A Self-Learning Scheduling in Cloud Software Defined Block Storage

Babak Ravandi
Computer and Information Technology
Purdue University
Email: bravandi@purdue.edu

Ioannis Papapanagiotou
Platform Engineering
Netflix Inc.
Email: ipapapa@ncsu.edu

***Abstract*—Software Defined Storage (SDS) separates the control layer from the data layer allowing the automation of data management and deployment of Commercial Off-The-Shelf (COTS) storage media rather than expensive traditional hardware-based solutions. Cloud block storage services lack an SDS framework that allows customization of block storage, policy enforcement, automate provisioning, and storage management. SDS decreases the human intervention and improves the resource utilization. Moreover, SDS allows cloud tenants to define customized functionalities based on their needs with guaranteed performance and high availability that meets Service Level Agreements (SLAs). However, maintaining SLAs requirements in cloud block storage is challenging due to the storage cluster features, the workload interference, the workload characteristics and other indirect related latent variables. To address the mentioned issues, cloud providers often over-provision the storage resources.**

Moving towards SDS, we initiate a framework for cloud block storage as an active storage system. Our framework provides customization of block storage services and optimized scheduling decisions based on the workload characteristics and performance of the underlying data layer leveraging a self-learning scheduler. The proposed scheduler treats the storage backend nodes as a black box and requires zero knowledge of their internal states. We showcase a practical application of the proposed scheduler in our private OpenStack deployment.

I. Introduction

Software Defined Storage (SDS) reduces the complexity of storage management in the cloud. SDS decouples the underlying storage hardware from the software that manages it [1]. The abstraction of storage from hardware prevents dependencies on a specific hardware or software. Developing an SDS framework for block storage systems received less attention within the research community compared to other types of storage systems.

As our first step towards developing an SDS for block storage, we propose and implement BSDS (Block Software Defined Storage), a novel scheduler to provide Quality of Service (QoS) for block storage systems without the need to have any knowledge of the underlying hardware. BSDS is an attempt to develop a black box self-learning scheduler that decouples the control from the storage layer. In our previous work, we developed a block storage simulator to assess the efficiency of using machine learning in making scheduling decisions [2]. Our workload-aware scheduler was able to maintain the QoS up to 98% of the simulation duration.

The fact that cloud resources are shared between multiple consumers makes it difficult for cloud vendors to maintain Service Level Agreements (SLAs) and its requirements Service Level Objectives (SLOs). Currently, from 57 drivers contributed to the OpenStack Cinder service only 10 drivers support QoS and they are designed for specialized hardware [3]. Such as the Dell EMC ScaleIO that integrates the EMC storage technologies with scalable multi-tenant cloud infrastructure [4]. ScaleIO contributed a driver for OpenStack with QoS support that operates on RHEL/CentOS, SLES and Ubuntu operation systems [5]. Our machine learning approach allows BSDS to learn the behavior of each backend independent of the underlying physical hardware. Therefore, BSDS enables QoS for commodity hardware as well. Also, BSDS can get integrated with existing storage solutions with built-in QoS support such as the NetApp SolidFire [6].

This work is the extension of our previous research [2] regarding the aforementioned scheduling issues of cloud block storage when developing an SDS framework for block storage. Such an approach enables cloud storage providers to provide guaranteed SLAs to customers and reduce the number of backend storage nodes. Another advantage of the proposed solution is the automatic migration of the risk factors. For example, a network traffic fluctuation could adversely increase the rate of SLA violations. The following recaps the main contributions of this work:

- Propose an SDS framework for block storage systems and investigate the utilization of our self-learning black box scheduler in a cloud deployment. This removes the requirement that the scheduler must make independent decisions per unit time.
- Treat backend storage systems as a black box that does not provide the internal states of backends to the scheduler. Thereby, the scheduler can make SLA-aware decisions independent of the vendor specific information.
- Provide the ability to control the rate of SLA violations as well as fairness in resource provisioning, i.e., optimize the QoS while maximizing the resource allocation.

Our contributions above are supported with experimental results. The rest of the paper is organized as follows: Section II provides background and related work, Section III briefly discusses the scheduling algorithms in the current OpenStack

block storage system, Cinder, along with its weaknesses. Section IV describes the BSDS architecture and presents its learning components and the technical implementation challenges. Section V introduces our experiments design and results to evaluate BSDS performance. Lastly, the section VI concludes our research.

