Towards Self-Managing Cloud Storage with Reinforcement Learning

Ridwan Rashid Noel
Dept. of Computer Science
University of Texas at San Antonio
San Antonio, Texas-78249
Email: ridwanrashid.noel@utsa.edu

Rohit Mehra
Dept. of Computer Science
University of Texas at San Antonio
San Antonio, Texas-78249
Email: rohit.mehra@my.utsa.edu

Palden Lama
Dept. of Computer Science
University of Texas at San Antonio
San Antonio, Texas-78249
Email: palden.lama@utsa.edu

Abstract—**Cloud storage services are often associated with various performance issues due to load imbalance, interference from background tasks such as data scrubbing, backfilling, recovery, and the difference in processing capabilities of heterogeneous servers in a datacenter. This has a significant impact on a broad range of applications that are characterized by massive working sets and real-time constraints. However, it is challenging and burdensome for human operators to hand-tune various control-knobs in a cloud-scale storage cluster for maintaining optimal performance under diverse workload conditions. Our study on an open-source object-based storage system, Ceph, shows that common load balancing strategies are ineffective unless they are adapted according to workload characteristics. Furthermore, positive effects of an applied strategy may not be immediately visible.**

To address these challenges, we developed a machine learning based system adaptation technique that enables a cloud storage system to manage itself through load balancing and data migration with the aim of delivering optimal performance in the face of diverse workload patterns and resource bottlenecks. In particular, we applied a stochastic policy gradient based reinforcement learning technique to track performance hotspots in the storage cluster, and take appropriate corrective actions to maximize future performance under a variety of complex scenarios. For this purpose, we leveraged system-level performance monitoring and commonly available control-knobs in object-based cloud storage systems. We implemented the developed techniques to build an Adaptive Resource Management (ARM) system for object based storage cluster, and evaluated its performance on NSF Cloud's Chameleon testbed. Experiments using Cloud Object Storage Benchmark (COSBench) show that, ARM improves the average read and write response time of Ceph storage cluster by upto 50% and 33% respectively, compared to the default case. It also outperforms a state-of-the-art dynamic load rebalancing technique in terms of read and write performance of Ceph storage by 43% and 36% respectively.

Keywords-**Performance Interference; Ceph; Cloud Storage; Reinforcement Learning.**

I. Introduction

Cloud storage services provide cost-effective, highly scalable and reliable platforms for storing large scale enterprise data. This is possible due to the underlying object-based storage technology (e.g OpenStack Swift [4], Ceph [26], Amazon S3, etc.). However, todays cloud services are associated with various performance issues [21, 27]. One of the factors influencing cloud storage performance is the interference from background tasks such as data scrubbing, recovery, rebalancing etc. running on a storage cluster. Furthermore, the difference in the processing capabilities of storage servers, which arises as servers are gradually upgraded and replaced in a cloud datacenter, can also be detrimental to the overall performance of the storage cluster. Existing cloud storage systems mostly rely on human operators to tune various control knobs to address performance issues. For example, a datacenter administrator needs to tune various configuration parameters to determine load balancing and data distribution strategies for the storage cluster. Manually tuning these control knobs for maintaining optimal performance is not only burdensome but may also be ineffective if they are not adapted to diverse workload patterns. In our motivational case study on an open-source object storage system, Ceph, we observed a complex interplay between various workload patterns (i.e. read-write ratio, data object size, etc.), underlying resource bottlenecks, and the performance of storage cluster under different system adaptation strategies. We also observed that the positive effects of an applied strategy may not be immediately visible, which further complicates the task of performance tuning.

There are recent studies that address performance issues caused by hardware and workload heterogeneity on distributed storage systems through workload-specific configuration of multiple independent micro object stores [6]. However, determining the optimal number of microstores and performing workload-to-microstore mapping on the fly is challenging in a multi-tenant cloud environment. Various works have enabled elastic scaling of cloud based storage [11, 23]. In contrast, our work focuses on addressing the adverse effects performance hotspots which may in fact arise due to the overheads of data movement when a cloud storage cluster is being scaled. There are client-centric approaches on addressing high latency variance associated with cloud storage services through request duplication [27], replica selection [21], and storage tiering [9, 20] . Such approaches are complementary to our work, which aims to reduce the burden of performance tuning from datacenter administrators by developing self-managing capabilities in a cloud storage system.

In this paper, we developed a reinforcement learning based system adaptation technique that enables a cloud storage system to manage itself through load balancing and data migration with the aim of minimizing the average response time in the face of diverse workload patterns and resource bottlenecks. Such self-managing and self-adaptive capability in cloud-scale storage systems can significantly reduce human efforts required for datacenter administration while delivering near optimal performance. In particular, we make the following contributions.

1) We analyzed system-level performance metrics to capture resource bottlenecks in a cloud storage system under various workload conditions and interference from background jobs. Utilizing system level metrics makes our approach easily portable to any storage software. We further studied the performance impact of commonly available control knobs such as adjusting load serving ratios and data migration among storage servers to develop effective system adaptation heuristics.
2) We applied a stochastic policy gradient based reinforcement learning technique to select the corrective actions needed to mitigate performance hotspots in the storage cluster under various workload conditions and resource bottlenecks. Reinforcement learning is particularly well-suited for this problem since it is able to generate policies optimizing a long-term goal instead of focusing only on immediate outcomes. This property enables cloud storage systems to take beneficial actions even though their positive effect may not be immediately visible. Furthermore, stochastic policies can robustly handle the uncertainties and performance variability of a cloud environment.
3) We implemented the developed techniques to build an Adaptive Resource Management (ARM) system for object based storage cluster. We evaluated ARM on NSF Cloud's Chameleon testbed. Our experiments using an open source Cloud Object Storage Benchmark (COS-Bench) [30] show that, ARM improves the average read and write response time of the Ceph storage by upto 50% and 33% respectively, compared to the default case. It also outperforms a state-of-the-art dynamic load rebalancing technique in terms of read and write performance of Ceph storage by 43% and 36% respectively.

The rest of the paper is organized as follows. Section II provides the background studies and motivation of this work. Section III outlines related work. Section IV describes the research challenges. Section V discusses reinforcement learning approach of system adaptation. Section VI presents the implementation details and Section VII evaluates the proposed system. Section VIII provides convergence and overhead analysis of the proposed solution. Section IX summarizes our contribution and discusses future work.

