial model that we iteratively extend to account for more complex scenarios.

During the system environment analysis, we define the system setup and identify the performance-relevant system aspects that need to be captured as part of the model. Typically, these comprise hardware resources, however, it might be required to include logical resources in the model explicitly, e.g., operating system schedulers. Furthermore, we create a typical workload characterization.

In the next phase, we develop an iterative model creation plan and identify the minimal workload scenarios that should be included in the model as a start. Based on the identified workload scenarios, we define the request classes of the model. Usually, different request types (e.g., read or write requests) are mapped to different classes possibly also differentiating further workload properties, e.g., sequential and random requests or requests from different virtual machines.

Then, we iteratively create and calibrate the performance model. Based on the resources that are accessed by the minimal workload scenarios, we create a minimal topology for each request class. This requires to identify the topology type (e.g., open or closed model), the required queues and their characteristics (e.g., the capacity and scheduling strategy), and the connection and routing between the queues. For each queue, the scheduling strategy is determined either empirically or based on heuristics.

To calibrate the model, we quantify the service times of each queue. This requires to estimate the service time distribution as well as the server characteristics, e.g., number of load-independent or load-dependent service stations. To conclude the iteration, the goodness-of-fit of the resulting queueing model is analyzed. If the model is acceptable, it can be extended to cover further workload scenarios and respective system components/layers involved in their processing. Such extensions might require to refine the model topology and/or the request classes, which in turn might require to re-calibrate the model.

Finally, we validate the performance model in a variety of scenarios including both interpolation and extrapolation scenarios to evaluate its accuracy and its predictive power for scenarios that have not been used as input for the calibration. We stop when the model is valid, i.e., when the performance predictions by the model match the measurements on the real system within a certain acceptable margin of error [7], [9]. For response time, a relative error of 20% is generally acceptable [9], [11]. Otherwise, the model needs to be refined iteratively to better reflect the system behavior. Every time the model is changed, it needs to be re-calibrated until it exhibits a sufficient model accuracy.

A detailed discussion on model calibration and validation techniques is available in [11].

IV. I/O Performance Modeling

In this section, we apply our performance model building methodology to the system under study following the steps shown in Figure 2. For model solving, we use a simulation-based approach.

A. System Environment Analysis

To model the I/O performance of the system under study, a detailed analysis of the system structure and its performance