II. Background and Related Work

In distributed systems, the data layer is usually recognized as the main bottleneck [7]. Several factors have a direct relationship in the performance of a block storage system. We categorize them into two categories. One which is related to the volume, (a) the create-volume request workload, and (b) the storage I/O workload once the block has been allocated. The second category is related to the performance of the block storage system such the overall status and performance of the cluster. The create-volume workload depends on the pattern of detach/attach requests issued by virtual machines to access the virtual volumes. The storage I/O workload depends on the applications that are using the attached volumes to perform data operations. The applications I/O usage have different patterns such as sequential, random read/write, diurnal I/O, etc. The status of the cluster is affected by the type of storage operations such as the garbage collection activity in the SSD drives, the compactions, the concurrent maintenance operations, the disks flapping, the network traffic fluctuations and the workload interference by collocating workloads. These factors affect the QoS and increase the complexity of designing a workload-aware scheduler for block storage systems. Such issues are not present in VM or container scheduler as some of the computations and resources can be strictly isolated.

Due to the aforementioned reasons, there has been a tremendous effort in improving performance of the storage systems. Black box models offer most promising solutions since they require minimum interactions with the underlying storage devices and they are designed to predict and adopt the workloads dynamics. Yin et al. explored the effectiveness of black box performance models in automating storage management using regression trees for modeling. They used a real storage system running on a RedHat Server Linux Kernel to collect data and perform experiments [8]. Zhang and Bhargava automated the parameters tuning of disk schedulers by introducing self-learning schemes and performed experiments on the Linux kernel [9]. Skourtis et al. acknowledged the issues with shared storage such as dealing with multiple type of workloads and performance targets. They proposed QBox (Queue Box), a black box controller that provides isolation between clients by forwarding the stream requests to the disks using the Kernel Asynchronous I/O (AIO) on Linux [10]. However, QBox approach requires to place a controller between the storage disks and the clients that might be impractical for the current datacenters to adopt. To compare, our proposed approach is designed for the cloud environment and utilizes the storage design of a cloud infrastructure that organically provides levels of isolation. Also, the installation process could be integrated with the currently popular configuration management tools such as the Puppet and Chef that increases its adaptability [11, 12].

In response to the aforementioned challenges, we worked on the scheduling issues in the cloud storage domain. We introduced an SLA-aware scheduler that can provide I/O performance management for block storage through scheduling policies leveraging a Multi-Vector Bin Packing (MVBP) algorithm. The proposed solution achieved more than 20% improvement in reducing the rate of SLA violations [13]. However, the scheduler assumed to have knowledge of the internal states of the underlying hardware and the cluster architecture. Hence, the solution was not adjustable to many types of block storage systems. We then investigated scheduling decisions that account multiple SLOs to scale up to concurrent arrival requests [14]. Subsequently, we designed Serifos [15] a workload consolidation and load balancing for SSD based storage systems and performed an empirical evaluation of IaaS infrastructures [16]. We finally proposed a self-learning scheduler that treats the backend nodes as a black box. This was a major step towards assuming a black box approach. We used simulations and block I/O traces to assess the performance of the machine learned scheduling [9].

III. OpenStack Cinder

Storage in OpenStack cloud environment is broken into two categories Ephemeral and Persistent. Ephemeral storage example is the disks associated with virtual machines, and they could get terminated if a virtual machine is deleted. Conversely, the persistent storage lives regardless of the virtual machines state because a persistent storage is handled outside of the virtual machines scope. Block storage is a type of persistent storage, and it provides virtual volumes that can get attached to the virtual machines while the volumes are physically located on storage backends. Various storage platforms such as Ceph [17] and NetApp [18] can be connected to Cinder as a storage backend in addition to the local Linux storage.

Cinder is the block storage service of OpenStack with three main components: Cinder API, Cinder Scheduler, and Cinder Volume [19]. Cinder API is a communication engine allows users and services access Cinder services through RESTful API. The Cinder scheduler decides on which backend a create-volume request should be allocated. Lastly, the Cinder volume component is a set of drivers designed to provide virtual volumes on a physical device, for example, the Logical Volume Management (LVM). A company providing specialized block storage hardware needs to develop a driver compatible with the Cinder volume component. Virtual volumes can dynamically get attached or detached to virtual machines, and they can be used as a second storage or boot device. Cinder-scheduler workflow contains two phases that are presented in Fig. 1:

i. *Filtering phase*: Upon receiving a create-volume request, the users' preferred filter will compare the request parameters (e.g. capacity, read IOPS and write IOPS) with the state of each backend to eliminate the backends that do not meet the request SLOs. The goal is creating a set of

