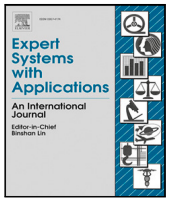
Expert Systems With Applications 237 (2024) 121443

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Data management of scientific applications in a reinforcement learning-based hierarchical storage system ☆

Tianru Zhang [a,*], Ankit Gupta [a], María Andreína Francisco Rodríguez [a,b], Ola Spjuth [b], Andreas Hellander [a], Salman Toor [a]

[a] Department of Information Technology, Uppsala University, 751 05, Uppsala, Sweden
[b] Department of Pharmaceutical Biosciences, Uppsala University, 751 24, Uppsala, Sweden

ARTICLE INFO

Dataset link: https://github.com/JSFRi/MSD-RLHSS.git

Keywords:
Data management
Scientific application
Hierarchical storage system
Reinforcement learning
Large scientific datasets

ABSTRACT

In many areas of data-driven science, large datasets are generated where the individual data objects are images, matrices, or otherwise have a clear structure. However, these objects can be information-sparse, and a challenge is to efficiently find and work with the most interesting data as early as possible in an analysis pipeline. We have recently proposed a new model for big data management where the internal structure and information of the data are associated with each data object (as opposed to simple metadata). There is then an opportunity for comprehensive data management solutions to account for data-specific internal structure as well as access patterns. In this article, we explore this idea together with our recently proposed hierarchical storage management framework that uses reinforcement learning (RL) for autonomous and dynamic data placement in different tiers in a storage hierarchy. Our case-study is based on four scientific datasets: Protein translocation microscopy images, Airfoil angle of attack meshes, 1000 Genomes sequences, and Phenotypic screening images. The presented results highlight that our framework is optimal and can quickly adapt to new data access requirements. It overall reduces the data processing time, and the proposed autonomous data placement is superior compared to any static or semi-static data placement policies.

1. Introduction

Recent advancements in the field of large-scale data management range from efficient interconnected storage devices to smart algorithms for efficient management at scale (Ikegwu, Nweke, Anikwe, Alo, & Okonkwo, 2022; Oussous, Benjelloun, Ait Lahcen, & Belfkih, 2018). These advancements have significantly enhanced storage capacity, management, and high-availability of large datasets, forming the basis of the big data revolution. Even so, the continuous generation of new datasets brings new requirements that need further research and development. These new requirements not only bring conventional challenges to a much larger scale but also add new challenges related to recently developed solutions. Conventional challenges include efficiency, scalability, and high throughput. New emerging challenges include data management based on different storage types (volume-based or object-based solutions), pay-as-you-go models offered by service providers, and solutions that are compliant with new data privacy and security regulations. It is evident that these multifold challenges require careful thinking and expertise from different disciplines.

Efforts to address these challenges include horizontal solutions, where different storage solutions are tightly or loosely connected with each other and offer transparent access to the available data. On the other side, hierarchical solutions are also available where different frameworks with varying capabilities manage datasets, known as vertical solutions. Both approaches are valid, however, static placement of datasets is a current limitation with both horizontal and vertical solutions.

The concept of hierarchical storage is the overarching approach of the herein presented solutions. The main idea is to connect different independent storage solutions and move the data between them

☆ This document is the results of the research project funded by the Swedish Foundation for Strategic Research.

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author at: Room POL 106281 hus 10, Lägerhyddsvägen 1, 751 05 Uppsala, Sweden.
E-mail addresses: tianru.zhang@it.uu.se (T. Zhang), ankit.gupta@it.uu.se (A. Gupta), maria.andreina.francisco@farmbio.uu.se (M.A.F. Rodríguez), ola.spjuth@farmbio.uu.se (O. Spjuth), andreas.hellander@it.uu.se (A. Hellander), salman.toor@it.uu.se (S. Toor).
URL: https://www.it.uu.se/katalog/tiazh991 (T. Zhang).

https://doi.org/10.1016/j.eswa.2023.121443
Received 21 March 2023; Received in revised form 1 September 2023; Accepted 1 September 2023
Available online 14 September 2023

according to criteria designed to meet a set of requirements. The hierarchies based on different underlying storage solutions ensure rational allocation of resources and therefore solve the problem of inefficiency due to imbalanced resource usage in traditional storage solutions. However, the actual realization of the concept is a non-trivial task. In one of our recent articles (Zhang, Hellander, & Toor, 2022), we have comprehensively described the underlying challenges related to storage hierarchies, and presented a solution based on reinforcement learning (RL). We have also built a simulation and a fully functional cloud-based framework for general purpose datasets. However, scientific datasets usually hold unique characteristics that requires further attention to better manage and analyze the datasets. This article is a dedicated effort to highlight the challenges related to scientific datasets and how a RL-based hierarchical storage solution can better address those challenges.

Different datasets have different characteristics that play important roles in large-scale data management. These characteristics include total size, individual object size, access patterns, regulatory requirements, etc. Scientific datasets are unique as the individual objects can be information-spare, such as in the case of automated microscopy. The information-sparsity plays an important role in the efficient data analysis and the sparsity varies with the kind of analysis under consideration. Faster access to the most relevant data would significantly improve the efficiency of analysis. For example, HASTE project (Blamey et al., 2021) is a unique effort to address challenges related to scientific datasets. The HASTE project takes a hierarchical approach to acquisition, analysis, and interpretation of image data. In this article, we focus on highlighting the unique characteristics of the scientific datasets, capitalize the underlying structure to efficiently build the storage hierarchies, and reduce the time and cost associated with the analysis pipelines. We have called those special characteristics the *interestingness values*, which can be a single value, a vector or a matrix that highlights the importance or unique characteristics of an individual object in a dataset.

Approaches based on both supervised and unsupervised learning have shown promising results to achieve efficient management of large datasets (Xiao et al., 2018). Our proposed framework based on storage hierarchies uses the reinforcement learning (RL) method for autonomous data placement in different tiers in the hierarchies. Reinforcement learning is an established branch of artificial intelligence where agents learn from the dynamics of the environment and adapt according to emerging needs. We have used RL agents to learn both about varying data access patterns and from the internal properties of the datasets. The framework helps in efficient data placement with minimal resource utilization as well as overall reducing the application's execution time.

In this article, our focus is to study the framework's capabilities to efficiently and autonomously manage scientific datasets and improve the analysis time by providing faster access to the datasets. Our results clearly illustrate that once the data is available in the system, the framework efficiently organizes the active/required subset of the data to have faster access while keeping the inactive data on the slower or less expensive tiers in the hierarchy. The framework achieves this behavior autonomously and also adapts accordingly to new emerging data access requirements. This makes it uniquely suitable for exploratory scientific analyses.

The remainder of the article is organized as follows. Section 2 presents the related works. Section 3 details storage hierarchies, migration policies and underlying mathematical foundations. To illustrate the utility of the proposed framework, Section 5 highlights four distinct scientific datasets and describes their characteristics and challenges of managing them efficiently. In Section 6 we conduct experiments based on the described datasets using six different policies. Finally, in Section 7 we conclude the work and comment on future directions.

2. Related work

One of the early ideas to address challenges related to data management in hierarchical storage is using cache replacement policies (Acharya, Alonso, Franklin, & Zdonik, 1996). Cache replacement policies are optimization algorithms targeting at managing the cache of information stored on the computer to achieve the best performance within a given limited memory space. These policies include simple random placement (that places data randomly). There were also Recency-based policies like Least Recently Used replacement (LRU-K) (O'Neil, O'Neil, & Weikum, 1993), which replaces the least recently used data by new data that are more important. Improved Recency-based policies such as Low Inter-reference Recency Set (LIRS) (Jiang & Zhang, 2002) were introduced later. Frequency-based policies were another group of policies that based on frequency, for example Least Frequently Used replacement (LFU) (Lee et al., 2001a). LFU replaces the least recently used data by new data that are more important. Hybrid policies that combine the benefits of LRU and LFU were also been explored. Examples include Least Frequent Recently Used replacement (LFRU) (Lee et al., 2001b), Adaptive Replacement Cache (ARC) (Megiddo & Modha, 2003), Multi-Queue replacement (MQ) (Zhou, Philbin, & Li, 2001). Data management policy for hierarchical storage system can be defined by adapting from these cache replacement policies. It is determined by combining the replacement policies with the hierarchy ranks information, and formed by considering the faster tiers as caches, and replacing old files in faster tier by new files according to the replacement policies (Brubeck & Rowe, 1996; Krish, Wadhwa, Iqbal, Rafique, & Butt, 2016; Sienknecht, Friedrich, Martinka, & Friedenbach, 1994).

Apart from using the cache replacement algorithms, there are other efforts bringing up data placement policies using evolutionary optimization methods. These policies include Discrete Particle Swarm Optimization with Genetic Algorithm operators (GA-DPSO) (Lin et al., 2019). The authors introduced the discrete solution DPSO based on the evolutionary computation technique Particle Swarm Optimization (PSO), and included a preprocessing method to optimize the structure of workflows to effectively compress the number of datasets and improve the execution efficiency of GA-DPSO. Another similar approach is Discrete Particle Swarm Optimization algorithm with Differential Evolution (DE-DPSO-DPS/DPA) (Du et al., 2020). Differential evolution (Storn & Price, 1995) is used instead of a genetic algorithm, motivated by differential evolution being demonstrated to be a more efficient algorithm in numerical multi-objective optimization (Tusar & Filipic, 2007).

Another research direction for large-scale data management focuses on the internal structure of the datasets. These methods include approaches such as (Yuan, Yang, Liu, & Chen, 2010), where they introduced a data placement strategy based on K-means and Bond Energy Algorithm (BEA) clustering that was able to reduce the number of data movements. Another method based on K-means was provided in Wang, Zhang, Dong, and Luo (2014). In addition, they also considered the data size and dependency relationships between datasets and tasks. Recent approach named LDM (Lineage-Aware Data Management) (Mishra & Somani, 2020) took into consideration the lineage property, i.e. in which the current writes are future reads. With trace-driven experiments.