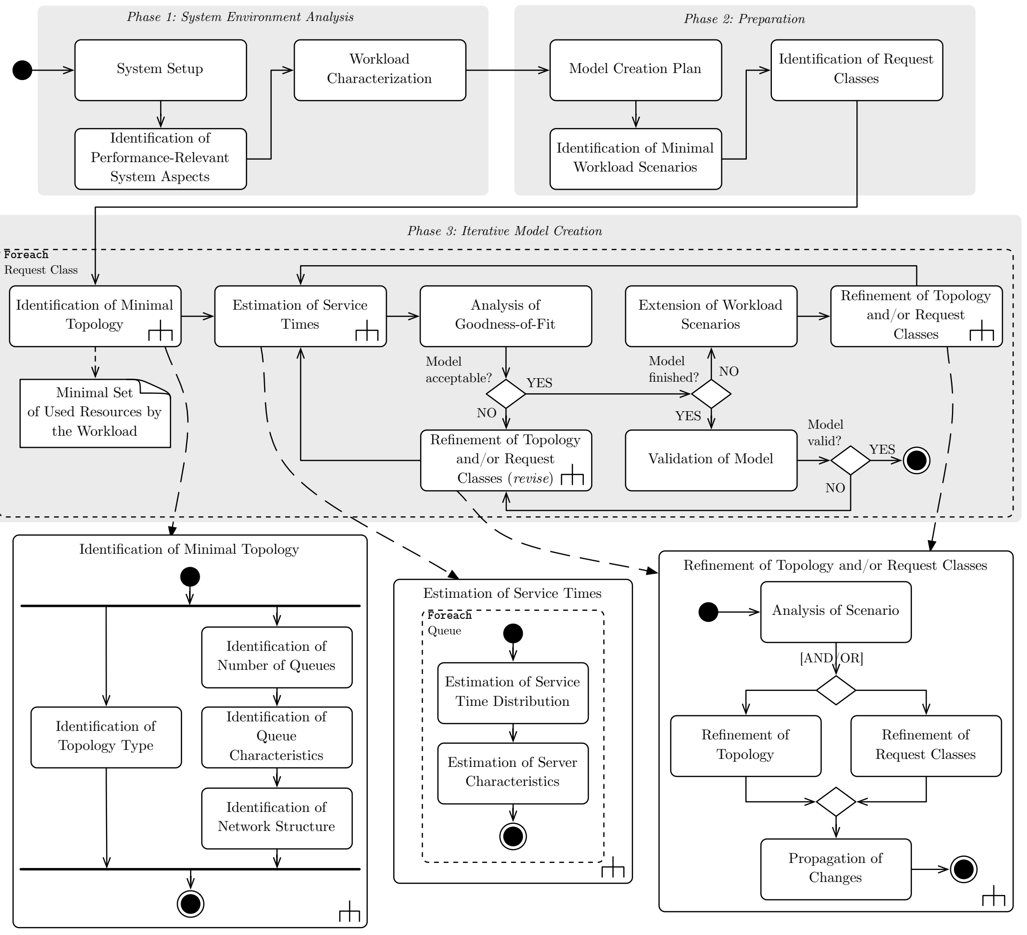

Phase 1: System Environment Analysis
System Setup
Workload Characterization
Identification of Performance-Relevant System Aspects
Phase 2: Preparation
Model Creation Plan
Identification of Request Classes
Identification of Minimal Workload Scenarios
Phase 3: Iterative Model Creation
Foreach Request Class
Identification of Minimal Topology
Estimation of Service Times
Analysis of Goodness-of-Fit
Extension of Workload Scenarios
Refinement of Topology and/or Request Classes
Minimal Set of Used Resources by the Workload
Model acceptable?
YES
NO
Model finished?
NO
YES
Refinement of Topology and/or Request Classes (revise)
Validation of Model
Model valid?
YES
NO
Identification of Minimal Topology
Identification of Number of Queues
Identification of Topology Type
Identification of Queue Characteristics
Identification of Network Structure
Estimation of Service Times
Foreach Queue
Estimation of Service Time Distribution
Estimation of Server Characteristics
Refinement of Topology and/or Request Classes
Analysis of Scenario
[AND OR]
Refinement of Topology
Refinement of Request Classes
Propagation of Changes

Fig. 2: Performance Model Building Methodology in Three Phases (depicted as gray areas)

influences is required. We analyzed the system environment in our previous work [12] and identified its performance-influencing factors as illustrated in Figure 3. Below, we briefly summarize the results of this analysis.

1) System Setup: In our system environment, the DS8700 contains 2 GB NVC and 50 GB VC with a RAID5 array containing seven HDDs. Calibration and validation measurements are obtained in a z/Linux virtual machine (VM) with 2 IFLs (cores) and 4 GB of memory. We focus our measurements on the storage performance using POSIX configuration and explicitly take into account the cache of the storage system by varying the overall size of data accessed in our workloads. As a basis for our experimental analysis, we use the open source *Flexible File System Benchmark* (FFSB)[1] due to its fine-grained configuration possibilities. FFSB runs at the application layer and measures the end-to-end response time covering all system layers from the application all the way down to the physical storage. For a given configuration of a benchmark run, a set of 16 MB files is created first. Then, the target number of workload threads are launched and they begin reading from and writing into the initial file set. For each workload thread, the read and write operations consist of 256 sub-requests of a specified size directed to a randomly chosen file from the file set. If the access pattern is sequential, the sub-requests access subsequent blocks within the file. Each thread issues a request as soon as the previous one is completed. For any configuration, we run FFSB for five minutes. During this time, the benchmark gathers millions of measurement samples, typically more than two million.

2) Identification of Performance-Relevant System Aspects: The performance-relevant hardware resources that should be captured in the model can be deduced from the schematic illustration in Figure 1. We model the storage server as a cache queueing station and the RAID array as a separate RAID queueing station. Moreover, a separate queueing station is usually required to model the operating system's I/O scheduler. In case the `NOOP` scheduler is used, it does not necessarily have to be modeled explicitly using a separate queue, since this scheduler only splits and merges requests without reordering. The I/O scheduler needs to be chosen carefully as the default schedulers are not necessarily best suited to virtualized environments [13]. In fact, in this paper, we use the `NOOP` scheduler since it is the most reasonable scheduler for our environment with the best performance, cf. [12]. Other

[1]http://github.com/FFSB-prime (extension of http://ffsb.sf.net)

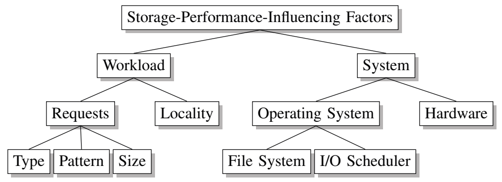


Fig. 3: Performance Influences (derived from [12])

schedulers performing significant optimizations, such as the `CFQ` *(Completely Fair Queueing)* scheduler or the `SFQ` *(Start-time Fair Queueing)* scheduler adopted in some hypervisors for instance, usually need to be modeled either explicitly or based on heuristics (cf. [4]).

3) Workload Characterization: The workload characterization we consider is illustrated in Figure 3. We distinguish the request type (i.e., read and write requests), the access pattern (i.e., sequential or random requests), and the request size. Furthermore, we explicitly consider the locality of the requests by analyzing the overall size of the set of files that is accessed in the workload. The request locality influences the storage server caching effectiveness (cf. [12]).

B. Preparation

Before creating the performance models, we develop a model creation plan describing the planned iterations. We then identify the minimal workload scenarios that we integrate in the first iteration. Finally, we identify the request classes to model the workload scenarios.

1) Model Creation Plan: Based on the results from the previous phase, we follow the model creation plan shown in Figure 4 to create and iteratively refine the I/O performance models that capture the performance-relevant system behavior. Initially, we only model the cache resource. We create homogeneous models considering read and write as well as random and sequential requests. We start with a fixed request size, which we vary in the next step. We then add the RAID resource to the models and extend the workload scenarios of the previous steps. In every step, we create the queueing network topology and scale the load (i.e., the number of clients = threads) stepwise for calibration.

2) Identification of Minimal Workload Scenarios: Initially, we only distinguish the request type and the access pattern. To analyze the behavior of the cache resource, we configure the set of files accessed by the workload (i.e., the file set) to 1.25 GB so that it fits in the caches completely.

3) Identification of Request Classes: We use the following four classes to encode the requests:

- Random read requests, R_r
- Sequential read requests, R_s
- Random write requests, W_r
- Sequential write requests, W_s

C. Iterative Model Creation: Iteration 1 & 2

In this section, we describe the first two iterations of our model creation plan to model the cache resource. We identify the queueing network topology and the service times.

