2014 IEEE 34th International Conference on Distributed Computing Systems Workshops

Automated Modeling of I/O Performance and Interference Effects in Virtualized Storage Systems

Qais Noorshams*, Axel Busch*, Andreas Rentschler*, Dominik Bruhn*, Samuel Kounev*, Petr Tůma†, Ralf Reussner*
*Karlsruhe Institute of Technology, Germany (e-mail: [lastname]@kit.edu)
†Charles University in Prague, Czech Republic (e-mail: tuma@d3s.mff.cuni.cz)

***Abstract*—Modern IT systems frequently employ virtualization technology to maximize resource efficiency. By sharing physical resources, however, the virtualized storage used in such environments can quickly become a bottleneck. Performance modeling and evaluation techniques applied prior to system deployment help to avoid performance issues. In current practice, however, modeling I/O performance is usually avoided due to the increasing complexity of modern virtualized storage systems. In this paper, we present an automated modeling approach based on statistical regression techniques to analyze I/O performance and interference effects in the context of virtualized storage systems. We demonstrate our approach in three case studies creating performance models with two I/O benchmarks. The case studies are conducted in a real-world environment based on IBM System z and IBM DS8700 server hardware. Using our approach, we effectively create performance models with excellent prediction accuracy for both I/O-intensive applications and I/O performance interference effects with a mean prediction error up to 7%.**

I. Introduction

Over the past decades, the I/O resource demands of modern IT systems have increased exponentially [1]. Many I/O-intensive applications, such as video streaming portals, online file storage services, and social networking applications, are increasingly often deployed in virtualized environments to exploit efficiency benefits. Sharing physical resources in such scenarios requires explicit analysis techniques of I/O performance and interference effects to avoid performance bottlenecks during operation.

In current practice, however, virtualized storage and its performance-influencing factors are often neglected or treated as a black-box due to their complexity. Several explicit modeling approaches considering I/O-intensive applications in virtualized environments have been proposed, e.g., [2] and our previous work in [3], which has shown, however, that such manual performance modeling approaches require a significant amount of time and expertise.

In this paper, we present a fully automated modeling approach to analyze I/O performance and interference effects in the context of virtualized storage systems based on three statistical regression techniques. Our goal is to maximize the practical usability for such models and to minimize the manual effort to create them. We demonstrate the benefit of our approach in three case studies creating performance models using two different I/O benchmarks. The case studies comprise models of general I/O performance-influencing factors, of mixed application workloads, and of I/O performance interference effects in mixed virtualized environments. The case studies are conducted in a real-world environment based on IBM System z and IBM DS8700 server hardware. Using our approach, we effectively create performance models with excellent prediction accuracy. Using the best regression technique, the mean prediction error is between 2.1% and 7% in the case studies.

In summary, the contribution of this paper is two-fold: i) We present a framework for automated modeling of I/O performance and interference effects in virtualized storage systems. ii) We demonstrate our approach in three case studies with two different benchmarks in a real-world environment based on the state-of-the-art server technology of the IBM System z and IBM DS8700. We extend our previous work [4] by i) presenting the architecture of our automation and by ii) evaluating our approach in two new case studies including the extraction of performance models for I/O performance interference effects in a heterogeneous, multi-VM scenario.

This paper is organized as follows: Section II presents our methodology. In Section III, we illustrate the design of our automation. An extensive evaluation of our approach in three case studies is given in Section IV. Finally, Section V reviews related work and Section VI concludes the paper.

II. Methodology

To systematically create performance models of virtualized storage systems, we identify a general process (cf. Figure 1):

1. *Modeling Target Specification*
   First, the system environment as well as the main modeling goals are specified, e.g., evaluating performance-influencing factors or analyzing performance interference effects.
2. *Measurement Space Configuration*
   Depending on the goals and the system environment, the *measurement configuration* (or *parameter space*) is defined. This is used to perform the measurements in the next step, and based on it, the *independent variables* used in the following steps are determined.
3. *Systematic Measurements*
   After the measurement space and the scenario are identified, the measurement space is explored by performing systematic measurements. The measurement metrics (e.g., response time and throughput) are used as *dependent variables* in the following steps.
4. *Regression Optimization*
   Regression techniques are used to model the effect of the *independent variables* on the *dependent variables*. Since the

optimal parameterization of regression techniques is usually scenario dependent, we use the measurement data to optimally tune the regression techniques for the specific scenario.

