A Black-Box Self-Learning Scheduler for Cloud Block Storage Systems

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

Ioannis Papapanagiotou
Cloud Database Engineering
Netflix
Email: ipapapa@ncsu.edu

Baijian Yang
Computer and Information Technology
Purdue University
Email: byang@purdue.edu

***Abstract*—A major requirement of cloud block storage services is guaranteed performance and high availability. However, offering guaranteed Service Level Agreements (SLAs) in cloud block storage services is often not straightforward. Cloud block storage performance may be affected by physical disk background operations, like garbage collection, storage cluster features, workload interference and the chracteristcs of the workload itself. On the other hand, the underlying physical storage drives do not expose the internal states to higher level block storage service offerings. Therefore, SLAs can only be satisfied by over-provisioning the storage resources. To address this issue, we propose a self-learning scheduler that can dynamically adapt based on the workload, and efficiently provide a scheduling decision with zero knowledge of the underlying hardware. We study two candidate algorithms based on Feedback learning and Two-Phase learning. We used workloads that were deducted from real-world block-level traces of an enterprise data center, and conducted extensive simulations. Our results indicate that the self-learning scheduling approach can reduce the SLA violations by mitigating the unexpected resource fluctuation, and the scheduler can also adapt dynamically to various workloads.**

I. Introduction

The cloud computing model decouples physical resources from the logical services. It provides dynamic provisioning, high flexibility, scalability and significant cost reduction [1]. The cloud vendor and consumer relationship is administered by Service Level Agreements (SLAs), with each specific requirement defined in an SLA called Service Level Objective (SLO). In the case of cloud block storage, the performance is measured by parameters such as Input/Output Operations Per Second (IOPS) and the latency distributions (identified by the corresponding percentiles). However, the challenge is that the dynamic nature of the cloud and the stochastic workloads make it difficult to provide a meaningful SLA in terms of the availability and the cost of services.

Significant amount of research has been done in the area of cloud computing resource provisioning (i.e. virtual machine allocation), yet cloud storage has received little attention [2]. In fact, in distributed systems, the storage performance tends to be the main bottleneck. The performance of the system can be directly affected by the patterns of the workload and concurrent workloads. In fact, workloads may follow different patterns, like diurnal, continuous, etc. I/O throughput and latency fluctuations may be caused by internal operations like garbage collection, compactions, flapping disks, or even maintenance changes in the underlying hardware, etc. Those operations may affect the QoS. At the same time, a scheduler must be configurable.

In our previous work [2], we designed an SLA-aware scheduler and corresponding scheduling policies that can enable I/O performance management for block storage systems. We achieved more than 20% improvement in reducing the SLA violations rate by leveraging a Multi-Vector Bin Packing (MVBP) algorithm. The scheduler was assumed to have knowledge of the underlying storage cluster architecture and hence, was not adjustable or scalable to different types of block storage clusters. Thereupon, we investigated the scheduling challenges [3] that are subject to multiple SLOs constraints and concurrent arrival of requests [4]. In this work, we assume that the scheduler has no knowledge of the internal structures of the backend systems (black-box approach); hence, it can be pluggable in different data center deployments. To investigate the performance and accuracy of the scheduling decisions, we used a number of workloads deducted from block-level traces collected from an enterprise data center. In the data center, Windows-based servers were running typical enterprise storage services such as distributed file sharing and web server [5].

Our design is based on a black-box approach, i.e. the performance of the scheduler is not affected by the internal state or operation of the storage array. In fact swapping the underlying hardware would only require re-training the algorithm. We use Feedback learning to identify and absorb unexpected changes in the workload patterns. At the same time, we propose a scheduling design with configurable components to dictate the behaviors of the learning algorithms. Hence, we enable cloud service providers to provide SLA guarantees, reduce the number of active storage nodes, and mitigate risk factors, such as traffic fluctuations that could adversely affect the SLA violation rate. We evaluate the performance of the proposed scheduler through extensive simulation based on the OpenStack block storage service, Cinder [6].

II. OpenStack Cinder Scheduling

OpenStack provides a large pool of services such as compute (named Nova), networking (Neutron), object storage (Swift), block storage (Cinder), identity (Keystone), etc. through a web-based dashboard, command-line, and RESTful APIs. It is the most widely used support platform for building and managing an Infrastructure as a Service (IaaS) [7]. Hence, we architected our scheduling proposal based on it. However, the ideas can be embraced by other platforms as well.

