Inside-Out: Reliable Performance Prediction for Distributed Storage Systems in the Cloud

Chin-Jung Hsu*, Rajesh K Panta†, Moo-Ryong Ra† and Vincent W. Freeh*
*Department of Computer Science, North Carolina State University
†AT&T Labs Research
Email: chsu6@ncsu.edu, rpanta@research.att.com, mra@research.att.com, vwfreeh@ncsu.edu

***Abstract*—Many storage systems are undergoing a significant shift from dedicated appliance-based model to software-defined storage (SDS) because the latter is flexible, scalable and cost-effective for modern workloads. However, it is challenging to provide a reliable guarantee of end-to-end performance in SDS due to complex software stack, time-varying workload and performance interference among tenants. Therefore, modeling and monitoring the performance of storage systems is critical for ensuring reliable QoS guarantees. Existing approaches such as performance benchmarking and analytical modeling are inadequate because they are not efficient in exploring large configuration space, and cannot support elastic operations and diverse storage services in SDS.**

This paper presents *Inside-Out*, an automatic model building tool that creates accurate performance models for distributed storage services. Inside-Out is a black-box approach. It builds high-level performance models by applying machine learning techniques to low-level system performance metrics collected from individual components of the distributed SDS system. Inside-Out uses a two-level learning method that combines two machine learning models to automatically filter irrelevant features, boost prediction accuracy and yield consistent prediction. Our in-depth evaluation shows that Inside-Out is a robust solution that enables SDS to predict end-to-end performance even in challenging conditions, e.g., changes in workload, storage configuration, available cloud resources, size of the distributed storage service, and amount of interference due to multi-tenants. Our experiments show that Inside-Out can predict end-to-end performance with 91.1% accuracy on average. Its prediction accuracy is consistent across diverse storage environments.

***Index Terms*—software-defined storage; performance prediction; storage; performance modeling;**

I. Introduction

Many storage systems are moving away from dedicated appliance-based storage model to software-defined storage (SDS), which separates software that provisions and manages storage from the hardware that provides raw physical storage. This trend is partly driven by the tremendous growth of data and the emergence of cloud applications that operate in a multi-tenant environment with diverse workload characteristics [1], [2], [3]. As a result, the rigid appliance-based model with tightly-coupled hardware and software features is no longer cost-effective, lacks flexibility and does not scale well. SDS systems are increasingly abandoning centralized storage services in favor of distributed systems like Ceph, HDFS, Swift [4], [5], [6], etc. Distributed storage systems are attractive because they scale well, allowing storage services to grow or shrink, based on storage demands. They are also better suited to handle diverse multi-tenant workloads.

Providing reliable quality of service (QoS) to storage applications is critical in an SDS environment shared by multiple applications with diverse usage patterns. However, in a distributed storage environment, it is challenging to provide storage QoS in a consistent and reliable manner. Practical deployments of modern distributed storage systems like Ceph are composed of a larger number of individual storage components that can interact in a complex manner. Diverse and time-varying storage workloads and performance interference in a multi-tenant environment further complicate the reliable assurance of storage QoS. Reliable and accurate monitoring of high-level storage performance metrics (e.g. throughput and IOPS) is critical for providing storage QoS guarantees. However, monitoring end-to-end storage performance is difficult in a distributed storage service. Instrumenting user applications to measure storage performance is not always practical. Performing benchmark tests in production systems also has practical limitations since they interfere with storage application workload. Furthermore, running exhaustive benchmark experiments to cover diverse application workloads, deployment topologies, and large configuration parameter space is time-consuming and impractical in many cases. Building accurate analytical performance models, on the other hand, is also difficult for the reasons mentioned above.

This paper proposes the idea of using low-level system metrics (e.g., CPU usage, RAM usage and network I/O) as a proxy for measuring high-level performance (e.g., end-to-end IOPS and throughput) of distributed storage applications. *We design, implement and evaluate a practical tool, called "Inside-Out", that applies machine learning techniques to the low-level metrics collected from individual components of a distributed storage system to accurately estimate high-level storage performance metrics—like throughput and IOPS—of the entire distributed storage system.* We believe that a tool like Inside-Out can serve as an important component of the overall SDS architecture.

Inside-Out takes a black-box modeling approach, which does not require knowledge about distributed storage system protocol, workload characteristics, and deployment topology. Inside-Out relies upon machine learning techniques to automatically derive an accurate end-to-end performance model. We explore several well-known machine learning algorithms

1060-9857/16 $31.00 © 2016 IEEE
DOI 10.1109/SRDS.2016.23

IEEE computer society

including linear regression, decision tree learning, and ensemble methods [7], [8], and conclude that there does not exist an one-size-fits-all algorithm that can work in all prediction cases. Hyperparameter tuning [9], [8], model selection [10] and feature selection [11], [12] all turn out to be too complicated for optimizing prediction accuracy. In contrast, Inside-Out uses a two-level learning method that automatically selects important features, boosts prediction accuracy, and achieves consistent prediction. This two-level learning method pipelines two supervised learning algorithms to eliminate irrelevant features while avoiding overfitting problems.[1]

Inside-Out offers several key benefits. Unlike traditional analytic performance modeling approach, Inside-Out is more generic, and therefore can be more easily applied to different storage services. Different from previous work [14], [15], [7], [16], [17], [8], [18], Inside-Out does not require information about system configuration and application workload. Due to the self-learning property, its performance prediction accuracy increases with more data, and it can adapt to changes in the system by continuously learning the system behavior.

We evaluate Inside-Out using Ceph [4] running on an OpenStack-based SDS platform. The low-level performance metrics are collected from participant virtual machines running various components of a Ceph storage service.[2] Our in-depth evaluation shows that Inside-Out generates end-to-end performance models with 91.1% prediction accuracy on average. More importantly, as discussed above, Inside-Out is generic in nature as it captures the behavior of the storage system by analyzing low-level system metrics (that are protocol and application agnostic). Furthermore, we demonstrate that Inside-Out can provide reliable hints for performance monitoring tasks even in the presence of evolving workload characteristics, changing storage configuration and interfering tenants. We also show that Inside-Out is reliable in estimating end-to-end performance even when the storage system expands or shrinks—in our evaluation, we find that Inside-Out provides reliable performance prediction even when the storage system is up to four times larger than the one used for building machine learning models during the training phase. Lastly, Inside-Out is able to learn new storage behavior over time (Section V-E).

