Autonomous Hierarchical Storage Management via Reinforcement Learning

Tianru Zhang
Supervised by Salman Toor, Andreas Hellander
Uppsala University, Uppsala, Sweden
tianru.zhang@it.uu.se

ABSTRACT

In the present era of big data, the challenges of data management have grown significantly. One crucial aspect is the management of data storage. As data volumes continue to expand, effective storage management becomes increasingly essential. Meanwhile, evolving hardware technologies offer various storage options, ranging from HDDs to SSDs and NVRAMs. To this end, hierarchical (multi-tier) storage systems (HSS) have emerged as a solution, organizing different storage devices hierarchically to provide various storage options. However, managing multiple storage tiers and their data, while optimizing performance and cost-efficiency, is extremely complex. In this paper, we discuss the challenges in the management of hierarchical storage system. We summarise our previous contributions on tackling these challenges, including the proposal of a reinforcement learning (RL) based data migration policy and the design of an autonomous hierarchical storage management framework HSM-RL. We also present the applications of HSM-RL in scientific data management to demonstrate its adaptability and scalability. Finally, we conclude our work to date and outline the future research plans.

VLDB Workshop Reference Format:
Tianru Zhang. Autonomous Hierarchical Storage Management via Reinforcement Learning. VLDB 2024 Workshop: VLDB Ph.D. Workshop.

1 INTRODUCTION

The advances in big data technologies and applications across diverse domains have catalyzed the generation and accumulation of vast amounts of digital information. While these advancements greatly enhance the development of powerful data-driven approaches, they also bring significant challenges in managing large-scale data effectively. Particularly, one essential aspect is storage management, where the primary objective is to achieve a balance between costs, performance, and capacity. To adapt the increasing amount of data, storage systems evolve to scaling horizontally and vertically, resulting in distributed storage (file) systems (DFS) such as HDFS, GPFS, Ceph, and hierarchical storage system (HSS) for instance HP AutoRAID [14], IBM storage hierarchies [3]. While DFS mainly focuses on high data availability, fault-tolerance, and system scalability, HSS arises from the natural structure of storage hierarchy considering the available technologies and their cost and performance characteristics. With the development of storage media, there are now various choices of storage devices ranging from RAMs and SSDs to HDDs and Tapes. Based on their different access speeds and storage capacities, HSS is formed by integrating various storage devices in a multi-tier hierarchical structure. Figure 1 illustrates an example of a three-tier HSS, where higher tiers offer fast read/write (R/W) speed, but are expensive and consequently small in size. Whereas lower tiers are less expensive, larger in capacity, but significantly slower in terms of input/output (I/O).

As the heterogeneity of available storage devices becomes increasingly remarkable, the HSS structure has gained significant attention in the design of recent advanced distributed storage and processing systems including StoRM-GPFS-TSM [1], OctopusFS [7], Hermes [8]. However, efficiently managing data across multiple tiers within HSS presents a highly challenging multi-objective optimization problem due to the diverse characteristics of storage devices and the complex, variable workloads they encounter.

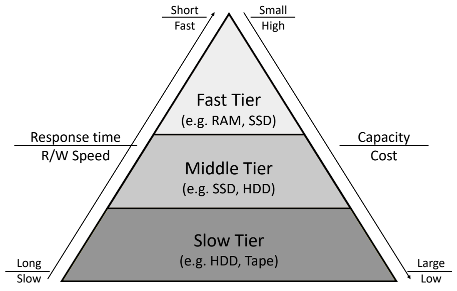

Figure 1: Example of a three-tier HSS, in which various storage devices are connected hierarchically in the system according to their characteristics.

An effective Hierarchical Storage Management (HSM) strategy is essential for balancing costs between infrastructures and end-user demands while maximizing resource utilization and performance. HSM achieves this balance by dynamically prioritizing frequently accessed and important data for placement on faster, more expensive storage tiers, while relegating less critical and infrequently accessed data to slower, low-cost tiers. Given the inherent variability in workloads and the resulting fluctuations in data access patterns [2], a dynamic HSM strategy capable of leveraging latent information to accurately determine data importance and optimal data placement is crucial. Such a strategy enhances performance and ensures cost-efficiency, effectively addressing both infrastructural and user-end requirements. To achieve this, we propose an autonomous HSM solution based on reinforcement learning (RL), which provides an intelligent approach to managing the complexities of HSS.