Cinder handles creation, attachment and detachment of volumes to virtual machines [6]. In addition to local Linux storage, various storage platforms such as Ceph [8] can be connected to Cinder as backend. Access to Cinder volumes is only associated with guest virtual machines. Cinder consists of 3 components: (a) Cinder-APIs: provides RESTful APIs to access cinder functionalities; (b) Cinder-volumes: volumes are dynamically attached to VM and can be used as a data storage device or boot device; (c) Cinder-scheduler: The scheduler decides which backend storage unit is selected to provision the requests, based on the state of the storage system and the request. Cinder-scheduler workflow contains two phases:

i. *Filtering phase*: upon receiving a create-volume request, a filter will compare the request parameters with the state of each backend and eliminate backends that do not meet the request properties. The goal is creating a candidate list for final scheduling decision.
ii. *Weighting phase*: The scheduler ranks each backend in the candidates list (received from the filtering phase) based on their available resources. Then, a backend with most available resources will be selected as the final scheduling decision.

Scheduling decisions can significantly impact the performance of storage services as well as the whole services provided in the cloud. This is because storage allocation will affect the performance of both new volumes and existing volumes. In addition, the internal operations of modern storage clusters are often not exposed to the upper layers. This means an optimized decision on storage allocation is not feasible by using existing approaches. More intriguingly, even if state of some of the operations, such as the garbage collector, are provided to the higher layer, Cinder still cannot use the information directly because there is another layer of abstraction sitting between the scheduler and the backend storage.

III. Architecture of the Block Storage Self-Learning Scheduler

In this section, we describe the challenges and architecture of the proposed black-box scheduling scheme. The system leverages machine learning such that we can make an intelligent decision on which backend storage array to allocate the volumes.

A. Performance Issues of Block Storage Schedulers

The QoS of a block storage node can be represented as a value $Q_{node}(q)$, where q denotes the expected level of QoS in the set $[1, 2, 3, 4]$. The levels of QoS are defined in Table I. In Equation 1, V_{node} denotes the set that contains allocated volumes, $v_{R_{vio}}$ denotes the number of SLO-IOPS violations that happened on volume v, and v_R denotes the number of times that the available IOPS of v is sampled. The proposed scheduler measures the available IOPS of each virtual volume within a configurable interval to approximately identify the rate of SLO-IOPS violations and each backend available IOPS. We name this process *Resource Evaluation*, which is represented by R in the equation. Hence, the QoS of a block storage backend node can be represented as:

TABLE I
Assessment Policies/Desired Level of QoS

Policy	Description
#1 DefaultCinder	Accept the request by capacity only. This is the default behavior of OpenStack Cinder.
#2 EfficiencyFirst	Accept the request if the allocation will not cause any SLA violations with at least 80% of chances.
#3 QoSFirst	Accept the request if the allocation will not cause any SLA violations with at least 90% of chances.
#4 StrictQoS	Accept the request if the allocation will not cause any SLA violations with at least 99% of chances.

$$Q_{node}(q) = \frac{\sum\limits_{v \in V_{node}} \frac{v_{R_{vio}}}{v_R}}{|V_{node}|} \tag{1}$$

B. Design Objectives

The architecture of the scheduler is shown in Fig. 1. The gray boxes represent the new modules that are added to the OpenStack Cinder. The *Self-learning core* uses the performance data collected from the backend nodes to train appropriate classifiers (Section III-D) for each backend node. In the event of receiving a create-volume request, the *filtering* module eliminates the backends that do not meet the capacity requirement, it then adds the remaining backends to a candidate set. Next, using the trained classifiers, the *filtering* module will predict the potential impact of the request on the QoS of each node in the candidate set.

TABLE II
Classification Features

Field	Description
clock	Record timestamp
TotReqIOPS	Total requested IOPS of live volumes
num	Number of live volumes
vioGroup	SLO-IOPS violation groups (defined in Table III)

The *weighting* module selects a backend node from the set of candidates, with the least probability of causing SLA-IOPS violations followed by the highest number of available capacity. Thereupon, based on the *Assessment Policy* specified by the user, the nodes that do not meet the SLO-IOPS requirements are removed from the candidate set.