II. Background

As a representative cloud storage system we use Ceph, an open-source distributed object storage platform, which is gaining increasing popularity due to its robust design and scaling capabilities. Ceph provides all data access methods (file, object, block) and appeals to IT administrators with its unified storage approach. Its core, RADOS, is a fully distributed, reliable and autonomous object store. Ceph's building blocks are called OSDs (Object Storage Daemons), which are responsible for storing objects on local filesystems, as well as working together to replicate data, detect and recover from failures, or migrate data when OSDs join or leave the cluster. An OSD generally consists of one ceph-osd daemon for one storage drive within a host machine. Ceph OSD daemons create object replicas on other Ceph nodes to ensure data safety. Ceph is highly scalable mainly due to its pseudo-random placement algorithm called CRUSH (Controlled Replication Under Scalable Hashing) [26], which allows OSDs and clients to compute object locations instead of looking them up in a centralized table. As a result, clients can directly interact with the OSDs for I/O after obtaining the most recent copy of the cluster map from a Ceph Monitor. Based on the hierarchies of failure domains for placing object replicas at different levels, e.g. disk, host, rack, etc. and a set of rules, CRUSH maps objects to placement groups (logical aggregations of objects inside one pool) and then maps each placement group to one primary OSD, and some replica OSDs as shown in Figure 1. Placement groups eliminate the computationally expensive task of tracking object placement and object metadata on a per-object basis.

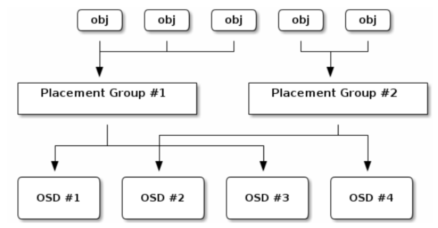


Fig. 1: Objects grouped into placement groups, and distributed to OSDs via CRUSH, a specialized replica placement function in Ceph.

Despite the flexibility and extreme scalability of object-based cloud storage systems, they are vulnerable to the detrimental effects of performance hotspots in the storage cluster. Here performance hotspots refer to a subset of storage servers that are slower than others due to various reasons. One of the primary reasons for this is hardware heterogeneity. Although storage clusters are initially setup using homogeneous configuration, hardware heterogeneity arises as servers are gradually upgraded, replaced and added in a cloud datacenter. Furthermore, with the advent of software-defined storage system such as Ceph, which supports rolling hardware and software upgrades, as well as the ability to run mixed hardware configurations, the storage nodes are even more likely to be heterogeneous.

Another important cause of performance hotspots is the interference from various background tasks. Object storage systems in general run data scrubbing tasks as a part of maintaining data consistency and cleanliness. In Ceph, OSD daemons perform data scrubbing periodically by comparing object metadata in one placement group with its replicas in placement groups stored on other OSDs to detect inconsistencies and filesystem errors. Furthermore, Ceph runs other background tasks such as data recovery and rebalancing in response to structural changes in the storage cluster. For example, when a component in a storage cluster fails, Ceph initiates recovery operations during which all data that was hosted on the failed OSD device is moved to other OSDs. Similarly when a new OSD is added to the storage cluster, the cluster map gets updated with the new OSD and Ceph starts rebalancing the cluster by migrating some of the PGs from existing OSDs to the new OSD.

III. Related Work

Performance interference in multi-tenant systems has attracted significant attention in recent year [8, 12, 19]. Delimitrou et al. [12] developed a collaborative filtering technique to classify an unknown, incoming workload with respect to how much interference it will cause to co-scheduled applications and how much interference it can tolerate. Govindan et al. [14] present a scheme to quantify the effects of cache interference between consolidated workloads. Chang et al. [8] proposed a statistical machine learning based I/O interference prediction model, and an interference-aware scheduler for data-intensive applications in virtualized environments. However, these techniques mainly focuses on performance isolation in the compute cluster and may not be easily extended to apply on cloud storage system.

Addressing high latency variance associated with cloud storage services has been a focus of several recent works. Anwar et al. [6] addressed the mismatch between the different applications requirements and capabilities of the object store by designing an architecture that supports independently configured micro stores each tuned dynamically to the needs of a particular type of workload. Wu et al. [27] focused on reducing the high latency variance associated with cloud storage services by augmenting GET/PUT requests issued by end-hosts with redundant requests, so that the earliest response can be considered. However, relying on redundant GET/PUT requests, and independently configured micro stores often incur additional costs, and inefficient resource utilization. Suresh et al. [21] reduced latencies in cloud data store by adaptively selecting one out of multiple replica servers to serve a request based on a continuous stream of in-band feedback about a servers load. These approaches are complementary to our work, which focuses on developing self-managing capabilities in a cloud storage system with system-level performance monitoring and machine learning.

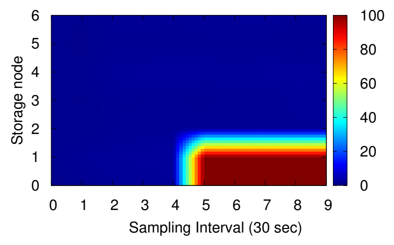
Storage tiering has also been explored in recent works [9, 20] . Cheng et al. [9] proposed a Cloud Analytics Storage Tiering solution that cloud tenants can use to reduce monetary cost and improve performance of analytics workloads. This approach performs offline workload profiling to construct job performance prediction models on different cloud storage services, and combines these models with workload specifications and high-level tenant goals to generate a cost-effective data placement and storage provisioning plan. In [20], the concept of storage tiering was extended from a single cloud storage to the wide-area and multiple data-centers. However, these approaches need application-specific fine tuning, and can not be generalized to a broad range of applications.

Scalability of storage clusters has been studied in the past. In [24], Wang et al. presented file and block I/O performance and scalability evaluation of Ceph for scientific high-performance computing (HPC) environments. Through systematic experiments and tuning efforts, they observed that Ceph can perform close to 70% of raw hardware bandwidth at object level and about 62% of at file system level. Gudu et al. [15] presented a multidimensional scalability evaluation of Ceph with the aim to achieve better understanding of how each dimension (number of OSDs, number of clients, object size) affects performance. Trushkowsky et al. [23] presented the SCADS Director, a control framework that reconfigures the storage system on-the-fly by adding or removing storage nodes in response to workload changes using a performance model of the system. In contrast, our work focuses on addressing the adverse effects of performance hotspots which may in fact arise due to the overheads of data movement when a cloud storage cluster is being scaled.