II. Background

In this section, we provide a brief overview of the SDS system that we are building. We present how Inside-Out fits in the overall SDS architecture. As shown in Figure 1, users specify high-level storage requirements to the SDS Planner component. These requirements include storage capacity, expected reliability (e.g., five 9s of storage reliability), expected throughput and IOPS, read/write workload specification, etc.

[1] Overfitting describes the situation when a model captures the relationship of noisy data but not the underlying relationship [13]. Overfitting becomes more prominent in the presence of high dimensional data, which is the case in this work.

[2] Our approach is not limited to VM-based environments. It can be applied to container-based and bare-metal storage servers as well.

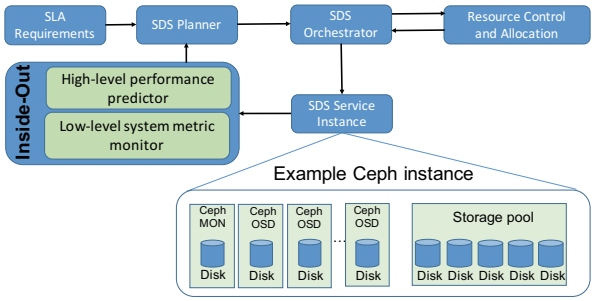

Fig. 1: SDS architecture. Inside-Out provides performance feedback to SDS Planner to continuously meet storage QoS requirements.

The SDS Planner is the brain of SDS. It takes requirements from users and creates a detail implementation plan that is expected to meet those requirements. We have developed some components of the SDS Planner and are in the process of developing others. We avoid details of the SDS Planner since that is not the focus of this paper. The SDS planner hands the generated plan to an SDS Orchestrator, which is responsible for creating an SDS storage service instance. In our current implementation, the SDS Orchestrator generates the OpenStack Heat template automatically based on the plan generated by the SDS Planner. The SDS Orchestrator uses the "Resource Control and Allocation" component to allocate the right amount of cloud resources (e.g. network bandwidth, storage IO bandwidth, etc.) to various components of the SDS storage service instance. The Resource Control and Allocation component also manages the placement of VMs, volumes and other cloud resources properly in the underlying cloud infrastructure to maximize resource usage. Again, we omit the details of this component since they are out of scope for this paper. Finally, the SDS Orchestrator creates the SDS instance. The Inside-Out component monitors the low-level system metrics of the generated instance and develops performance models for predicting high-level storage performance metrics like throughput and IOPS. The output of Inside-Out is used iteratively by the SDS Planner to ensure that the user-specified performance requirements are met.

III. Mapping from Low to High

This section discusses the guiding principles and challenges in using low-level performance metrics to build accurate end-to-end performance models for a distributed storage system.

A. Important Considerations

1) General low-level metrics: Since our goal is to provide a tool for estimating the end-to-end performance of a diverse set of storage systems, the inputs to our model need to be generic in nature, i.e. they need to be independent of storage application or the distributed protocols used by such applications. An SDS provider should be able to obtain the input metrics without instrumenting storage application or requiring domain knowledge about the storage application. Low-level

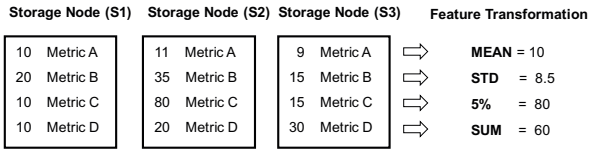

Fig. 2: Four statistical features used in Inside-Out to capture load and internal status of a distributed storage system. The numbers and metrics represent low-level performance data collected from storage nodes.

system metrics (e.g. CPU utilization, memory usage, network IO, etc.) satisfy these requirements. DeepDive [19] uses low-level metrics to identify performance anomaly for a running VM . To the best of our knowledge, this paper presents the first study that maps low-level system metrics to high-level end-to-end performance of a distributed storage service.

2) Capture important features of a distributed storage system: A distributed storage system can expand or shrink on demand based on dynamic storage requirements. The performance model should be able to capture the current scale of the deployment, the bottlenecks, and the average and variance in performance of individual components of the distributed system. For each low-level system metric collected from various components of the distributed system, we use four statistical variables to characterize the behavior of a distributed system (see Fig. 2). The statistical variable *mean* and *std* describe whether the impact of the workload is evenly distributed among storage components. The *sum* variable represents the scale of the deployment, while the variable *5%* (top 5 percentile) captures the hot spot situations. The feature transformation from raw system metrics to these four statistical values also allows Inside-Out to apply the uniform input format for developing performance models for distributed systems at different scales.

B. Feature Selection

In this work, we collect 32 low-level performance metrics, using dstat, from two components of Ceph namely monitor (MON) and Object Storage Daemons (OSD). These measurements are then transformed using the process described in Fig. 2 (refer to Section IV-A for more details).

Selecting the "right" features is a challenging task [11], [12]. Furthermore, for our case, the right feature set is not deterministic. Table I shows the *model accuracy* of different learning methods when modeling read throughput. We see that all learning methods achieve high model accuracy even though they choose different features. The model accuracy was obtained using k-fold cross validation (k=10), a common technique for assessing model accuracy. The training data is partitioned into k disjoint sets. A single data partition is used for validation purpose and the remaining $k-1$ partitions are used for training data. Although all models yield good model accuracy, they perform poorly and inconsistently when the storage environment changes. In Fig. 3, we show the prediction accuracy under three types of changes in the storage environment—increase in the size of the distributed storage system, read workload and individual storage IO request size. These algorithms (discussed later in Section IV-B) do not yield consistent prediction accuracy any more. For example, Lasso can still predict well when workload has changed but Decision Tree cannot. On the contrary, Decision Tree performs better than Lasso when the size of the storage system increases. We suspect this is caused by the large feature space, which leads to the overfitting problem [13], [20]. Next, we manually remove most features and select only a few with a trial-and-error strategy. As shown in Figure 3, we see significant improvement in some cases, but not all. Since an SDS environment can change over time, it is important for our model to provide consistent prediction accuracy in the presence of such changes.

Although Hyperparameter tuning [9], [8], model selection [10] and feature selection [11], [12] have been proposed as potential solutions, it is challenging to use them in practice, not to mention the complexity of automating this task. PCA (Principle Component Analysis) is another potential solution [21]. PCA transforms original data into a lower dimension while keeping high fidelity. However, PCA has several limitations. First, PCA is not scale invariant. Not all performance metrics are comparable and therefore, there is no standard way to scale these metrics. Second, PCA assumes Gaussian distribution in data points; however, many storage workloads have Pareto distribution [22]. Third, determining a good number of components is also a challenging task. In our case, PCA does not address the problems. In fact, Fig. 3 shows that it can further degrade prediction accuracy.