This thesis explores the utilisation of RL in designing efficient autonomous HSM. We first formulated the HSM problem into a Markov Decision Process (MDP) to bridge the connection between HSM and RL [17]. In the paper we also proposed a RL-based data

Proceedings of the VLDB Endowment. ISSN 2150-8097.

migration policy and a scalable HSM framework built upon the policy (HSM-RL). We testified the proposed approach via experiments on both simulation and cloud-based environments. Subsequently, we zoomed into the scenario of scientific data management [16], where we analysed the unique characteristics of data management in scientific applications and provided corresponding solutions with the HSM-RL framework. Specifically, we presented four different scientific datasets, and dedicated efforts on using the HSM-RL framework to manage them. With the empirical results, we further demonstrated the effectiveness and scalability of the HSM-RL.

The remaining of this paper is structured as follows: Section 2 discusses recent developments in related fields and current limitations; Section 3 elaborates the theoretical base and detailed design of our HSM-RL framework; Section 4 covers the applications in scientific data management and extends the discussion on adaptability and scalability; Section 5 summarises the experimental results from our previous works; Section 6 concludes our contributions and presents the prospects for future research.

2 RELATED WORKS

Early designs of HSM controlled data migration between tiers by employing classical cache replacement policies, for instance LRU, LFU, ARC, MQ, Greedy-Dual-Size [11]. These approaches rely on predefined parameters such as recency, frequency, size, or their mixtures. While the usage of certain parameter(s) make them straightforward and simple to implement, it also weaken their adaptability to changing workloads. Subsequent efforts aimed at incorporating more information from the workloads, for example EXD [5], DUX [9], ReCa [12]. Yet these parameter-driven approaches require manual reconfiguration when the workload changes, and have no balance between performance and cost.

Machine learning and optimization methods were also explored to develop more intelligent policies. K-means clustering was used for data placement in [15]. Classification model XGBoost was applied to decide the upgrade and downgrade of files [6]. Neural Network was also widely used in recent researches, including but not limited to [13]. Evolutionary optimization methods were also explored, such as Discrete Particle Swarm Optimiztion (DPSO) [4]. However, these methods either need extensive supervised pretraining or have slow convergence with changing request patterns.

Another major paradigm of machine learning and optimal control, RL however, has not been extensively explored in HSM. RL focuses on training intelligent agent to take actions in a dynamic environment to maximize cumulative rewards. In fact, RL is highly suitable for HSM, which aims to manage data placement (actions) in HSS (environment) under various workloads (dynamic) to achieve high performance (reward).

3 HSM-RL

In this section, we motivate the usage of RL in HSM and present the detailed design of HSM-RL.

3.1 Reinforcement Learning

RL is a broadly acknowledged decision-making approach aimed at developing intelligent agents capable of learning from environmental interactions to make optimal decisions, adapt to new circumstances, and efficiently achieve defined objectives. RL accomplishes these objectives by solving the Markov Decision Process (MDP), which is an environment formulated as $< S, A, P, R, \gamma >$. Here S is the set of states, A is the set of actions, P is the transition probability matrix, R stands for reward and γ is the discount factor. RL agent solves MDP by learning the best policy ($\pi : S \longrightarrow A$) that determines the most appropriate action based on the current state. This determination of the best action derives from optimizing both the state-value function and action-value function under the chosen policy and action. The agent continuously updates its value functions, learning from the states transitions, enabling it to perform the best action based on the current state and its refined parameters.