5. *Regression-based Performance Modeling*
Finally, the performance models are created using multiple regression techniques, where a model is created for each dependent variable. The models can be compared to choose the best model for the scenario.

Steps 3 – 5 are fully automated and are only briefly elaborated in the following due to space constraints. More details can be found in [4].

A. Systematic Measurements

In a virtualized storage environment, a wide variety of performance-influencing factors exists, cf. [5]. To evaluate the factors, we support multiple scenarios based on different benchmarks. A first scenario is applying fine-grained measurements to evaluate the impact of the factors on the performance. This allows to create a performance model of the system environment with fine-grained configuration aspects. A second scenario is analyzing mixed application workloads. This allows to evaluate the system environment in typical scenarios and to create a performance model, e.g., to predict the effect of workload scaling when the number of users increases. Finally, we also consider combinations of both to analyze interference effects on co-located virtual machines (VMs). For each scenario, the parameter space is then fully explored to extract a system profile with systematic measurements, which are used for performance modeling. We demonstrate each of the three scenarios in Section IV.

B. Regression Optimization

Regression techniques usually have configuration parameters (e.g., the maximum number of modeling terms) that influence their effectiveness in a certain application scenario. To optimally tune the regression techniques, i.e., find the best configuration parameters for a given scenario, we apply a heuristic search algorithm (introduced in [4]). For given data, we iteratively search for best fitting regression parameters that minimize the average root mean square error of a 10-fold cross-validation.

C. Regression-based Performance Modeling

In the following, we briefly introduce the three regression techniques considered in this paper.

1) *Multivariate Adaptive Regressions Splines (MARS)* [6] 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 $\beta_0, \dots, \beta_n$. In this paper, we consider MARS with *interaction terms*, which includes terms that are a product of one or more hinge functions.

2) *Classification and Regression Trees (CART)* [7] are binary decision trees with conditions in their non-leaf nodes and constant values in their leaf nodes. To determine the value of the dependant variable corresponding to a set of values of the independent variables, the evaluation starts at the root and the condition in this node is checked. If the condition is true, the left edge is followed, otherwise the right edge. This is repeated until a leaf is reached.

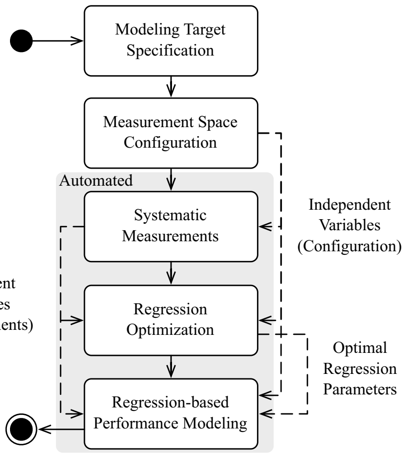
Modeling Target Specification
Measurement Space Configuration
Automated
Systematic Measurements
Regression Optimization
Regression-based Performance Modeling
Independent Variables (Configuration)
Dependent Variables (Measurements)
Optimal Regression Parameters

Fig. 1: Performance Modeling Process (dashed: data flow)

3) *Cubist* forests [8], [9] are based on *M5* trees [10], which are binary decision trees with conditions in their non-leaf nodes and linear regression models in their leaf nodes. Compared to M5, Cubist introduces two extensions. First, it follows a boosting-like approach, i.e., it creates a set of trees instead of a single tree. Second, it combines model-based and instance-based learning, i.e., it can adjust the prediction of unseen points by the values of their nearest training points.

III. Automation Design

Our methodology is automated as part of our *Storage Performance Analyzer (SPA)* framework[1]. Next, we present the high-level design and the components of SPA.