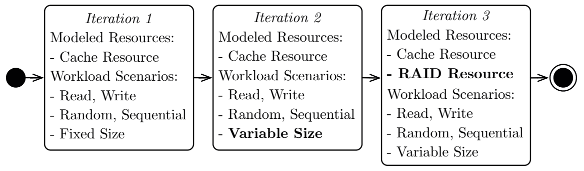


Fig. 4: Model Creation Plan

1) Identification of Topology: We model our system environment as a *closed network*. Since, at this stage, we consider only the storage server cache resource, we abstract the environment as a *single cache queue* with *unlimited capacity* and *First-Come-First-Serve (FCFS)* scheduling strategy. The resulting queueing network topology is illustrated in Figure 6.

2) Estimation of Service Times: We estimate the service times based on response time measurements. We start with a fixed request size and extend the models to consider variable request sizes in the next step.

Fixed Request Size. To estimate the service time distribution, for every request class we run measurements under low workload intensity with only one workload thread and 4 KB request size. We model the service times using a *gamma-distribution* whose parameters are determined based on the measurements. To estimate the service times and analyze the server under resource contention, for every request class we scale the workload up to 100 threads in steps of five. We observe a strong correlation between the number of workload threads and the mean response time with a *coefficient of determination* R^2 of 0.9972 and 0.9987 for read and write requests, respectively. Therefore, we model a load-dependent service station and parametrize the mean service times depending on the number of threads. This technique is routinely used in modeling complex systems [14] and is inspired by the Flow Equivalent Server (FES) method [9], [10], where the load-dependent service station replaces a more complex sub-network.

Variable Request Size. As a next step, we parametrize the service times according to the request size. We analyze request sizes of 4 KB, 8 KB, 16 KB, 32 KB, and 64 KB while scaling the load up to 100 threads in steps of 10 for every request class. We observe a strong correlation between the number of workload threads t, the request size s and the mean response time. For read requests, R^2 is 0.9958 and 0.9966 for random and sequential requests, respectively. For write requests, R^2 is 0.9999 for random and sequential requests. Thus, using the method of least squares [15], we parametrize the mean of the gamma-distributed service times μ and fit them to the measurements as follows:

$$\mu(t, s) = c_1\, ts + c_2\, t + c_3\, s + c_4;\ c_i \in \mathbb{R}. \quad (1)$$

3) Analysis of Goodness-of-Fit: Figure 5 shows the relative calibration error between the queueing models and the measurements on the real system when the number of threads varies between 10 and 100 by increments of 10. For larger read requests (16 KB, 32 KB, 64 KB), the mean error (depicted as small crosses) is less than 7.5%. For smaller read requests (4 KB, 8 KB), the error is less than 18%, however, since

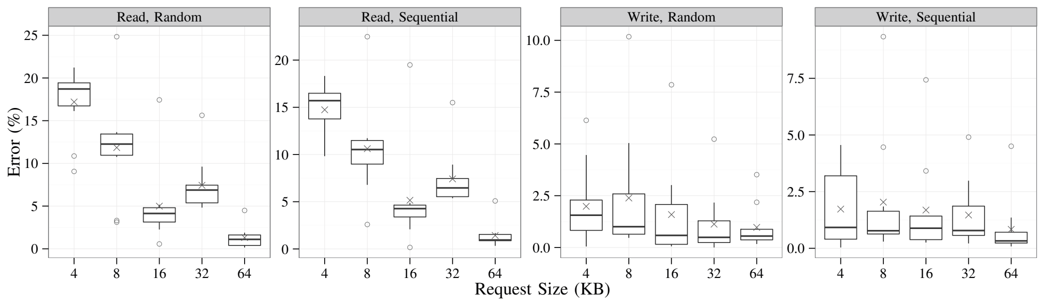


Fig. 5: Relative Calibration Error for the Initial Model for Summarized Number of Threads along the Request Sizes

the measurements are in the range of a few milliseconds, the absolute calibration error is very small. For write requests, the calibration error is always less than 2.5%.

D. Iterative Model Creation: Iteration 3

Next, we describe the third iteration of our model creation plan in which the RAID system is integrated into the queueing models.

1) Extension of Workload Scenarios: To analyze the behavior of the RAID resource, we configure the file set to be significantly larger (up to 180 GB) than the cache size so that the RAID system needs to be accessed frequently. Since the type of a request has an impact on the queueing network topology and the service times, we show the extension stepwise for every request class.

2) Random Read Requests (R_r): Refinement of Topology. We extend the topology by a second queue with unlimited capacity and FCFS scheduling strategy representing the RAID resource. Initially, the requests arrive at the cache queue. After being served by the cache, two alternatives are possible for each request: Either the request arrives at the RAID queue with probability p_1 or the request is completed and leaves with probability $p_2 = 1 - p_1$. The topology is illustrated in Figure 8.

Recall that read requests are served by the 50 GB VC if possible. The cache hit rate of the workload cannot be easily estimated due to the complex pre-fetching algorithm. Therefore, to estimate p_1 and p_2, we scale the workload up to 100 threads in steps of 10 and the file set size between 80 GB and 180 GB in steps of 20 GB. We observe two different situations, one for number of threads between 1 and 30, and one for number of threads between 40 and 100. If the number of threads is between 1 and 30, we observe that the response times follow two clearly separate distributions, one for requests served by the cache and one for requests served by the RAID array. The requests served by the cache are recognizable by response times close to the response times when having a small, fully cached file set. Separating the distributions and calculating the relative number of requests for the two distributions leads to (p_1, p_2) of $(48.32\%, 51.68\%)$ on average with a standard deviation of 9.64%. If the threads are between 40 and 100, however, we observe a significant change in the two distributions. If we apply the separation strategy as above, we obtain for p_1 a value of 99.91% on average with a standard deviation of 0.03%. This

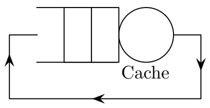


Fig. 6: Cache Resource Model

behavior is probably due to the increased contention and the cache pre-fetching algorithm. If the load is higher, which is the case for more than 30 threads, the requests served by the cache and RAID resources are no longer clearly distinguishable solely by the distribution of the response times. At this point, however, we omit further refining the queueing network topology.