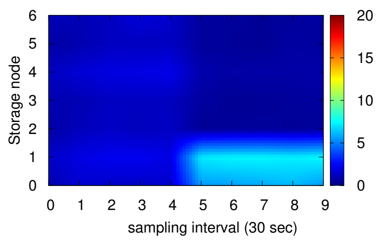
Dynamic load redistribution (DLR) technique was applied by Noel et al [18] to improve the performance of a Ceph cluster in the presence of performance hotspots caused by hardware heterogeneity and interference from background workloads. DLR relies on manually tuned thresholds on system level performance metrics to trigger load balancing among Ceph OSDs by adjusting their load serving ratio. However, our study shows that such approach is ineffective in the presence of diverse workload characteristics and resource bottlenecks.

Machine learning techniques have been applied for automated resource management of networked systems in the past [7, 12, 13, 28, 29] . Dutreilh et al. [13] applied reinforcement learning for automated resource allocation of cloud applications, and introduced further optimization for faster convergence of the learning algorithm. Xu et al. [28] applied reinforcement learning to automate the configuration processes of virtualized machines and appliances running in the virtual machines. Yadwadkar et al. [29] applied supervised machine learning technique to improve task scheduling in distributed data processing frameworks. Cano et al. [7] presented a reinforcement learning based algorithm for storage tiering in enterprise clusters. In particular, they applied Q-learning technique [25] to decide how much data to keep in SSDs and HDD under diverse workload conditions.

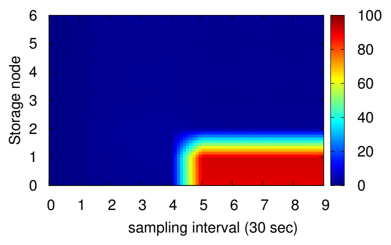
In this paper, we apply reinforcement learning based adaptive load balancing and data migration to improve cloud storage performance. Unlike the related works that apply deterministic reinforcement learning approach (i.e Q-learning), we apply a stochastic policy gradient technique to determine an action in the face of diverse workload conditions and resource bottlenecks. Stochastic policies are well-suited to an uncertain cloud environment, where executing a fixed action in a given state may not always be optimal.



(a) CPU utilization

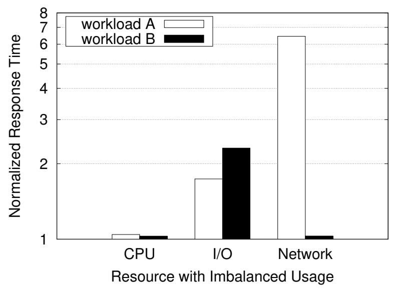


(b) %Iowait

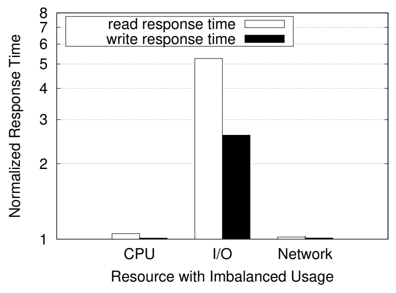


(c) Network utilization

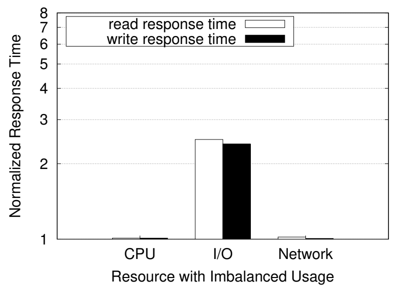
Fig. 2: Heat maps of various system-level metrics in the Ceph storage cluster.



(a) Workload A and B



(b) Workload C



(c) Workload D

Fig. 3: Ceph performance degradation due to imbalance in CPU, disk I/O and network usage for various workload mixes.

IV. Challenges in Managing Cloud Storage Performance

A. *Selecting Appropriate System Adaptation Heuristics*

Existing cloud storage systems provide various control knobs that can be used to adapt system behavior. However, it is challenging to determine which adaptation technique should be applied under what conditions and how much adaptation is needed to mitigate performance issues in the storage cluster. In this paper, we focus on two adaptation techniques provided by Ceph, which involves adjusting the primary affinity and weight associated with Ceph OSDs.

In Ceph, a primary affinity value determines the probability that an OSD will act as the primary OSD. Any read operation is always done from the primary OSD. By default, all OSDs have a primary affinity value of 1 and decreasing an OSDs primary affinity value reduces the amount of read workload that it serves. Hence, primary affinity can be utilized to adjust the read workload serving ratio of Ceph OSDs. However, this has little effect on write-heavy workloads. This is because a write operation in Ceph is performed on all OSDs within a placement group (PG) regardless of the primary affinity values associated with them. While writing, data is first written to primary OSD, and then replicated to the other OSDs.

Another mechanism of system adaptation in Ceph is to adjust the OSD weights. The OSD weights control the amount of data stored in the OSDs by mapping PGs to OSDs in proportion to their weights. Although OSD weights influence both read and write performance, changing the weights result in data migration between OSDs, and incur associated overheads.

TABLE I: Ceph COSBench workload profile

Workload	Obj. size	Operation distribution	App. scenario
A	1-8MB	G: 90%, P: 5%, D: 5%	Online video sharing
B	1-8MB	G: 5%, P: 90%, D: 5%	Enterprise backup
C	1-128KB	G: 50%, P: 50%, D: 0%	General
D	1-8MB	G: 50%, P: 50%, D: 0%	General

B. *Performance Hotspots under Various Workload Mixes*

Determining a corrective action to improve cloud storage performance in response to system-level performance metrics is non-trivial due to a complex interplay between various workload patterns (i.e. read-write ratio, data object size, etc.), underlying resource bottlenecks, and the performance of storage cluster. To analyze this complex behavior, we setup an 8-node Ceph cluster on the NSF Clouds Chameleon testbed. One of the node was used as Ceph Admin, and the remaining 7 nodes were used as storage nodes, each hosting one OSD. Consideration of multiple OSDs per node is out of scope for this paper. The Ceph version used was 9.2.1 (Infernalis-stable).

We examined four different workload mixes derived from COSBench [30] (version 0.4.2) to represent real-world applications that use cloud object storage as listed in Table I. The workload mixes have different ratio of read, write and delete operations. Workload A is read-heavy with 90% GET requests, 5% PUT requests and 5% DELETE requests. Workload B is write-heavy with 5% GET requests, 90% PUT requests and 5% DELETE requests. The remaining workloads have balanced read and write ratio. The object size distribution is uniform in the range of 1 to 8 MB for workloads A, B, D, and 1 to 128 KB for workload C. We setup two storage containers in the Ceph cluster and worked with a total number of 500 objects for workload A, B and D and 5000 objects for workload C. Each COSBench job was run for 5 minutes.

