Reinforcement Learning based Load Balancing in a Distributed Heterogeneous Storage System

Jooyoung Park, Seunghwan Jeong and Honguk Woo†
Department of Computer Science and Engineering
Sungkyunkwan University, Suwon, South Korea
{aig031,party1996,hwoo†}@skku.edu

2022 International Conference on Information Networking (ICOIN) | 978-1-6654-1332-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICOIN53446.2022.9687277

***Abstract*—With the growing demand for big data storage and processing, distributed storage systems with heterogeneous devices have become the majority in cloud data centers. However, hardware heterogeneity and workload variety make it challenging to maintain optimal performance in those storage systems. In this work, we present a learning-based control method that optimizes the performance of a distributed storage system. Specifically, to provide automatic parameter tuning upon dynamic workload patterns on a tiered storage architecture, we employ deep reinforcement learning (RL) and implement a simulation environment for a Ceph storage system. Through simulation tests, we demonstrate our RL method shows better performance than other heuristic approaches for a task of load balancing based on the primary affinity settings in Ceph.**

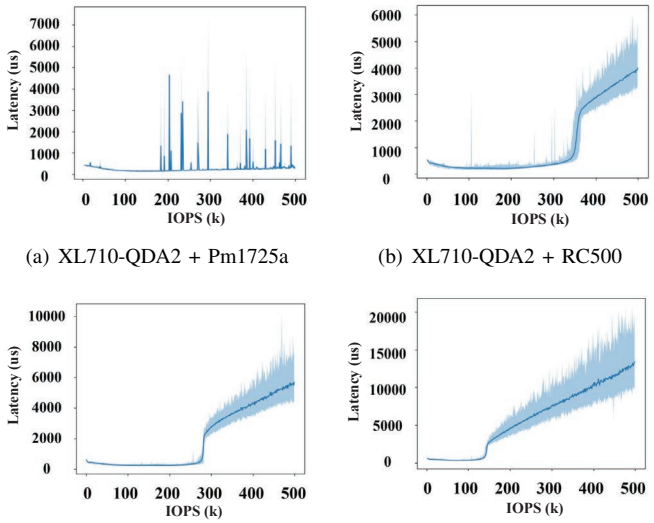
I. Introduction

Recently, distributed, multi-tiered storage systems with heterogeneous devices have become popular. While those systems are notably cost-efficient to handle the massive scale of data, their hardware heterogeneity increases the complexity and difficulty in operation, often causing performance hotspots. For example, with a Ceph storage system where the CRUSH algorithm is used to manage objects, in [1], several experiments demonstrated that performance hotspots in randomly selected nodes caused significant performance degradation of up to 55% in throughput and 2.3-times increase in latency.

In this paper, we present a deep reinforcement learning (RL)-based approach to optimize the task of dynamic load balancing in Ceph through a mechanism of primary affinity control. Specifically, we develop the primary affinity RL (PARL) method by which affinity settings are continuously adjusted to ensure high quality-of-service (QoS) in storage applications. By using the primary affinity setting, it is feasible to reduce the load on a specific node without reducing the amount of data it contains, because the preference between primary and secondary is updated and propagated to clients [2].

Through simulation experiments, we demonstrate that our PARL outperforms other heuristic methods under various system conditions, e.g., 10% improvement over a state-of-the-art Ceph CRUSH heuristic algorithm, DLR [1]

The rest of the paper is structured as follows. Section II introduces the background and motivation of this paper. Overall architecture and formulation of PARL are given in Section III. Section IV evaluates the proposed approach. Section V covers related work. Section VI concludes the paper.



(a) XL710-QDA2 + Pm1725a (b) XL710-QDA2 + RC500

(c) X710-DA2 + RC500 (d) X710-DA2 + Pm981

Fig. 1: Different device performance in latency with respect to IOPS. The x-axis denotes the total amount of IOPS (K) and the y-axis denotes the average latency (us) achieved by devices.