C. A Two-Level Approach

Instead of performing feature selection or dimension reduction, we propose a generic two-step approach that can improve the consistency of prediction accuracy. In the first step, we use some heuristic methods to filter out irrelevant features. Then, in the second step, we apply machine learning algorithms to build performance models with the reduced feature set. The intuition behind this idea is that it is difficult to determine the most important performance features but it is relatively easy to eliminate unimportant features. For example, the features which are not in the top 100 list after step one can be labeled as unimportant features.

IV. The Inside-Out Design

In this section, we present the design of Inside-Out. We also discuss the trade-offs among a set of representative machine learning algorithms and propose a two-step learning technique for mitigating overfitting problems.

A. Collecting and Pre-Processing Low-Level Metrics

Inside-Out collects general low-level system metrics from individual machines running the distributed storage service. However, the raw collected data suffers from various problems due to inefficiency of data collectors, system clock skews, incomparable data formats, workload outliers, bursty system

TABLE I: Important features selected by different algorithms are not deterministic

Lasso			Ridge			Elastic Net			Decision Tree			Random Forest		
osd	network.send	sum	**osd**	**network.recv**	**mean**	osd	network.send	sum	osd	disk.read	sum	osd	disk.read	sum
osd	disk.writ	sum	**osd**	**disk.read**	**5%**	osd	disk.writ	sum	osd	network.send	sum	osd	network.send	sum
osd	**cpu.sys**	**sum**	**osd**	**load.15m**	**std**	osd	disk.read	sum	osd	network.recv	sum	osd	disk.writ	sum
osd	io.read	sum	osd	network.send	sum	osd	cpu.sys	sum	osd	disk.writ	sum	**osd**	**network.recv**	**sum**
osd	vm.minpf	mean	osd	tcp.tim	std	osd	tcp.lis	sum	**mon**	**memory.buff**	**mean**	osd	memory.cach	sum
mon	**memory.used**	**5%**	osd	network.recv	std	osd	io.read	std	osd	cpu.sys	sum	osd	memory.buff	mean
mon	**memory.cach**	**5%**	osd	load.5m	std	osd	io.read	sum	osd	vm.alloc	sum	osd	memory.buff	5%
osd	tcp.lis	sum	osd	cpu.idl	sum	osd	vm.minpf	mean	osd	vm.minpf	5%	mon	io.writ	sum
osd	io.read	std	osd	cpu.wai	sum	osd	io.writ	mean	**mon**	**cpu.idl**	**std**	osd	memory.buff	sum
osd	io.writ	mean	osd	cpu.sys	sum	**mon**	**memory.used**	**sum**	mon	memory.cach	sum	mon	vm.free	sum
96.20%			96.60%			96.18%			96.78%			96.94%		

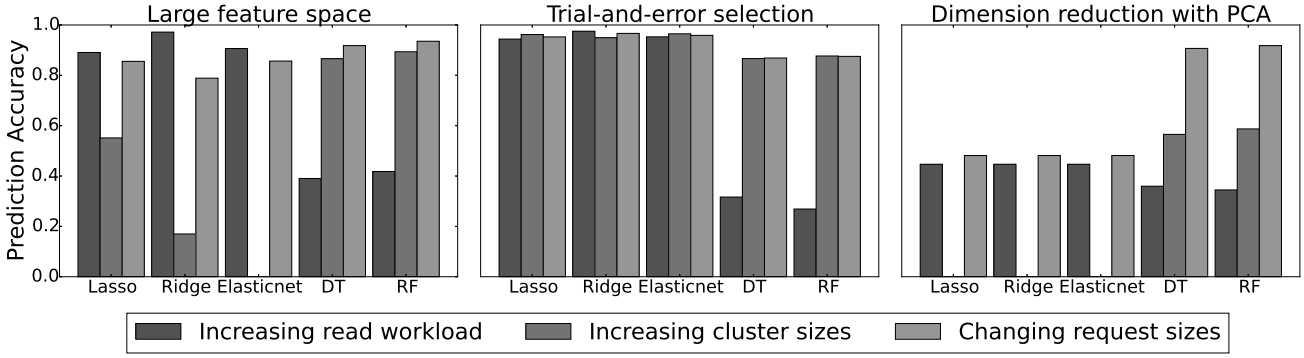

Fig. 3: Prediction accuracy is inconsistent due to the large feature space. Learning methods fail to select the right features in some cases. Dimension reduction (PCA with 10 components) does not help in this case. In the trial-and-error case, we select a subset of metrics, e.g. *mean(disk.read)*, *sum(network.recv)* and *std(cpu.usr)*.

anomalies, etc. The noisy data can lead to unstable and inaccurate performance models. Inside-Out performs a series of data pre-processing functions to address these issues.

1) Monitoring storage components: We collect low level system metrics of the underlying operating system to capture resource utilization (e.g. cpu, memory, disk, network usage, etc.). The low-level performance metrics are sampled with one-second granularity. Such data can be collected from *libvirt*, Ganglia, instrumented hypervisors [23] and *Ceilometer* in OpenStack. We use *dstat* monitoring tool (with option: -tcly -mg –vm -dr -n –tcp –float) to collect these data.

2) Data smoothing: Building a performance model with data collected at one second granularity is challenging because system data can exhibit high variance at small time scales, e.g. due to dynamic/bursty workloads and interference among co-located tenants. Furthermore, the storage IO operation needs to pass through a series of software layers between the storage client and the back-end raw physical storage device. The long storage IO path can introduce high variability in resource utilization at smaller time scales. For example, HDFS and Ceph both replicate data blocks across storage nodes distributed in physically disjoint servers, racks or even datacenters. To address the uncertainties due to complex IO path spanning several software layers, we compute the moving average of the collected performance data. We have empirically found that an one-minute window for processing the moving average is sufficient to eliminate outliers from the raw data.

3) Timestamp alignment: Proper time synchronization among participating servers is essential to correlate data collected from those servers. We use NTP for time synchronization. The average timestamp of all nodes is taken as the basis for time alignment.