Alongside hierarchical data management policies, efforts focusing on horizontal management of scientific data have also contributed significantly. In large-scale scientific projects such as Large Hadron Collider (LHC) (LHC, 2008) and Square Kilometre Array (SKA) (SKA, 2019), storage and computing resources are usually heterogeneous and distributed at various geographical locations belonging to different administrative domains and organizations. Such projects generate massive datasets and require complex workflows for multi-step analyses. To address these challenges, several frameworks have been introduced, such as the eXtreme DataCloud (XDC) project (Cesini et al., 2020). The project aimed at developing scalable technologies for federating

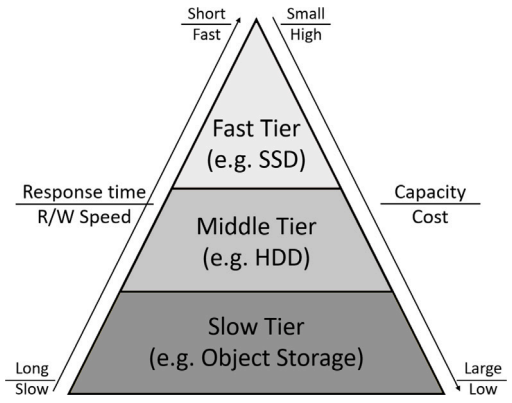


Fig. 1. Example of a three-tiers Hierarchical Storage System. Higher tiers have faster response time and higher cost. Whereas lower tiers are slower, larger in capacity, and less expensive in comparison with the higher tiers.

storage resources. The presented framework manages data coming from a range of scientific disciplines including: life science, biodiversity, clinical Research, astrophysics, high energy physics and photon science. Another example is Rucio by Barisits et al. (2019), a software framework that provides functionalities to organize, manage, and access data at scale. Their presented results specifically focus on the management of scientific datasets.

While all of the above-mentioned efforts achieved good performance in their settings, it is important to jointly consider the efficiency, self-adaptability, and customizability. For example, some methods work well under one request pattern, but as soon as the request pattern changes they have difficulty in adapting to the new pattern. Other methods might be able to adapt quickly, but with a high cost in data transfers between storage tiers. In the following sections, we introduce our reinforcement learning based hierarchical storage management framework. We also test its performance with real-world scientific datasets and corresponding analysis pipelines. Meanwhile, we have also compared proposed RL-based policy with commonly known relevant policies including random placement, LRU replacement, LFU replacement, K-means policy, and a prefixed minimal/maximal feature policy.

3. Methods

3.1. Storage hierarchies

Hierarchical Storage System (HSS) (Wilkes, Golding, Staelin, & Sullivan, 1996), is also called as multi-tier storage system. HSS solutions connect multiple storage tiers in a hierarchical structure. Each tier in a HSS is an allocated storage medium based on a software framework, for instance cache memories, hard disks, object storage, etc. The mediums are placed in the hierarchical structure according to their respective features. The list of features generally includes size, speed, efficiency, cost, security, and resilience. This list may further include custom features specific to the underlying mediums' components and hardware in use. The aim of a HSS architecture is to manage the combination of different mediums in a systematic and coherent manner.

Fig. 1 shows an example of a three-tiers HSS. The hierarchy is defined by the access speed and storage capacity. Tiers in high rank have fast read/write (R/W) speed, but are expensive and consequently small in size. Whereas lower tiers are less expensive and significantly larger in size, but are slow in terms of input/output (I/O). This is mainly due to the internal architecture of the frameworks and the underlying hardware used to build the solutions (e.g. Solid State Drives (SSDs), Hard Disk Drives (HDDs) or Tape Drives) . A higher rank tier (fast tier) is used to store important and frequently requested data. Less important or less frequently accessed data are stored in slower tiers. To reveal the full benefits of the hierarchical structure, a HSS should be able to automatically move data between fast and slow storage tiers. Therefore, autonomous, online, and efficient data placement strategies to optimally utilize the available resources in different tiers are essential.

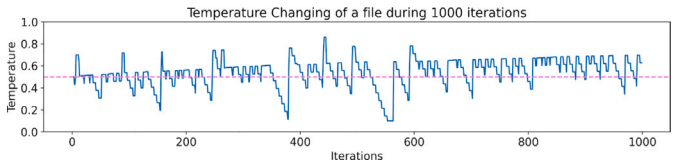


Fig. 2. Temperature changing of a file with 100 random requests during 1000 iterations.

3.2. Data migration policy

In HSS, a systematic way to decide which file should be stored in which tier is crucial. For example, if a file is being accessed frequently, then it should be stored in a fast tier to reduce the response time. Frequently accessed files are called *hot* files, in contrast, files that are rarely accessed are labeled as *cold* files. The access frequency of a file is called *hotness-level* and it is measured by the file temperature, which takes values between 0 and 1. Where 0 is the *hotness-level* for *coldest* files and 1 for *hottest* files.

The changing mechanism of the file temperature is important and needs to be defined precisely, since it reflects the access frequency and priority of files. In order to present the temperature in a mathematical way, we defined the rules of file temperature changing by two part, increasing and decreasing. The detailed expressions are as follow:

- $Temperature\ increasing$: File temperature will increase when there are more access requests to this file. We describe this process using an exponential formula: $temperature = 1 - 0.5/exp(0.2 * num\_req)$, where num_req is the total number of requests to the file.
- $Temperature\ decreasing$: When a file has not been requested for a period, the temperature of this file should drop because the importance and priority of this file has become lower. Therefore, we define the temperature decreasing rule to be: *After 5 iterations without requests, file temperature will decrease 0.05, until it become 0.0 (the lowest temperature)*. As long as new request come to a file, its temperature will start increasing, following the temperature increasing rule above.

With this mechanism, we define the file temperatures changing dynamics. Fig. 2 shows an example a temperature changing process of a file with 100 random requests in 1000 iterations.

As the file properties (which in general include *hotness-level*, size, type, and use-case dependent properties) change, files will be moved between tiers. For instance, if one file is currently stored in slow tier but has recently been requested frequently, it becomes a hot file and should be moved to faster tier. On the contrary, for file in fast tier but has not been requested for some time, then this file will be counted as a *cold* file and moved to a slow tier in order to free up space for *hot* files. These file movements are in general called migrations.

The process of file migrations consumes a significant amount of compute and network resources. Thus a policy that can control these file migrations in an efficient way is important. Previous work on well-defined policies include Least Recently Used Replacement (LRU) (O'Neil et al., 1993), Least Frequently Used Replacement (LFU) (Lee et al., 2001a) and Size-Temperature Replacement (STR) (Zhang et al., 2022), and so on. However, these policies are mostly static and require strict initial assumptions. However, data placement under changing access patterns is highly dynamic, and it is increasingly challenging with the growing amount of data in the system. To address the need for

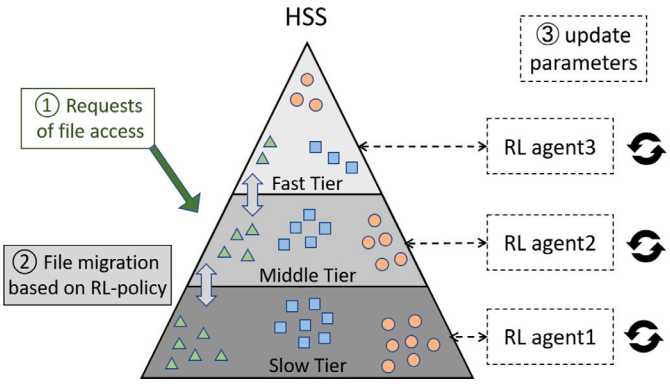


Fig. 3. A three-tier HSS with RL-based policy, and general processes in the system.

autonomous data migration between tiers and efficient utilization of available resources, we have designed and developed an online data migration policy that adapts according to the incoming file access pattern, migrates data between tiers and consumes minimal system resources. The following subsection presents the mathematical model used to architect the online data migration policy.

3.3. Reinforcement learning based policy

Recently, we have developed a hierarchical storage management framework using reinforcement learning (HSM-RL) (Zhang et al., 2022). The framework contains reinforcement learning (RL) agents served for each tier, and a data migration policy based on the tier information variables and the RL agents. In comparison to the previous study, the unique contributions of this article are highlighted at the end of Section 1. Fig. 3 shows a summarized structure of the HSM-RL framework. Step ①, ②, ③ briefly describe how the system deals with file requests, how the files are migrated between each tier, and how the RL agents update themselves according to the requests and new file distributions.

More in detail, the RL agents output data migration actions based on the cost value function and the state variables. Once a file F is requested, if it is currently not placed in the fast tier, then the framework will use a policy to decide whether this file should be upgraded to the faster tier or not. The RL-based policy is: file F now in tier i will be upgraded to tier $i+1$ if

$$v^i_{up} \cdot \bar{t}^i_{up} + v^{i+1}_{up} \cdot \bar{t}^{i+1}_{up} < v^i_{not} \cdot \bar{t}^i_{not} + v^{i+1}_{not} \cdot \bar{t}^{i+1}_{not} \tag{1}$$

where v_{up} is the cost value function of tier $i/i+1$ after file F is upgraded, and v_{not} is before upgrade. $\bar{t}_{up}$ is the average temperature of all files in tier if file F is upgraded, and $\bar{t}_{not}$ is the same variable before upgrade. As if the upper tier $i+1$ does not have enough space for upgrading file F, files in tier $i+1$ with the lowest temperature will be downgraded to leave enough space for file F.

In Eq. (1), the cost value function v is fundamentally the state-value function $v_\pi(s)$ of Markov Decision Processes (MDPs), which is the general aim of RL to solve. It is a function of the state s and represents the expected return starting from state s to the next state s' following policy π. For MDPs with finite discrete states, this state-value can be represented by the expectation matrix of the transition matrix. However, for MDPs with continuous states, it is impossible to form a matrix for infinite states. To solve the problem, functional approximation is commonly used (Sanghi, 2021).