II. Background

Ceph is an open-source software-defined storage service, deployed in private and public cloud data centers. It includes several modules such as monitors, managers, and object storage daemons (OSD), where managers track status information generated by monitors and OSDs manage data.

In Ceph, CRUSH [3], a pseudo-random data distribution algorithm, is responsible for managing placement group (PG) in OSDs, where each PG is a group of objects, and several replicas of each PG are located on different OSDs. By default, CRUSH uniformly distributes PGs and their replicas in a cluster of OSDs, and selects a primary OSD for each PG and its replicas. CRUSH also supports a configuration parameter *primary affinity* that controls primary OSDs. The value of primary affinity corresponds to the probability that some OSD is selected as primary by PGs. Once selected, the primary OSD is responsible for managing all requests to the PGs. Given this configurable primary affinity, CRUSH enables rapid load redistribution over the cluster with numerous OSDs, without

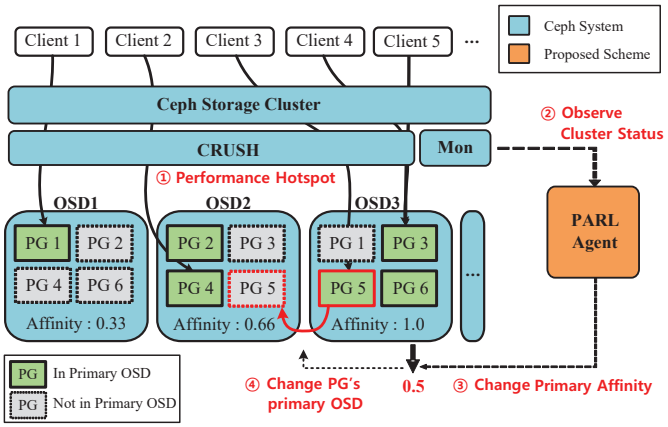


Fig. 2: Overall architecture of PARL-based distributed storage systems. The numbered descriptions in red illustrate the procedure of PARL.

direct data migration. In principle, setting primary affinity values is critical for storage performance, since they control the priority of OSDs that determines which OSD responds to arriving requests [1]. However, optimization techniques for primary affinity have rarely been investigated in CRUSH, and this limitation often incurs performance hotspots and undesirable results on load balancing.

Figure 1 shows the performance patterns of several storage devices, on which the fio benchmark [4] is tested. Those testing devices include (a) Samsung PM1725a with 40G network connectivity, (b) Toshiba RC500 with 40G, (c) Toshiba RC500 with 10G, and (d) Samsung PM981 with 10G. It is observed that the bottleneck point when the served latency increases drastically differs for each device. This experiment result necessities further investigation on performance optimization techniques specific to heterogeneous storage systems.

III. Affinity Control with RL

In this section, we describe a primary affinity control method using deep RL, namely PARL, that addresses the performance hotspot problem. Figure 2 shows the overall architecture with PARL for distributed storage systems.

A. RL formulation

To implement a deep RL method, we formulate a Markov decision process (MDP) with a tuple $(S, A, \mathcal{P}, R, \gamma)$, a state space S, an action space A, a transition probability $\mathcal{P} : S \times A \times S \rightarrow [0, 1]$, a reward function $R : S \times A \rightarrow \mathbb{R}$, and a discount factor $\gamma \in [0, 1]$.

- **State** A state $s \in S$ includes seven features representing the current state of OSDs. For each OSD o, s_t^o comprises the OSD tier (T_t^o), the total amount of I/O requests (IO_t^o), the amount of I/O responses (SIO_t^o), the average required latency of requests (RL_t^o), the average latency of responses (L_t^o), the number of PGs in the OSD which registered as primary OSD ($|PG_t^o|$), and the current primary affinity value (PA_t^o) at time-step t. All state values are normalized between 0 and 1.
- **Action** An action, $a_t \in A$ at time-step t, denotes the target primary affinity value between 0 and 1 for an OSD, where the action size corresponds to the number of OSDs in a system. The OSD with the largest primary affinity distance $|PA_{t-1}^o - PA_t^o|$ only takes the action value.
- **Reward** A reward signal is given by $QoS(\cdot)$ in a target system, e.g.