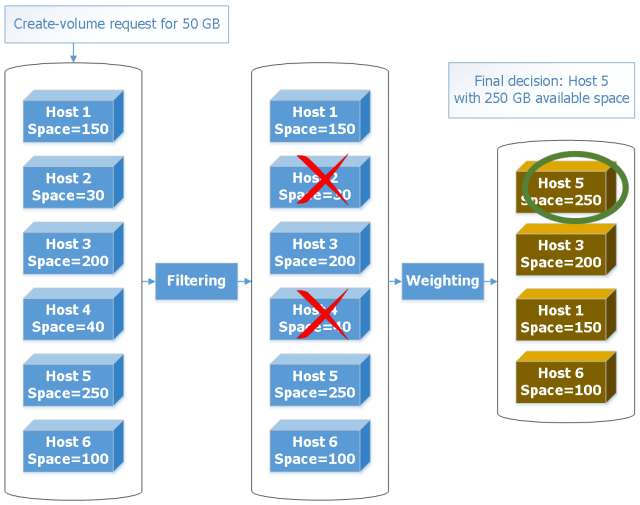

Fig. 1. Workforce of OpenStack Cinder consisting the filtering and weighting phases.

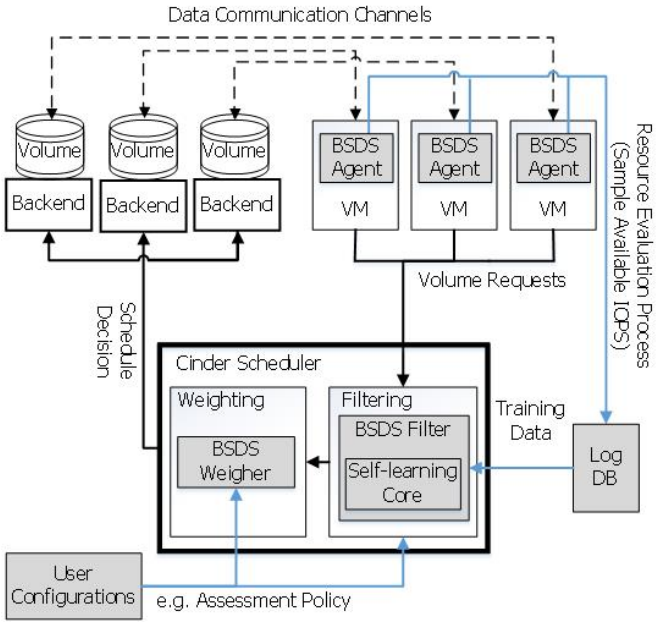

Fig. 2. BSDS high level architecture overview. The gray boxes are BSDS modules and white boxes are Cinder components. Cinder scheduling workflow starts from a virtual machine issuing a create-volume requests.

backends as candidates for the final scheduling decision. For example, in the Fig. 1 if a create-volume request asks for 50 GB capacity, then the filtering module will eliminate the Hosts 2 and 4 because they do not have enough space.

ii. *Weighting phase*: The goal of weighting phase is selecting the best backend within the candidate set received from the Filtering phase. The users' preferred weigher ranks each backend based on their available resources. Then, a backend with the most available resources will be selected as the final scheduling decision. For example, in Fig. 1 the Host 5 is chosen because it has the most available capacity within the candidates set.

The Cinder default filter and weigher only consider the available capacity of backend nodes to make a scheduling decision. BSDS is designed to recognize the available read and write IOPS of the nodes to provide QoS.

IV. Architecture

This section presents the BSDS architecture, the machine learning approach and the technical implementation challenges.

A. Design Objectives

Figure 2 shows the architecture of BSDS with the gray boxes depicting the BSDS modules and white boxes showing Cinder modules. The blue arrows indicate communications between BSDS modules and the black arrows indicate the Cinder scheduling workflow that starts with a virtual machine issuing a create-volume request. The dotted arrows are I/O pipes that connect the attached virtual volumes to the virtual machines through the Cinder volume service.

BSDS consists of two main components. First, the BSDS weigher and filter modules implemented based on OpenStack's Cinder interfaces. Second, the BSDS Agent that is installed on the cloud VMs and is responsible for measuring the available read/write IOPS of every attached volume to a VM i.e. executing the *resource evaluation* process. The measured available read and write IOPS will be stored in the *Log DB* module to generate the training dataset.

The process of scheduling starts upon receiving a create volume request. First phase is Cinder filtering that activates the *BSDS filter* module. Upon activation, the *Self-learning Core* create and cache classification models for each backend node using the training data stored in the *Log DB*. Then, using the models the available expected level of SLO-violation for each backend is predicted. Next, the backends that do not satisfy the requested SLOs will be filtered. Algorithm 1 demonstrates the process. The algorithm creates and maintains a separate classifier for each backend because a backend could have a unique behavior depending on its configurations and workload. Hence, it is not practical to predict all the backends behaviors with a single learning model.