To introduce performance hotspots in the storage cluster, we ran various background jobs, one at a time, on two of the Ceph nodes starting at time 2 minutes. The background jobs worked on creating specific resource contention in the Ceph OSDs. We used sysbench [17] cpu benchmark for stressing the CPU resource, fio [2] tool for introducing I/O contention, and iperf [3] tool for introducing contention in the network bandwidth. More details on the experimental setup and background jobs are described in section VII-B. Figure 2 shows the heat maps of CPU utilization, %iowait and network utilization in the Ceph storage cluster. Here, %iowait provides the percentage of time that the CPU or CPUs are idle during which the system has an outstanding disk I/O request. Since both read-heavy and write-heavy workloads resulted in similar heat maps, we only show one set of data. We observe that the background jobs cause various degrees of imbalance in the utilization of CPU, disk I/O and network resources across the Ceph storage nodes starting at the 4th sampling interval.

Figure 3 shows the impact of imbalance in the CPU utilization, %iowait and network utilization of the storage nodes on the average response time of COSBench workloads. We observe that in the case of high imbalance in CPU utilization among the storage nodes, the performance of COSBench workloads is largely unaffected. In the case of imbalance in network utilization, read-heavy workload A experienced a large performance degradation (6X increase in average read response time). However, the remaining workloads were not affected. This is due to the presence of significant amount of write operations in workload B, C and D. Since every single write operation involves writing on all the OSD replicas across the network, workloads B, C and D already cause high network utilization among Ceph OSDs and suffer from large network latency even when there is no network interference from background jobs. Hence, the presence of additional network interference does not have much impact on performance. In the case I/O interference from background jobs, we found that even a small amount of imbalance, in term of %iowait, among the storage nodes cause significant performance degradation for all of the COSBench workloads.

C. Delayed Effect of System Adaptation

One of the key challenges of system adaptation is that sometimes the positive effects of an applied strategy may not be immediately visible. To demonstrate this phenomenon, we measured the impact changing the OSD weights on the performance of COSBench workload in the presence of I/O interference from background jobs. In this experiment, starting at the second sampling interval (time 60 seconds), we reduced the OSD weights of the two storage nodes where the background jobs were running. As shown in Figure 4, the average read response time of COSBench workload A initially increases due to the data movement caused by OSD reweight. However, the performance improves significantly starting at sampling interval 5. This is because data migration from the two low-performing OSDs to the remaining OSDs, divert majority of the subsequent read requests to the well-performing OSDs. These results motivate the need to incorporate such delayed effects in designing effective system adaptation techniques.

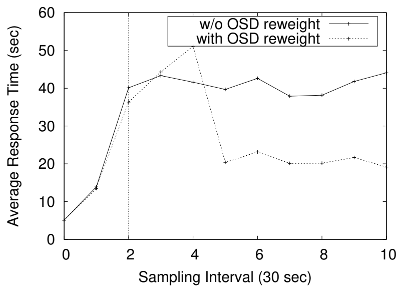


Fig. 4: Delayed effect of OSD reweight on the performance of COSBench workload in the presence of I/O interference on a subset of storage nodes. OSD reweight is applied at sampling interval 2.

V. Reinforcement Learning Based Adaptive Load Balancing and Data Migration

A. A Case for Reinforcement Learning

In this paper, we use reinforcement learning to adaptively determine how the primary affinity and weights of Ceph OSDs should be adjusted for mitigating performance hotspots in the storage cluster. Reinforcement learning (RL) is a process by which a machine or an agent can learn to achieve desired goal by interacting with the environment. This autonomous agent has the capability to sense the state of environment and take actions leading to other states. As agent transitions from one state to another, it only receives a numerical reward signal. At the start of training, agent is unaware of the best actions, but with interactions it discovers and learns which actions yield the most reward in specific situations or state of the environment.

We choose RL based system adaptation approach for the following reasons:

- Unlike supervised machine learning techniques, RL algorithms do not require labelled training data. In a dynamic cloud environment with ever changing workloads, obtaining the training dataset is difficult and time consuming.
- RL can generate policies optimizing a long-term goal of maximizing cumulative rewards instead of focusing only on immediate outcomes. This property can be utilized to incorporate delayed effects of system adaptation in the learning algorithm.

- RL's model-free approach is application-agnostic in its design, and hence is applicable to any type of workloads.

B. Problem Formulation

RL problems can be formulated as a Markov decision process (MDP), which consists of a set of states and several actions for each state. Every state in MDP satisfies "Markov" property, which refers to the fact that the future only depends on the current state and not the history. Hence, the current state contains enough information to choose optimal actions to maximize future rewards. RL algorithms assume that the problems to be learned are (at least approximately) Markov decision processes.

Consider a set of states S and a set of actions A. The MDP is defined by the transition probability, $P(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a)$ and expected reward function, $R(s,a) = E[r_{t+1}|s_t = s, a_t = a]$. At each step t, the agent perceives its current state $s_t \in S$, takes an action $a \in A$, transits to the next state s_{t+1} and receives an immediate reward r_{t+1} from the environment. The goal of the RL agent is to develop the policy of choosing actions $\pi(s,a) = P(a|s)$, which can maximize the cumulative rewards through iterative interactions with the environment. If the sequence of rewards received after time step t until the end of an episode is $r_{t+1}, r_{t+2}, r_{t+3}, \ldots, r_T$ then the objective of learning is to maximize the expected discounted return. The discounted return G_t^T is of the form:

$$G_t^T = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \tag{1}$$

Here, γ is called the discount factor whose value ranges between 0 and 1. $\gamma = 0$ will make the learning agent short sighted and it will consider only immediate rewards; $\gamma \rightarrow 1$ will make it look into long term future rewards.

We describe the State-action-reward configurations of our problem as follows.

States: We define state s at time step t by the following tuple $s_t = (iowait, rtps, wtps, net)_t$, where each element is a vector that denotes the %iowait value, disk read requests per second, disk write requests per second, and network utilization respectively on each storage node. The %iowait and network usage data helps the agent determine the resource bottleneck. Similarly, rtps and wtps help in determining the proportion of read and write operations associated with a given workload. These metrics also help the learning agent to figure out whether the workload has small object size or large object size. Workloads with small object sizes typically cause more reads and writes per second than workloads with large object sizes.