$$QoS(G,t) = \frac{1}{|G|} \sum_{g \in G} \left(\frac{SIO_t^g}{IO_t^g} + clip\left(1 - \frac{L_t^g - RL_t^g}{2 * RL_t^g}, 0, 1\right)\right) \quad (1)$$

where G is a set of PGs, and $clip(\cdot)$ is defined as

$$clip(x, a, b) = \begin{cases} a, & x \leq a \\ b, & x \geq b \\ x, & otherwise \end{cases}. \quad (2)$$

B. Learning environment for RL

Deep RL algorithms require a large number of data samples to learn the optimal policy. In general, it takes large time to collect a sufficient number of data samples in a storage system environment. Thus, we develop a simulation environment for the CRUSH algorithm, which takes the updated primary affinities from an agent and derives corresponding I/O results according to a given workload. The CRUSH simulation consists of four steps.

(1) An agent with the current state generates an action containing the primary affinity values for OSDs. Only one of OSDs changes its primary affinity value as the action value, which has the largest primary affinity distance. If the primary affinity is decreased, the PGs associated with the OSD are mapped to another OSD by the reduced ratio of the primary affinity.
(2) After the primary affinity setting, I/O operations (with I/O requests and demanded latency) are generated based on the workload trace.
(3) I/O results (with I/O responses and response latency) are calculated from OSDs with the given the primary affinity setting. Each OSD performance is derived from the performance patterns of testing devices as shown in Figure 1.
(4) The QoS of a target system is calculated from the I/O results and used as reward signals.

By iteratively performing the above steps in the CRUSH simulation, training rollouts with states and rewards are obtained to train PARL as illustrated in Algorithm 1.

IV. Evaluation

In this section, we conduct simulation experiments to compare the cluster performance of our PARL and other methods.

A. Experiment Settings

Workloads. We generate an object-based workload by modifying the MSR-Cambridge workload [5] where a real I/O trace with request time, logical address, size, type was recorded from 13 servers of the Microsoft data center for seven

Algorithm 1 PARL

π_θ is an agent parameterized by θ
PA is a set of OSDs' primary affinity value
PG^o is a set of placement group placed in o^{th} OSD
G is a set of placement group in the system
D is a transition buffer, N is a batch size
loop
  $t = 0, D \leftarrow \emptyset$
  while $t < MaxEpisodeLength$ **do**
    $PA_t \leftarrow \pi_\theta(s_{t-1})$
    $o \leftarrow argmax_o(|PA^o_{t-1} - PA^o_t|)$
    while $|PG^o| < |PG^o| \times (PA^o_t / PA^o_{t-1})$ **do**
      random sampling g in PG^o
      $g.setPrimaryOSD(g.getSecondaryOSD())$
    end while
    $w_t \leftarrow$ Workload$[t]$
    $s_t \leftarrow$ Simulation(s_{t-1}, w_t)
    $r_t \leftarrow QoS(G, t)$
    $D \leftarrow D \cup (s_t, a_t, r_t, s_{t+1})$
    if $|D| = N$ **then**
      $Optimize(\theta, D)$, $D \leftarrow \emptyset$
    end if
    $t = t + 1$
  end while
end loop

days. For our experiments, each object is set to a unit of 4 MB logical address and the time-step period is set to be one minute.

TABLE I: Hyper-parameter settings

Hyper-parameter	Our setting
Learning rate	0.00025
Batch size	1024
Network architecture	{128, 256, 256, 128}
Activation function	Leaky Relu
Num of workers	16

Hyper-parameters. We implement our RL method based on Proximal Policy Optimization (PPO) [6], where hyper-parameter settings are listed in Table I. PPO is known to have promising performance on continuous environments such as a cluster storage environment.

Simulation Setting. The default settings of our simulation are listed in Table II. The replicas of PGs are distributed on OSDs with respect to the replication factor.

