ReStore: RL-Based Data Migration for Multi-Tiered Storage

Tarikul Islam Papon, Tianru Zhang, Salman Toor, Manos Athanassoulis

Rise of Big Data. Data generation is growing at an unprecedented rate due to sources like social media, digital transactions, sensors, and the expanding network of Internet of Things (IoT) devices. As a result, a key challenge relates to data storage, retrieval, and maintenance, leading to the need for new scalable storage architectures that balance performance with cost while providing near-optimal performance.

Tiered Storage Architecture for Big Data Management. To address these needs, cost-effective storage solutions have been developed that scale (i) horizontally, resulting in distributed storage (file) systems (DFS) such as GFS, Cassandra, HDFS, Ceph; and (ii) vertically, resulting in the development of tiered storage systems like HP AutoRAID, IBM Storage Tank. Data-intensive applications are increasingly employing *tiered-storage systems* that offer a unified interface to a collection of devices organized in *tiers*, each with varying levels of performance, capacity, and cost similar to the classical memory hierarchy. Generally, the top tier's devices are faster and smaller, where the lower tier devices are slower and larger. However, unlike the classical memory hierarchy, applications can read from any tier directly without always having to push to a higher level [1]. An effective tiered storage management strategy is essential for balancing costs between infrastructures and end-user demands while maximizing resource utilization and performance [2]. It achieves this balance by dynamically prioritizing frequently accessed and important data for placement on faster, more expensive storage devices while relegating less critical and infrequently accessed data to slower, low-cost devices. We now ask the question: *How do modern storage device properties affect the performance of a tiered storage system?*

Storage Devices and Their Properties. Modern tiered storage systems typically incorporate a variety of storage devices like Solid State Drives (SSDs) with variable performance and cost and Hard Disk Drives (HDDs) for slow or archival storage. Since SSDs are the fastest secondary storage device, multiple (or all) tiers of a tiered storage system typically employ SSDs, hence, we focus on how to better exploit these devices. Typically, SSDs exhibit a high degree of internal parallelism (termed *concurrency* in this paper) that can be harnessed to increase performance [3, 4]. On the other hand, for flash-based SSDs, the cost of reading is generally lower than the (amortized) cost of writing, leading to an SSD read/write asymmetry where writes can be up to one order of magnitude slower than reads. These two properties of SSDs are crucial: (i) proper use of SSD concurrency enables better device utilization, and (ii) special care of expensive writes can optimize overall workload execution [5, 6, 7]. Hence, proper performance modeling of SSDs that captures (and exploits) device properties can help tiered storage systems improve resource utilization and performance.

Challenges of Tiered Storage Management. The best strategy for managing a multi-tiered environment depends on data type, access patterns, storage device, and long- and short-term availability of the data. A recurring challenge is the cost associated with the data stores. Data access patterns typically evolve over time, and keeping the data on the fastest storage devices is economically challenging as faster tiers are generally expensive and small. On the contrary, pushing all the data on the slow tiers (slow SSD, HDD, or even an object store) degrades the overall performance of the applications. Therefore, it is essential to have an effective data migration policy in multi-tiered storage that governs when and how data is moved between tiers. Most common migration policies are recency-based, frequency-based, or a hybrid of these two approaches, and all face three major challenges.

Challenge 1: Optimal Data Placement. For many applications the *optimal data placement strategy* depends on the workload. The data placement strategy or migration strategy (across tiers) is *the most crucial design decision* for any multi-tiered storage system. Ideally, we want the *hot* data in the faster tiers while the *cold* data should be in the slower tiers. In case of dynamic workloads, this is not straightforward as data hotness can vary drastically over time. Systems that rely on simple rule-based strategies (e.g., LRU-style or LFU-style policies) for data placement are designed primarily for a simpler problem setting (e.g., two tiers), leading to suboptimal performance when workloads drift.

Challenge 2: Capturing Storage Device Properties. SSD concurrency and asymmetry have not been exploited in tiered storage systems. As an example, there is a 40× increase in the read bandwidth of the 1TB PCIe Intel P4510 SSD when using full concurrency compared to no concurrency and 4KB writes are 3× slower than 4KB reads [4]. Without capturing these storage properties accurately, the devices can remain vastly underutilized.

Challenge 3: Finer Migration Granularity. Existing data migration policies for tiered storage are designed for data files/objects. However, many applications (i.e., DBMS, KV-store) running on tiered storage operate on pages, creating the need for new page migration policies for tiered storage systems. To work on such finer granularity, data migration policies must be *lightweight* while capturing workload and device properties.