Second phase is Cinder Weighting that activates BSDS weigher with the task of selecting a backend with least probability of causing read/write-SLO-IOPS violations among the eligible backends. Variable $pref.IsReadPriority$ determines SLO-read-IOPS has more priority (explained with details in the next section). Algorithm 2 demonstrates the BSDS Weigher process.

B. Self-Learning Approach

Self-learning approach allows treating the backends as a black box. In other words, BSDS ensures the QoS without requiring to have any knowledge of the underlying hardware state. Table I presents the classification features. The *clock* feature reflects the time that a *performance evaluation* is

Algorithm 1 BSDS Filter

procedure FILTERING($VolRequest$)
  $Classifiers \leftarrow Run\_BuildClassifiers(Backends)$
  $read\_can \leftarrow \{\varnothing\}$
  $write\_can \leftarrow \{\varnothing\}$
  for each $classifire \in Classifiers$ **do**
    $pred \leftarrow classifire.Predict(VolRequest)$
    if $pred$ **satisfy** $VolRequest.readSLO$ **then**
      $read\_can \leftarrow read\_can \cup prediction.Node$
    end if
    if $pred$ **satisfy** $VolRequest.writeSLO$ **then**
      $write\_can \leftarrow write\_can \cup prediction.Node$
    end if
  end for
  return $read\_can, write\_can$
end procedure

Algorithm 2 BSDS Weigher

//$Pref$ denotes the user preferences
procedure WEIGHTING($read\_can, write\_can$)
  $final\_can \leftarrow read\_can \cap write\_can$
  $final\_decision \leftarrow none$
  if $final\_can$ **is equal** $\varnothing$ **then**
    if $pref.IsReadPriority$ **=** $True$ **then**
      $final\_decision \leftarrow Best(read\_can)$
    else
      $final\_decision \leftarrow Best(write\_can)$
    end if
  else
    $final\_decision \leftarrow Best(final\_can)$
  end if
  if $final\_decision$ **is** $none$ **then**
    return '$reject$'
  end if
  return $final\_decision$
end procedure

executed to measure the available read/write IOPS of a virtual volume. Considering the time as a feature in our learning model allows have the periodic performance fluctuations in account while making scheduling decisions (e.g. rush hours and regular maintenance). The remaining variables are collected by aggregating the *performance evaluation* results in a given clock.

The *TotReqReadIOPS* and *TotReqWriteIOPS* features account for the total number of requested read/write SLOs from a backend in a given clock. Similarly, the *num* variable is the total number of live volumes in a given clock. Lastly, the *vioGroup* is the model response variable and represents the level of SLO-IOPS violation on a clock. The *vioGroup* feature is the discretized transformation of the number of the *performance evaluation* records that identified an SLO violation in a given clock. We decided to discretize the number of SLO violations because the nature of proposed scheduling issue falls into supervised learning since the learning features are known. The correctness of discretization in supervised learning is proven by Dougherty et al. [20].

TABLE I
LEARNING FEATURES

Field	Description
clock	the *Performance Evaluation* Record timestamp
TotReqReadIOPS	Total requested read-IOPS of live volumes
TotReqWriteIOPS	Total requested write-IOPS of live volumes
num	Number of live volumes
vioGroup	SLO-IOPS violation groups (defined in Table II)

C. Candidate Learning Algorithms

This research evaluates the C4.5 Decision Tree and Bayesian Network learning algorithms. In our previous work we compared the accuracy of the C4.5 Decision Tree, Support Vector Machine (SVM), Naïve Bayes and Bayesian Network K-fold validation on the test dataset [2]. The results showed linear boundary algorithms (SVM and Naïve Bayes) perform poorly, but algorithms with highly non-linear decision boundary perform better (C4.5 and Bayesian Network). We omitted the logistic regression and K-nearest neighbor (KNN) algorithms because the training data does not fit well in them, and KNN is not lightweight. For those reasons, we do not evaluate the SVM and Naïve Bayes. Below, we present a brief discussion on the candidate machine learning algorithms.

- *C4.5 Decision Tree*: C4.5 builds a classifier tree based on the ID3 algorithms. Each leaf of the tree determines a class of prediction and nodes split the feature domains. Decision trees are not probabilistic based learning models. However, there are probabilistic models that incorporate decision trees. We use the Probability Estimation Tree to have the final predictions ranked as probabilities of classes [21].
- *Bayesian Network (BN)*: BN is a type of probabilistic graphical model defined on a directed acyclic graph (DAG). BN applies the Bayes' theorem on the conditional probability distributions to create connections between the learning features and propagate changes in the values of those features in the network [22].