In our method, we use a Fuzzy Rule Based (FRB) function for the approximation. FRB function is a mapping from an input $s \in \mathbb{R}^k$ to an output $y \in \mathbb{R}$ in the form of a weighted average of the outputs p^i of all the fuzzy rules: Rule i: IF $s_1 \subset A^i_1, s_2 \subset A^i_2, \ldots, s_k \subset A^i_k$ THEN p^i, where $A^i_1, \ldots, A^i_k$ are fuzzy categories, which the input $s_1, \ldots, s_k$ may belong to. The output y is then a weighted average of p^i, and defines the approximation of value function $\hat{v}$:

$$y = \frac{\sum_{i=1}^N p^i w^i(s)}{\sum_{i=1}^N w^i(s)} = \hat{v}(s) \tag{2}$$

where p^i is the output parameter of rule i, N is the number of rules, $w^i(x)$ is the weight of rule i computing by $w^i(x) = \prod_{j=1}^k \mu_{A^i_j}(x_j)$, and $\mu_{A^i_j}(x_j)$ is the membership function measures how much does an input x_j belong to a category A^i_j. Two fuzzy categories $\{Small, Large\}$ are used here, and $\mu_{Large}(x_j) = 1/(1 + a_j e^{-b_j x_j})$, $\mu_{Small}(x_j) = 1 - \mu_{Large}(x_j)$, where a_j, b_j are the hyperparameters.

For the state s, it is important to use state variables that can strongly present all the properties of the dataset and the access patterns. More precisely, given unique characteristics in scientific datasets such as *interestingness value*, state variables $s\_$ requires special definition. In Section 6, we will further show details of the usage of different state variables for different datasets.

The above paragraphs described the design of the migration policy based on RL agents and its value functions. However, since storage systems always receive new requests, the file distributions in each tier are not static. For the RL agents, this indicates a change of the environment, thus it is necessary for RL agents to update themselves in order to maintain the effectiveness of the migration policy. To update the RL agents, it is equivalent to update the cost value function $v(s)$. We uses a widely-used and well-proved way of updating state-value function, the TD(λ) algorithm. The updating process can be expressed by the following formula:

$$\begin{aligned} &\hat{v}(s) = \hat{v}(s) + \alpha(R_n + \gamma \hat{v}(s_{n+1}) - \hat{v}(s_n))E_n(s), \\ &R_n = \frac{1}{X_n}\sum_{i=1}^{X_n} r_i e^{-(t_{n,i}-t_n)}, E_n(s) = \lambda\gamma E_{n-1}(s) + \mathbb{1}(s = s_n) \end{aligned} \tag{3}$$

where $\hat{v}(s)$ is the approximation of $v(s)$, α is the learning rate, R_n is the rewards at state s_n, X_n is the total number of requests in state s_n, r_i is the response time of each request, $t_{n,i}$ is the arrival time of request i, and t_n is the time arrived at state s_n, γ is the discounting factor, and E_n is the eligibility trace, which is initialized to 0 and updated by the formula above and λ is the trace-decay parameter of TD(λ). The convergence of (3) is given by the convergence of TD(λ) algorithm, which has been proved in Dayan (1992).

This section established the theoretical basis of our HSM-RL framework, previous experiments (Zhang et al., 2022) also showed the feasibility of the framework in data management of general datasets. However, as mentioned above, previous experiments only demonstrated the efficiency of the framework in single and general dataset. Therefore, in the following sections we will present experiments on specific scientific datasets, together with multiple datasets in one system. Accordingly modified setups are also introduced in order to verify the effectiveness and the adjustability of HSM-RL in scientific datasets.

4. Focused research questions

In our previous study, we demonstrated through both simulations and real-world experiments that hierarchical storage solutions are critical for developing next-generation, efficient, and cost-effective data management frameworks (Zhang et al., 2022). The key to enabling such frameworks is to implement a smart data placement policy. We proposed an RL-based policy and compared it with several other data placement policies. Our results showed that the RL-based policy required significantly fewer transfers between tiers than contemporary policies, as presented in the left part of Fig. 4. However, we also noted that while our proposed RL-based policy outperformed other policies in terms of the number of transfers required, other policies eventually achieve similar file distributions with significantly more work, as shown in the right part of Fig. 4.

This raises an essential question:

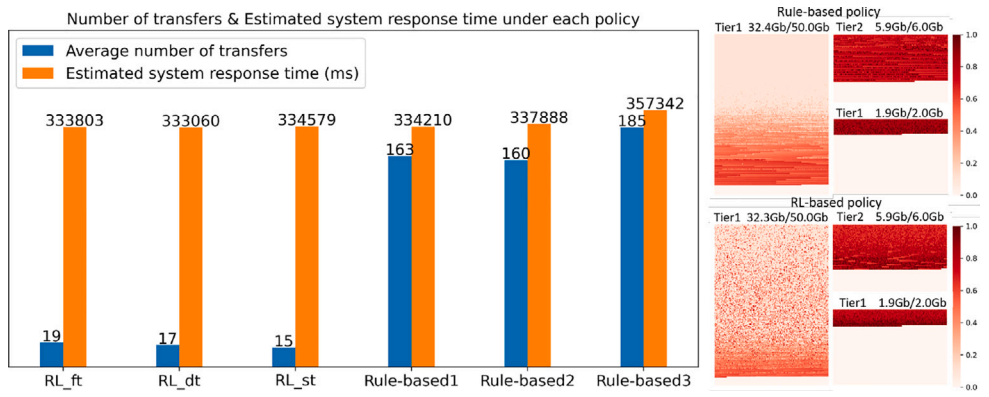


Fig. 4. Left: Number of transfers and estimated system response time under each policy, in cloud based experiment. Right: File temperature distribution on the cloud distributed system with dynamic dataset.
Source: Figure from (Zhang et al., 2022).

- How effective is the proposed policy in terms of application response time, given that application response time is highly dependent on file distribution?

Moreover, it is crucial to understand:

- How will the proposed RL-based policy perform when there is a varying number of state-space variables in the framework for highly challenging access patterns? Particularly in scientific applications.
- Finally, how will the framework manage file distributions when using multiple datasets from different use cases?

These questions require extensive framework evaluation and are the focus of this article. We conducted six different experiments using four different datasets to highlight the usefulness of hierarchical storage for a range of challenging applications. The following sections provide detailed information about the datasets and experiment settings. Our findings indicate that the RL-based data migration policy in a hierarchical storage management solution offers a better or comparable response time compared to contemporary policies. The RL-based policy's adaptive behavior towards unseen access patterns makes it superior to its counterparts.

5. Scientific datasets

This study focuses on scientific datasets. Scientific datasets tend to be big datasets where large experiments require Petascale or Exascale storage solutions (Elworth et al., 2020; Scaife, 2020). An individual object in a scientific dataset tend to be larger than in conventional datasets and it also often has a defined structure. Examples include images of varying sizes (one megabyte to several hundred megabytes), matrices based on integers, floats or double precision numbers, or very long text files consisting of DNA sequences. Previously, efforts like ROOT (Antcheva et al., 2011), HDF5 (Soumagne et al., 2022) and similar file formats have been used to exploit the underlying structure to improve the overall application's performance. These successful efforts clearly highlight that scientific datasets need special attention and the performance of the scientific applications can be significantly improved by exploiting the underlying structure in the datasets. Another major difference from conventional datasets is the access pattern. In scientific datasets, on many occasions, a specific part of the data is required to perform a specific analysis. If the subset of the data is available on fast storage, the analysis can be finished much faster. However, identifying the required subset in advance, with manual placement on faster storage requires significant effort. Also, this same effort will be required every time there is a change in the analysis. The proposed hierarchical storage framework presents a unique data management solution that addresses the above-mentioned challenges at scale. It uses the internal properties of the dataset as well as the incoming request pattern to autonomously place data in storage hierarchies. To the best of our knowledge, so far, the internal properties of the data have not been used for the data placement strategy. In the following sections, we describe the datasets used for our case-study, and detailed experimental settings for each case.

5.1. Protein translocation dataset

The first dataset consists of fluorescence microscopy images from the publicly available Broad Bioimage Benchmark Collection (BBBC) (Ljosa, Sokolnicki, & Carpenter, 2012). The images contain varying doses of two drugs in human U2OS cells that are grown in a 96-well plate. The images showcase the translocation of protein tagged with green fluorescent protein (GFP) from the cytoplasm to the nucleus of the cells. As the drug dose increases, the GFP expressed in the cytoplasm decreases and GFP expressed in the nuclei increases. The quantification of the cells with GFP expression in the nuclei or cytoplasm with respect to the drug dosage is important in understanding effects of the drug.

Here, SimSearch (Gupta, Sabirsh, Wählby, & Sintorn, 2022) is used to generate the counts of the cells with three categories: "GFP in cytoplasm", "GFP in nuclei", and "No GFP expression". SimSearch is a deep learning based ROI (Region Of Interest) detection framework for quickly annotating microscopy dataset. We increased the size of the dataset to 1056 images by rotating, flipping, and transposing the original images. The counts in different categories remains the same with the operations mentioned above. The example images with the annotated cells are shown in Fig. 5. In the experiments, we used the counts of different categories as the *interestingness value* of the image. The *interestingness value* is a measurement that defines the importance of an individual object/image in the dataset.

In order to discover the effects of given drugs, analyses to inspect images with high *interestingness value* are usually conducted. The idea is to check the high GFP expression area, which can be represented by high *interestingness value*. Therefore, an autonomous data management policy will significantly increase the efficiency. In Section 6 we will further discuss the experimental details.

5.2. Airfoil angle of attack dataset

In the field of aerodynamics, the design of an airfoil is crucial. It helps determine important properties for example the lift and the drag forces. These forces can be calculated by the velocity and pressure fields of the air passing through the wing, which are described by the Navier–Stokes equations. To solve the equations, researchers (Nazarov, 2011) wrote a $2D$ Navier–Stokes solver based on the Finite Element Method.

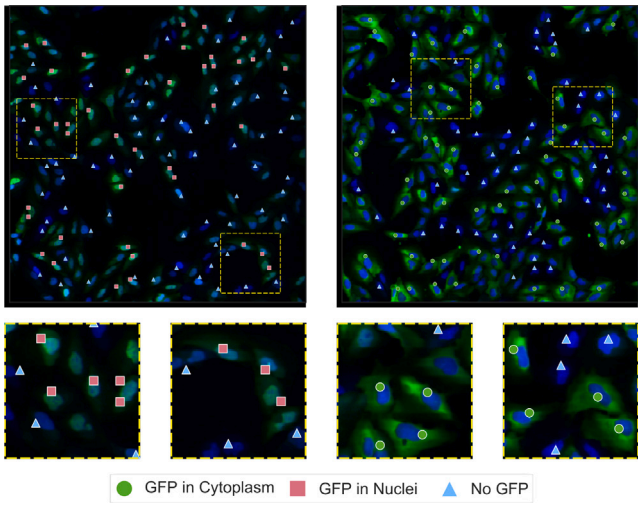


Fig. 5. Display of the SimSearch results on the GFP translocation images. The centers of the detected ROIs of the different classes are marked.