A. High-level Architecture

Illustrated in Figure 2, our SPA framework basically consists of a benchmark harness that coordinates and controls the execution of benchmarks and a tailored analysis library used to process and evaluate the collected measurements. The benchmark harness runs on a controller machine managing the measurement process. Using SSH connections, the benchmark controller first configures the benchmark, then it executes the target workload, and it finally collects the results into an SQLite database. The benchmark controller guarantees a synchronized execution of experiments on multiple targets, i.e., on multiple VMs that can be deployed on the same system. Currently, we have integrated two benchmarks into our framework. We use the open source *Flexible File System Benchmark*[2] (FFSB) for a fine-grained analysis and the *Filebench* benchmark[3] to emulate mixed application workloads, e.g., a file server workload. The evaluation is automated using analysis functions implemented using the open source statistics tool *R* [11]. The analysis library comprises the analysis, optimization and regression functions we created and applied for regression optimization and performance modeling, cf. Section II-B and II-C.

[1] http://sdqweb.ipd.kit.edu/wiki/SPA
[2] https://github.com/FFSB-prime (extension of http://ffsb.sf.net)
[3] https://github.com/Filebench-Revise (includes fixes)

B. Components

Basically, our framework comprises a composite *benchmarking component*, a composite *performance modeling component*, and a *persistence component* that serves as an interface between the former two components.

The benchmarking component realized in Java contains a *benchmark controller* that explores the parameter space and coordinates the benchmark runs accordingly. The benchmark controller is connected to the *benchmark driver*, which is used to configure and execute the benchmark. The benchmark driver uses an internal *remote execution component* to communicate with the actual benchmark, which is deployed on the target system. In our implementation, the remote execution component employs SSH connections, but it could be easily changed to use another connection type. The benchmark controller saves the measurement results using the persistence component.

The performance modeling component is integrated into *R*. The *datastore interface* can load and prepare the measurement data, e.g., by filtering irrelevant data. Both the *regression optimization* and the *regression modeling* component can further process this data or use other data specified by the user. The regression optimization component comprises the optimization algorithm outlined in Section II-B and uses the *regression techniques* whose implementations are provided by *R* libraries. The regression modeling component automatically creates the models with the considered regression techniques.

IV. Case Study

In this section, we present three case studies demonstrating our approach in different scenarios.

A. System Under Study

In our case studies, we consider a representative virtualized environment based on the IBM mainframe *System z* and the storage system *DS8700*. The System z combined with the DS8700 represents a high-end virtualized environment that can be used as a building block of private cloud infrastructures. The System z provides processors and memory, whereas the DS8700 provides storage space.

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 [12].

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 z/Linux VMs with ext4 file system and, unless specified otherwise, NOOP I/O scheduler as it has recently been used as default scheduler in virtualized environments [13]. We focus the 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.

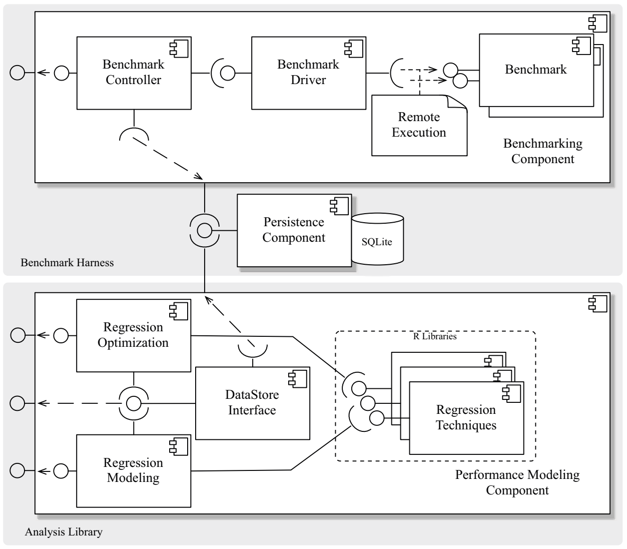

Fig. 2: Framework Architecture and Components

TABLE I: FFSB Experimental Setup Configuration

I/O scheduler	CFQ, NOOP
Threads	100
File set size	1 GB, 25 GB, 50 GB, 75 GB, 100 GB
Request size	4 KB, 8 KB, 12 KB, 16 KB, 20 KB, 24 KB, 28 KB, 32 KB
Access pattern	random, sequential
Read percentage	0%, 25%, 30%, 50%, 70%, 75%, 100%