D. BSDS Configurations

BSDS configurations allow a user to define their proffered level of QoS. Table II presents the main settings. The *ViolationGroups* represents intervals to discretize the number of violations happened on a given clock that defines a domain for the response variable, *vioGroup*. *IsReadPriority* determines preventing read-SLO-IOPS has more priority over write-SLO-IOPS. For example, in a case that none of the backends can satisfy the write-SLO-IOPS and the *IsReadPriority* is enabled, BSDS will accept the request if a backend could satisfy the requested read-SLO-IOPS. The *MLAlgorithm* defines which classification algorithm BSDS may use to create the learning models. Lastly, the *AssessmentPolicy* sets in what level the

TABLE II
BSDS CONFIGURATIONS

Parameter	Description	
ViolationGroups	Defines intervals to discretize the number of SLO. i.e. G1: 0 violations; low violation rate.	
IsReadPriority	if true, satisfying read-SLO is priority	
MLAlgorithm	Learning algorithm	
AssessmentPolicy	#1 EfficiencyFirst	Accept if with 80% chance, the allocation will not cause any SLO violations.
	#2 QoSFirst	Accept if with 90% chance, the allocation will not cause any SLO violations.
	#3 StrictQoS	Accept if with 99% chance, the allocation will not cause any SLO violations.

QoS needs to be maintained. The more strict an assessment policy is, the less SLO-IOPS violation is guaranteed. However, the requests rejection rate will increase to preserve IOPS resources.

E. Technical Challenges

We used the FIO software (Flexible I/O Tester synthetic benchmark) to measure the available read/write IOPS of virtual volumes [23]. However, running concurrent FIO tests on multiple virtual volumes that are attached to a VM can cause hash inconsistency between the FIO jobs. Therefore an isolated environment is required to concurrently run the FIO jobs on multiple virtual volumes to be able to generate synthetic I/O workloads and measure the available IOPS of those volumes. To address the issue, we used a Docker-containerized environment to force isolation within multiple concurrent FIO executions. We used the ClusterHQ FIO-Tool that is a Dockerized FIO Container tool to run FIO jobs [24]. Each virtual volume path was mounted to a Docker running the ClusterHQ FIO-Tool container to execute the required set of FIO jobs.

V. EXPERIMENTS

In this section, we analyze the performance of BSDS implemented in our private OpenStack cloud.

A. Experiment Design

We used 6 VMWare VSphere ESXi 6.0 Hypervisors to implement our private OpenStack cloud using the OpenStack Mitaka release. The deployment had 9 storage backend nodes running on the Ubuntu Server 16.04, and each node had 4 GB RAM memory and 2 cores. Two nodes had Western Digital Red 1TB NAS 5400 RPM SATA6 Gb/s 64MB Cache hard drives. Plus other three nodes with Western Digital Caviar Blue 250GB SATA3 Gb/s 7200 RPM 64MB Cache hard drives and the five remaining nodes had Seagate Barracuda 1TB SATA6 Gb/s 7200 RPM hard drives. On average, the hard drives achieved up to 286 write IOPS and 1772 read IOPS throughput measured by the aforementioned *resource evaluation* process.

During the experiment, twelve Ubuntu Server 16.04 virtual machines were requesting maximum 3 Cinder volumes every 200 seconds up to 400 successful create-volume requests. The volume requests were repeated sequentially after a volume is detached. The requests Capacity, read-SLO-IOPS, and write-SLO-IOPS were randomly chosen using the uniform random distribution within the following sets of values [5GB, 10GB, 15GB], [600, 700, 800] and [2050, 350, 400] respectively. Each volume lifetime was randomly chosen between [160, 180, 200] seconds.

We used the FIO software to generate multiple types of storage workload depicted in the Table IV. FIO is used for two purposes. First, to generate a synthetic storage workload (i.e. backup a video streaming server). Second, to execute the *resource evaluation* process with the goal of sampling the available read and write IOPS of a virtual volume. Therefore, collect the training data and model the behavior of each backend.

TABLE III
CONFIGURATION PARAMETERS FOR THE EXPERIMENT

Parameter	Value	Description
ViolationGroups	V1: (0); V2: (1 or 2) V3: (3 or 4) V4: (5+)	violation classes for the experiments
IsReadPriority	if true, satisfying read-SLO is priority	
MLAlgorithm	C4.5 Decision tree	
AssessmentPolicy	CinderDefault, EfficiencyFirst, QoSFirst, StrictQoS	

We conducted two experiments to evaluate BSDS. First experiment used a sequential read workload to benchmark a media streaming scenario. The second experiment used a random read/write workload to create a backup server scenario. The workloads are synthetic and generated using FIO software. Table IV represents the workload generation details.