3.2 RL-based policy

To address the HSM challenge using RL, we first formulate the HSM problem into an MDP. In this MDP, the states S include variables that represent the status of tiers, while actions A correspond to data migrations between these tiers. The transition probabilities P are assumed to follow a uniform distribution among all potential states. The reward R is defined as the negative of the system response time, and the discount factor γ is considered as a hyperparameter.

Explicitly, with identifying the frequency, recency, size of file as the most influential factors to data migration process, we define the tier-wise state variables $S = \{s_1, s_2, s_3\}$ as follows:

- s_1 : Average temperature of files in a tier, where the temperature (or the hotness level) of a file is an measurement of access frequency and recency[1].
- s_2 : Average weighted temperature, calculated as the average of file size times its temperature, taking into consideration the file temperature as well as the file size.
- s_3 : current queuing time in a tier, indicating the latency level in the tier.

These three state variables are defined in the way that all influential factors are considered. However, the definition of state S is not limited to these three variables, but a flexible design adaptable to different use cases [16].

Subsequently, we define the value function $v_\pi(s)$, which is the expected return starting from state $s \in S$, following the policy π. Since the state variables are continuous, to present the value function we use a functional approximation in the form of the Fuzzy Rule-Based function (FRB). The FRB function maps the input $x \in \mathbb{R}^k$ to a scalar output y by a combination of rules such as *IF* $x_1 \subset C_1^i, x_2 \subset C_2^i, ..., x_k \subset C_k^i$ *THEN* p^i, where $x_1, ..., x_k$ is the components of x, $C_1^i, ..., C_k^i$ are fuzzy categories, and p^i is the output parameter of this rule. The output of the rule-based function is then a weighted average of p^i: $v(\cdot) = y = \frac{\sum_{i=1}^N p^i w^i(x)}{\sum_{i=1}^N w^i(x)}$, where N is the number of rules, and $w^i(x)$ is the weight of rule i.

With the definitions above, we form the migration policy for making decisions of file transfers between tiers i and j in the following form: *file k should be upgraded from tier i to tier j if*

$$v_{\text{up}}^i \cdot \tilde{s_1}^i + v_{\text{up}}^j \cdot \tilde{s_1}^j < v_{\text{not}}^i \cdot s_1^i + v_{\text{not}}^j \cdot s_1{}^j \tag{1}$$

where $v_{\text{up/not}}^{i/j}$ is the value function of tier i/j after/before file k is upgraded; $\tilde{s_1}/s_1$ is the average temperature of all files in tier if file k is upgraded / not upgraded.

[1]For detailed explanation of the file temperature and how it is modeled, refer [16, 17]

As the environment changes, the value function should be updated to adapt to the new circumstance. We use an off-policy method, Temporal Difference learning (TD(λ)). It updates the value function $v(s)$ as follows:

$$\begin{aligned} v(s) &= v(s) + \alpha(R_t + \gamma v(s_{t+1}) - v(s_t))z_t(s), \\ z_t(s) &= \lambda\gamma z_{t-1}(s) + \mathbb{1}(s = s_n) \end{aligned} \tag{2}$$

where α is the learning rate, R_t is the rewards at state s_t at time t, γ is the discount factor, z_n is the eligibility trace, and λ is the trace-decay parameter of TD(λ).

The HSM process with RL-based policy can be abstracted in three steps, as shown in Figure 2. For each file access request, the RL-based policy determines whether the file should be upgraded to faster tier according to Equation 1, and sends migration request to the storage system. Afterwards, the RL agents update their parameters by Equation 2 in the back-end. These processes have also been implemented in a cloud-based environment. For more details about the framework structure and deployment, see [17].

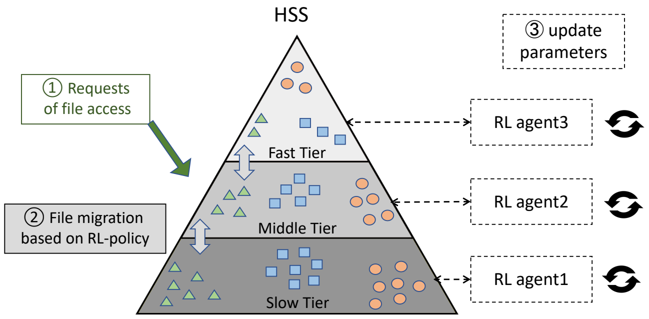

Figure 2: General processes of RL-based policy in HSM of the three-tier HSS.