4) Feature transformation for a distributed storage system: As mentioned earlier, elasticity is an important feature of SDS, since it needs to adjust its size based on storage demand. Thus our model should be able to accurately predict end-to-end performance at arbitrary deployment scales. However, the data collected from different scales may have different dimensions. For instance, Ceph with 10 Object Storage Servers (OSDs) generates 10 copies of low-level performance metrics, while Ceph with 5 OSDs generates less number of data points. This makes it hard to train and build a unified model. As mentioned in Section III-A, we use *mean, sum, std, and 5%* statistical variables to capture *load-balanced, non-load-balanced, hotspot, and aggregate* workload situations.

In summary, Inside-Out collects 32 raw low-level system metrics with one-second granularity. Inside-Out applies proper time alignment and moving average with one-minute windows for stabilizing performance data. Then it calculates *mean*, *std*, *sum* and *5%* of individual metrics collected from multiple machines. This ensures that our performance model can accept input data for systems with varying scales of deployment, while preserving important characteristics of a distributed storage system. For training and validation purposes, we

measure end-to-end performance metrics (IOPS, throughput, and latency) every 5 seconds using COSbench [24], and take average over the one-minute window. Next, we describe how we build end-to-end performance models in order to capture the relationship between low-level system metrics and end-to-end throughput and IOPS.

B. Exploring Learning Methods

Our goal is to build a model that accurately predicts end-to-end throughput and IOPS by analyzing only the low-level metrics of a distributed storage system. We explore several algorithms, including statistical regression [25], [20], decision tree learning and random forests learning [20], [7]. For statistical regression, we mainly focus on linear regression techniques, which can be extended to support non-linear regression by expanding features that simulate, for example, quadratic terms [26]. We did not find this necessary in our application and exclude the discussion in this paper.

Lasso is a least square linear regression technique with L1-norm regularization. The L1 penalty function leads to a sparse solution, which has an effect of restricting the number of selected variables. This property is useful for figuring out important features, especially when the number of variables or features is large. *Ridge* is similar to Lasso but instead uses L2-norm regularization, which has the effect of group selection of variables. This property does not restrict the number of variables selected by the prediction model and therefore, the prediction accuracy might degrade and become inconsistent when the number of input features to the training model is large. *Elastic Net* combines both advantages—it does group selection while enforcing sparsity. Based on our data set, Lasso and Elastic Net have similar prediction performance, and Ridge shows larger variance. The *Decision Tree* (DT) learning uses a top-down approach and recursively partitions data to fit target values. The tree-based model is easy to interpret and scales well to large datasets. *Random Forests* (RF) is an ensemble method that uses multiple decision trees [20]. RF improves a single decision tree in many ways, e.g., accuracy, efficiency, and robustness.

To summarize, linear regression models assume a linear relationship and might oversimplify the storage behavior. Nonetheless, it has the potential to exhibit better generalization for extrapolating performance prediction for the unknown behavior case (the pattern not included in the training dataset). On the other hand, the tree-based learning can achieve good model accuracy (perfectly fits the training data), but it can easily lead to overfitting problems. Its prediction accuracy decreases, for example, under different storage workloads, as shown in Fig. 3.

C. Two-level Training

The fundamental challenge in building an effective prediction model from a large set of features is the overfitting problem. One way to address this problem is to perform manual feature selection. However, this approach is problematic because the right set of features depend on application types, deployment topology, resource constraint, etc.

Instead, we propose a two-level training process that filters out irrelevant features at the first step and then builds models by using the reduced set of features in the second step. To this end, Inside-Out pipelines Ridge and Lasso together, where Ridge filters features in coarse-granularity and then Lasso builds the prediction model. We choose Ridge as the filtering algorithm because it is not a sparse solution and considers all features. We then apply exhaustive grid search to find the optimized score for important features. We use $\alpha\times$ *median(coefficients)* derived from Ridge as the threshold.

For comparison, we consider Decision Tree with Lasso (Auto-DTL) and RandomForest with Lasso (Auto-RFL). Our evaluation shows Inside-Out outperforms consistently across all prediction cases, and boosts prediction accuracy in several scenarios, where the linear regression models fail to generalize the behavior of a distributed storage system. We also experimented by using Lasso and Elastic Net as the filter algorithm but did not find comparable performance with Inside-Out.

Inside-Out uses the following pseudo code to generate an end-to-end performance model.

Algorithm 1 Inside-Out Model Building

Input: low-level performance metrics from distributed nodes
Output: an end-to-end performance model
Initialisation
$thresholds = \{\alpha_1, ..., \alpha_N\}$
$m1$ = filtering algorithm $\rightarrow Ridge$
$m2$ = model algorithm $\rightarrow Lasso$
k = k-fold cross validation
$score = 0$
Data preprocessing (refer to Section IV-A)
alignment of input data
calculate moving average across metrics
feature transformation for the distributed scenario
Grid Search
for all $t \in thresholds$ **do**
  $features$ = execute $m1$ with threshold t
  $score, m$ =max(crossvalidation(k, $m2$, $features$))
end for
return m with maximum $score$

V. Evaluation

In this section, we present a comprehensive evaluation of Inside-Out. We demonstrate that Inside-Out can accurately predict end-to-end performance, i.e., throughput and IOPS, using low-level system metrics and is applicable to a wide range of realistic scenarios.

A. Setup

We choose Ceph [4] as a target distributed storage service for our evaluation and use COSBench [24] to generate various types of storage workloads. COSBench supports several object storage protocols, including *librados* for Ceph, and provides

a set of knobs to change storage traffic pattern. Table III lists Ceph and COSBench configurations used in our experiments.

We collected benchmarking data from an OpenStack-based SDS platform. The cluster has 16 machines, and each machine has 16 cores, 24GB memory and 250GB disk space. Each machine has 1Gbps network interface connected to a 10Gbps switch. The dataset is collected from about 5300 benchmark runs. The total dataset is composed of about 15.2 million records, each of which is a vector of 32 low-level performance data. The end-to-end performance data collected from COSBench contains 3 million records. The combined dataset is about 24GB, collected over two weeks.

B. The Comparison Method

Our goal is to find a function $f(X_t)$ that predicts the end-to-end performance, where X_t is a vector that describes the internal status at time t of a distributed storage service. We say a model is accurate if $f(X_t) = \hat{y_t} \simeq y_t$, where y_t is the ground truth (measured at the client side) and $\hat{y_t}$ is the predicted values. To interpret performance models, we are interested in four indicators: 1) the overall prediction accuracy, 2) the goodness-of-fit, 3) the consistency across diverse scenarios and 4) the consistency across prediction instances.