Table II describes the four features selected for the proposed supervised learning. Feature *vioGroup* denotes the number of SLO-IOPS violations (identified by the *Resource Evaluation* process) as a categorical variable that is transformed via

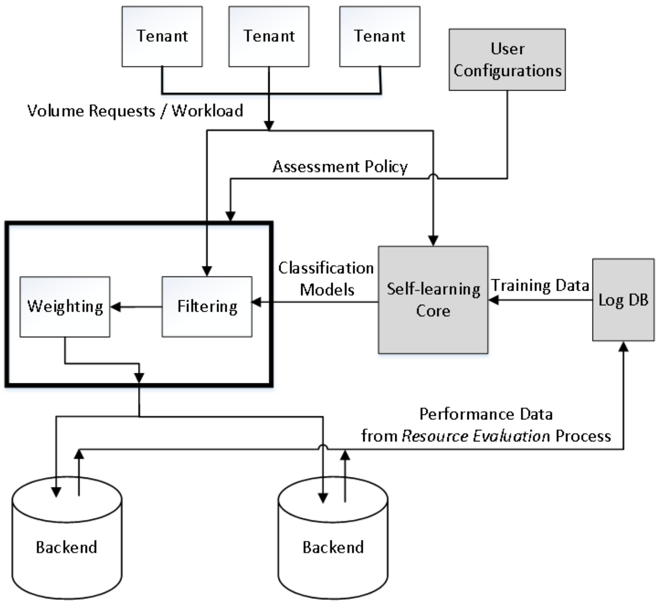


Fig. 1. Architecture of the self-learning scheduler

variable discretization heuristics. The domain of $vioGroup$ is presented in Table III. Variable discretization decreases the entropy of the learning models, and its correctness in classification learning is presented by Irani [9].

TABLE III
SELF-LEARNER SCHEDULER CONFIGURATION PARAMETERS

Parameter	Description
ResMClockGap	Threshold of IOPS viloation evaluations.
ViolationGroups	Defines intervals to discretize the number of SLO. *i.e.* G1: 0 to 5 violations; low violation rate.
MLAlgorithm	Machine learning algorithms selected.
AssessmentPolicy	Scheduling policy defined in Table I.

The learning objective is to model the pattern of SLO-IOPS violations for each backend. Therefore, feature selection is based on linking the behavior of recurring workloads to the performance of the underlying block storage system. Table III presents the self-learner configurations. Users can configure the level of QoS based on their needs.

C. Self-Learning Core Algorithms

We leverage supervised machine learning algorithms to schedule storage requests. There are two phases involved. Phase 1 is to build the classification models, and Phase 2 is to make scheduling decisions using the classifiers built in Phase 1. Users can also choose to rebuild the classifiers if the performance results of Phase 2 are not satisfactory. This is denoted as Feedback learning scheme in this paper. Otherwise if the classifiers remain unchanged in Phase 2, we refer to it as Two-Phase learning scheme.

1) Algorithm 1 - Training Phase: The training phase only considers capacity for scheduling the create-volume requests (same as Cinder default scheduler). We chose to do it so that our scheduler can be applied in the OpenStack environment. The goal of training phase is collecting the data required to build the initial classifiers for each backend (line 11) while treating them as a black-box. The classifiers are then used in the decision phase to make accurate scheduling decisions. The steps above are integrated with Cinder filtering and weighting modules respectively.

Algorithm 1 Training Phase

```
    //$Pref$ denotes the user preferences
1:  while training is not done do
2:      $clock \leftarrow clock + 1$ // Each $Clock$ is a second
3:      if $Request$ != $\emptyset$ //Checks for requests arrival then
            //$Best$ denotes the chosen backend by scheduler
4:          $Best \leftarrow CinderFilteringWeighting(Request)$
5:          $Allocate(Best, Request)$
6:      if $Pref.ResMClockGap > random.next()$ then
            //sample training data from the guest VMs
7:          $Run\_ResourceEvaluation()$
8:  end while
9:  $Run\_BuildClassifiers(Backends)$
```