Estimation of Service Times. Before estimating the service times, we analyze the influence of the request locality in a preparation step. To this end, we scale the file set size between 60 GB and 180 GB in steps of 20 GB such that it exceeds the volatile cache size. We then evaluate the mean response time for 50 and 100 threads for request sizes of 4 KB and 8 KB. For each number of threads and request size combination, we observe a strongly natural logarithmic correlation between the file set size and the mean response time. For 50 threads, we observe an R^2 of 0.9902 and 0.9932 for 4 KB and 8 KB requests, respectively. For 100 threads, we observe an R^2 of 0.9803 and 0.9957 for 4 KB and 8 KB requests, respectively. To reflect this, we include the file set size in the service times logarithmically as part of the next step.

To estimate the service times of the second queue, we measure the response times under low workload intensity and scale the intensity stepwise. Similar to the initial model, we observe a load-dependent behavior. Thus, we model the second server as a load-dependent server with gamma-distributed service times.

For the calibration, we distinguish between the two situations explained above when discussing the topology. More specifically, we distinguish between case s_1, where the number of threads is less than or equal to 30, and case s_2, where the number of threads is greater than 30. We analyze the response time distributions for 4 KB, 8 KB, 16 KB, 32 KB, and 64 KB requests for 10, 30, 50, 80, and 100 threads, while scaling the file set size between 80 GB and 160 GB in steps of 20 GB. Note that the file set cannot be fully cached leading to regular cache misses. We separate the response time distributions in cache and RAID array response times. For each case s_1, s_2

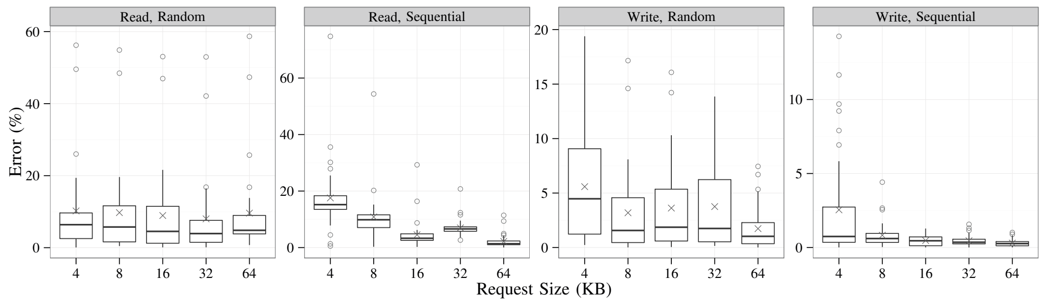


Fig. 7: Relative Calibration Error for the Refined Model for Summarized Number of Threads along the Request Sizes

and each queue, we fit the mean service times μ with file set size f to the measurements of the form

$$\mu(t,s,f) = c_1\, ts \cdot \ln(f) + c_2\, t \cdot \ln(f) + c_3\, s \cdot \ln(f) + c_4\, ts + c_5\, t + c_6\, s + c_7 \ln(f) + c_8;\ c_i \in \mathbb{R}. \quad (2)$$

To simplify the service time parametrization, we prune insignificant terms, i.e., if the p-value of a term in the parametrization is greater than 0.05. We obtain R^2 values of 0.9327, 0.9937, 0.9951, and 0.9917 for the cache(s_1), cache(s_2), RAID(s_1), and RAID(s_2) queues and case s_i, respectively.

Analysis of Goodness-of-Fit. Figure 7 shows the calibration error for 10, 30, 50, 80, and 100 threads and file set sizes between 80 GB and 160 GB in steps of 20 GB. The mean calibration error is between 8.01% and 10.21% depending on the request size, thus, exhibiting a good fit.

3) Sequential Read Requests (R_s): Refinement of Topology. To determine the topology for sequential read requests, we repeat the analysis of the previous step and scale the file set size starting from 1.25 GB and doubling until 160 GB. We observe that the requests seem to be almost always served by the storage server cache due to the effective pre-fetching algorithm. For large file set sizes significantly exceeding the storage cache, however, we observe a slight decrease in performance due to some cache misses. Therefore, we model the system using two cases s_1 and s_2 for the cache without pre-fetching (i.e., when the file set is fully cached) and the cache with pre-fetching and frequent RAID array accesses (i.e., when the file set size exceeds the cache size significantly), respectively. We use two queues representing each case. The queueing network topology is illustrated in Figure 10.

Estimation of Service Times. To parametrize the model, we analyze the mean response times for 4 KB, 8 KB, 16 KB, 32 KB, and 64 KB requests for 10, 30, 50, 80, and 100 threads, while scaling the file set size as above up to 160 GB. The cases s_1 and s_2 apply for cached data and data exceeding the cache size significantly, respectively. For the queue in case s_1, we fit the mean service times of the server μ to the measurements of the form in Equation (1). For the queue in case s_2, we fit the mean service time of the server μ to account for the cache misses due to the file set size f as follows:

$$\mu(t,s,f) = c_1\, ts + c_2\, t + c_3\, s + c_4 \cdot \ln(f) + c_5;\ c_i \in \mathbb{R}. \quad (3)$$

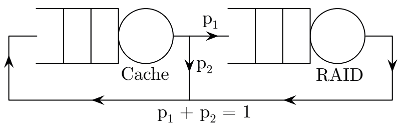


Fig. 8: Cache and RAID Resource Model for R_r

Again, we prune insignificant terms and obtain R^2 values for the queues in cases s_1 and s_2 of 0.9962 and 0.9954, respectively.