First, we use mean absolute percentage error (MAPE) to compute prediction accuracy as

$$max(1 - \frac{\sum_{t=1}^{n} |\frac{y_t - \hat{y_t}}{y_t}|}{n}, 0) \quad (1)$$

where n is the length of the observation period. We restrict the scope of prediction accuracy between 0 to 1 because the prediction accuracy can be negative (e.g. when y_t is small).

Second, we use the coefficient of determination R^2 to interpret *Goodness-of-Fit*, which is less than or equal to one [8]. Third, we examine whether a performance model can present consistent prediction in various SDS scenarios. Last, we further analyze the probability density function of prediction decisions for different categories of prediction scenarios.

We consider prediction of throughput and IOPS for both read and write operations, and use the following terms TP_r, TP_w, OP_r and OP_w for read throughput, write throughput, read IOPS and write IOPS, respectively.

C. Baseline: Prediction Performance on Static Deployment

We evaluate prediction accuracy of Inside-Out under a variety of scenarios with different storage workloads and configurations (Table II). In this subsection, we focus on a static deployment scenario with one storage tenant running on a distributed Ceph storage service that does not expand or shrink in terms of number of VMs used for running Ceph. Later, we evaluate more challenging scenarios in which the Ceph cluster expands or shrinks based on user demand, and storage traffic of multiple tenants interfere with each other.

1) Can Inside-Out handle diverse workloads?: An SDS application needs to handle various request volumes, object/file sizes and different ratios of read/write workloads. First we examine whether Inside-Out can achieve accurate and consistent predictions when workload changes.

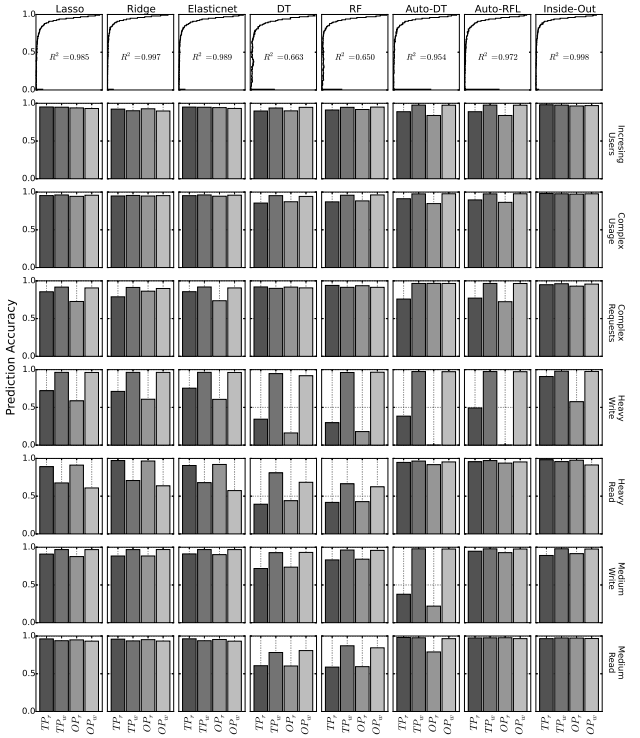

Fig. 4: Analysis of performance models with diverse workloads. Each bar is the average prediction accuracy. The top row is the probability density function of prediction accuracy for each performance model.

Changing user behavior. We increase the number of concurrent clients to stress the Ceph cluster. The "increasing users" scenario changes the number of COSBench clients and the "complex usage" scenario increases the worker threads of each client. As shown in Fig. 4, all prediction models perform well. The linear regression technique performs slightly better than the tree-based learning. The linearly increasing load is well captured by linear models because of proportional change in low-level metrics. When we switch to the "complex request" scenario, the variable request size slightly changes the behavior of Ceph, affecting prefetching and caching. We observe that the linear regression methods (Lasso, Ridge and Elastic Net) show drops in accuracy, e.g. 20% in the OP_r case; however, Inside-Out maintains good accuracy. The tree-based learning shows comparable predictions (5-10% lower) with Inside-Out in these settings.

Varying I/O pattern. Next, we consider workloads with different read/write ratios. Fig. 4 shows that varying workload poses a big challenge to performance models. The linear regression methods (Lasso, Ridge and Elastic Net) present better prediction accuracy than tree-based models (DT, RL). In addition, we observe that several models make poor predictions of TP_r and TP_r. The reason is that read behavior is largely affected by *cache*, and large read variance contributes to low prediction accuracy. Inside-Out performs consistently well,

TABLE II: Common scenarios that storage behavior can change in a software-define storage environment

	Scenario	Training Dataset	Prediction Dataset	Explanation
Changing Workload	Increasing users	{1, 2}	{4}	The number of client virtual machines running COSBench.
	Complex usage	{1, 2, 4, 8}	{16, 32}	The number of threads for all benchmark clients.
	Complex request	512KB	1-1024KB	The request size (either static or variable) of the workload, configured in COSBench.
	Write intensive	{50, 75, 100}	{25, 0}	The percentage of read operations specified in the workload generation. The read and write percentages are 100 in total.
	Read intensive	{0, 25, 50}	{75, 100}	
	Medium write intensive	{0, 50, 100}	{25}	
	Medium read intensive	{0, 50, 100}	{75}	
Reconfiguration	Reconfigure Ceph	{1}	{2}	The number of Ceph monitor daemons.
	Scale-up instances	m1.small	m1.medium	The instance type of the virtual machines running Ceph is upgraded to a powerful one. A m1.small instance has one core and 2GB memory and m1.medium has two cores and 4GB memory. Note that in this setting, the configuration of disk I/O remains the same.
	Medium network SLO	unrestricted	500 Mbps	The network bandwidth of virtual machines is limited at 500 Mbps. We use the Linux tool *tc* for network throttling
	Low network SLO	unrestricted	250 Mbps	Network bandwidth is limited at 250 Mbps.
Elasticity	Scale out to *n*	{4, 6, 8, 10}	{20, 30, 40}	The total number of Ceph OSDs. Note that each OSD is running in a virtual machine and different OSDs can run on the same physical servers (10 servers in total).
	Shrink in to *n*	{20, 30, 40}	{4, 6, 8, 10}	Similar to the above, but the cluster size is decreased.

TABLE III: Ceph and COSBench settings for data collection.