2) Algorithm 2 - Decision Phase: After the initial training phase the behavior of each backend is modeled. Thereby, the proposed scheduler takes the requested IOPS into account. Upon receiving a request, the scheduler initiates a candidate set (line 4) and makes a decision in two steps. First, the filtering phase matches the backends that has enough capacity (line 6). Then, the scheduler predicts the request potential impact on each backend (line 7) and will only add the backends that could satisfy the users' preferred level of QoS (line 8) to the candidate set. The second step is weighting of the selected candidates and either allocate the request or reject it. The scheduler selects a backend with maximum available capacity and the least probability of increasing the SLO-IOPS violation rate (line 12). Note, the classifiers return a vector (*pred* in line 7) that contains predictions in form of probability percentages for each *vioGroup* (defined by user).

3) Feedback Learning Scheme: The goal of feedback learning is re-building the predictive models in case a backend does not have good performance. User can enable feedback learning by setting value of the $AllowFeedback$ variable to $true$ in the configurations. The $Check\_BadPerformance$ function (line 21 of algorithm 2) detects sudden changes in the level of QoS for each backend by comparing recent changes in their Q_{node} (Equation 1) within ranges of time. If the value of Q_{node} has a high variance within few consecutive ranges of time, then the scheduler will re-build the classifiers using the most recent sampled data collected through the *Resource Evaluation* process (line 19).

D. Potential Machine Learning Algorithms

We evaluated the C4.5 decision tree and Bayesian Network (BayesNet) learning algorithms that had accuracies of 90% and 92% respectively [10]. The accuracy is calculated using the K-fold cross validation (with $k = 10$) [11]. We compared

Algorithm 2 Decision/Feedback Phase

```
    //$Pref$ denotes the user preferences
1:  while true do
2:      $clock \leftarrow clock + 1$
3:      if $Request$ != $\emptyset$ then //Checks for requests arrival
4:          $CandidateBackends = \{\}$
    /*$Filtering$ phase*/
5:          for each $B$ in $Available\_Backends$ do
6:              if $B.AvlCapacity > Request.Capacity$ then
                    //Classify the request
7:                  $pred \leftarrow B.Classify(Request)$
                    //$Pref.AP$ denotes $AssessmentPolicy$
8:                  if $Pref.AP.Satisfy(pred)$ then
9:                      $CandidateBackends.Append(B)$
10:             end if
11:         end for
    /*$Weighting$ phase*/
12:         $bk \leftarrow$ from $CandidateBackends$ select
                backend $b$ such that:
                    1) $b$ has max available capacity
                    2) $Request$ has the least probability of
                       increasing SLA-IOPS violation rate on $b$
13:         if $bk$ != $\emptyset$ then
14:             $Allocate(bk, Request)$
15:         else
16:             $reject(Request)$
17:     end if
18:     if $Pref.ResMClockGap > random.next()$ then
            //sample training data from the guest VMs
19:         $Run\_ResourceEvaluation()$
20:     if $Pref.AllowFeedback == true$ then
21:         $BadPerformance = Check\_Performance()$
22:         if $BadPerformance == true$ then
23:             $Run\_BuildClassifiers(Backends)$
24: end while
```

the accuracy of C4.5 decision tree, Support Vector Machine (SVM), BayesNet and Naïve Bayes. However, we decided to eliminate Naïve Bayes and SVM with accuracy of 54% and 83% respectively. First, the linear boundary algorithms (SVM and Naïve Bayes) performed poorly, but the decision boundary algorithms that are highly non-linear performed better (C4.5 and BayesNet). Second, their accuracies are relatively lower than C4.5 and BayesNet. Also, We omitted the logistic regression and K-nearest neighbor (KNN) algorithms because the training data does not fit well in them. Moreover, KNN is not lightweight.

IV. Experiments

In this section, we analyze the performance of the proposed self-learning scheduler, and Cinder scheduler in a self-developed simulation environment of cloud storage systems.

A. Experiment Design

The experiment design was based on the storage deployment proposed by Rackspace [12] with 6 storage backend nodes. Each node contains 18 × 2TB 7.2K SAS hard drives configured with RAID 0+1. Each node also has 6 GB RAM memory and 1 core CPU. Each hard drive can achieve up to 190 IOPS throughput.