B. Tune Primary Parameters

Table III shows the primary configurations used within all of the experiments. For example, assume on a certain $clock$ the executions of *Resource Evaluation* process identifies 3 SLA violations on a volumes. Then, the scheduler discretize the value into the *V3* category as the *VioGroup* for its respective *Resource Evaluation* sample.

C. Experiment Results

Each experiment conducted in two modes: training and decision. The goal of training mode is collecting the available read/write IOPS of each live volume to create a training dataset for each backend nodes. Since measuring the available IOPS on every second is not practical, we perform the *resource evaluation* process periodically based on the users' configurations.

The decision mode activates BSDS filter and weigher modules to enable QoS assurance for read/write IOPS. Then, classification models were created for each backend using the training dataset obtained from the training mode. Lastly, the

TABLE IV
FIO CONFIGURATION TYPES

FIO Config For	Restart Interval (seconds)	I/O Type	Read / Write	block size	size
Resource Evaluation (Sampling) Process	20	mix random read/write	70% / 30%	4k	40 mb
Experiment I: Sequential Read	4	Sequential reads	100% / 0%	4k	48 mb
Experiment II: Random Read/Write	4	mix random read/write	50% / 50%	4k	128 mb

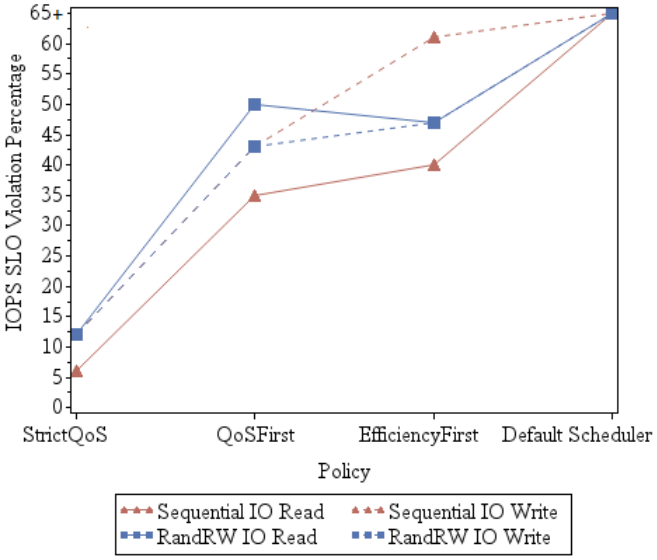

Fig. 3. SLO IOPS Violations Percentage for Read/Write using Decision Tree

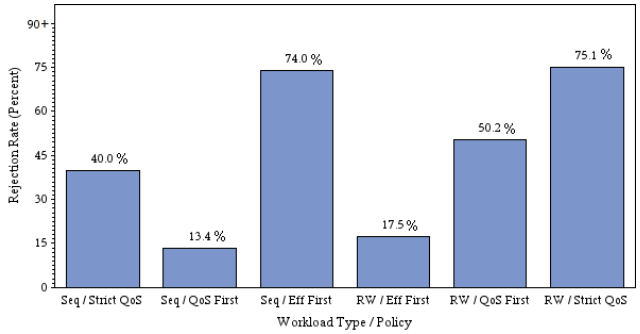

Fig. 4. SLO IOPS Violation Percentage for Each *Assessment Policy* using C4.5 Decision Tree