Actions: We define five possible actions A to choose for every state. The actions are 1) I/O-based affinity control, 2) I/O-based OSD reweight, 3) network-based affinity control, 4) network-based OSD reweight or 5) take no action (no-op). The policy maps a state to a probability function over the actions. The RL problem only focuses on which action should be taken under different workload conditions. To determine how much of an action should be taken, we apply a system adaptation heuristic for increasing or decreasing the primary affinity and weights of an OSD in proportion to the difference between the OSD's system-level performance metrics, and the average value measured among all OSDs. Let $iowait_i$ be the %iowait and net_i be the network utilization measured at OSD i. We update the primary affinity of OSD i by subtracting a value $2*\delta_i$, where $\delta_i = (1 - iowait_{avg}/iowait_i)$ for I/O-based affinity control and $\delta_i = (1 - net_{avg}/net_i)$ for network-based affinity control. On the other hand, we update the OSD weight less aggressively by subtracting δ_i value. This is to avoid excessive data movement among the OSDs due to change in weights.

Rewards: Finally, we define the rewards according to the workload type. As the state observations provide disk read and write statistics, we can detect whether the running workload is read-heavy or write-heavy. We use the reciprocal of the average read (GET request) response time and the reciprocal of average write (PUT request) response time of COSBench as a $performance\_score$ for ready-heavy and write-heavy workloads respectively. For read-write balanced workloads, we calculate the $performance\_score$ as $2/(read\_response\_time + write\_response\_time)$.

C. RL Solution

We use a stochastic policy gradient method [1, 22] to solve the RL problem. A stochastic policy determines the probability of taking a certain action at given state. Such policy allows our agent to explore the state space without always taking the same action for a given state. As a consequence, it handles the exploration/exploitation trade off without hard coding it. In contrast to deterministic or quasi-deterministic approaches such as Q-learning [25], stochastic policy gradient method is very effective in overcoming the problem of perceptual aliasing [10], where different states appear to be the same but require different actions. This property is useful in learning good policies in a partially observable domain such as a complex cloud storage system where the observed state may be insufficient in fully describing the system.

Policy gradient methods aim to optimize the policies directly by running a policy for a while, observing what actions led to high rewards, and increasing the probability of those actions. The policy $\pi_\theta(s_t, a_t)$ is represented as a parameterized function with respect to θ. We use a simple feed-forward neural network with two hidden layers to approximate the policy function. Each time the agent interacts with the environment, the parameters θ of the neural network are tweaked so that "good" actions will more likely be sampled in the future and vice versa. This process is repeated until the policy network converges to the optimal policy π^*.

Formally, the main objective of Policy Gradients is to maximize the total future expected rewards $E[G_t^\infty]$, where G_t^∞ represents the sum of all future discounted rewards. The parameters θ of the policy network is iteratively tweaked so that $E[G_t^\infty]$ is maximized. This is achieved by calculating the gradient of $E[G_t^\infty]$ which can be formalized as follows:

$$\nabla_\theta E[G_t^\infty] = E[G_t^\infty \nabla_\theta log P(a)] \tag{2}$$

Algorithm 1 REINFORCE Algorithm

```
1: function REINFORCE
2:     Initialize $\theta$ arbitrarily
3:     for each mini-batch of K episodes do
4:         Run every episode of the mini-batch
5:         for each episode
6:             $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
7:             for $t = 1$ to $T - 1$ do
8:                 $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t)(G_t^T - b)$
9:             end for
10:        end for
11:    end for
12:    return $\theta$
13: end function
```

Intuitively, G_t^∞ is the scaling factor which dictates how the probability $P(a)$ of taking action a should change in order to maximize the expected future rewards. Eventually, good actions will have an increased likelihood to get sampled in future iterations. The derivation of Equation 2 can be found in [1].

As shown in Algorithm 1, the sequence of (state, action, reward) values $< s_t, a_t, r_{t+1} >$ sampled from the environment at each training episode. After each mini-batch of $K = 6$ episodes, the data samples are used to estimate the gradient of $E[G_t^\infty]$. An extension of gradient ascent technique called Adam [16] optimization is applied to maximize $E[G_t^\infty]$ by updating the parameters θ of the policy network. In this REINFORCE algorithm, α is the learning rate that determines how aggressively the policy parameters θ are updated. $\pi_\theta(s_t, a_t)$ is the policy (which maps state to action probabilities), and G_t^T is the expected discounted reward after time step t. Here, G_t^T is an approximation of G_t^∞ on each episode.

Algorithm 1 differs from vanilla REINFORCE algorithm in two major ways. (1) We apply a widely used variation of REINFORCE that subtracts a baseline value b from the return G_t^T to reduce the variance of gradient estimation while keeping the bias unchanged and facilitate faster convergence. Here, b is the performance score measured from the environment at the beginning of each training episode when the OSD weights and affinities are reset to Ceph's default values. This ensures that the agent gets a negative reward when its actions reduces the performance from the baseline, and such actions are discouraged. Although measuring the baseline performance in the beginning of each episode may result in sub-optimal performance for one sampling interval, the benefit of faster convergence outweighs the overhead. One possible improvement (in future work) is to measure the baseline performance only when a change in workload is detected. (2) Policy network update is delayed until the end of a mini-batch. In the face of performance variability of cloud environment, change in workloads, and the variability introduced by stochastic policy, this approach avoids unnecessary fluctuations in policy after every episode, thereby facilitating faster convergence.

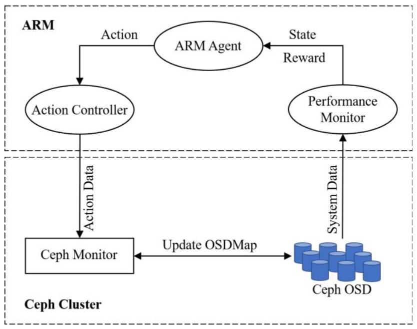


Fig. 5: System Architecture

D. Hyperparameter Tuning

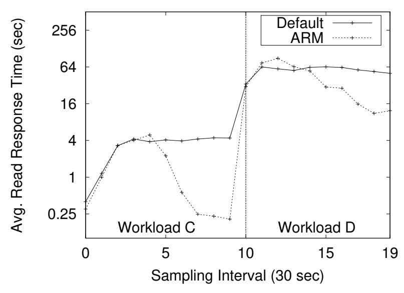
We now present the methodology for tuning various hyperparameters used in Algorithm 1. The initial learning rate α is set to 0.001, which is a default value for Adam optimization. As the training progresses, the learning rate is adapted according to the Adam optimization technique. In contrast to classical stochastic gradient ascent (descent) that maintains a single and static learning rate for all parameters of the policy network, Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. This method is well suited for problems with large number of parameters and noisy gradients.