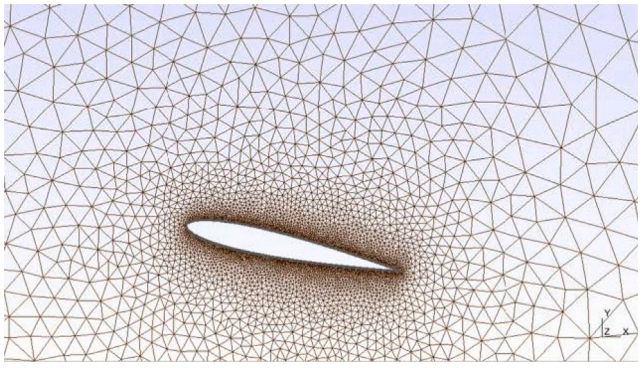


Fig. 6. Example of a Gmsh mesh around an airfoil.

However, another factor that affects the forces is the angle of attack of the airfoil. To find the optimal angle of attack, one can use the solver repeatedly with different geometries as input and for each of them calculate the solutions and forces. These processes generate multiple mesh files with different parameters such as angels, number of nodes, and level of refinements. Fig. 6 shows an example of a mesh file visualized by the software Gmsh (Geuzaine & Remacle).

In this dataset, 30 angles (0°–30°), 5 levels of refinements, and 20 nodes (100–300) were used to generate the mesh files. The results are based on 3000 mesh files, with each mesh file sizes from 500 Kb to 170 Mb depending on different parameters. The total size of the dataset is 128.5 Gb. This dataset is completely different from the previous image dataset. It is based on varying sizes of different geometry files containing high-precision numerical values. In Section 6, we will present the analysis pipeline and the proposed process to accelerate the execution by providing faster access to the relevant geometry files.

5.3. 1000 Genomes dataset

The 1000 Genomes project (Consortium et al., 2015) is an international research effort to establish a detailed catalog of human genetic variation. The dataset of the phase 3 release contains the DNA sequences of 23 pairs of human chromosomes of 3115 samples. Samples are taken from 3115 individuals belonging to 27 populations in different continents. Fig. 7 presents the geographical locations of the populations and their different distributions of one DNA.

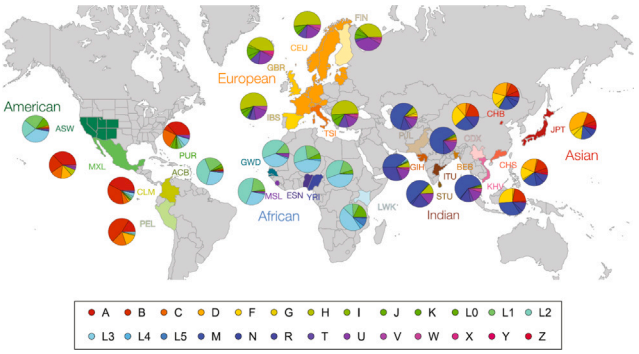


Fig. 7. Location distributions of the populations in the 1000 Genomes phrase 3 dataset over continents, and their different MtDNA distributions (Rishishwar & Jordan, 2017).

In total, the 1000 Genomes dataset contains 3115 matrices, where each matrix represents all the DNA sequences of an individual together with its label information (e.g. sample name, sex, population, super-population). The overall size of the dataset is 17Gb. Here it is important to note that the nature of this dataset is different from the previously presented datasets. Each matrix in the format of vcf file contains information including chromosome number, genetic locus, genotype quality, and etc. Given these large amounts of data, lots of analyses could be investigated. In Section 6, we will present an analysis pipeline using the dataset: population-wise comparison, and the dedicated efforts of speeding up the analyzing process by using our method.

5.4. Phenotypic screening dataset

This dataset originates from a cell-based drug repurposing screen. Cells were first infected by the virus and then exposed to a library of existing drugs. After 24 hours exposure, cells were subjected to multiplexed fluorescence staining using the Cell Painting protocol (Bray et al., 2016) and imaged using automated high-content imaging. The experiments were carried out in 384-well plates, and 9 sites were imaged in each well in 5 channels, producing approximate 15,000 images per plate. An antibody was added to stain viral proteins allowing to measure the rate of infection on a single-cell basis, according to our previously developed method (Rietdijk et al., 2021).

The objective of the screen was to identify existing drugs that are able to reverse the disease phenotype, so driving cells from infected state to non-infected state. The entire screen consists of 32 plates, resulting in approximate 500,000 images occupying 4 TB storage. CellProfiler software (McQuin et al., 2018) was used to calculate image-based features on single-cell basis, and median values per compound were calculated. A principal component analysis (PCA) showed that principal component one (PC1) could be used to distinguish between infected and non-infected cells. Furthermore, the dataset also contains mean antibody intensity as an important feature to indicate the level of infection.

It is worth to be highlighted here that this dataset is not only a multi-feature dataset with complex characteristics, but also contains multiple experimental groups (i.e. data are generated in different period and thus with different value scales). Meanwhile, two analyses on this dataset with totally different file access patterns are considered in this article. We will describe more details regarding the analyses requests pattern and implementation of different policies in Section 6.2.

6. Experiments

In order to address the challenges related to data management, next-generation frameworks need to be highly available and scalable. We have proposed a hierarchical storage system with a dynamic policy

using reinforcement learning. In this section we are presenting five experiments using the proposed framework to manage scientific datasets. We have used a real-world cloud environment to run our experiments. The setup was deployed as a three-tier HSS using three volumes with different sizes and I/O speed in the SNIC Science Cloud (SNIC, 2017; Toor et al., 2017). The three tiers are structured as illustrated in Fig. 1, with the read/write speed of the fast tier setup as 1000 Mb/s, middle tier 500 Mb/s, and slow tier 100 Mb/s. The following sections showcase the HSM-RL policy's impact in terms of system response time for different scientific applications. The presented use cases highlight two major points. First, the diverse data request patterns in different scientific applications, and second, the effect of different data migration policies on system response time. The experiments also highlight the self-adapting behavior of the RL-based policy which makes the proposed policy better or comparable to the best-performing policy.

Another important feature of the RL-based framework is the ability to re-distributing files to proper tiers after files being dropped down to slow tier in the purpose of reducing cost when data is not in use. In case there are not many requests related to some files, the framework decreases the temperature of those files and pushes them to the slowest tier to save resources. Thus the overall cost of hosting the dataset will be less compared to keeping the data in faster storage hierarchies. This cost-reducing feature is also incorporated into other policies through supplementary mechanisms. However, when compared to those policies, RL-based policy offers a significant advantage in efficiently redistributing files to appropriate tiers, leveraging the inherent memory capabilities of the RL agent.

Also, we would like to highlight that all the codes related to the deployment of the HSS based on cloud, the implementation of the data management policies, and the conduction of experiments are publicly available on the Github repository.

6.1. System performance based on different policies

6.1.1. Protein translocation dataset

We first explored the data management and processing requirements of the Protein Translocation dataset. The dataset has three different categories, and the analysis of category 1 usually requests images with high *interestingness value*. The analysis is about cell counting for high-quality images with *interestingness value* > 80. In this experiment, the analysis process ran for 1000 iterations. For each iteration, $18 \times 18 = 324$ qualified images were selected. To serve these requests for images, we have used various storage strategies including our proposed HSM-RL framework. We have monitored the response time of different policies during the entire 1000 iterations to verify their capabilities.

As described in Section 5.1, the Protein Translocation dataset consists of 1056 images of the same sizes. This brings an interesting challenge for the proposed framework to setup the state variables. In general, datasets consist of different file sizes. In previous settings, we have used file size as a state variable. This challenge highlights the need of giving extra attention to scientific datasets. However, the above-mentioned two features, the *interestingness value* and request pattern are the available data-centric information. Hence, we used them to setup the state variables s of the HSM-RL framework for the Protein Translocation dataset use case:

- s_1 : average temperature of all files in the tier, where file temperature takes value in $[0, 1]$ and stands for the request frequency of the file. Frequently requests to files will increase their temperature, while the temperature of files being untouched for a long time will decrease.
- s_2 : average weighted temperature in the tier, where the weight is defined by the *interestingness value*. s_2 is calculated by the average among the values of (*interestingness value* $\times$ temperature) of all files in the tier.
- s_3 : the interestingness index, computed by the sum of the interestingness index of files being requested in the tier. The interesting index of a file is $100/e^{3 \cdot |intr|}$, where $|intr|$ is the normalized value of the *interestingness value* of the file.

Here, it is important to note that the data-centric information is not dependent on the processing pipeline. It allows domain-specific knowledge to better manage the dataset independent of the computational requirements. With the newly defined state variables, we offer the hierarchical storage system with RL-based migration policy. To test and verify the performance and efficiency of the RL-based system, we launched the storage system with initial parameters and run the experiment based on 1000 iterations. In each iteration, 324 image requests were sent to the storage system. Since the read/write speed of each tier was different, the overall system response time varied based on the file distribution. Therefore, we monitor the system response time to measure the performance of the RL-based policy.

Apart from RL-based policy, we have implemented five other policies as well. In general, the most basic solution for file distribution is random placement. This policy randomly places files among available spaces, it is intuitively inefficient (therefore considered as a baseline policy). A more common and efficient policy is Least Recently Used (LRU) replacement (O'Neil et al., 1993) policy. LRU records the access history and downgrades files that are least recently used to lower tier, and replaces them with more recently used files. A similar approach is Least Frequently Used (LFU) replacement (Lee et al., 2001a) policy. Instead of replacing the least recently used files, LFU replaces the least frequently used files. Given the special feature of Protein Translocation dataset, the *interestingness value*, we formed a prefixed policy based on the *interestingness* feature as well. This policy places images with higher *interestingness value* to upper tiers, hence we called this policy Maximal *interestingness* policy.