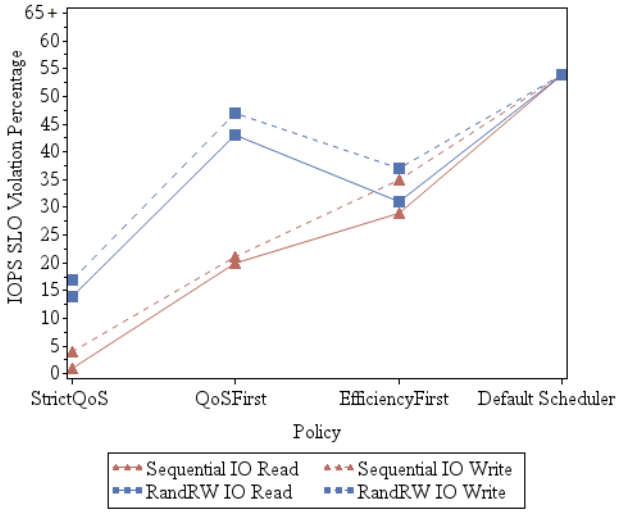

Fig. 5. SLO IOPS Violations Percentage for Read/Write using Bayesian Network

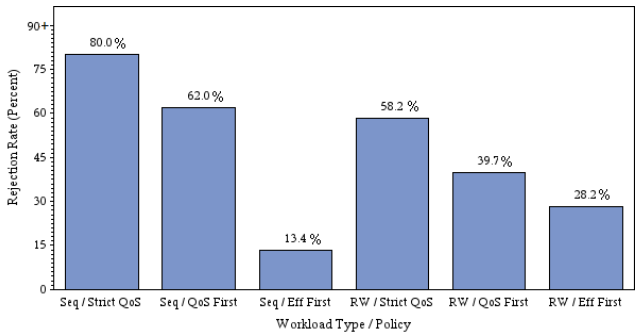

Fig. 6. Simulation results

prediction values were assessed based on the *assessment policies* to achieve the expected level of QoS. In our experiment we defined three policies represented in the Tables II and III.

Our experiment used the Bayesian Network and C4.5 Decision Tree machine learning algorithms to make SLA-aware scheduling decision [25]. Based on our previous work the non-linear learning algorithms accuracy is higher than linear algorithms [2]. Therefore, we decided to use those algorithms.

Figure 3 and 4 shwocase the percentage of SLO-IOPS violations and the rejection rate compared to the training mode using the C4.5 Decision Tree for the sequential read and random read/write workloads. The experiments are configured to have higher priority for ensuring the read-SLO-IOPS over write-SLO-IOPS. This is the reason for having more write-SLO-IOPS violations. The graphs indicated that using the *Strict QoS* policy BSDS can lower the rate of read-SLO-IOPS violation to 3% with the trade off as rejecting 74.55% of overall create-volume requests. In contrast, using the *QoS First* maintains 33% violation rate while rejecting 13.4% of the create volume requests.

Figures 5 and 6 present the performance of BSDS while using the Bayesian Network. Overall, the C4.5 Decision Tree and Bayesian Network were able to control the SLO-IOPS violation rate around the same level for both random-read/write and sequential-read workloads. On the other hand, for the sequential workload the Bayesian network performs better than C4.5 Decision Tree.

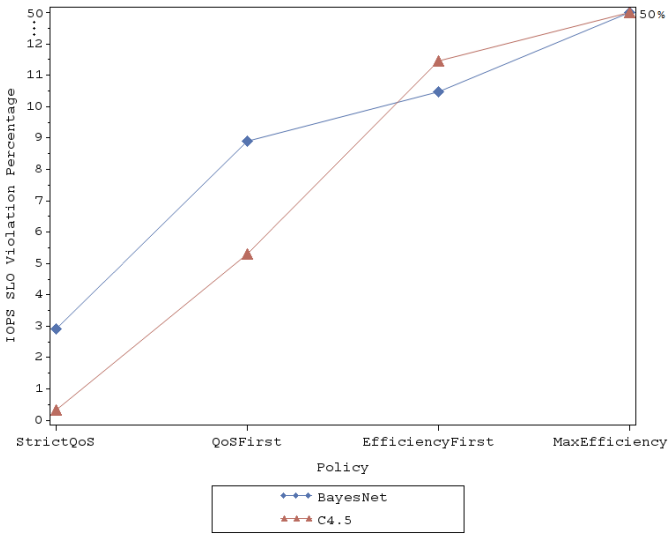

Fig. 7. Simulation Results Based on Our Previous Work [2]

D. Comparison with Simulation Results

Our previous work assessed the proposed black box self-learning approach using simulation [2]. In this section, we compare the results from simulation and BSDS implementation. We performed simulations with workloads induced by real-world block-level traces of an enterprise data center. The results showed that self-learning scheduling could mitigate the unexpected resource fluctuation and dynamically adapt to various workloads. Figure 7 presents the SLO-IOPS violations using the simulation. Both read and write I/O were simulated as a single variable in the simulation. Comparisons between the Figure 7 and Figures 3 and 5 (SLO-IOPS violations using Bayesian and Decision Tree algorithms respectively) indicate the following: (a) using the Bayesian network the experiment results shows the same pattern as the simulation for both read and write IO measurements (b) however, using the Decision Tree algorithm the experiment results shows more fluctuations compared to the simulation results. The similarity between the results of simulation and implementation shows that the black box approach is effective in designing schedulers for cloud storage systems.