TABLE II: Default environment settings

Env. parameter	Our setting
The number of tiers	4
The number of OSDs	4
The number of PGs	500
Replication factor of PGs	3
OSD tier ratio	1:1:1:1
Action cycle	1

Compared Methods. For evaluation, we compare our PARL method with several heuristic methods.

- The initial primary OSDs are stationary as the default setting of CRUSH, namely *default*.
- The primary affinity is controlled by the parameters ($iowait$, thresholds) investigated in DLR (Dynamic Load Redistribution) [1].

B. Stability on various OSD tier ratios

To investigate the stability of our method upon various environment settings, we evaluate the cluster performance on various OSD tier ratios of a target system, where the tier ratio is set to 1:1:3:3, 1:3:1:3, 2:2:2:2, 3:1:3:1, and 3:3:1:1, respectively. In addition, the number of PGs and OSDs are set to 1000 and 8, respectively.

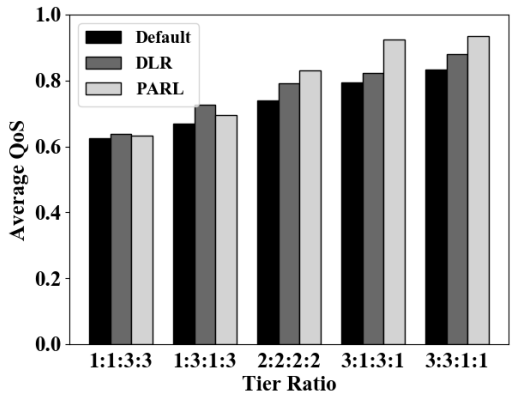


Fig. 3: Cluster performance in various OSD tier ratio settings

Figure 3 represents the stability of PARL and other methods with respect to various OSD tier ratio environments. For all target environments where the x-axis denotes various tier ratio settings, our method (PARL) achieves comparable cluster performance in average QoS in the y-axis over the other methods (Default, DLR). In the high tier enough environments (2:2:2:2, 3:1:3:1, 3:3:1:1), PARL maintains the best performance of up to 10% improvement against DLR. This result indicates that PARL can achieve high storage resource utilization in cluster environments with middle and light workloads.

C. Robustness in Scale

To demonstrate the robustness of our RL method on various system scales, we evaluate the cluster performance with respect to the numbers of OSDs of a system. As the number of OSDs increases to 4, 8, and 12, the number of PGs increases to 500, 1000, and 1500, respectively. Figure 4 shows the robustness of our method in various scales. PARL outperforms the other methods in all cases. For example, PARL achieves better average QoS up to 7.7% than DLR.

D. Action Cycle

Considering that it is non-trivial to guarantee that action commands are timely delivered to a system within each time-step period, we test PARL with respect to various action cycles, as shown in the x-axis in Figure 5. PARL shows performance improvement of 8% regardless of action cycles to

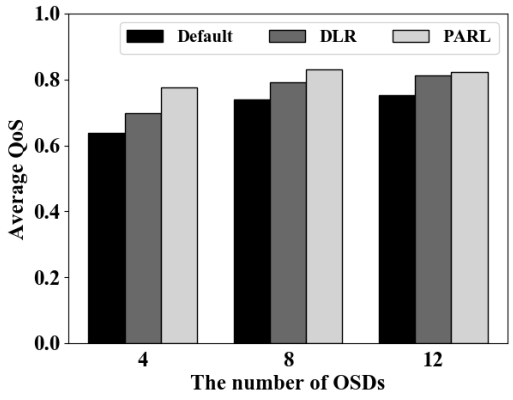


Fig. 4: Cluster performance in various system scales

the other methods. Meanwhile, the performance gain of DLR against Default decreases as the action cycle increases.