Analysis of Goodness-of-Fit. For the calibrated measurement set, we obtain an error as illustrated in Figure 7. While for 4 KB requests, the mean calibration error is 17.56%, for larger requests, the error is less than 10.84%. Again, the absolute calibration error constitutes only a fraction of a millisecond such that the models exhibit a reasonable fit.

4) Random Write Requests (W_r): Refinement of Topology. Unlike read requests, write requests are *always* served by the cache and stored asynchronously on the RAID array. To analyze this effect, we measure the response times of random write requests while scaling the file set size starting from 1.25 GB and doubling until 160 GB. We observe that the cache is able to buffer the requests for a file set size of up to 5 GB. If the file set size is scaled further, the mean response time increases logarithmically. This effect is similar to the observed effect for sequential read requests. Therefore, we use the same queueing network topology as illustrated in Figure 10. The cases s_1 and s_2 define cached data (i.e., the file set size is up to 5 GB) and uncached data with frequent destaging (i.e., the file set size is greater than 5 GB), respectively.

Estimation of Service Times. Similarly as for the sequential read requests, we parametrize the mean service times of the queue in case s_1 as in Equation (1). For the queue in case s_2, we use Equation (2). For both, we again prune insignificant terms. For the queues in cases s_1 and s_2, we obtain R^2 values of 1 and 0.993, respectively.

Analysis of Goodness-of-Fit. Figure 7 shows a very good fit of the model to the calibrated measurements with an average error of 5.89% for 4 KB requests and less for larger requests.

5) Sequential Write Requests (W_s): Refinement of Topology. Repeating the analysis of the previous section for sequential write requests, we observe that the requests are almost entirely buffered by the cache even when we scale the file set size up to 160 GB. Therefore, we keep the queueing network topology for sequential write requests as shown in Figure 6.

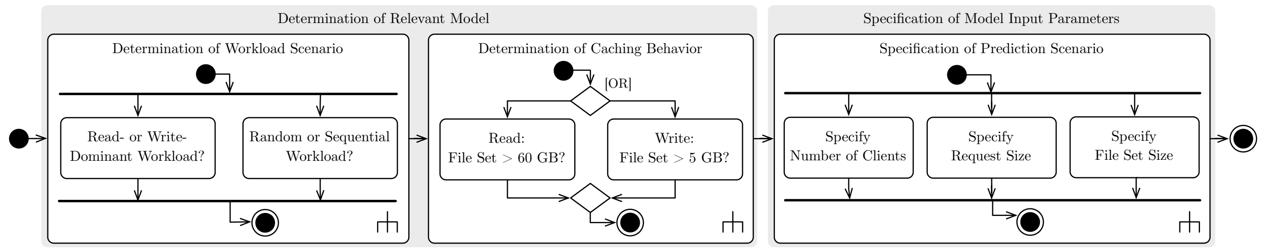

Determination of Relevant Model
Specification of Model Input Parameters
Determination of Workload Scenario
Read- or Write-Dominant Workload?
Random or Sequential Workload?
Determination of Caching Behavior
[OR]
Read: File Set > 60 GB?
Write: File Set > 5 GB?
Specification of Prediction Scenario
Specify Number of Clients
Specify Request Size
Specify File Set Size

Fig. 9: Performance Prediction Process

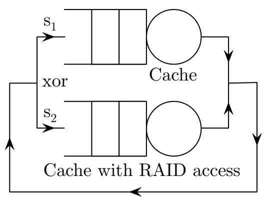


Fig. 10: Cache and RAID Resource Model for R_s, W_r

Estimation of Service Times. We calibrate the system similar to the random write requests and fit the service times to the measurements of the form of Equation (3). We obtain an R^2 of 1.0000.

Analysis of Goodness-of-Fit. We obtain a mean calibration error for each request size of up to 2.55%, cf. Figure 7.

E. Prediction Process

Figure 9 illustrates the process when using the queueing models for prediction. First the workload scenarios as well as the caching behaviour has to be determined to use the relevant model. Then, the prediction scenario has to be specified, which is comprised by the parameters used as model input.

The workload scenario can be read or write and sequential or random workload. The caching behaviour is determined by the file set size. The empirically determined thresholds are 60 GB and 5 GB for read and write workloads, respectively. If the workload scenario comprises random write requests for instance, the relevant model is the W_r model as illustrated in Figure 10 with the respective service time parametrization for write requests. Furthermore, case s_1 applies for the queueing network if the file set size is less than or equal to 5 GB; case s_2 applies otherwise.

The parameters used as input for the model are the number of clients, the request size, and the overall size of the file set accessed by the clients. The model predicts the mean response time for the specified prediction scenario.

V. Evaluation

In this section, we evaluate the predictive power of our I/O performance models in three scenarios: i) interpolation, ii) extrapolation with respect to the number of clients, and iii) extrapolation with respect to the number of VMs. We present the results for the different request classes explicitly distinguishing between cached and uncached data for random access requests resulting in six different workload scenarios.

TABLE I: Parameter Ranges for Interpolation Scenarios

Parameter	Range
Request type	{read, write}
Access pattern	{random, sequential}
Request size	[4 KB, 64 KB] rounded to multiples of 512 bytes
Number of threads	[10, 100]
File set size	[1.25 GB, 180 GB] rounded to multiples of 16 MB

A. Interpolation

For every evaluated workload scenario, we compare mean response time measurements of 200 completely random configurations within the ranges indicated in Table I against results obtained using the queueing models. Thus, in this section, we evaluate 1200 completely random measurement configurations in total. Depending on the configuration, mean read and write measurements are in [0.51 ms, 30.58 ms] and [0.68 ms, 36.05 ms], respectively. The prediction results are shown in Figure 11a and Figure 11e and discussed next.