4 SCIENTIFIC APPLICATIONS

In our first article [17] we introduced a RL-based solution for HSM in file systems, and demonstrated the efficiency and effectiveness through experiments conducted on various regular datasets and workloads in both simulation and real cloud-based environments. However, following interdisciplinary discussions with colleagues, we identified additional untapped potentials and challenging problems, particularly within scientific applications. Scientific datasets often hold unique characteristics, for example in large-scale scientific projects, multiple rounds of experiments are repeated to investigate certain factors while maintaining consistent settings. In these scientific datasets individual objects tend to have a defined identical structure, resulting in uniform file sizes. Whereas unique features related with the properties of applications differ each object. These features also significantly impact access patterns, as on many occasions a specific part of the data is required to perform a specific analysis.

We showcased these special characteristics by presenting four different scientific datasets: the BBBC fluorescence microscopy images dataset, DNA sequences data from 1000 Genomes project, Geometric mesh files generating by Finite Element Method for solving airfoil angel of attack problems, and a phenotypic screening dataset from a cell-based drug repurposing screen [16]. Each of these datasets consists unique features, such as *interestingness value*, *population*, *angel of attack*, *antibody intensity*. Meanwhile, as earlier mentioned the file sizes in each dataset are equal, which makes the file-size-related state variable s_2 ineffective. To tackle these challenges, we introduced new versions of HSM-RL with different variables definition tailored to each use case.

The workloads in each scientific application are also diverse. We presented six different workloads across the four datasets. These workloads exhibit various access patterns, for instance repeated request of files with their specific features valued within a range, batch accesses of certain parts divided by unique features, decreasing number of file requests after gradually applying filters. Additionally, we conducted experiments in a multi-dataset scenario where multiple datasets generated from different groups of settings were simultaneously managed in one HSS. Further details about the experimental settings will be described in the next section, along with discussions on the results.

5 EXPERIMENTAL RESULTS

We first conducted simulation experiments with a synthetic dataset as the proof-of-concept. 20,000 files with varying sizes between 10 KB and 200 MB were comprised, with the total size of the whole dataset to be 20 GB. The access pattern of each file followed Poisson distributions with different expectations. The left side of Figure 3 shows one result from the simulation experiments. We compared three RL-based policy with different initialization against three rule-based policies based on file frequency, recency, and size. To evaluate their efficiency and effectiveness, we measured the average number of file migrations in time intervals and the estimated system response time at certain timestep. Both RL-based policies and the rule-based policies achieved similar system response time (the orange bars), indicating their effectiveness in managing file distributions among tiers according to the workload. However, the rule-based policies resulted nearly ten times more file migrations than the RL-based policies, illustrating the high efficiency of RL-based policy.

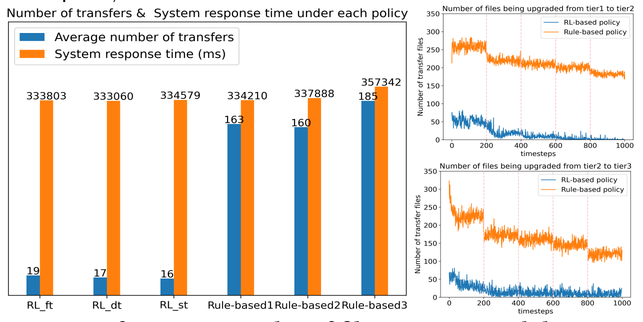

Figure 3: Left: Average number of file migrations and the system response time of experiments under different policies. Right: File migrations amount in bulk deletion experiment

To further demonstrate the adaptability and scalability of the HSM-RL, we carried out more experiments on various scenarios, such as access patterns from different distributions, continuous adding of new files, bulk deletions at certain points. The right side of Figure 3 shows the explicit file migrations number in the experiment of bulk deletion. From the plot it can be observed that the file migration pressure resulted by the RL-based policy is constantly much lower than rule-based policy.