Overall, existing tiered storage systems rely on predefined rules (e.g., recency-based, frequency-based) for data files/object

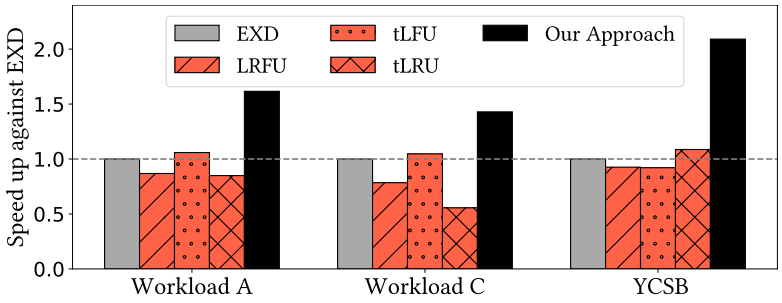

Figure 1: State-of-the-art policies like EXD [8] and LRFU [9] that rely on a utility function to capture workload features, ignore device properties, and struggle with workload drift. tLRU and tLFU that adapt the LRU and LFU caching policies for migration capture access statistics, but are slow to adapt to drift and do not consider device properties. Our approach captures workload features and device properties and quickly adapts to workload drift. From the three different workloads, A is more skewed, and YCSB has more drift. Our approach always outperforms the state of the art by up to 2.2×.

placement and migration, which do not work well for page-level migration and do not adapt well to changing workload patterns and access frequencies. Further, *rule-based approaches do not interact with the system and, thus, do not take into account device properties, rather, rely on basic heuristics that lead to suboptimal data placement.*

Reinforcement Learning to the Rescue. Reinforcement Learning (RL) uses intelligent *agents* to make decisions by *interacting* with the environment and receiving feedback in the form of rewards or penalties aiming to maximize a cumulative reward. The RL formulation typically represents the state of the modeled system, and, thus, is well-suited for learning how to migrate data in tiered storage systems, as it can adapt to changing workloads and an evolving device state to optimize data placement. Further, an RL-based data migration approach can incorporate new storage technologies as they emerge, providing a future-proof solution.

`ReStore`: An RL-Based Tiered Storage Migration Policy. To address the above challenges, we propose `ReStore`, an RL-based multi-tiered storage migration policy, which can be integrated into any multi-tiered storage system that allows for **page-level** accesses using multiple storage devices. As part of the RL formulation, we define state variables that capture the workload features (recency and frequency) and device properties (concurrency and asymmetry), and use a reward function tied to system runtime. The policy is then optimized by maximizing the accumulated rewards through value functions that quantify the quality of the current system state, which in turn minimizes workload execution cost. To ensure adaptability, we employ Temporal Difference learning to update the value functions dynamically. A prime benefit of `ReStore` is that its RL-based data migration gets feedback from the system through interaction and adapts accordingly. Compared to state-of-the-art approaches, `ReStore` is the only policy that considers device properties and responds well to dynamic workloads.

We compare `ReStore` with different rule-based policies, highlighting that while both approaches can achieve similar data placement, the RL approach accomplishes this with significantly fewer data migrations, maximizing efficiency and reducing cost. This is because `ReStore` (i) takes into account storage device properties like concurrency and asymmetry to ensure optimal device utilization, and (ii) balances the tradeoff between cost, performance, and resource utilization. Figure 1 compares `ReStore` with other baseline migration policies in case of workload drift. The figure shows that `ReStore` achieves up to 2.2× speedup compared to other baselines for changing workloads, showcasing `ReStore`'s flexibility to adapt to dynamic workloads.

References

[1] H. Herodotou and E. Kakoulli. "Automating Distributed Tiered Storage Management in Cluster Computing," *VLDB*, 2019.

[2] T. Zhang. "Autonomous Hierarchical Storage Management via Reinforcement Learning," *VLDB PhD Workshop*, 2024.

[3] T. I. Papon, and M. Athanassoulis. "The Need for a New I/O Model," *CIDR*, 2021.

[4] T. I. Papon, and M. Athanassoulis, "A Parametric I/O Model for Modern Storage Devices," *DAMON*, 2021.

[5] T. I. Papon, and M. Athanassoulis, "ACEing the Bufferpool Management Paradigm for Modern Storage Devices," *ICDE*, 2023.

[6] T. I. Papon, T. Chen, S. Zhang and M. Athanassoulis, "CAVE:Concurrency-Aware Graph Processing on SSDs," *PACMMOD*, 2024.

[7] T. I. Papon, "Enhancing Data Systems Performance by Exploiting SSD Concurrency & Asymmetry," *ICDE PhD Symposium*, 2024.

[8] A. Floratou, N. Megiddo, N. Potti, F. Özcan, U. Kale, and J. Schmitz-Hermes, "Adaptive Caching in Big SQL using the HDFS Cache," *SoCC*, 2016.

[9] S. Min, D. Lee, C. Kim, J. Choi, J. Kim, Y. Cho, and S. Noh. 2001, "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," *IEEE Trans. Comput.*, 2001.