Random Read Requests. For cached data, i.e., if the file set size is up to 60 GB, we observe a mean error of 8.40%. There are a few higher error values occuring when the file set size is close to 60 GB. We conclude that for some configurations, the cache effect applies already for such file set sizes. For uncached data, we obtain a mean error of 8.25%. A few higher error values exist, mainly occuring for low load when the number of threads is less than 20. For such configurations, every small absolute deviation constitutes a high relative error. Overall, the mean response times are predicted very well on average.

Sequential Read Requests. For the requests, the mean error is 5.00% and all requests are predicted very well.

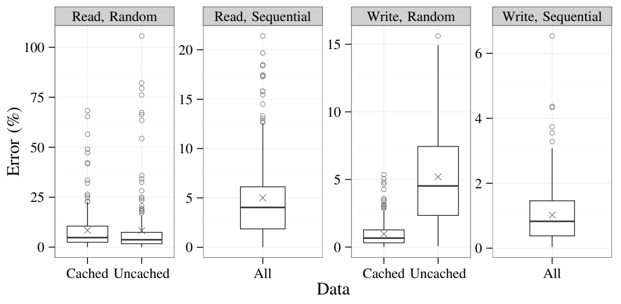
Random Write Requests. For cached data, i.e., if the file set size is up to 5 GB, the model performs very well with a mean error of 0.97%. For uncached data, the model also exhibits high prediction quality. The mean error is 5.19%. Overall, the prediction accuracy is very high.

Sequential Write Requests. For all requests, the model exhibits an excellent accuracy with a mean error of 1.02%.

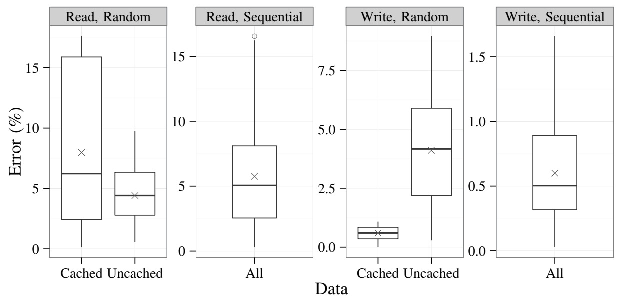
Summary. On average, the performance models exhibited high interpolation quality given the fact that the configurations considered in the evaluation were chosen completely randomly. The mean error was 4.81%. More specifically, read requests were predicted slightly worse than write requests with a mean error of 7.22% and 2.39%, respectively. Furthermore, random

TABLE II: Parameter Values for Extrapolation Scenarios

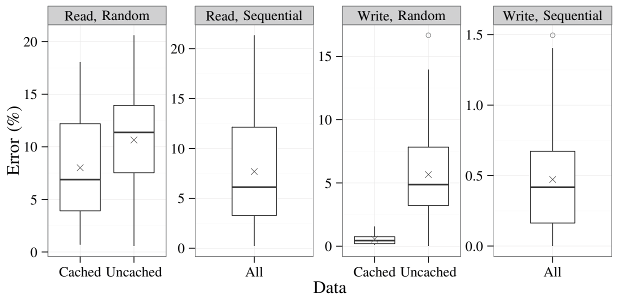
Parameter	Extrapolation Clients	Extrapolation 2 VMs	Extrapolation 3 VMs
Request type	{read, write}	{read, write}	{read, write}
Access pattern	{random, sequential}	{random, sequential}	{random, sequential}
Request size	{4 KB, 8 KB, 16 KB, 32 KB, 64 KB}	{4 KB, 8 KB, 16 KB, 32 KB, 64 KB}	{4 KB, 8 KB, 16 KB, 32 KB, 64 KB}
Number of threads	{125, 150}	{50, 75, 100}	{33, 50, 100}
File set size	{2.5 GB, 40 GB, 80 GB, 160 GB}	{2.5 GB, 40 GB, 80 GB, 160 GB} / #VMs, rounded to multiples of 16 MB	



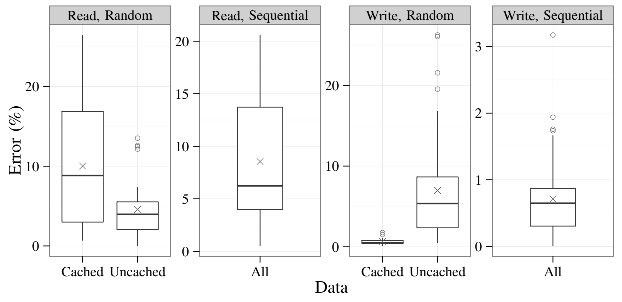
(a) Interpolation Error



(b) Extrapolation Error (Clients)



(c) Extrapolation Error (2 VMs)



(d) Extrapolation Error (3 VMs)

Scenario	R_r Cached	R_r Uncached	R_s All	W_r Cached	W_r Uncached	W_s All	Arithmetic Mean – $\frac{1}{6}\Sigma$
Interpolation	8.40	8.25	5.00	0.97	5.19	1.02	4.81
Extrapolation Clients	7.98	4.41	5.76	0.60	4.11	0.60	3.91
Extrapolation 2 VMs	8.01	10.65	7.69	0.55	5.66	0.47	5.51
Extrapolation 3 VMs	10.02	4.59	8.55	0.68	6.98	0.72	5.26

(e) Mean Error (%)

Fig. 11: Evaluation

read requests exhibited the highest prediction error with a mean error of 8.32%, however, still a very good value.