Parameters	Values
Ceph version	9.2 (Infernalis)
# of physical nodes	16
Storage back end	Logic Volume (iSCSI)
# of storage nodes	{4, 6, 8, 10, 20, 30, 40}
# of drivers	{1, 2, 4}
# of workers	{1, 2, 4, 8}
Request size	{512KB, 1-1024KB}
Duration	180 sec
# of containers	{64}
# of objects	{1024}
read/write ratio	{100/0, 75/25, 50/50, 25/75, 0/100}

whereas the three linear regression techniques show accuracy drops. One exception is the OP_r prediction in the *write-intensive* scenario even though TP_r prediction is accurate. As we will show later in Section V-E, "over- and under-predictions" cause such behavior. The self-learning property of Inside-Out improves its prediction accuracy as it keeps learning the new storage behavior.

Summary. The linear regression models achieve high prediction accuracy, great goodness-of-fit (> 0.98) and consistency in prediction for many instances (see the distribution of prediction accuracy in Fig. 4), but they are not consistent across all prediction scenarios. Inside-Out achieves good prediction accuracy across all cases consistently because the two-level approach filters out many irrelevant features in the first step, thereby presenting a smaller relevant feature space to the second step. The tree-based learning methods (DT and RF) do not show consistent prediction across all scenarios. Auto-DT and Auto-RFL, which use DT and RF as the filter algorithms, are not as consistent as Inside-Out.

2) Can Inside-Out handle different system configurations?: We study whether low-level metrics can capture the storage behavior when it is reconfigured by tenants. The results are reported in Fig. 5.

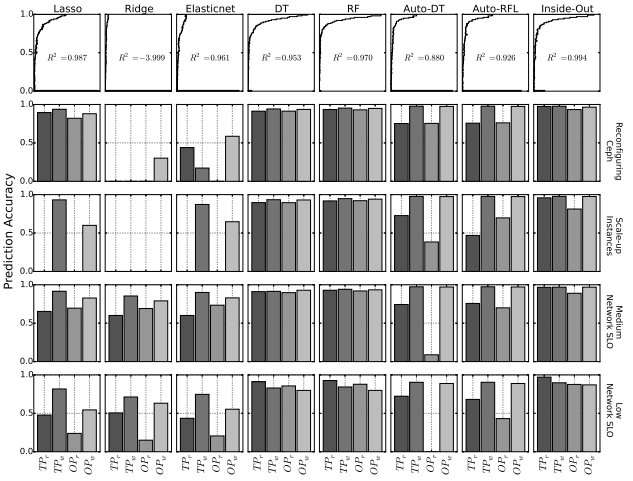

Fig. 5: Comparison of performance models when the storage service is reconfigured: Ceph, VMs and network SLOs

Reconfiguring Ceph. The first change is to add one extra Ceph monitor daemon. Ridge and Elastic Net fail to generate consistent predictions, but Lasso is able to achieve around 80% to 90% prediction accuracy. DT, RF and Inside-Out have very close prediction accuracies, but Auto-DRL and Auto-RFL perform slightly worse in predicting TP_r and OP_r.

Scale-up instances. Increasing CPU and memory allocation to Ceph VM instances improves Ceph's ability to handle more requests. In this test, we change the instance type from m1.small (1 vCPU, 2GB memory) to m1.medium (2 vCPUs, 4GB memory). The linear models are unable to predict TP_r and OP_r, but Inside-Out's two-level learning performs well by avoiding the overfitting problem.

Network SLOs. Here we consider the case where the amount of network bandwidth allocated to Ceph VMs is

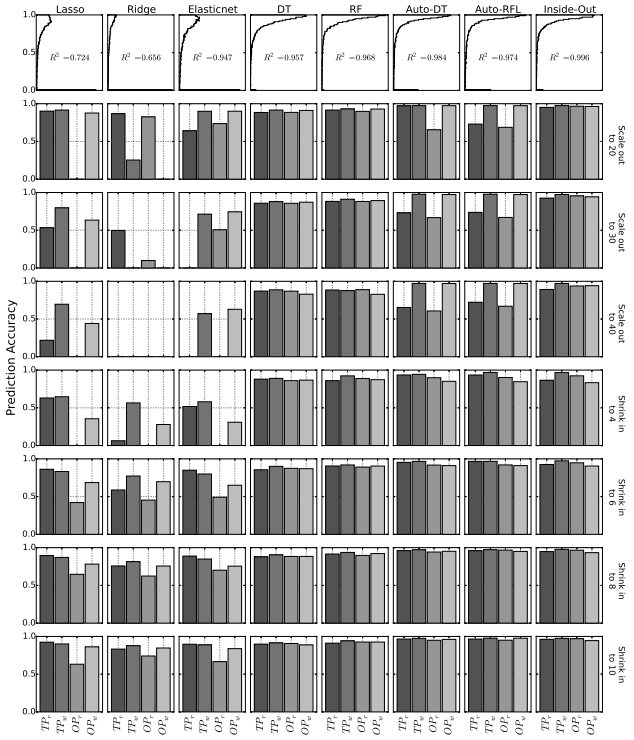

Fig. 6: Comparison of model performance in the on-demand scaling scenario. In the scale-out scenario, a performance model trained with 10 Ceph nodes is used to predict the performance of Ceph cluster with 20, 30 and 40 nodes.

limited. We use Linux network throttling tool *tc* to limit network bandwidth at 500 Mbps and 250 Mbps for medium and low bandwidth SLOs, respectively. We observe that linear models without the two-level method do not show comparable prediction accuracy across both throughput and IOPS predictions. The tree-based learning models, on the other hand, achieve 80% to 90% accuracy, comparable to Inside-Out.

Summary. Tree-based learning (DT, RF) models demonstrate promising prediction in terms of prediction accuracy and consistency. Lasso, Ridge and Elastic Net show inconsistent behavior in the above four scenarios. Inside-Out, on the other hand, provides consistent predictions and improves Lasso, from 23.9% to 87.6% in the extreme case.

D. Prediction Performance in a Multi-tenant Cloud

1) Elastic Storage (On-demand Scaling): A storage service needs to grow or shrink its capacity on demand. We evaluate Inside-Out's ability to capture the storage behavior at different system scales. As shown in Fig. 6, we use training data collected from 4, 6, 8, and 10 nodes, and then predict the performance of 20, 30, and 40 nodes. We also evaluate prediction accuracy in the "shrink-in" scenario. For both read and write throughput predictions, the linear models exhibit high variance. In the OP_r and OP_w cases, the prediction results are not even comparable to the other methods. Inside-Out, on the other hand, helps mitigate this issue, and achieves more than 90% accuracy. With increasing sizes of the storage, the prediction accuracy decreases because the prediction target becomes increasingly different from the training data. Running a benchmark test against a very large system is time-consuming. Here we demonstrate that Inside-Out can predict performance for systems that are four times larger than the system for which training data was collected.