The workloads are generated using block level I/O traces and random distributions. Each create-volume request includes Arrival-time, Capacity, SLO-IOPS, and Deletion-time. The Arrival-time is induced from 50,000 sequential I/O write-requests collected from real world Windows enterprise storage servers [5]. Also, we normalized the Arrival-time from milliseconds to seconds to have a concurrent batch of requests. The requests Capacity and SLO-IOPS are randomly chosen between [100GB, 500GB, 1000GB] and [200, 350, 450] respectively using the Normal random distribution. The requests Deletion-time is generated based on the Poisson distribution with a mean value of 20 seconds. We chose the Poisson distribution for simplicity. Based on our experiments, different types of distributions would not affect the methodology of our experiments. Since machine learning is based on finding patterns in data, however, it is more challenging to discover patterns in the maximum entropy probability distributions (such as the Normal distribution). To illustrate, we ran 1000 simulations using workloads generated with Poisson and Normal distributions. Each experiment simulated 5,600 minutes of I/O operations. The results are presented in Table IV, they show that workloads generated using the Normal distribution increase the average rate of SLO-IOPS violations compared to the Poisson distribution. Thereby, the proposed scheduler can control the rate of SLA-IOPS violations for workloads with higher entropy distributions.

TABLE IV
Comparison of Random Distributions for Workload Generation

Random Distribution	Bayes Net Accuracy	C4.5 Accuracy	Average rate of SLO-IOPS Violations
Poisson	90%	92%	6.5%
Normal	83.3%	84.3%	12%

The initial 9,000 requests are used as the training workload to build 6 classifiers (one classifier for each backend). This will avoid overfitting and guarantee that for each backend, at least 800 *Resource Evaluation* records will be available for training. The remaining 41,000 requests were used as the validation workload to assess the performance of the proposed scheduler. Lastly, in order to simulate the resource fluctuation, the number of available IOPS of each backend was randomly (50% chance) added or subtracted between 100 and 600 on every 250 seconds of simulation. Lastly, each experiment simulated 5,600 minutes of operations.

B. *Tune Primary Parameters*

Table V presents the simulator configurations used for our experiments. For example, assume on a certain *clock* the *Resource Evaluation* event identifies 3 SLA violations within the volumes of a backend node. Then, the scheduler will assign the *V3* category as the *VioGroup* for the respective *Resource Evaluation* record.

TABLE V
CONFIGURATION PARAMETERS FOR THE EXPERIMENT

Parameter	Value	Description
ResMClockGap	0.5	50% chance of running *Resource Evaluation*
ViolationGroups	V1: (0); V2: (1 or 2) V3: (3 or 4) V4: (5+)	violation classes for the experiments
MLAlgorithm	C4.5 Decision tree and Bayesian network	
AssessmentPolicy	CinderDefault, EfficiencyFirst, QoSFirst, StrictQoS	

C. *Simulation Results*

We first simulated the training phase with the OpenStack default scheduler and the training workload to collect initial performance data and build the classification models. Next, we assessed the proposed scheduler performance based on the *Two-Phase* and *Feedback* learning schemes.

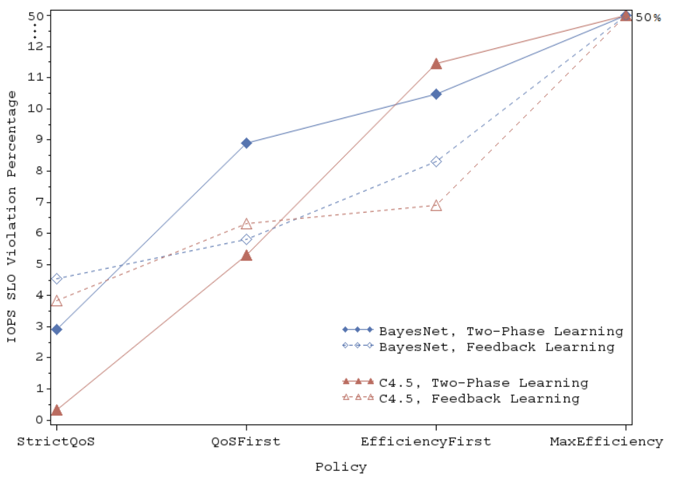
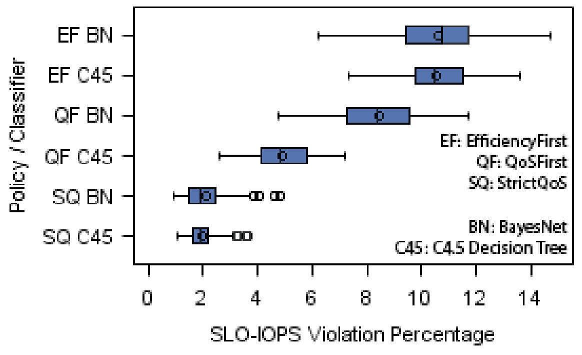
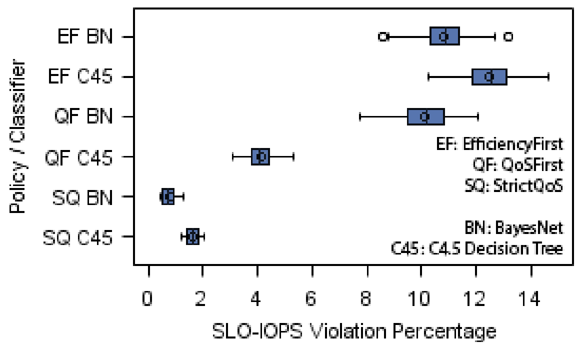