The length of a training episode and the sampling interval within each episode are heuristically chosen to be 300 seconds and 30 seconds respectively. If the sampling interval is too small, then the observed data can become noisy and if the interval is too large, then the RL agent will be too slow in reacting to the changes in the environment. Similarly, if the training episode is very long, then it will take longer time to collect the discounted return G_t^T needed for updating the policy network, thus slowing down the entire training process. On the other hand, if the episode is too short then the RL agent cannot sufficiently explore the various states and actions for a given episode.

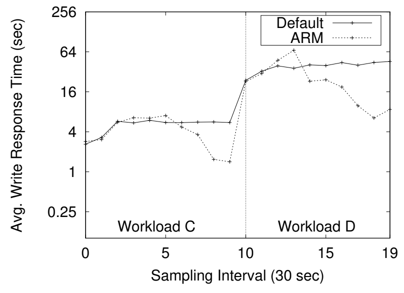
VI. Implementation

A. System Architecture

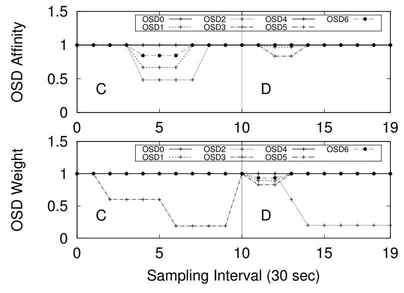
In this section, we present the implementation details of our RL based Adaptive Resource Management (ARM) system. As shown in Figure 5, ARM mainly consists of three modules: the Performance Monitor, ARM agent and Action Controller. The performance monitor periodically measures the system level performance metrics of the Ceph OSDs, and the response time of Ceph workloads. These metrics are sent to the ARM agent as the state information (s_t) and reward information (r_t) respectively. The ARM agent applies stochastic policy gradient method to periodically update the policy network as shown in Algorithm 1. Furthermore, it selects one of the five possible actions describe in Section V-B based on their probabilities. Finally, the Action Controller applies the system adaption action suggested by the ARM agent.



(a) Read Response Time

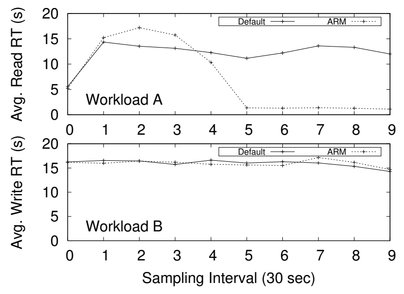


(b) Write Response Time

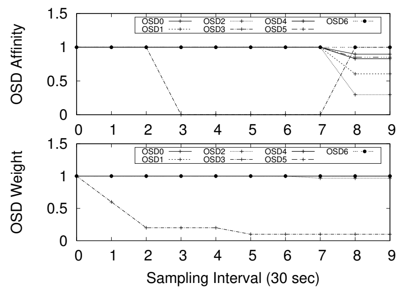


(c) Actions

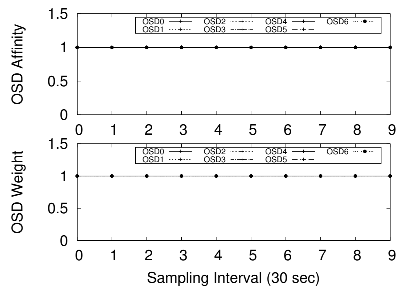
Fig. 6: Performance of COSBench workloads, and the corresponding system adaptation actions taken by ARM in the presence of I/O interference on a subset of storage nodes. The workload mix changes from workload C to D at sampling interval 10.



(a) Response Time



(b) Actions for Workload A



(c) Actions for Workload B

Fig. 7: Performance of COSBench workloads, and the corresponding system adaptation actions taken by ARM in the presence of network interference on a subset of storage nodes.

B. Performance Monitor

The performance monitor fetches the system data from the Ceph OSDs at 30 second intervals by using the Linux SAR (System Activity Report) [5] utility. For the I/O data, it collects the %iowait value from each storage node. To measure the network utilization, it collects rxkb/s and txkb/s, which is the amount of network data (kilobytes) received and transmitted from a storage node, and divides these values by the overall network bandwidth to get the network utilization. It also collects rtps and wtps, which is the number of read and write requests to the disk. Together these metrics provide the state information for the ARM agent. For the reward data, the performance monitor measures the average read response time and write response time of COSBench's GET and PUT requests respectively.

C. ARM Agent

The ARM Agent is responsible for learning optimal policies that maximize the total future expected rewards. We used TensorFlow (version 1.1.0) to implement a neural network with two hidden layers as the policy function approximator. The hidden layers of the neural network uses ReLU activation function. For any given state, the output layer uses a Softmax function to calculate the probability distribution over the five actions described in Section V-B. The ARM agent initializes the policy network parameters θ randomly, and updates them after mini-batches of six episodes as shown in Algorithm 1.

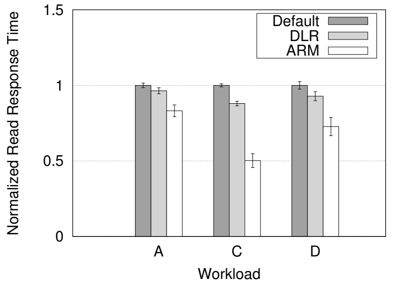
D. Action Controller

According to the decision made by the ARM agent about which type of adaption needs to be applied on the storage cluster, the Action Controller takes the appropriate action as described in V-B. The Action Controller invokes the Ceph monitor to update the OSDmap of the Ceph cluster so that the new adaptation parameters such as OSD weights and primary affinity take effect. Ceph monitors are responsible for maintaining the cluster map, which includes information on the cluster topology, a list of OSDs, pools, placement groups, and mapping of placement groups to OSDs.