B. Extrapolation with Respect to the Number of Clients

In this section, we increase the workload intensity and set the number of threads to 25% and 50% above the calibrated range. More specifically, we compare 160 mean response time measurements for all combinations of parameters shown in Table II against results obtained using the queueing models. Depending on the configuration, mean read and write measurements are in [2.74 ms, 44.86 ms] and [5.29 ms, 59.05 ms], respectively. The prediction results are illustrated in Figure 11b and Figure 11e and discussed next.

Random Read Requests. For cached data, the mean error is 7.98%. In general, the model exhibits a higher accuracy for larger request sizes. Still, for small request sizes, the absolute error is less than 1 ms. For uncached data, the model also exhibits a very high prediction quality. The mean error is 4.41%. Overall, the models predict the system performance very well.

Sequential Read Requests. For the requests, the model exhibits a high accuracy with a mean error of 5.76%. Also, the model exhibits a higher accuracy for larger request sizes.

Random Write Requests. For cached data, the model again exhibits a high prediction accuracy. The mean error is 0.60%. For uncached data, we obtain a mean error of 4.11%.

Sequential Write Requests. Overall, the model exhibits an excellent accuracy with a mean error of 0.60%.

Summary. When extrapolating the number of threads, the performance models exhibited very high quality. The mean error was 3.91%. Similar to the interpolation scenario, read requests were predicted slightly worse than write requests with a mean error of 6.05% and 1.77%, respectively. While random read requests exhibited the highest prediction error with a mean error of 6.19%, the error was just slightly worse than for sequential read requests.

C. Extrapolation with Respect to the Number of VMs

By using two and three virtual machines, we increase the load on the system up to 300% of the calibrated range. More specifically, we compare 480 mean response time measurements for all combinations of the parameters shown in Table II against results obtained using the queueing models. Depending on the configuration, mean read and write measurements are in [2.12 ms, 90.33 ms] and [4.10 ms, 148.20 ms], respectively. The prediction results are depicted in Figure 11c, Figure 11d, and Figure 11e and discussed in the following.

Random Read Requests. For cached data, the model exhibits a high accuracy with a mean error of 9.01%. For uncached data, the model predicts also well with a mean error of 7.62%.

Sequential Read Requests. Again, the model exhibits a consistently high accuracy. The mean error is 8.12%.

Random Write Requests. For cached data, the model exhibits an excellent prediction accuracy with a mean error of 0.62%. For uncached data, the mean error is 6.32%.

Sequential Write Requests. Here, the model exhibits an excellent prediction accuracy with a mean error of 0.59%.

Summary. Even when using two and three virtual machines and scaling the load up to 300% of the calibrated range, the performance models exhibited a very high prediction accuracy with a mean error of 5.38%. Similar to the previous scenarios, the prediction for read requests was not as accurate as for write requests with a mean error of 8.25% and 2.51%, respectively. While the prediction accuracy was consistent for read requests, write requests were predicted best for sequential requests and random requests on cached data.

VI. Related Work

Many modeling approaches for storage systems in native environments exist, e.g., [16], [17], [18], [19], [20], however, they are only shortly mentioned here as such approaches strongly rely on low-level instrumentation and monitoring data, e.g., allocation of data to disk sectors, disk seek times, disk rotation time, or single disk utilization. In typical virtualized environments, such information is hardly available for storage users, if available at all, hampering the reliable parametrization of these models.

The work closely related to the approach presented in this paper can be classified into two groups. The first group is focused on modeling storage performance in virtualized environments. Here, Kraft et al. [4] present two approaches based on queueing theory to predict the I/O performance of consolidated virtual machines. Their first, trace-based approach simulates the consolidation of homogeneous workloads. The environment is modeled as a single queue with multiple servers having service times fitted to a Markovian Arrival Process (MAP). In their second approach, they predict storage performance in heterogeneous workload consolidation scenarios. They create linear estimators based on mean value analysis (MVA). Furthermore, they create a closed queueing network model, also with service times fitted to a MAP. Both methods use monitored measurements on the block layer that is lower than typical applications run. Moreover, both methods are focused on performance prediction of consolidation scenarios only without considering the performance effects due to changes in the workload intensity. In [21], Ahmad et al. analyze the I/O performance of VMware's ESX Server virtualization. They compare virtual to native performance using benchmarks. They further create mathematical models for the virtualization overhead. The models are used for prediction of I/O throughput degradation. To analyze performance interference in a virtualized environment, Koh et al. [22] manually run CPU bound and I/O bound benchmarks. While they develop mathematical models for prediction, they explicitly focus on the consolidation of different types of workloads, i.e., CPU and I/O bound. By applying different machine learning techniques, Kundu et al. [23] use artificial neural networks and support vector machines for dynamic capacity planning in virtualized environments. Further, Gulati et al. [24] present a study on storage workload characterization in virtualized environments, but perform no performance analysis.

The second group of related work deals with benchmarking and performance analysis of virtualized environments not specifically targeted at storage systems. Hauck et al. [25] propose a goal-oriented measurement approach to determine performance-relevant infrastructure properties. They examine OS scheduler properties and CPU virtualization overhead. Huber et al. [26] examine performance overhead in VMware ESX and Citrix XenServer virtualized environments. They create regression-based models for virtualized CPU and memory performance. In [27], Barham et al. introduce the Xen hypervisor comparing it to a native system as well as other virtualization platforms. They use a variety of benchmarks for their analysis to quantify the overall Xen hypervisor overhead. Iyer et al. [28] analyze resource contention when sharing resources in virtualized environments. They focus on modeling cache and core effects.