Special use cases such as the scientific applications we discussed in section 4 were also investigated in [16]. The focused research questions include the effectiveness of proposed policy in terms of application response time, the adaptability of RL-based policy with varying numbers of state variables, and the scalability of managing multiple datasets in one system. Table 1 records the average response time per round of file accesses in each experiment of different datasets under diverse policies. The results demonstrate the adaptability of our RL-based policy in effectively managing various workloads and multi-datasets scenarios.

Table 1: Average response time (ms) per round for workloads in each scientific dataset using different policies.

Policy	Experiments on different dataset				
	BBBC	Airfoil	Genomes	Screen	Multi-
LFU	9061	3601	21581	18099	8132
LRU	9056	3534	15150	19431	9704
MF	8906	3523	20129	18062	8970
K-means	8810	3599	21078	18096	8076
RL	**8681**	**3460**	**15006**	**17924**	**7988**

6 CONCLUSION AND PROSPECT

In this paper we discuss the challenges in hierarchical storage management. We propose a reinforcement learning based approach for autonomous HSM, and a scalable framework HSM-RL with flexible design space. We also study the special scenarios in scientific applications, where we show the adaptability of the RL-based approach to different use cases and the scalability of HSM-RL framework in managing multiple datasets. The experimental results further demonstrate the efficiency and effectiveness of our RL-based approach.

For future research plans, we envision two aspects. First we note that modern storage devices offer increasingly fast read/write access, with new technologies like NVMe, PCIe, M.2, Optane. More specifically, modern SSDs differ from traditional HDDs in two major ways: read/write performance asymmetry and access concurrency [10]. Therefore, we are designing a new RL-based policy that is aware of the asymmetry and concurrency of devices in the HSS, in order to fully exploit their I/O performance. Additionally, beyond the general data placement at the file level in file systems, page placement—where page is the smallest unit of data—is also worth investigating, especially in data systems with page-oriented bufferpools. Thus, we extend our RL-based method from file-level data management to page-level data management.

Apart from the storage hierarchy we studied in previous works, we introduce a new concept called Information Hierarchy. While the storage hierarchy leverages access patterns and file properties (such as temperature, size, and other use-case-specific features) along with the hierarchical characteristics of storage devices to enable efficient data management, the Information Hierarchy focuses on utilizing the latent information within the data itself to create the hierarchy. The information hierarchy could not only help users to comprehend their datasets from scratch without the costly efforts from domain experts, but also further guide the management of the storage hierarchy. Currently we are investigating the recent advances in self-supervised and unsupervised learning to explore their potential applications in data management.

ACKNOWLEDGMENTS

This work was supported by the Swedish Foundation for Strategic Research(SSF) under Grant No. BD15-0008. We would also like to acknowledge Swedish National Infrastructure for Computing (SNIC) for providing cloud resources, and support from eSSENCE, a Swedish strategic collaborative research program in e-science.

REFERENCES

[1] A Cavalli, L dell'Agnello, A Ghiselli, D Gregori, L Magnoni, B Martelli, M Mazzucato, A Prosperini, P P Ricci, E Ronchieri, V Sapunenko, V Vagnoni, D Vitlacil, and R Zappi. 2010. StoRM-GPFS-TSM: A new approach to hierarchical storage management for the LHC experiments. *Journal of Physics: Conference Series* 219, 7 (apr 2010), 072030. https://doi.org/10.1088/1742-6596/219/7/072030

[2] Yanpei Chen, Sara Alspaugh, and Randy Katz. 2012. Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads. *Proc. VLDB Endow.* 5, 12 (aug 2012), 1802–1813. https://doi.org/10.14778/2367502.2367519

[3] E. I. Cohen, G. M. King, and J. T. Brady. 1989. Storage hierarchies. *IBM Systems Journal* 28, 1 (1989), 62–76. https://doi.org/10.1147/sj.281.0062