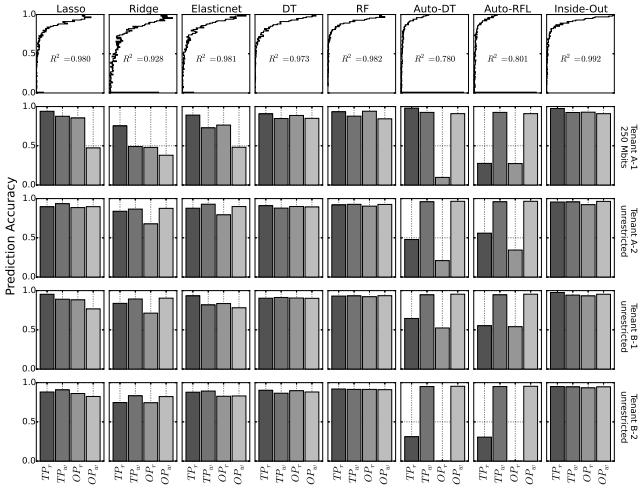

Fig. 7: Prediction accuracy in a multi-tenancy scenario. Tenant A-1 is co-located with Tenant B-2. Tenant A-1 is throttled at 250Mbps. Tenant B-1 and B-2 are co-located without any traffic throttling.

2) Multi-Tenancy: Next we evaluate Inside-Out's ability to adapt to performance interference among storage tenants. We consider two cases for this evaluation. Each tenant runs a Ceph cluster with 10 OSDs separately, but tenants share the same 10 physical machines. In the first case, we restrict the bandwidth of only the first tenant at 250Mbps. In the second case, we run two concurrent Ceph clusters but without network throttling. Fig. 7 shows that most prediction models are able to achieve more than 80% accuracy. The linear models like Ridge and Elasticnet yield lower prediction accuracies in some cases; however, Inside-Out performs well consistently. Performance interference is challenging for a performance model designed for an isolated environment. This evaluation demonstrates that the low-level performance metrics are good proxies for measuring the end-to-end storage performance, even in a shared SDS environment.

E. Online Self-Learning

Next we create several synthetic workloads with mixed read/write ratios. This synthetic workload spans 12 hours with 720 stages. Each stage is 60-second long on average, with a standard deviation of 20 seconds. We run four COSBench virtual machines for benchmarking and up to eight threads per COSBench client, with 10 Ceph OSDs and one monitor daemon. We use Inside-Out to build an initial performance model with the training dataset described in Section V. Fig.

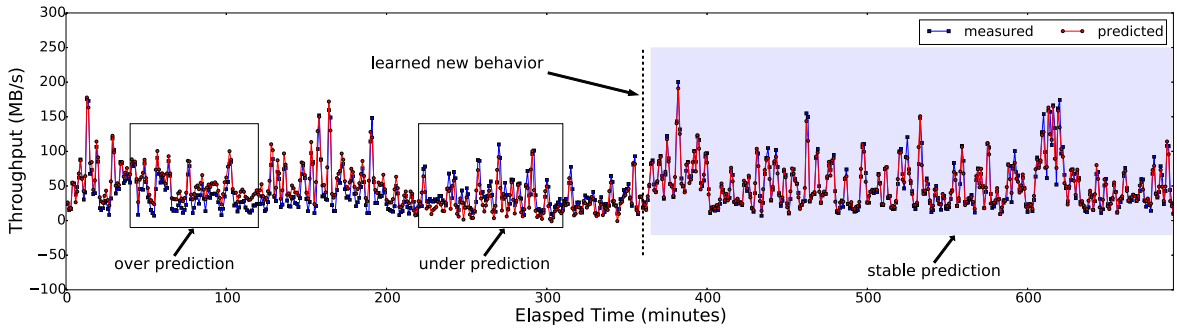

Fig. 8: Application of Inside-Out to real time prediction of read throughput on a 10-node Ceph cluster. Inside-Out starts from a simple prediction model trained by our collected benchmarking data. Inside-Out keeps learning the storage behavior while improving prediction accuracy over time.

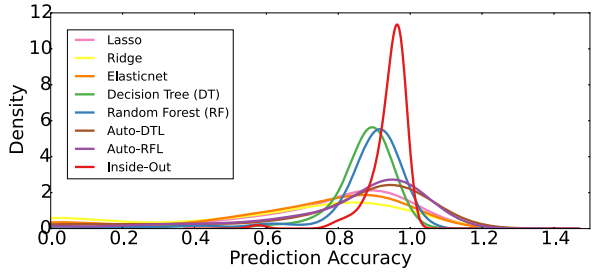

Fig. 9: Kernel density function of prediction accuracy from Fig. 4 to Fig. 7. Each colored line represents the density function of a modeling approach. Inside-Out is more consistent and accurate across almost every prediction case.

8 shows the prediction result for read throughput. We can observe that the generated model can capture the overall trend, but suffers from over and under predictions. This is because our training dataset is generated from a relatively clean environment, i.e. the OS memory is flushed before any benchmarking process. However, in the online prediction setting, cache is continuously consumed by non-stop client requests, which causes the real time storage behavior to be different from the training dataset. With continuous monitoring of the performance of the storage service, we use Inside-Out to generate a new performance model at the sixth hour. Fig. 8 shows that Inside-Out learns the new storage behavior and therefore, the over- and under-prediction issues are greatly mitigated. By continuously learning the storage behavior, SDS can accurately capture performance changes and therefore is able to provide reliable storage service.

F. Discussion

We have shown that low-level performance metrics are useful to predict end-to-end throughput and IOPS. Our evaluation has shown that low-level performance metrics are good indicators of end-to-end throughput and IOPS. Most existing performance models exhibit an inconsistent prediction behavior in the presence of diverse storage scenarios, such as changing workload, storage reconfigurations, growing/shrinking storage and multi-tenancy environments. Our proposed two-level learning method can greatly improve prediction accuracy and yield consistent behavior. Machine learning provides powerful tools, but they need to be used intelligently to achieve the best prediction accuracy. Fig. 9 shows the kernel density function of prediction accuracy across all prediction scenarios. Inside-Out is a clear winner in terms of accuracy and consistency. More importantly, Inside-Out is able to learn new storage behavior, thereby enabling the performance model to adapt to the complex SDS environment.