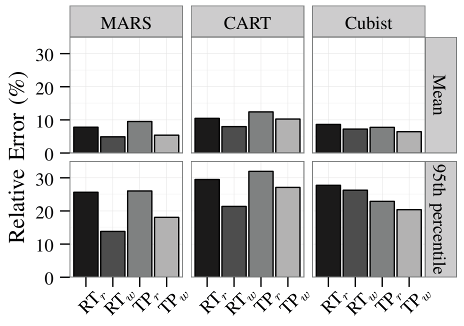

Fig. 3: Prediction Quality (Case Study I)

B. Modeling Performance Factors

Setup. In our first case study, we use FFSB to systematically benchmark the system environment and major performance-influencing parameters. The detailed setup configuration is chosen representatively and shown in Table I. The parameter space is fully explored leading to a total of 1120 measurement configurations. For each configuration, we configure a one minute warm up and a five minute measurement phase. The measurement phase consists of five intervals of one minute length each. In one minute, the benchmark obtains ~575 000 measurement samples on average and between ~90 000 and 2 800 000 measurement samples depending on the configuration, while the mean response times of read and write requests are in [2.2 ms, 70.4 ms] and [2.8 ms, 60.7 ms], respectively, and the throughputs of read and write requests are in [4.02 MB/s, 386.70 MB/s] and [2.99 MB/s, 171.1 MB/s], respectively.

Performance Models. For each regression technique we create four different models: A read response time model

(RT$_r$), a write response time model (RT$_w$), a read throughput model (TP$_r$), and a write throughput model (TP$_w$).

Prediction Accuracy. We evaluate 100 configuration scenarios with parameter values chosen completely randomly within the configured ranges (e.g., 80 GB file set size, 30 KB request size and so on). For each scenario, we compare the model predictions with measurements on the real system.

Figure 3 shows the mean and the 95th percentile, i.e., the value below which 95% of the prediction errors fall, of the relative error for the various models. Overall, the models perform very well and especially MARS and Cubist exhibit excellent prediction accuracy with less than 7% and 8% error, respectively. The CART trees are highly splitted due to the parameter tuning step, yet with ~10% mean error their accuracy is acceptable. Across the four models, the mean of the 95th percentile of the prediction error is 20.89%, 24.32%, and 27.48% for MARS, Cubist, and CART, respectively.

Optimization Improvement. Finally, to evaluate the improvements in model accuracy achieved through our regression optimization step, we compare the accuracy of the models when using the optimized regression parameters vs. the standard parameters, respectively. We evaluate the performance prediction error for each model with 100 completely random configurations within the configured ranges.

Overall, especially MARS and CART benefit from the parameter optimization exhibiting an average error reduction of 66.30% and 74.08%, respectively. The error reduction for Cubist is 15.7%. We evaluate the statistical significance of the optimization results in a *paired t-test*. Here, the p-value of both MARS and CART is less than $2.2e^{-16}$ and the p-value of Cubist is $3.39e^{-4}$, thus, confirming that the optimization is statistically significant.

C. Modeling Mixed Applications

Setup. In our second case study, we use Filebench to emulate a composite application workload. We use the configuration shown in Table II emulating a mail server workload consisting of mixed file system operations, such as file creation and deletion as well as whole file reads and append operations of random size. We analyze all combinations of varying the number of clients (threads), the number of files, and the mean file sizes as summarized in Table II, leading to a total of 576 measurement configurations. For each configuration, we use a one minute warm up phase and a five minute measurement phase. During the latter, the benchmark obtains ~980 000 read and append samples on average and between ~825 000 and 1 100 000 samples depending on the configuration, while the mean response times of read and append requests are in [0.36 ms, 2.74 ms] and [0.40 ms, 1.70 ms], respectively, and the throughputs of read and append requests are in [5.95 MB/s, 62.95 MB/s] and [5.60 MB/s, 7.60 MB/s], respectively.