All the above-mentioned policies are static policies. To make a fair comparison, we have also implemented a machine learning-based dynamic hierarchical data management approach, the K-means policy. In each iteration, it determines the data distribution among tiers according to the result of K-means clustering over all files. More in detail, by having the accumulated number of requests (i.e. the total number of how many times this file has been requested) and the *interestingness value* of each file, the K-means algorithm is able to partition the files in a given number of clusters (in our implementation $k = 50$). Files are then placed in different tiers according to their cluster and accumulated number of requests in descending order. Files belonging to the cluster whose center has the highest accumulated number of requests are placed in the fast tier, followed by the second highest and so on until the fast tier gets full. The rest of the files are placed in a similar way into the middle and last/slow tier.

Based on the above-mentioned data placement policies, we have compared their performance in this experiment with the response time as the standard measure. We have monitored the system response time in each iteration under each policy. The results are shown in Fig. 8. Among all the tested policies, random placement performed the worst with an average response time $12,554$ ms (to avoid the effect of randomness we selected the best result over 5 random placements). It has the longest response time in each iteration, which indicated that the file distribution under this policy was not optimal. LRU and LFU worked better, achieved average responses of 9056 ms and 9061 ms. Both of them traced the requests history, so they benefited from the previous request pattern and were able to distribute files more optimally. Unlike LRU/LFU, Maximal *interestingness* policy is based on the *interestingness value* instead of request patterns. It performed better than the previous policies (8906 ms on average). The analysis processes specifically focused on high *interestingness value* images. However, if the request patterns change in the analysis, the *interestingness value* might not be the index of importance anymore. In this case, to keep the optimal system response, a new policy is needed to be defined and the file

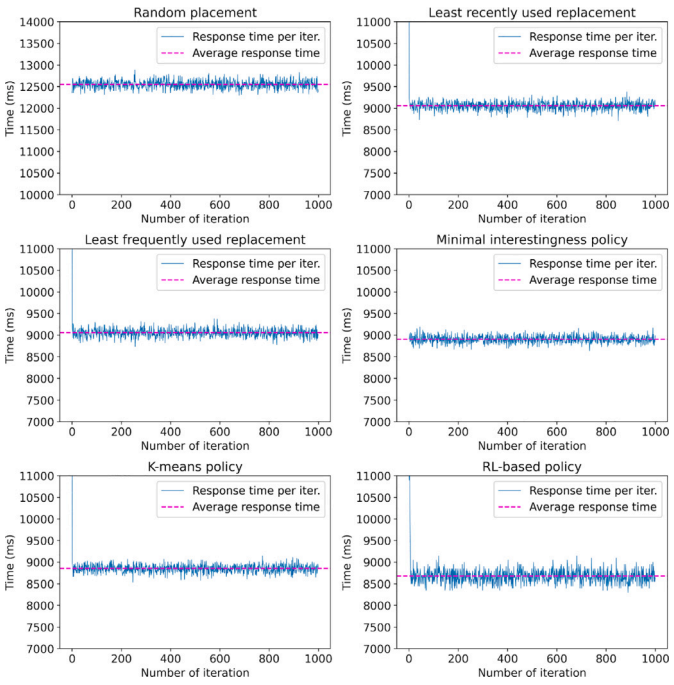


Fig. 8. System response time based on each iteration using Protein Translocation dataset. We have used RL-based policy, Random placement, LRU replacement, LFU replacement, K-means policy, and Maximal *interestingness* policy. **Additional remark:** Random placement subfigure has a different y-axis scale due to a significantly higher average response time.

distribution needs to be rearranged manually. The K-means policy uses the information of both *interestingness value* and the number of requests, together with the optimal clustering ability of K-means algorithm. It hence outperformed the other static policies (8810 ms on average). However, K-means algorithm is highly sensitive to the selection of k value and initial cluster center, thus usually requires large efforts in tuning. Meanwhile, it is weak for situations when there are many outliers in the feature distribution or non-convex scenarios.

The RL-based policy performed with a system response time 8681 ms on average. The average response time is superior to the best-performing policy, the K-means policy. Here it is important to note that the RL-based policy achieved this response time without the need of any initial assumptions on the access pattern. This is the first experiment that shows the ability of RL-based policy to offer a comparable system response time to the best-performing policy.

Apart from the system response time, we have also argued in Section 6 that the RL-based method holds another important advantage in fast re-distributing files. This feature is demonstrated in Fig. 9, where the file distributions among each tier during the experiment period are shown. The first part of Fig. 9 (timestep 0–1000) presents the occupancy of each tier when the 1000 iterations analysis started, and how RL-policy managed to fill faster tiers with the most important files. The second part (timestep 1000–1010) shows how the temperature decreasing mechanism (introduced in Section 3.2) worked in cooling down files that were not frequently used anymore and dropping them down to the slow tier to reduce cost. The final part (timestep 1101) illustrates the efficient re-distributing ability of RL-policy when repeating the same experiment again.

6.1.2. Airfoil mesh dataset

The second dataset we used to run the computations is the Airfoil angle of attack. As we introduced in the section , there are 3000 meshes files with different parameters (angels, nodes, etc.). The purpose of analyzing this dataset is to find the best angle of attack on airfoils. In

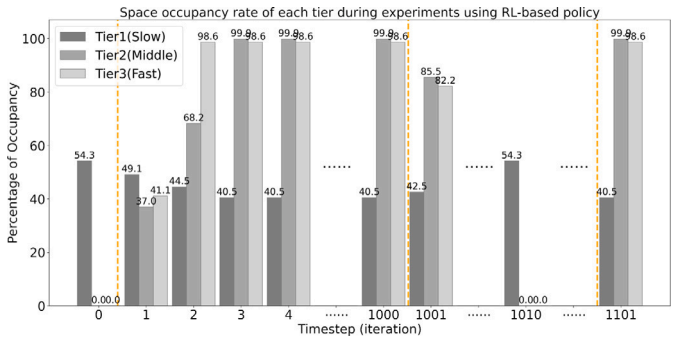


Fig. 9. File distributions in each tier during the experiments on Protein Translocation dataset, in terms of the space occupancy rate of each tier. Iteration 0 was when everything was at rest, iteration 1 analysis started. At iteration 1001 analysis completed, 1010 again at rest and finally 1101 when the experiment started again.

order to achieve this, we use the Navier–Stokes solver with 5 degrees of angles in each round (i.e. 0°–4° in the first round, 5°–9° in the second, and so on). Within each round, we calculate the solutions by using the solver with 10 different sets of parameters for 10 iterations to get the required convergence results. This overall requires a request pattern of 600 rounds (iterations), with 500 requests to mesh files in each iteration.

Similar to the experiment on the Protein Translocation dataset, we used 5 other policies to compare with our RL-based policy. The 5 other policies include Random Placement, LRU, LFU, K-means policy and Maximal feature policy. While the first three policies are exactly the same as the previous ones, the K-means policy and the Maximal feature policy are different. For the K-means policy, the clustering is based on the accumulated number of requests and the file size, which is an important feature instead of *interestingness value*. Maximal *interestingness* policy also changed to Maximal temperature policy, where file temperature is a feature valued between 0 and 1, proportional to $\frac{number\ of\ requests}{file\ size}$, given by the temperature changing mechanism we introduced in Section 3. This Maximal temperature policy places files with higher temperatures into the upper tiers.

Regarding the RL-based policy, since the internal characteristic of this dataset is the varying file size, we designed the RL-based policy with state variables s in the following way:

- s_1 : average temperature of all files in the tier.
- s_2 : average size-weighted temperature in the tier (the average of size×temperature of files in the tier).
- s_3 : current queuing time for arriving requests in the tier. A high value implies a large latency in this tier.

The policies mentioned above were used in managing the dataset during the 600 iterations. Same as the last use case, we evaluated their performance in terms of system response time per iteration. Fig. 10 illustrates that once again the RL-based policy offers a comparable response time to the best available policy. From the figure it can be seen that the shape of the system response time varies with different policies. This is due to the changing requests pattern and the various file sizes in the dataset, and different factors that each policy is based on. LRU and LFU showed different shapes because of their differences in using recency or frequency. While K-means policy had a very similar shape with LFU, mainly because the K-means clustering was also partly based on the frequency. Regarding the RL-based policy, with the combination of all the features (frequency, temperature, file size and etc.), it showed an integrated shape of all the other policies (the first 100 iterations are similar to the K-mean or LFU and the rest are closer to LRU and Maximal temperature). In this experiment, the best-performing policy out of the first five is the Maximal temperature policy and once again the results of the RL-based policy are comparable to the best one. Thus this experiment also clearly shows that rather than fine-tuning a policy for a special use case, it is better to use the RL-based policy to get the best response time.

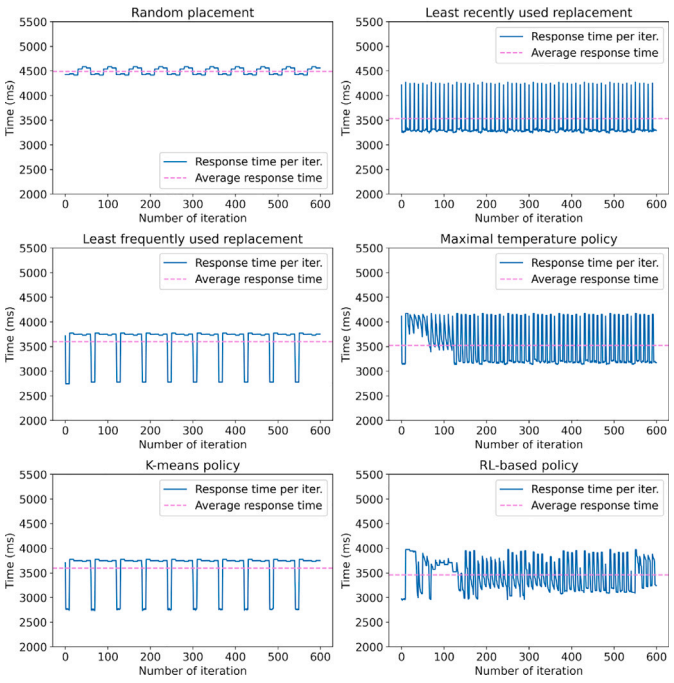


Fig. 10. System response time using the Airfoil mesh dataset. We have used RL-based policy, Random placement, LRU replacement, LFU replacement, K-means policy, and Maximal temperature policy.