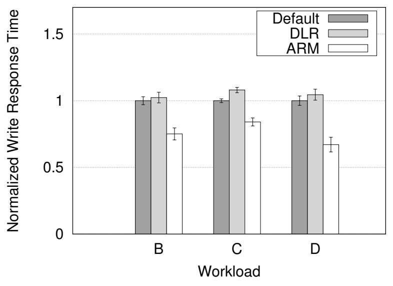
VII. Evaluation

A. TestBed

For the experiments, we setup an 8-node Ceph cluster with Ceph version 9.2.1 (Infernalis-stable) on the NSF Cloud's Chameleon testbed. One of the nodes is configured as Ceph Admin and Ceph monitor. The remaining seven nodes are configured with one OSD per node. Two storage containers each with 512 placement groups are setup in the Ceph cluster. Each of the node in the cluster is equipped with 2.3 GHz Intel Xeon E5-2650 v3 "Haswell" processors. The nodes run

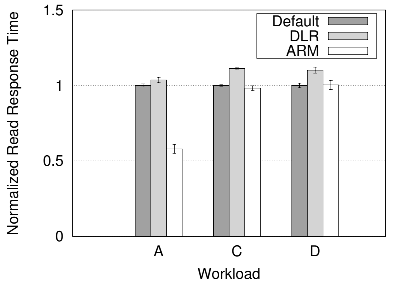


(a) Read Response Time

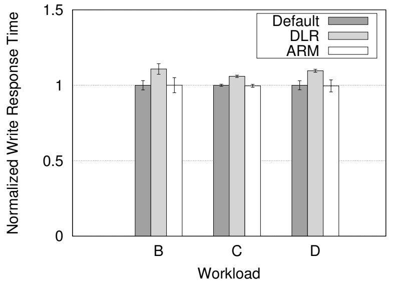


(b) Write Response Time

Fig. 8: Performance Comparison among Default Ceph, DLR and ARM in the presence of I/O Interference on a subset of storage nodes.



(a) Read Response Time



(b) Write Response Time

Fig. 9: Performance Comparison among Default Ceph, DLR and ARM in the presence of network interference on a subset of storage nodes.

Ubuntu Linux 14.04.1 LTS with kernel 3.13.0 and each has 80GB of hard disk space and 16GB of memory.

B. Benchmarks

For measuring the cloud storage performance, we setup Intel Cloud Object Storage Benchmark (COSBench) [30] version 0.4.2 in our Ceph cluster. The benchmark has provision for running workloads with different read-write ratio and object sizes. We ran 4 workloads with different read-write ratio and object sizes. The workload profiles are provided in Table I. For the I/O background jobs, we ran Flexible I/O tester (fio) [2]. We setup fio version 2.1.3 with the following configurations: libaio ioengine, iodepth of 16 and blocksize of 4kB. We ran 4 simultaneous random read jobs of size of 2GB each. We used iperf [3] version 3.1.3 in the cluster nodes as the background jobs that create network interference. We setup iperf with window size of 8MB in the server and client nodes where 50 client threads ran in parallel with the server.

C. Experimental Results

We compared the performance of our ARM system with the Ceph default performance, where no system adaptation actions are taken, and with the performance of dynamic load rebalancing technique, DLR [18]. In our experiments, we only used the background jobs that create I/O and network interference, since motivational case study in section IV show that cpu resource usage does not have much impact on the performance of COSBench workloads. We ran our ARM agent for a total of 1000 episodes with each episode having a duration of 300 seconds. The final 200 episodes were used for evaluation. Each episode ran a particular COSBench workload which was selected randomly from the workload mix shown in Table I. The duration of the workloads coincided with the episode length. The background jobs were run on two randomly selected nodes in the Ceph storage cluster. After each episode all the benchmarks were stopped and the OS cache was cleared to remove any caching effect.

Figures 6 (a), (b) and (c) show the average read and write response times of randomly selected representative episodes running COSBench workloads C, D and the corresponding actions taken by ARM in the presence of I/O interference on two of the Ceph nodes. We observe that ARM is able to decrease the response time of each workload after a few sampling intervals, and gain significant performance improvement over the Ceph default case. As shown in Figure 6 (c), the primary-affinity and weights of the Ceph OSDs are updated according to the system adaptation heuristic by ARM. The updates are mostly done in the earlier stages of the workloads but the effects of the adaptations can be seen in the later stages of the workloads due to the delayed effect of the adaptations. Figures 7 (a), (b) and (c) show the average response times of randomly selected representative episodes running workload A, B and the corresponding actions taken by ARM in the presence of network interference on two of the Ceph nodes. For the read-heavy workload A, we observe that the read response time initially increases but decreases significantly after few sampling intervals. However, the write response time of the write-heavy workload B remains almost the same in both Ceph

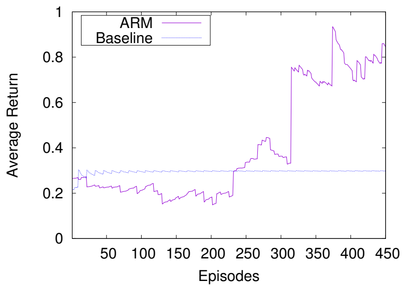


Fig. 10: Convergence of ARM's policy gradient algorithm.

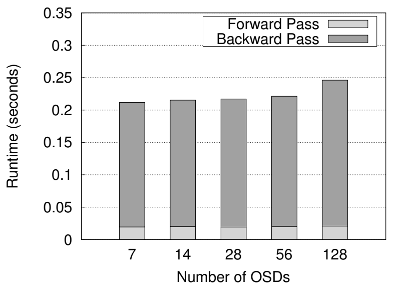


Fig. 11: Overhead of Tensorflow

default and ARM case. This is because, as shown in Figure 7 (c), ARM does not take any action for workload B in this case. Since there was not much degradation in write performance due to network interference in the first place, adapting the system unnecessarily may actually degrade the performance owing to the associated overheads of system adaptation.

Figure 8 and 9 compare the performance among the Ceph default, DLR and ARM in the I/O and network interference cases respectively. The results presented here are an average of the final 200 episodes. The response times of the COSBench workloads are normalized with respect to their response times in the Ceph default case. In the I/O interference experiments, ARM outperforms Ceph default and DLR for all the workloads A, C and D. The read response time of workload C improved the most by up to 50% and 43% compared to the Ceph default and DLR respectively. The write response time improved the most for workload D, where the improvement were 33% and 36% compared to Ceph default and DLR respectively. DLR did not perform well in these experiments since it uses fixed threshold values for triggering system adaptation irrespective of the different workload types.