Performance Models. For each regression technique we create four different models: A read response time model (RT$_r$), an append response time model (RT$_a$), a read throughput model (TP$_r$), and an append throughput model (TP$_a$).

TABLE II: Filebench Experimental Setup Configuration

Workload type	Mail server workload
Threads	16$^{(*)}$, 32, 48, 64, 80, 96
Files	1000$^{(*)}$, 5000, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000
Mean file size	4 KB, 16 KB$^{(*)}$, 32 KB, 48 KB, 64 KB, 96 KB, 128 KB, 192 KB

$^{(*)}$ default value

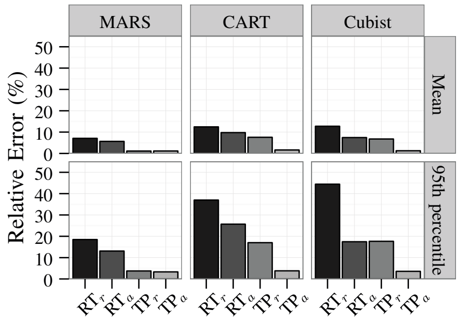

Fig. 4: Prediction Quality (Case Study II)

Prediction Accuracy. We evaluate 100 configuration scenarios with parameter values chosen randomly within the configured ranges. For each scenario, we compare the model predictions with measurements on the real system.

Figure 4 shows the mean and the 95th percentile of the relative error for the various models. Overall, the models perform very well and especially MARS exhibits excellent performance prediction accuracy with less than 4% mean error. The Cubist and CART models are also very accurate with ~7% and 8% mean prediction error, respectively. Across the four models, the mean of the 95th percentile of the prediction error is very good for MARS with 9.66%, while it is 20.78% and 20.88% for Cubist and CART, respectively.

Optimization Improvement. We evaluate the improvements in model accuracy achieved through our regression optimization step, similar as in the previous case study. Overall, the optimization reduces the prediction error by 4.2%, 27.3%, and 8.7% for MARS, CART, and Cubist, repectively. In a *paired t-test*, the p-value of MARS, CART, and Cubist is $5.26e^{-4}$, $9.63e^{-14}$, and $2.19e^{-3}$, respectively. Thus, the optimization is again statistically significant.

D. Modeling Performance Interference

Setup. In our third case study, we analyze the performance interference among VMs, i.e., the performance of a workload in a given VM as a function of the workload running in a co-located VM. We explicitly focus on machines with a constant and equal workload intensity, but with different workload types, e.g., read- or write-intensive workload. Since I/O performance isolation in virtualized environments is widely an open challenge, varying the workload intensity would lead to obvious performance interference. For the measurements, we use both FFSB as well as Filebench emulating a file server workload consisting of mixed file system operations, such as file creation and deletion as well as whole file reads, whole file writes, and append operations of random size. The benchmarks run in respective virtual machines with the configurations chosen representatively and shown in Table III. We use all combinations shown leading to a total of 200 measurement configurations. We use a one minute warm up phase and a five minute measurement phase.

A first indication of the performance interference is the number of read, append, and write operations of the file server workload emulated with Filebench in VM1. Depending on the configuration of FFSB in VM2, the number of operations in VM1 varies between ~375 000 and 700 000 with a mean of ~550 000 operations, while the number of operations in VM2 varies between ~420 000 and 3 100 000 with a mean of ~1 300 000 operations. For VM1, the mean response times of read, append, and write requests are in [13.08 ms, 34.08 ms], [11.63 ms, 32.12 ms], and [17.90 ms, 52.81 ms], respectively, and the throughputs of read, append, and write requests are in [54.80 MB/s, 108.30 MB/s], [3.40 MB/s, 6.20 MB/s], and [55.30 MB/s, 109.10 MB/s], respectively, depending on the configuration. For VM2, the mean response times of read and write requests are in [4.34 ms, 26.75 ms] and [7.45 ms, 37.59 ms], respectively, and the throughputs of read and write requests are in [1.38 MB/s, 218.00 MB/s] and [2.21 MB/s, 95.20 MB/s], respectively, depending on the configuration.