VII. Conclusion

Summary. We proposed a generic iterative performance model building methodology for virtualized storage systems and created I/O queueing models of a real-world representative environment based on IBM System z and IBM DS8700 server hardware. We analyzed the system environment in detail and created the models in three iterations. First, we modeled the storage server cache for requests with a fixed size on fully cached data. Then, we parametrized the service time of the service stations with the request size. Finally, we included the storage RAID array in the model to consider requests on uncached data. We evaluated the models in interpolation and extrapolation scenarios as well as scenarios where the workload was distributed on multiple virtual machines. Using our approach, we effectively created performance models with high predictive power. In the worst case, the prediction error was less than 11%. On average, the prediction error was less than 5% and 4% for interpolation and extrapolation, respectively. Even in scenarios where the workload was distributed on multiple virtual machines and scaled up to 300% of the calibration range, the prediction error was less than 6% on average.

Lessons Learned. Our main insight is the identification of an abstraction level for I/O performance models that allows easy parametrization yet enabling excellent prediction for a complex virtualized environment. More specifically: i) Our approach using iterative refinement captured the performance characteristics of the considered complex virtualized systems using only few load-dependent queueing stations. ii) Our fine-grained analysis revealed interesting relationships between the workload factors and the system performance. iii) We could effectively calibrate the performance models using end-to-end response time data, which is normally easy to obtain.

Approach Application. Generally, our approach is targeted as a generic systematic methodology for modeling the performance of virtualized storage systems. The models created in this paper assume a typical environment, where some of the I/O requests are handled by a storage cache and some by a RAID array. In virtualized environments, a user cannot control and hardly track where the data is stored and how it is allocated, therefore, the models do not depend on low-level system information or monitoring still providing predictions for average I/O performance. Once the relationships between the workload factors and the system performance are identified, the calibration for using the models requires only a small set of measurement points. Therefore, in a similar environment, the performance models can be created with approximately only as many measurements as needed to estimate the calibration coefficients and the workload scenario thresholds. Our current performance models support (predominantly) homogeneous workloads, e.g., logging, backup, and archiving services as well as media streaming applications and digital libraries. They can be used in different scenarios for capacity planning when the number of clients and the workload intensity increases. Furthermore, the models can be applied to evaluate deployment decisions and virtual machine consolidation scenarios.

Future Work. Next, we plan to extend the workload scenarios and the models to account for mixed workloads and more complex scenarios. Moreover, we plan to evaluate our performance modeling approach in further system environments.

Acknowledgments

This work was funded by the German Research Foundation (DFG) under grant No. RE 1674/5-1 and KO 3445/6-1, and the Czech Science Foundation (project GACR P202/10/J042). 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.

References

[1] TechNavio, "Global Server Virtualization Market 2012-2016," http://www.technavio.com/content/global-server-virtualization-market-2012-2016, 2013, last accessed: Mar 2013.

[2] S. Oliveira, K. Furlinger, and D. Kranzlmuller, "Trends in computation, communication and storage and the consequences for data-intensive science," in *IEEE HPCC-ICESS'12*, pp. 572 –579.

[3] 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: Mar 2013.

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

[5] P. Mell and T. Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, 2009.

[6] 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.

[7] S. Kounev, "Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets," *IEEE TSE*, vol. 32, no. 7, July 2006.

[8] D. Menascé and V. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall, 2000.

[9] D. Menascé, V. Almeida, L. Dowdy, and L. Dowdy, *Performance by Design: Computer Capacity Planning by Example*, ser. Prentice Hall science explorer. Prentice Hall, 2004.

[10] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, ser. Wiley-Interscience publication. Wiley, 2006.

[11] D. Menascé, V. Almeida, and L. Dowdy, *Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems*. New Jersey: Prentice-Hall, 1994.

[12] Q. Noorshams, S. Kounev, and R. Reussner, "Experimental Evaluation of the Performance-Influencing Factors of Virtualized Storage Systems," in *EPEW '12*, ser. LNCS, vol. 7587. Springer, 2012.

[13] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling passe," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, 2010.

[14] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson, "Estimating service resource consumption from response time measurements," in *VALUETOOLS '09*.

[15] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., ser. Springer Series in Statistics. Springer, 2011.

[16] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, and Contributors, *The DiskSim Simulation Environment - Version 4.0 Reference Manual*, Carnegie Mellon University, Pittsburgh, PA, 2008.

[17] P. Harrison and S. Zertal, "Queueing models of RAID systems with maxima of waiting times," *Performance Evaluation*, vol. 64, 2007.

[18] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt, "Analytical and Simulation Modelling of Zoned RAID Systems," *The Computer Journal*, vol. 54, 2011.

[19] E. K. Lee and R. H. Katz, "An analytic performance model of disk arrays," *SIGMETRICS Perform. Eval. Rev.*, vol. 21, no. 1, 1993.

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

[21] 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.

[22] 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*.

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

[24] A. Gulati, C. Kumar, and I. Ahmad, "Storage workload characterization and consolidation in virtualized environments," in *VPACT '09*.

[25] M. Hauck, M. Kuperberg, N. Huber, and R. Reussner, "Ginpex: deriving performance-relevant infrastructure properties through goal-oriented experiments," in *QoSA-ISARCS '11*.

[26] N. Huber, M. von Quast, M. Hauck, and S. Kounev, "Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments," in *CLOSER '11*.

[27] 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," *SIGOPS Oper. Syst. Rev.*, vol. 37, 2003.

[28] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell, "VM3: Measuring, modeling and managing VM shared resources," *Computer Networks*, vol. 53, 2009.