VI. Conclusion

In this paper, we introduced the Block Software Defined Storage (BSDS) as a step towards designing Software Defined Storage (SDS) systems for cloud block storage. BSDS is based on a self-learning black box scheduler that provides Quality of Service (QoS) without requiring any knowledge of the underlying storage hardware, as an alternative for the expensive specialized equipment that currently is the only QoS based solution in block storage systems. However, BSDS provides QoS on deployment of Commercial Off-The-Shelf (COTS) storage media. We integrated BSDS with OpenStack Cinder to assess its performance. We defined three policies to control the expected level of QoS. The policies are *StrictQoS*, *QoSFirst* and *EfficiencyFirst* that guarantee the level of QoS from strict to efficient respectively. Strict level means having the minimum chance of having Service Level Objectives (SLO) violations occurrence. In contrast, the efficient level offers a balance between the IOPS resource reservations and the chance of having SLO violations occurrence. We performed experiments using Bayesian Network and Decision Tree classifiers and BSDS was able to control the percentage of read/write-SLO violations up to 3%, 21% and 32% using *StrictQoS*, *QoSFirst* and *EfficiencyFirst* respectively.

In the future, we plan to work on the dynamic migration of volumes and implementing feedback learning to address sudden changes in the rate of SLO violations.

References

[1] J. Samp, P. Garca-Lpez, and M. Snchez-Artigas, "Vertigo: Programmable micro-controllers for software-defined object storage," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, June 2016, pp. 180–187.

[2] B. Ravandi, I. Papapanagiotou, and B. Yang, "A black-box self-learning scheduler for cloud block storage systems," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, June 2016, pp. 820–825.

[3] "Cinder support matrix." [Online]. Available: https://wiki.openstack.org/wiki/CinderSupportMatrix.

[4] "Software defined block storage - scaleio." [Online]. Available: https://www.emc.com/en-us/storage/scaleio/index.htm.

[5] "Emc scaleio block storage driver configuration." [Online]. Available: https://docs.openstack.org/draft/config-reference/block-storage/drivers/emc-scaleio-driver.html.

[6] "Solidfire all-flash array." [Online]. Available: http://www.netapp.com/us/products/storage-systems/solidfire/index.aspx.

[7] J. Shafer, "I/o virtualization bottlenecks in cloud computing today," in *Proceedings of the 2nd conference on I/O virtualization*. USENIX Association, 2010, pp. 5–5.

[8] L. Yin, S. Uttamchandani, and R. Katz, "An empirical exploration of black-box performance models for storage systems," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on*. IEEE, 2006, pp. 433–440.

[9] Y. Zhang and B. Bhargava, "Self-learning disk scheduling," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 1, pp. 50–65, 2009.

[10] D. Skourtis, S. Kato, and S. Brandt, "Qbox: Guaranteeing i/o performance on black box storage systems," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*. ACM, 2012, pp. 73–84.

[11] "Puppet. cloud management solution." [Online]. Available: https://puppet.com/.

[12] "Chef. continuous automation for continuous enterprise," [Online]. Available: https://puppet.com/.

[13] Z. Yao, I. Papapanagiotou, and R. D. Callaway, "SLA-aware resource scheduling for cloud storage," in *IEEE International Conference on Cloud Networking (CloudNet)*. IEEE, 2014.

[14] ——, "Multi-dimensional scheduling in cloud storage systems," in *International Communications Conference (ICC)*. IEEE, 2015.

[15] Z. Yao, I. Papapanagiotou, and R. Griffith, "Serifos: Workload consolidation and load balancing for SSD based cloud storage systems," *arXiv preprint arXiv:1512.06432*, 2015.

[16] Z. Yao and I. Papapanagiotou, "A trace-driven evaluation of cloud computing schedulers for IaaS," in *International Communications Conference (ICC), 2017 IEEE*, May 2017.

[17] "Block device and openstack," [Online]. Available: http://docs.ceph.com/docs/master/rbd/rbd-openstack/.
[18] "Netapp data management and cloud storage solutions." [Online]. Available: http://www.netapp.com/us/index.aspx.
[19] "Openstack cinder," [Online]. Available: https://wiki.openstack.org/wiki/Cinder.
[20] J. Dougherty, R. Kohavi, M. Sahami *et al.*, "Supervised and unsupervised discretization of continuous features," in *Machine learning: proceedings of the twelfth international conference*, vol. 12, 1995, pp. 194–202.
[21] F. Provost and P. Domingos, "Tree induction for probability-based ranking," *Machine Learning*, vol. 52, no. 3, pp. 199–215, 2003.
[22] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," in *AAAI*, 1982, pp. 133–136.
[23] "Fio - flexible i/o tester synthetic benchmark," [Online]. Available: https://github.com/axboe/fio/blob/master/HOWTO.
[24] "Clusterhq docker hub," [Online]. Available: https://hub.docker.com/r/clusterhq/fio-tool/.
[25] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.