Fig. 2. Feedback learning scheme vs. Two-Phase learning scheme.

Feedback learning can mitigate unexpected workload changes by dynamically updating the classification models when sudden changes in SLA violation rates are detected (Section III-C2). It increases the overall accuracy of the learning models at the cost of adding computation overhead to the storage scheduler. The comparison of Two-Phase and Feedback learning schemes are presented in Figure 2. Results showed that the Feedback learning scheme with Bayesian network classifier significantly dropped the rates of SLO-IOPS violations for the *QoSFirst* and *EfficiencyFirst* policies to 5%. Also, the C4.5 decision tree performed better than Bayesian network in the Two-Phase learning scheme. Overall, the rate



(a) Feedback learning scheme



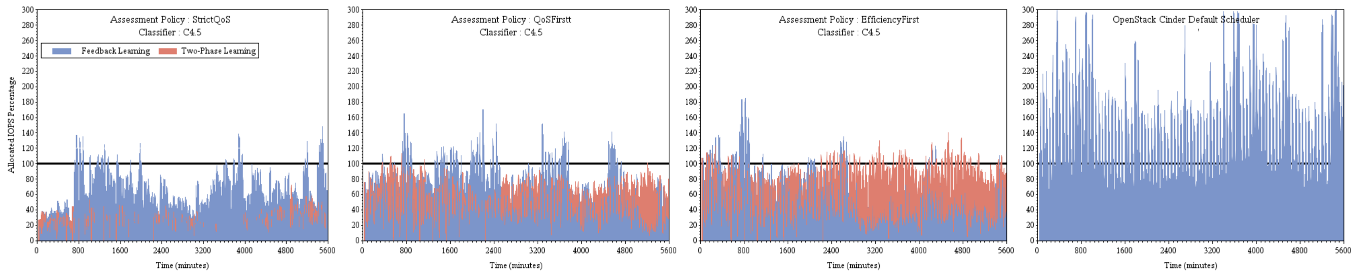
(b) Two-Phase learning scheme

Fig. 3. Performance of the self-learning scheduler in controlling the rate of SLO-IOPS violation within 1000 experiment per each combination of each assessment policies and learning schemes.

of SLO-IOPS violations was regulated with the choice of *Assessment Polices*, e.g. the *StrictQoS* policy achieved the minimal violation rates. On the other hand, the rejection rate of the create-volume requests was increased, thereby more back-end nodes were required to schedule a higher number of the requests. Figure 3 presents the performance of the self-learning scheduler in controlling the rate of SLO-IOPS violations in 1000 experiments when combining the assessment policies and the learning schemes.