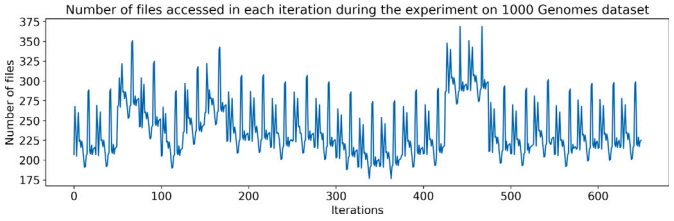


Fig. 11. File access pattern of the population-wise comparison experiment on the 1000 Genomes dataset. On the y axis is the number of files being requested in each iteration.

6.1.3. 1000 *Genomes dataset*

The 1000 Genomes project produces an extensive catalog of human genetic variation. The aim of this experiment was to compare specific DNA sequences between samples from different populations. More in detail, in each iteration, samples from one population and another population were selected to investigate their differences. This required a request pattern of 650 iterations, with a number of files being requested in each iteration varying from 177 to 369 depending on the number of samples in each population. More precisely, Fig. 11 shows the access pattern in terms of the number of files being accessed in each iteration.

With this requests pattern, once again we applied the six policies to manage the dataset and monitored the response time to evaluate their performance. Same as in previous experiments, Random placement, LRU, LFU were implemented in the exact same way. While for the K-means policy, we used the population code and the accumulated number of requests as the features for clustering, since population diversity is a unique characteristic of this dataset. For the RL-based policy, in the experiment we have used 2 state variables:

- s_1 : average temperature of all files in the tier.
- s_2 : entropy of the population distribution in the tier. For instance, if there are 26 samples of the population 'Iberian', 41 samples of 'Finnish', and 33 samples of 'British' in the tier, then $s_2 = entropy([26, 41, 33]) = 1.081$.

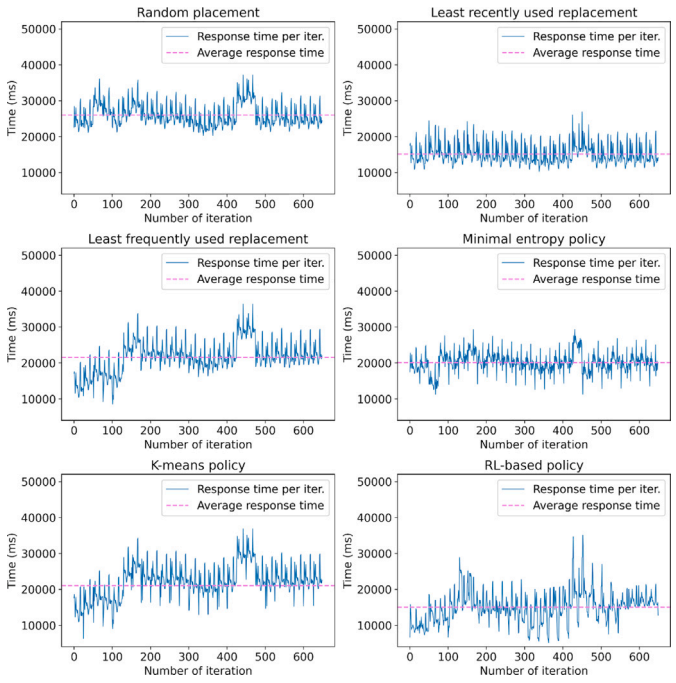


Fig. 12. System response time using 1000 Genomes dataset. The experiments include RL-based policy, Random placement, LRU replacement, LFU replacement, K-means policy, and Minimal entropy policy.

We also used the entropy of population distribution to define a Maximal/minimal feature policy, the Minimal entropy policy. As long as one file is requested, the Minimal entropy policy will decide whether this file should be upgraded to the upper tier or not based on the entropy of the upper tier. The file will be upgraded if the file upgrading will decrease the entropy, otherwise not. Take the same example of a fast tier containing 26 'Iberian', 41 'Finnish', and 33 'British'. If a new file from other tiers belonging to another population (e.g. 'Japanese') is being requested, then the entropy after upgrading this file is $entropy([25, 41, 33, 1]) = 1.124 > 1.081$, and according to the Minimal entropy policy, this file will not be upgraded to the fast tier. On the other hand, if the file belonged to 'Finnish', then the entropy after upgrading this file is $entropy([25, 42, 33]) = 1.076 < 1.081$, and this file will be upgraded to the fast tier.

Fig. 12 presents the system response time of each iteration in the population-wise comparison of the 1000 Genomes dataset. Unlike the previous experiments, LRU and LFU performed differently. LRU gained huge benefits from considering the repeating request patterns, and thus had an average response time of $15,150$ ms. On the other hand, LFU replacement ($21,581$ ms in terms of the average response) and K-means policy ($21,078$ ms) performed worse because of their tight coupling with the frequency/accumulated number of requests. The Minimal entropy policy only uses the population feature but not the repeating pattern, thus it did not work as outstanding as in previous experiments, with the average response time being $20,129$ ms. In this experiment, the best-performing policy is the LRU replacement policy, and once again RL-based policy performed comparably ($15,006$ ms in terms of the average response) to the best-performing policy. Here it is important to note that in this experiment we have used 2 state variables. This experiment also validates our claim that RL-based policy can quickly adapt according to any access pattern and performs better or comparable to a hand-picked policy.

6.2. *Scalable management of multi-feature datasets*

In addition to the three datasets mentioned above, we have also conducted experiments on the Phenotypic screening dataset, which is

a more complex use case. As introduced in Section 5.4, the high-content imaging screen consists approximately of 500,000 images in 32 plates, with 15,000 images in each plate. In the following experiments, we used the data from one plate after preprocessing to filter out damaged or polluted images. We have used 11,222 images, in total 97.5 GB dataset, with each image size 8.9 MB. The 11,222 images belonged to two experiments, with 5176 in experiment A and 6046 in experiment B. Images in Experiment A were collected as the first part, and were labeled as group A. While images in Experiment B (group B) were generated from the same microscopy but with different settings. We further described the detailed implementations of these two experiments in the following subsections.

6.2.1. Single dataset experiment

The initial experiment was a first-phase screening of non-infected cells. As introduced in Section 5.4, mean antibody intensity is a feature that can indicate the level of infection. Therefore, data for the first experiment were selected from images that have a low intensity in group A. The analysis runs for 100 iterations, with 500 images being screened in each iteration.

Again we used six policies (RL-based policy, Random placement, LRU, LFU, Maximal policy, and K-means policy) to manage the file distribution, and tested them in the 100 iterations to verify their effectiveness. However, since there is no feature as the *interestingness value* in the previous dataset, the RL-based policy and the maximal policy for this dataset need different settings. Alternatively, there are two features in the Phenotypic screening dataset that can play the same role as the *interestingness value* in the Protein Translocation dataset. The mean antibody intensity and the principal component 1 (pc1) are significant features to indicate the importance of images. Hence, we defined the states variable of RL-based policy to be:

- s_1 : average temperature of all files in the tier.
- s_2 : average weighted temperature in the tier (intensity × temperature).
- s_3 : importance index, computed by the sum of pc1 of files being requested in the tier.

In addition to the RL-based policy, the Maximal feature policy was also modified to the Minimal intensity policy. This policy places images with lower intensity into upper tiers. The K-means policy clustered data based on the accumulated number of requests, the antibody intensity, and the pc1 value. Random placement, LRU, and LFU were kept the same as in previous experiments.

Fig. 13 presents the response time of the system using each of the six policies. With respect to the response time, RL-based policy achieved an average of 17,924 ms, which is better than the best-performing policy, the Minimal intensity policy (18,062 ms in average). Close to the Minimal intensity policy, K-means policy achieved an average of 18,096 ms. As for the LRU/LFU replacement, the average system response time of LFU is 18,099 ms, lower than the LRU with an average 19,431 ms. The reason why LFU performed better than LRU is the ability to learn and continue from the previous requests patterns. Once again this experiment showcases the ability of the RL-based policy to quickly adapt according to the incoming request pattern and offer the best possible response time.

6.2.2. Multi-datasets experiment

The previous experiment was a first-phase analysis that only involved images from group A. As new data in group B was added, we increase the storage system capacity and setup the second round of analysis. In this experiment, we trained a CNN (Convolution Neural Network) model to classify images that were able to reveal effective drugs. The training process went on for 200 epochs, i.e. 200 iterations of requests were sent to collect images as input training data. The model was trained to identify high-quality images with a large number of healthy cells, thus images with low-intensity values were most often

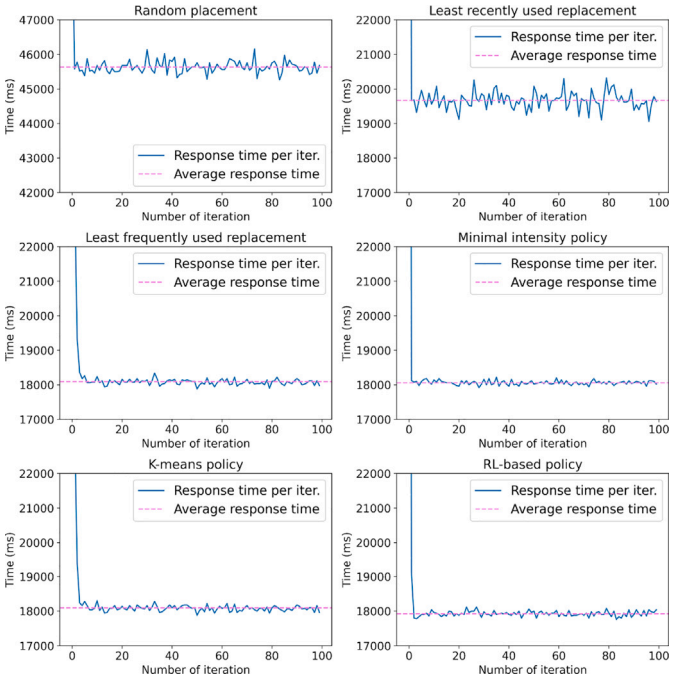


Fig. 13. System response time in the first phrase screening of Phenotypic screening dataset using RL-based policy, Random placement, LRU replacement, LFU replacement, K-means policy, and Minimal intensity policy. **Additional remark:** Random placement subfigure has a different y-axis scale due to a significantly higher average response time.