Performance Models. To model the interference effects, for each regression technique, we create a total of 10 models: For VM1, we use the configuration in VM2 as independent variables and create a read response time model (RT^1_r), an append response time model (RT^1_a), a write response time model (RT^1_w), a read throughput model (TP^1_r), an append throughput model (TP^1_a), and a write throughput model (TP^1_w). For VM2, we do not need to use the configuration in VM1 as independent variables explicitly as the configuration in VM1 remains constant in this case study. We create a read response time model (RT^2_r), a write response time model (RT^2_w), a read throughput model (TP^2_r), and a write throughput model (TP^2_w) using the configuration in VM2 as independent variables.

Prediction Accuracy. We evaluate 100 configuration scenarios with parameter values chosen randomly within the configured ranges. We use these parameter values as configuration for the FFSB benchmark and predict both the performance of the FFSB benchmark and the performance interference on the co-located VM running Filebench. For each scenario, we compare the model predictions with measurements on the real system.

Figure 5 and 6 show the mean and the 95th percentile of the relative error for the various models. Especially interesting is the prediction error indicated in Figure 5 showing how the models are able to predict very accurately how different workloads, e.g., write-intensive or read-intensive, affect the performance on the co-located virtual machine. This is significant since, as mentioned above, the response time spreads between 261% and 295% for the operations depending on the co-located workload.

Overall, the models perform significantly well for both virtual machines. As before, MARS exhibits the best performance prediction accuracy with ~2.1% and 5.0% mean prediction error for VM1 and VM2, respectively. The Cubist models are also very accurate with ~2.6% and 10.0% mean prediction error for VM1 and VM2, respectively. Finally, CART models

TABLE III: Hybrid Experimental Setup Configuration

Filebench Workload Parameters @VM1	
Workload type	File server workload
Threads	50$^{(*)}$
Files	10000$^{(*)}$
Mean file size	128 KB$^{(*)}$
FFSB Workload Parameters @VM2	
Threads	50
File set size	1 GB, 2 GB, 5 GB, 10 GB
Request size	4 KB, 8 KB, 16 KB, 32 KB, 64 KB
Access pattern	random, sequential
Read percentage	10%, 30%, 50%, 70%, 90%

$^{(*)}$ default value

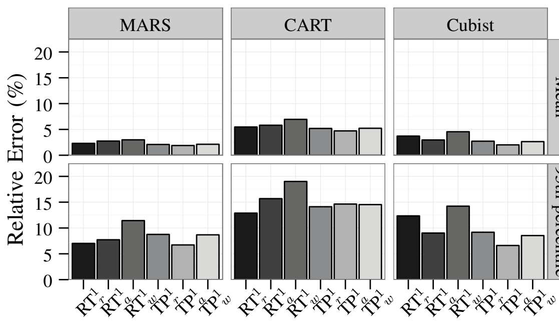

Fig. 5: Prediction Quality VM1 (Case Study III)

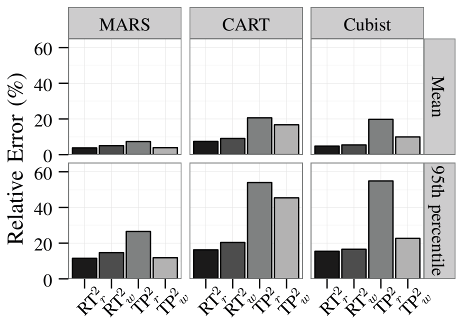

Fig. 6: Prediction Quality VM2 (Case Study III)

exhibit the highest error in this case study with ~5.2% and 13.5% mean prediction error for VM1 and VM2, respectively. For VM1, the mean of the 95th percentile of the prediction error across the six models is very good for MARS and Cubist with 8.39% and 9.99%, respectively, while it is 15.14% for CART. For VM2, the mean of the 95th percentile of the prediction error across the four models is 16.15%, 27.39%, and 34.00% for MARS, Cubist, and CART, respectively.