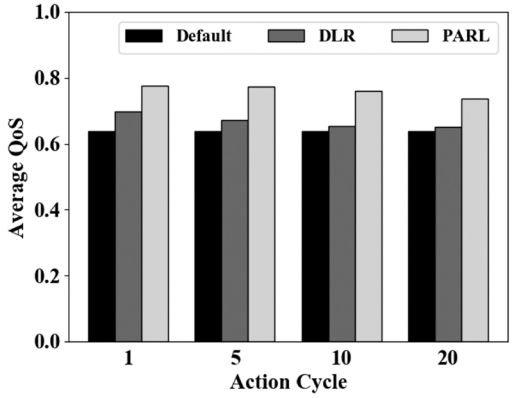


Fig. 5: Cluster performance with respect to action cycle

V. Related Work

Several heuristic-based solutions have been introduced for performance optimization problems in distributed storage systems. For example, Noel *et al.* [1] presented a heuristic mechanism for load balancing by exploiting various system-level performance measurements, and Wu *et al.* [7] developed the SDN-based storage management technique to provide fine-grained granularity for CRUSH weighting factors. Wang *et al.* [8] extended the CRUSH algorithm with an extra time-dimensional map to support stable data migration during the cluster expansion.

Several works have discussed RL approaches for storage optimization problems. Liu *et al.* [9] adopted Q-learning with deep neural networks for data placement scenarios, and Liao *et al.* [10] showed the optimization scheme in the dimension of an input matrix for Q-learning based storage management. Furthermore, there has been an effort to manage Ceph object storage systems with RL. Noel *et al.* [11] presented RL-based handcrafted Ceph actions. While that work has such a limitation that only static environments are considered, our PARL focuses on dynamic environment settings with various aspects of storage devices, tier ratios, and Ceph action cycles.

VI. Conclusion

In this paper, we presented the deep RL-based storage management method by which the primary affinity setting in Ceph can be optimized to establish robust performance upon dynamic storage workloads. Through simulation tests, the proposed method demonstrated its effectiveness and robustness, balancing the I/O loads well across distributed storage devices. A deep RL-based mechanism can be effective in storage applications with various configurable knobs, e.g., the number of replicas or the bucket size. Our future work is to implement an RL-based storage framework running in a real cluster with various knobs.

Acknowledgment

We thank all reviewers for their valuable comments and suggestions. This research was supported by IITP under Grant 2021-0-00900 and by Samsung Electronics. Honguk Woo is the corresponding author.

References

[1] R. R. Noel and P. Lama, "Taming performance hotspots in cloud storage with dynamic load redistribution," in *Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, Honolulu, HI, June 2017, pp. 42–49.

[2] "Ceph primary affinity," [Online]. Available: https://ceph.io/en/news/blog/2014/ceph-primary-affinity/, accessed on: Nov 29, 2021.

[3] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, New York, NY, Nov. 2006, pp. 31–31.

[4] "fio," [Online]. Available: https://linux.die.net/man/1/fio, accessed on: Nov 29, 2021.

[5] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, pp. 1–23, 2008.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[7] D. Wu, Y. Wang, H. Feng, and Y. Huan, "Optimization design and realization of ceph storage system based on software defined network," in *Proceedings of the 2017 13th International Conference on Computational Intelligence and Security (CIS)*, Hong Kong, Hong Kong, Dec. 2017, pp. 277–281.

[8] L. Wang, Y. Zhang, J. Xu, and G. Xue, "Mapx: Controlled data migration in the expansion of decentralized object-based storage systems," in *the 18th USENIX Conference on File and Storage Technologies (FAST '20)*, Santa Clara, CA, Feb. 2020, pp. 1–11.

[9] K. Liu, J. Peng, J. Wang, B. Yu, Z. Liao, Z. Huang, and J. Pan, "A learning-based data placement framework for low latency in data center networks," *IEEE Transactions on Cloud Computing*, pp. 1–12, 2019.

[10] Z. Liao, J. Peng, Y. Chen, J. Zhang, and J. Wang, "A fast q-learning based data storage optimization for low latency in data center networks," *IEEE Access*, vol. 8, pp. 90 630–90 639, 2020.

[11] R. R. Noel, R. Mehra, and P. Lama, "Towards self-managing cloud storage with reinforcement learning," in *Proceedings of 7th IEEE International Conference on Cloud Engineering (IC2E)*, Prague, Czech, June 2019, pp. 34–44.