In case of network interference, the performance of ARM is almost similar to the Ceph default case except for workload A. Since only the performance of read-heavy workload A gets degraded by the background network tasks, as explained in section IV, ARM takes the necessary actions to improve the read response time of workload A by up to 42% and 44% compared to Ceph default and DLR respectively.

VIII. Convergence and Overhead Analysis

Figure 10 shows the average return over episodes during the training of ARM agent. It also shows the average reward over episodes for baseline, which is computed based on the performance of Ceph default case. Initially, ARM's average return is small since the actions taken by the ARM agent are mostly exploratory. However as the agent learns to take more meaningful actions from episode 250 onward, the average return increases significantly until it converges to a stable value around episode 400. The convergence of ARM's policy gradient algorithm is facilitated by the use of baseline comparison and mini-batch update techniques described in Algorithm 1.

The ARM implementation incurs minimal overhead. Its Performance Monitor module gathers system utilization data of the storage nodes in parallel, similar to the approach used in [18]. Furthermore, our Tensorflow based implementation of the ARM agent's policy network is highly scalable. For scalability analysis, we simulated state observations of large scale storage clusters and fed the data to equivalently sized policy networks with large number of inputs. The ARM agent was deployed in the admin node of the Ceph cluster and its hardware configuration is described in Section VII-A. Figure 11 shows the average time taken by forward pass and backward pass operations through the policy network with increasing number of OSDs in the Ceph cluster. Although the size of policy network grows rapidly with the increasing number of OSDs, the runtime overhead of using the policy network increases marginally. Furthermore, the overhead is negligible compared to ARM's sampling interval of 30 seconds.

IX. Conclusions And Future Work

In this paper, we presented a machine learning based adaptive resource management technique which enables a cloud storage system to manage itself, and provide superior performance in the presence of diverse workloads and resource bottlenecks. We applied a stochastic policy gradient based reinforcement learning technique to detect performance issues in cloud storage and take necessary actions in the form of load balancing and data migration to improve storage performance. We implemented our technique in Ceph, a software defined storage solution and evaluated the performance in NSF Cloud's Chameleon testbed using Cloud Object Storage Benchmark. Experimental results show that, our approach improves storage read and write performance by up to 50% and 33% respectively comparing to the default case and up to 43% and 36% comparing to a state-of-the-art load redistribution technique. In future, we will evaluate our approach in a larger scale dynamic storage cluster, and will investigate the scalability of our approach. We also plan to apply diverse workloads with dynamic I/O traffic and compare our approach with other RL techniques, e.g. actor-critic, DQN, etc.

Acknowledgment

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation. We thank the anonymous reviewers for their many suggestions for improving this paper. In particular we thank our shepherd, Mike Spreitzer.

References

[1] Deep reinforcement learning: Pong from pixels. http://karpathy.github.io/2016/05/31/rl/.

[2] fio(1) – linux man page. https://linux.die.net/man/1/fio.

[3] iperf(1) – linux man page. https://linux.die.net/man/1/iperf.

[4] Openstack swift. http://docs.openstack.org/developer/swift.

[5] Sar tool. https://linux.die.net/man/1/sar.

[6] A. Anwar, Y. Cheng, A. Gupta, and A. R. Butt. Mos: Workload-aware elasticity for cloud object stores. In *Proc. of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '16, 2016.

[7] I. Cano, S. Aiyar, V. Arora, M. Bhattacharyya, A. Chaganti, C. Cheah, B. N. Chun, K. Gupta, V. Khot, and A. Krishnamurthy. Curator: Self-managing storage for enterprise clusters. In *NSDI*, pages 51–66, 2017.

[8] R. C. Chang and H. H. Huang. Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments. In *Proc. Int'l Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[9] Y. Cheng, M. S. Iqbal, A. Gupta, and A. R. Butt. Cast: Tiering storage for data analytics in the cloud. In *Proc. of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2015.

[10] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI*, volume 1992, pages 183–188. Citeseer, 1992.

[11] S. Das, D. Agrawal, and A. El Abbadi. Elastras: An elastic, scalable, and self-managing transactional database for the cloud. *ACM Trans. Database Syst.*, 38(1), Apr. 2013.

[12] C. Delimitrou and C. Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proc. Int'l Conference on Architecture Support for Programming Language and Operating System (ASPLOS)*, 2013.

[13] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, pages 67–74, 2011.

[14] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam. Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proc. ACM Symposium on Cloud Computing (SoCC)*, 2011.

[15] D. Gudu, M. Hardt, and A. Streit. Evaluating the performance and scalability of the ceph distributed storage system. In *Proc. of the IEEE International Conference on Big Data*, 2014.

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv: Computing Research Repository*, 1412.6980, 2014.

[17] A. Kopytov. Sysbench manual. *MySQL AB*, 2012.

[18] R. R. Noel and P. Lama. Taming performance hotspots in cloud storage with dynamic load redistribution. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 42–49. IEEE, 2017.

[19] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proc. of the 2013 USENIX Conference on Annual Technical Conference*, 2013.

[20] K. Oh, A. Chandra, and J. Weissman. Wiera: Towards flexible multi-tiered geo-distributed cloud storage instances. In *Proc. of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '16, 2016.

[21] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[22] R. Sutton and A. Barto. *ReinforcementLearning: An Introduction*. MITPress, 1998.

[23] B. Trushkowsky, P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. The scads director: Scaling a distributed storage system under stringent performance requirements. In *Proc. of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, 2011.

[24] F. Wang, M. Nelson, S. Oral, S. Atchley, S. Weil, B. W. Settlemyer, B. Caldwell, and J. Hill. Performance and scalability evaluation of the ceph parallel file system. In *Proceedings of the 8th Parallel Data Storage Workshop*, pages 14–19. ACM, 2013.

[25] C. J. C. H. Watkins and P. Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

[26] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proc. of the 7th USENIX symposium on Operating systems design and implementation (OSDI)*, 2006.

[27] Z. Wu, C. Yu, and H. V. Madhyastha. Costlo: Cost-effective redundancy for lower latency variance on cloud storage services. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[28] C.-Z. Xu, J. Rao, and X. Bu. Url: A unified reinforcement learning approach for autonomic cloud management. *Journal of Parallel and Distributed Computing*, 72(2):95–105, 2012.

[29] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz. Wrangler: Predictable and faster jobs using fewer resources. In *Proc. of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014.

[30] Q. Zheng, H. Chen, Y. Wang, J. Duan, and Z. Huang. Cosbench: A benchmark tool for cloud object storage services. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 998–999. IEEE, 2012.