[4] Xin Du, Songtao Tang, Zhihui Lu, Jie Wet, Keke Gai, and Patrick C.K. Hung. 2020. A Novel Data Placement Strategy for Data-Sharing Scientific Workflows in Heterogeneous Edge-Cloud Computing Environments. In *2020 IEEE International Conference on Web Services (ICWS)*. 498–507. https://doi.org/10.1109/ICWS49710.2020.00073

[5] Avrilia Floratou, Nimrod Megiddo, Navneet Potti, Fatma Özcan, Uday Kale, and Jan Schmitz-Hermes. 2016. Adaptive Caching in Big SQL using the HDFS Cache. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '16)*. Association for Computing Machinery, New York, NY, USA, 321–333. https://doi.org/10.1145/2987550.2987553

[6] Herodotos Herodotou and Elena Kakoulli. 2019. Automating distributed tiered storage management in cluster computing. *Proc. VLDB Endow.* 13, 1 (sep 2019), 43–56. https://doi.org/10.14778/3357377.3357381

[7] Elena Kakoulli and Herodotos Herodotou. 2017. OctopusFS: A Distributed File System with Tiered Storage Management. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 65–78. https://doi.org/10.1145/3035918.3064023

[8] Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. 2018. Hermes: a heterogeneous-aware multi-tiered distributed I/O buffering system. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing* (Tempe, Arizona) *(HPDC '18)*. Association for Computing Machinery, New York, NY, USA, 219–230. https://doi.org/10.1145/3208040.3208059

[9] K. R. Krish, Bharti Wadhwa, M. Safdar Iqbal, M. Mustafa Rafique, and Ali R. Butt. 2016. On Efficient Hierarchical Storage for Big Data Processing. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 403–408. https://doi.org/10.1109/CCGrid.2016.61

[10] Tarikul Islam Papon and Manos Athanassoulis. 2021. A Parametric I/O Model for Modern Storage Devices. In *Proceedings of the 17th International Workshop on Data Management on New Hardware* (Virtual Event, China) *(DAMON '21)*. Association for Computing Machinery, New York, NY, USA, Article 2, 11 pages. https://doi.org/10.1145/3465998.3466003

[11] Stefan Podlipnig and Laszlo Böszörmenyi. 2003. A survey of Web cache replacement strategies. *ACM Comput. Surv.* 35, 4 (dec 2003), 374–398. https://doi.org/10.1145/954339.954341

[12] Reza Salkhordeh, Shahriar Ebrahimi, and Hossein Asadi. 2018. ReCA: An Efficient Reconfigurable Cache Architecture for Storage Systems with Online Workload Characterization. *IEEE Transactions on Parallel and Distributed Systems* 29, 7 (2018), 1605–1620. https://doi.org/10.1109/TPDS.2018.2796100

[13] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. 2021. A hierarchical neural model of data prefetching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 861–873. https://doi.org/10.1145/3445814.3446752

[14] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. 1996. The HP AutoRAID hierarchical storage system. *ACM Trans. Comput. Syst.* 14, 1 (feb 1996), 108–136. https://doi.org/10.1145/225535.225539

[15] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. 2010. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems* 26, 8 (2010), 1200–1214. https://doi.org/10.1016/j.future.2010.02.004

[16] Tianru Zhang, Ankit Gupta, María Andreína Francisco Rodríguez, Ola Spjuth, Andreas Hellander, and Salman Toor. 2023. Data management of scientific applications in a reinforcement learning-based hierarchical storage system. *Expert Systems with Applications* 237 (2023), 121443. https://doi.org/10.1016/j.eswa.2023.121443

[17] Tianru Zhang, Andreas Hellander, and Salman Toor. 2022. Efficient Hierarchical Storage Management Empowered by Reinforcement Learning. *IEEE Transactions on Knowledge and Data Engineering* 35, 6 (2022), 5780–5793. https://doi.org/10.1109/TKDE.2022.3176753