VI. Related Work

Storage performance modeling has been extensively explored in many prior works. Three common modeling techniques are analytical, simulation and data-driven approaches [15], [16], [18]. The analytical (mathematical) model requires domain knowledge to manually identify the factors that affect performance[15], [16]. Kelly et al. use a probability model to predict response time for an enterprise storage array. Ruemmler et al. found that a disk is too complex to model with analytical methods and designed a disk simulator to characterize storage behavior [14]. However, the simulation approach becomes inefficient when searching a large design space [16].

The data-driven or the measurement-based approach uses measurement data to derive a prediction model. Wang et al. [7] adopt classification-and-regression-tree (CART) to predict the response time of a single disk and a disk array. The authors propose request-level and workload-level device models for different prediction granularities. Yin et al. also use the regression tree to predict storage throughput and latency [17]. Their work mainly focuses on multiple workloads, and proposes a scalable model, by combining related workload features. Noorshams et al. extensively analyze four different types of algorithms (including linear regression and CART models) and apply to IBM storage servers [8]. They also propose an optimization technique to search for parameters that can improve prediction accuracy. Their proposed parameter optimization complements our work for improving prediction accuracy. To sum up, Inside-Out uses only low-level

performance metrics and does not require workload profiles and storage configurations. Unlike these studies, Inside-Out is primarily designed for diverse storage services that need to be reconfigured frequently to meet users' demand.

Mesnier et al. [27] propose a novel black-box approach that can describe the performance difference between two storage devices. It may be possible to borrow this idea and apply it to the unseen configuration scenarios as described in Section V-C2. With this approach, we can study the performance difference between two configurations and create a combined model with better prediction accuracy. Bodik et al. propose an exploration policy for quick collection of essential data required to train a performance model [28]. This policy can reduce the time required for offline and online model training.

Chen et al. propose SLA decomposition that combines profiling and queuing model to derive resource thresholds for meeting application SLA [29]. Machine learning has also been applied to performance modeling for virtual machines (VMs). DeepDive uses the classification technique to detect performance anomaly among VMs [19]. In [26], the authors apply regression and artificial neural network to model performance of a single VM. Our work focuses on performance prediction of a distributed storage system that includes multiple software and hardware entities.

VII. Conclusion

Ensuring end-to-end performance in software-defined storage (SDS) requires accurate performance models. This paper presents Inside-Out, a tool that provides reliable and consistent prediction of end-to-end performance of distributed storage systems using low-level system metrics. Our evaluation indicates that Inside-Out is able to generate accurate prediction models even when the storage environment differs significantly from the training phase. Inside-Out is generic in nature because it does not use application or protocol specific data for building performance models. Although we used Ceph as an example distributed storage service to evaluate Inside-Out, we believe it should be applicable to other storage systems as well with minor modifications.

References

[1] Software-Defined Storage. http://www.research.att.com/articles/featured_stories/2015_09/software-defined-storage.html

[2] E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu, "IOFlow: A Software-Defined Storage Architecture," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13)*, Farmington, Pennsylvania, USA, November 2013, pp. 182–196.

[3] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the tenant-provider gap in cloud services," in *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC 2012)*, San Jose, CA, USA, October 2012, pp. 1–14.

[4] Ceph. http://ceph.com

[5] Apache Hadoop. http://hadoop.apache.org/

[6] OpenStack. http://www.openstack.org

[7] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. Ganger, "Storage device performance prediction with CART models," in *Proceedings of the joint international conference on Measurement and modeling of computer systems (SIGMETRICS'04/Performance'04)*, New York, NY, USA, June 2004, pp. 588–595.

[8] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner, "Predictive performance modeling of virtualized storage systems using optimized statistical regression techniques," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE 2013)*, Prague, Czech Republic, April 2013, pp. 283–294.

[9] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Machine Learning*, vol. 46, no. 1-3, pp. 131–159, 2002.

[10] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," *International Joint Conference on Artificial Intelligence*, vol. 14, pp. 1137–1143, 1995.

[11] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[12] Y. Saeys1, I. Inza2, and P. Larrañaga2, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, pp. 2507–2517, 2007.

[13] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, Oct. 2012.

[14] C. Ruemmler and J. Wilkes, "Introduction to disk drive modeling," *Computer*, vol. 27, no. 3, pp. 17–28, mar 1994.

[15] E. Shriver, A. Merchant, and J. Wilkes, "An analytic behavior model for disk drives with readahead caches and request reordering," in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems (SIGMETRICS '98/PERFORMANCE '98)*, Madison, WI, USA, June 1998, pp. 182–191.

[16] T. Kelly, I. Cohen, M. Goldszmidt, and K. Keeton, "Inducing Models of Black-Box Storage Arrays Inducing Models of Black-Box Storage Arrays," HP Laboratories, Tech. Rep., 2004.

[17] L. Y. L. Yin, S. Uttamchandani, and R. Katz, "An Empirical Exploration of Black-Box Performance Models for Storage Systems," in *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS 2006)*, Monterey, CA, USA, September 2006, pp. 433–440.

[18] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing: modeling techniques and their applications," *Journal of Internet Services and Applications*, vol. 5, no. 1, p. 11, September 2014.

[19] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, and R. Bianchini, "DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments," in *Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC'13)*, San Jose, CA, USA, June 2013, pp. 219–230.

[20] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[21] J. Shlens, "A tutorial on principal component analysis: derivation, discussion and singular value decomposition," pp. 1–16, March 2003.

[22] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemyer, "Workload characterization of a leadership class storage cluster," in *PDSW'10*, 2010.

[23] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *ISPASS 2007*, San Jose, CA, USA, April 2007, pp. 200–209.

[24] COSBench. https://github.com/intel-cloud/cosbench

[25] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani *et al.*, "Least angle regression," *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[26] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao, "Application performance modeling in a virtualized environment," in *HPCA)*. IEEE, 2010, pp. 1–10.

[27] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger, "Modeling the relative fitness of storage," in *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS 2007)*, San Diego, CA, USA, June 2007, pp. 37–48.

[28] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, and D. a. Patterson, "Automatic Exploration of Datacenter Performance Regimes," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds (ACDC'09)*, Barcelona, Spain, June 2009, pp. 1–6.

[29] Y. Chen, S. Iyer, X. Liu, D. Milojicic, and A. Sahai, "SLA decomposition: Translating service level objectives to system level thresholds," in *Fourth International Conference on Autonomic Computing, ICAC'07*, Jacksonville, FL, USA, June 2007.