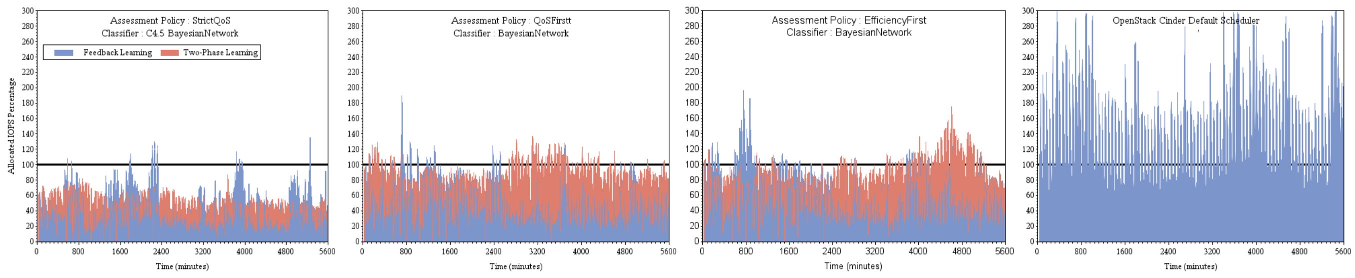
Figure 4 presents the percentage of IOPS allocation for each *Assessment Policies* using the *Two-Phase* and *Feedback* learning schemes. White spaces below the 100% line refer to the reserved resources to guarantee QoS, and the allocations above the 100% line denote over-allocation of the available IOPS to increase the resource utilization. The results showed that the Cinder default scheduler over-allocates the IOPS resources as it does not consider the QoS. Therefore, it has the highest rates of SLO-IOPS violations (Figure 2). In contrast, for each backend node, the proposed QoS-aware policies only allocate a sufficient number of volumes to assure QoS for the future create-volume requests.

The proposed scheduler met the certain levels of QoS indicated by the user configurations and at the same time maximized the resource utilization. Consequently, depending on the task, the proposed scheduler determine and configure backend block storage nodes to provide efficient scheduling decisions.



Assessment Policy: StrictQoS | QoSFirst | EfficiencyFirst | OpenStack Cinder Default Scheduler

(a) C4.5 Decision Tree



Assessment Policy: StrictQoS | QoSFirst | EfficiencyFirst | OpenStack Cinder Default Scheduler

(b) Bayesian Network

Fig. 4. Percentage of resource allocation during the simulations for each *Assessment Policy*, evaluated within both Two-Phase and Feedback learning.

V. Conclusion

In this work, we introduced a self-learning scheduling algorithm for cloud block storage systems. The proposed system can be easily integrated to OpenStack deployments. The scheduling decision assumes that the internal states of the backend nodes are unknown, and by monitoring the performance across multiple workloads, we were able to cluster the performance and schedule the resources. We also defined three assessment policies that can dictate how the violation rates of SLO-IOPS should be controlled offering another layer of Quality of Service to cloud tenants. We evaluated the performance of the self-learning schedulers. The violation rates of SLO-IOPS were well controlled within 85% of accuracy for all three policies for most experiments. Particularly, the rate of SLO-IOPS can be reduced to 2% when the *StrictQoS* policy is selected. Overall, the proposed scheduler makes SLA-aware scheduling decisions independent of the workload, the internal states of the backend nodes, and the state of the cloud (i.e. network traffic). Therefore, the scheduler can reduce the cost of cloud storage infrastructure and increase the resource utilization.

Our short term plan is to open source our code base to the OpenStack community repositories as well as study the effectiveness of our architecture in a larger hardware deployment.

References

[1] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," *EECS Department, University of California, Berkeley*, vol. 28, p. 13, 2009.

[2] 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.

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

[4] 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.

[5] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, p. 10, 2008.

[6] "Openstack cinder," [Online]. Available: https://wiki.openstack.org/wiki/Cinder.

[7] S. Yadav, "Comparative study on open source software for cloud computing platform: Eucalyptus, openstack and opennebula," *International Journal Of Engineering And Science*, vol. 3, no. 10, pp. 51–54, 2013.

[8] "Block device and Openstack," [Online]. Available: http://docs.ceph.com/docs/master/rbd/rbd-openstack/.

[9] K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," *Proceedings of the International Joint Conference on Uncertainty in AI*, vol. Q334 .I571, pp. 1022–1027, sep 1993.

[10] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.

[11] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2, 1995, pp. 1137–1145.

[12] "Laying cinder block (volumes) in openstack, part 2: Solutions design," [Online]. Available: http://www.rackspace.com/blog/laying-cinder-block-volumes-in-openstack-part-2-solutions-design.