Optimization Improvement. We again evaluate the improvements in model accuracy achieved through our regression optimization step, similar as in the previous case studies. In summary, the optimization improvement is between 5.0% and 32.3% and the p-value is at most $1.46e^{-3}$ indicating that the optimization is statistically significant.

V. Related Work

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. [2] 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 consolidation scenarios with heterogeneous workloads. 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. In [14], Ahmad et al. analyze the I/O performance in 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 I/O throughput degradation predictions. By applying different machine learning techniques, Kundu et al. [15] use artificial neural networks and support vector machines for dynamic capacity planning in virtualized environments.

The second group of related work analyzes I/O performance interference effects in virtualized environments. Closest to our work, Chiang et al. [16] use linear and second degree polynomials to model I/O performance interference. They use the models for scheduling algorithms to manage task assignments in virtualized environments. As input in their model, they use read and write request arrival rates as well as local and global CPU utilization. However, they do not distinguish between request sizes or sequential and random requests, for instance. Our measurements have shown that such factors have a significant impact on I/O performance. In [17], Yang et al. present a framework that uses a set of pre-defined workloads to identify characteristics of the hypervisor I/O scheduler. Furthermore, they show how this information can be exploited to deteriorate the I/O performance of co-located virtual machines. To analyze performance interference also across resources, Koh et al. [18] manually run CPU-bound and I/O-bound benchmarks. They develop mathematical models for prediction of normalized performance compared to the isolated performance of the benchmark. In an experimental study, Pu et al. [19] analyze CPU and network I/O performance interference in a Xen-based environment. They conclude that the least performance degradation occurs for workloads with different resource demands, i.e., CPU and network I/O demand or mixing small with large network demands.

VI. Conclusion

We presented a fully automated approach to systematically create and optimize I/O performance models of virtualized storage systems based on three statistical regression techniques. We demonstrated the benefit of our approach in three case studies creating performance models with two different I/O benchmarks. The case studies comprised models of general I/O performance-influencing factors, of mixed applications workloads, and of I/O performance interference effects in mixed virtualized environments. The case studies were conducted in a real-world environment based on IBM System z and IBM DS8700 server hardware. Overall, we effectively created performance models with excellent prediction accuracy. Interestingly, of the three considered regression techniques, MARS performed better than CART and Cubist in every scenario. The mean prediction error of MARS was between 2.1% and 7% in the case studies. This fact, however, was also due to the regression parameter optimization approach reducing the error of MARS by up to 66.3%. Moreover, the regression parameter optimization approach reduced the prediction error of every considered technique with statistical significance in every case study and every regression technique considered in the paper.

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

References

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

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

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

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

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

[6] J. H. Friedman, "Multivariate Adaptive Regression Splines," *Annals of Statistics*, vol. 19, no. 1, pp. 1–141, 1991.

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

[8] RuleQuest Research Pty Ltd, "Data Mining with Cubist," http://rulequest.com/cubist-info.html, 2012, last accessed: Jan 2014.

[9] M. Kuhn, S. Witson, C. Keefer, and N. Coulter, "Cubist Models for Regression," http://cran.r-project.org/web/packages/Cubist/vignettes/cubist.pdf, 2012, last accessed: Jan 2014.

[10] J. R. Quinlan, "Learning with Continuous Classes," in *AI '92*. World Scientific.

[11] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org

[12] 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.

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

[14] 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.

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

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

[17] Z. Yang, H. Fang, Y. Wu, C. Li, B. Zhao, and H. Huang, "Understanding the Effects of Hypervisor I/O Scheduling for Virtual Machine Performance Interference," in *CloudCom '12*.

[18] 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*.

[19] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," in *CLOUD '10*.