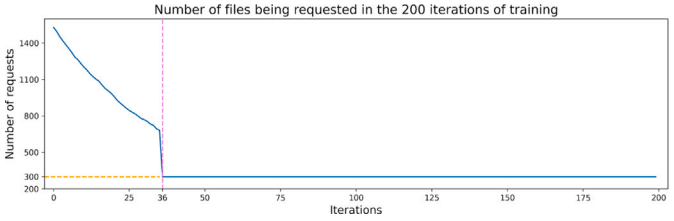


Fig. 14. Number of files being requested per iteration in the model training process involving group A and group B of the Phenotypic screening dataset.

chosen as the input. However, the significance thresholds of intensity were different between group A and group B. Therefore, we selected images with intensity < 0.009 in the first iteration, then reduced the intensity limitation 0.00002 per iteration until it reached the significance threshold of group B 0.0083. These reductions ended in 35 iterations, from the 36th iteration, we randomly selected 300 images from images in group A and B with intensity lower than the significance threshold.

According to the data selection routine described above, we eventually formed a requests pattern with 1529, 1497, 1455, 1420, …, 300, 300, …, 300 images in each iteration. These numbers of files being requested are also visualized in Fig. 14. Same as in previous experiments, we used six policies to manage the file distribution and recorded the system response times to evaluate these policies. The first 5 policies were exactly the same as the single group experiment, but the RL-based policy had some differences. Since the intensity value and the PC1 value of images in group A and B are at different scales, one group of RL agents cannot manage two groups of data because each agent can only have one pair of hyperparameters for one dataset. Therefore, we added 3 more RL agents in the system to manage data from group B. They worked together with the original 3 RL agents to determine the data migration policy. More in detail, for the formula (1), if the involved file F is in group A, then the system will use the cost function from RL agents for group A as v. Else if the involved file is in group

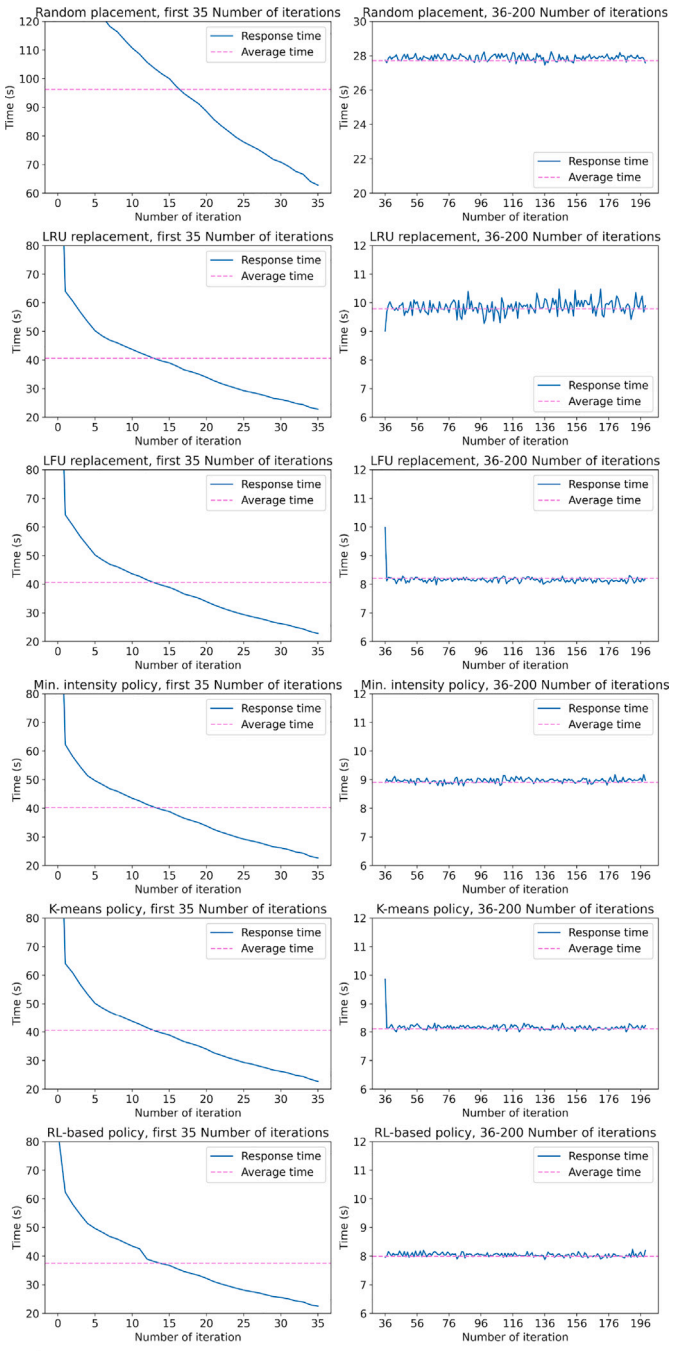


Fig. 15. System response time for the CNN model training using group A and group B dataset. We have used RL-based policy, Random placement, LRU replacement, LFU replacement, K-means policy, and Minimal intensity policy. **Additional remark:** Random placement subfigure has a different y-axis scale due to a significantly higher average response time.

B, then v will be the cost function from RL agents for group B. Here it is important to note that the agents only run as back-end processes in the framework, the front-end services are exactly the same as previous experiments.

Fig. 15 illustrates the system response time of all the 200 iterations in the model training process. The left part of the figure is the response time in the first 35 iterations, where the number of files being requested decreases in every iteration. Therefore, every policy showed a decreasing response time. In this phase (first 35 iterations) the four policies LFU, LRU k-means and Minimal intercity have a mean response time of around 40 seconds whereas the RL-based policy is at 37 seconds. Again RL-based policy due to its self-adapting behavior performs better than the rest of the policies.

Table 1
File distributions of Group A and B in Phenotypic screening dataset under the RL-based policy during 200 iterations.

Iter.	Tier1 Group		Tier2 Group		Tier3 Group	
	A	B	A	B	A	B
0	22.5%	26.3%	0.0%	0.0%	0.0%	0.0%
1	17.6%	26.3%	96.1%	3.9%	99.6%	0.4%
2	17.6%	26.3%	97.0%	3.0%	97.8%	2.2%
35	18.1%	24.7%	77.2%	22.8%	54.7%	45.3%
36	18.1%	24.7%	24.7%	75.3%	14.8%	85.2%
200	18.1%	24.7%	32.0%	68.0%	14.8%	85.2%

The right part of Fig. 15 shows the system response time in the 36th–200th iterations, where the number of files being requested was constantly 300. It reflects the ability of each policy to adapt to new request patterns. In this phase, the best average response time is from the K-means policy, and once again RL-based policy outperforms the K-means policy with its adaptive behavior.

In addition, Table 1 presents the file distributions in each tier during the 200 iterations using RL-based policy. Initially, all the files are on the slowest tier (Tier1). The percentages in the table represent the space used by the datasets. In the first 35 iterations, the system received more requests of files in group A than in group B, thus in the fastest tiers (Tier2 and Tier 3) there were more files of group A. From iteration 36 to 200, files being requested were mainly in group B, thus group B files started to migrate to faster tiers. This corresponds to the system response time shown in Fig. 15, and supports the efficiency and self-adjustability of the RL-based framework more intuitively. Furthermore, it also demonstrates the scalability and availability of RL-based policy in terms of managing multiple datasets in a single storage system.

7. Conclusion and future direction

In this paper, we have presented an RL-based hierarchical storage management (HSM-RL) framework together with the usage of internal features of the datasets. The aim is to offer better response time to the analysis pipelines. This study has a special focus on scientific datasets. We have presented four scientific datasets and associated six experiments. Together with the proposed RL policy, we have also evaluated other policies including Random placement, Least Recently Used replacement, Least Frequently Used replacement, K-means based policy, and prefixed minimal/maximal feature policies.

More specifically, our experiments illustrate that the RL-based policy with its unique self-adapting capability can perform effectively in challenging use cases. The results also describe the response time of the other potential policies that perform differently in different scenarios. The RL-based policy always gives better or comparable mean response time to contemporary policies. We have also presented the results of the RL-based policy using a different number of state variables (in the case of 1000 Genomes Dataset). Our findings clearly showcase that the RL-based data migration policy is the optimal way forward to design next-generation hierarchical storage solutions.

In the future, we will further evaluate the framework with more complex scenarios and also investigate the parameter tuning of the framework.

CRediT authorship contribution statement

Tianru Zhang: Conceptualization, Methodology, Software, Formal analysis, Visualization, Writing. **Ankit Gupta:** Resources, Visualization, Data curation, Writing – original draft. **María Andreína Francisco Rodríguez:** Resources, Visualization. **Ola Spjuth:** Writing – original draft,

Project administration, Funding acquisition. **Andreas Hellander:** Writing – review & editing, Supervision, Funding acquisition. **Salman Toor:** Conceptualization, Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Tianru Zhang reports financial support was provided by Uppsala University.

Data and code availability

All codes related to the deployment based on cloud, the implementation of the data management policies, and the conduction of experiments are publicly available on the Github repository. https://github.com/JSFRi/MSD-RLHSS.git Some datasets are not publicly available, but in case of needed they can be requested from the authors.

Acknowledgments

This research is supported by the Swedish Foundation for Strategic Research (SSF), project HASTE, under Grant No. BD15-0008. We would also like to acknowledge Swedish National Infrastructure for Computing (SNIC) for providing cloud resources, project number SNIC 2022/22-835, and support from eSSENCE, a Swedish strategic collaborative research program in e-science.

References

Acharya, S., Alonso, R., Franklin, M., & Zdonik, S. (1996). Broadcast disks: Data management for asymmetric communication environments. In T. Imielinski, & H. F. Korth (Eds.), *Mobile computing* (pp. 331–361). Boston, MA: Springer US, http://dx.doi.org/10.1007/978-0-585-29603-6_12.

Antcheva, I., Ballintijn, M., Bellenot, B., Biskup, M., Brun, R., Buncic, N., et al. (2011). ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications*, *182*(6), 1384–1385. http://dx.doi.org/10.1016/j.cpc.2011.02.008.

Barisits, M., Beermann, T., Berghaus, F., Bockelman, B., Bogado, J., Cameron, D., et al. (2019). Rucio: Scientific data management. *Computing and Software for Big Science*, *3*(1), 1–19. http://dx.doi.org/10.1007/s41781-019-0026-3.

Blamey, B., Toor, S., Dahlö, M., Wieslander, H., Harrison, P. J., Sintorn, I.-M., et al. (2021). Rapid development of cloud-native intelligent data pipelines for scientific data streams using the HASTE Toolkit. *GigaScience*, *10*(3), http://dx.doi.org/10.1093/gigascience/giab018.

Bray, M.-A., Singh, S., Han, H., Davis, C. T., Borgeson, B., Hartland, C., et al. (2016). Cell painting, a high-content image-based assay for morphological profiling using multiplexed fluorescent dyes. *Nature protocols*, *11*(9), 1757–1774. http://dx.doi.org/10.1038/nprot.2016.105.

Brubeck, D., & Rowe, L. (1996). Hierarchical storage management in a distributed VOD system. *IEEE MultiMedia*, *3*(3), 37–47. http://dx.doi.org/10.1109/93.556538.

Cesini, D., Donvito, G., Costantini, A., Aguilar Gomez, F., Duma, d. C., Fuhrmann, P., et al. (2020). The extreme-DataCloud project solutions for data management services in distributed e-infrastructures. *EPJ Web of Conferences*.

Consortium, . G. P., et al. (2015). A global reference for human genetic variation. *Nature*, *526*(7571), 68. http://dx.doi.org/10.1038/nature15393.

Dayan, P. (1992). The convergence of TD(λ) for general λ. In R. S. Sutton (Ed.), *Reinforcement learning* (pp. 117–138). Boston, MA: Springer US, http://dx.doi.org/10.1007/978-1-4615-3618-5_7.

Du, X., Tang, S., Lu, Z., Wet, J., Gai, K., & Hung, P. C. (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* (pp. 498–507). http://dx.doi.org/10.1109/ICWS49710.2020.00073.

Elworth, R. A. L., Wang, Q., Kota, P. K., Barberan, C. J., Coleman, B., Balaji, A., et al. (2020). To Petabytes and beyond: Recent advances in probabilistic and signal processing algorithms and their application to metagenomics. *Nucleic Acids Research*, *48*(10), 5217–5234. http://dx.doi.org/10.1093/nar/gkaa265.

Geuzaine, C., & Remacle, J.-F. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, *79*(11), 1309–1331. http://dx.doi.org/10.1002/nme.2579.

Gupta, A., Sabirsh, A., Wählby, C., & Sintorn, I.-M. (2022). SimSearch: A human-in-the-loop learning framework for fast detection of regions of interest in microscopy images. *IEEE Journal of Biomedical and Health Informatics*, 1. http://dx.doi.org/10.1109/JBHI.2022.3177602.

Ikegwu, A. C., Nweke, H. F., Anikwe, C. V., Alo, U. R., & Okonkwo, O. R. (2022). Big data analytics for data-driven industry: a review of data sources, tools, challenges, solutions, and research directions. *Cluster Computing*, 1–45. http://dx.doi.org/10.1007/s10586-022-03568-5.

Jiang, S., & Zhang, X. (2002). *LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance*. New York, NY, USA: Association for Computing Machinery.

Krish, K. R., Wadhwa, B., Iqbal, M. S., Rafique, M. M., & Butt, A. R. (2016). On efficient hierarchical storage for big data processing. In *2016 16th IEEE/ACM international symposium on cluster, cloud and grid computing* (pp. 403–408). http://dx.doi.org/10.1109/CCGrid.2016.61.

Lee, D., Choi, J., Kim, J.-H., Noh, S., Min, S. L., Cho, Y., et al. (2001a). LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, *50*(12), 1352–1361. http://dx.doi.org/10.1109/TC.2001.970573.

Lee, D., Choi, J., Kim, J. H., Noh, S. H., Min, S. L., Cho, Y., et al. (2001b). LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies, 50 (12). [ISSN: 0018-9340].

LHC (2008). Large hadron collider, https://home.cern/science/accelerators/large-hadron-collider.

Lin, B., Zhu, F., Zhang, J., Chen, J., Chen, X., Xiong, N. N., et al. (2019). A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing. *IEEE Transactions on Industrial Informatics*, *15*(7), 4254–4265. http://dx.doi.org/10.1109/TII.2019.2905659.

Ljosa, V., Sokolnicki, K. L., & Carpenter, A. E. (2012). Annotated high-throughput microscopy image sets for validation. *Nature Methods*, *9*(7), 637. http://dx.doi.org/10.1038/nmeth.2083.

McQuin, C., Goodman, A., Chernyshev, V., Kamentsky, L., Cimini, B. A., Karhohs, K. W., et al. (2018). CellProfiler 3.0: Next-generation image processing for biology. *PLoS Biology*, *16*(7), Article e2005970. http://dx.doi.org/10.1371/journal.pbio.2005970.

Megiddo, N., & Modha, D. S. (2003). ARC: A self-tuning, low overhead replacement cache. In *2nd USENIX conference on file and storage technologies*.

Mishra, P., & Somani, A. K. (2020). LDM: Lineage-aware data management in multi-tier storage systems. In K. Arai, & R. Bhatia (Eds.), *Advances in information and communication* (pp. 683–707). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-030-12388-8_48.

Nazarov, M. (2011). *Trita-CSC-A, Adaptive algorithms and high order stabilization for finite element computation of turbulent compressible flow* (Ph.D. thesis), (2011:13), (p. xii, 54). KTH Royal Institute of Technology, KTH, Numerical Analysis, NA, QC 20110627.

O'Neil, E. J., O'Neil, P. E., & Weikum, G. (1993). *The LRU-K page replacement algorithm for database disk buffering*. New York, NY, USA: Association for Computing Machinery.

Oussous, A., Benjelloun, F.-Z., Ait Lahcen, A., & Belfkih, S. (2018). Big data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*, *30*(4), 431–448. http://dx.doi.org/10.1016/j.jksuci.2017.06.001.

Rietdijk, J., Tampere, M., Pettke, A., Georgiev, P., Lapins, M., Warpman-Berglund, U., et al. (2021). A phenomics approach for antiviral drug discovery. *BMC Biology*, *19*(1), 1–15. http://dx.doi.org/10.1186/s12915-021-01086-1.

Rishishwar, L., & Jordan, I. K. (2017). Implications of human evolution and admixture for mitochondrial replacement therapy. *BMC Genomics*, *18*(1), 1–11. http://dx.doi.org/10.1186/s12864-017-3539-3.

Sanghi, N. (2021). Function approximation. In *Deep reinforcement learning with python: with pytorch, tensorflow and OpenAI Gym* (pp. 123–154). Berkeley, CA: A Press, http://dx.doi.org/10.1007/978-1-4842-6809-4_5.

Scaife, A. (2020). Big telescope, big data: Towards exascale with the square kilometre array. *Philosophical Transactions of the Royal Society, Series A*, *378*(2166), Article 20190060. http://dx.doi.org/10.1098/rsta.2019.0060.

Sienknecht, T. F., Friedrich, R. J., Martinka, J. J., & Friedenbach, P. M. (1994). The implications of distributed data in a commercial environment on the design of hierarchical storage management. *Performance Evaluation*, *20*(1), 3–25. http://dx.doi.org/10.1016/0166-5316(94)90003-5, Performance '93.

SKA (2019). Square kilometre array, https://www.skatelescope.org/the-ska-project/.

SNIC (2017). Swedish National Infrastructure for Computing, https://www.snic.se.

Soumagne, J., Henderson, J., Chaarawi, M., Fortner, N., Breitenfeld, S., Lu, S., et al. (2022). Accelerating HDF5 I/O for exascale using DAOS. *IEEE Transactions on Parallel and Distributed Systems*, *33*(4), 903–914. http://dx.doi.org/10.1109/TPDS.2021.3097884.

Storn, R., & Price, K. (1995). Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, *23*.

Toor, S., Lindberg, M., Falman, I., Vallin, A., Mohill, O., Freyhult, P., et al. (2017). SNIC science cloud (SSC): A national-scale cloud infrastructure for Swedish academia. In *2017 IEEE 13th international conference on e-science* (pp. 219–227). http://dx.doi.org/10.1109/eScience.2017.35.

Tusar, T., & Filipic, B. (2007). Differential evolution versus genetic algorithms in multiobjective optimization, 4403. (pp. 257–271). ISBN: 978-3-540-70927-5, http://dx.doi.org/10.1007/978-3-540-70928-2_22.

Wang, M., Zhang, J., Dong, F., & Luo, J. (2014). Data placement and task scheduling optimization for data intensive scientific workflow in multiple data centers environment. In *2014 second international conference on advanced cloud and big data* (pp. 77–84). http://dx.doi.org/10.1109/CBD.2014.19.

Wilkes, J., Golding, R., Staelin, C., & Sullivan, T. (1996). The HP autoraid hierarchical storage system, 14 (1). [ISSN: 0734-2071].

Xiao, J., Xiong, Z., Wu, S., Yi, Y., Jin, H., & Hu, K. (2018). *Disk failure prediction in data centers via online learning*. New York, NY, USA: Association for Computing Machinery.

Yuan, D., Yang, Y., Liu, X., & Chen, J. (2010). A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems*, *26*(8), 1200–1214. http://dx.doi.org/10.1016/j.future.2010.02.004.

Zhang, T., Hellander, A., & Toor, S. (2022). Efficient hierarchical storage management empowered by reinforcement learning. *IEEE Transactions on Knowledge and Data Engineering*, http://dx.doi.org/10.1109/TKDE.2022.3176753.

Zhou, Y., Philbin, J., & Li, K. (2001). The multi-queue replacement algorithm for second level buffer caches. In *Proceedings of the general track: 2001 USENIX annual technical conference* (pp. 91–104). USA: USENIX Association, http://dx.doi.org/10.5555/647055.715773.