Forecasting file lifecycles for intelligent data placement in hierarchical storage

Adrian KHELILI
Eviden Atos BDS R&D Data Management
Li-PaRAD, UPSaclay-UVSQ, France
adrian.khelili@eviden.com

Sophie ROBERT HAYEK
Eviden Atos BDS R&D Data Management
Echirolles, France
sophie.robert@eviden.com

Soraya ZERTAL
Li-PaRAD, UPSaclay-UVSQ
Guyancourt, France
soraya.zertal@uvsq.fr

***Abstract*—The increasing disparity between computing speed and data access latency poses significant challenges in managing data storage, particularly for massively parallel supercomputers. To address this issue, storage systems have evolved into hierarchical architectures with different tiers offering varying performance, cost, and capacity based on specific hardware technologies. This heterogeneous and hierarchical nature of storage comes with the need for an optimal data placement strategy. Existing strategies have primarily approached this problem from a block perspective, focusing on analyzing application I/O behavior. However, such approaches fail to capture the contextual usage of information. To overcome this limitation, considering file-level usage patterns and adopting a file-centric perspective for data placement can leverage the context and semantics of files, leading to more efficient data placement strategies.**

This study proposes a novel file-based approach for data placement, focusing on the concept of file re-use and representing files through their life cycles (FLCs). The FLC of a file captures the sequence of operations it undergoes during its lifetime, referred to as FLCevents. By analyzing the time series data associated with the FLCs of actively used files, our algorithm detects repetitive patterns of FLCevents to predict future events using pattern matching and facilitate the anticipation of file movement. This allows for optimal data placement that aligns with the expected near-future usage. To validate our approach, we conducted experiments using traces extracted from representative high-performance computing (HPC) applications, namely NEMO, NAMD, LQCD, and an IO-Benchmark. The results we obtained are highly promising, demonstrating that in term of file prediction accuracy our proposed F-LRU (File-based Least Recently Used) achieves from 77% to 55% precision in most difficult scenarios to 100% for the others. Also, it is at least as effective as traditional LRU and LFU and can increase the hit rate by a factor of 1.94 against LFU and of 1.06 against LRU for LQCD and IO-Bench respectively. These findings highlight the potential of our approach to significantly enhance hierarchical storage performance, particularly in HPC environments.

***Index Terms*—I/O Prediction, File lifecycles, Time Series, Data temperature, Hierarchical storage, High Performance Computing**

I. INTRODUCTION

The growing disparity between computing and storage performance in modern supercomputers is causing severe performance challenges, especially as applications become more and more data-intensive [1] [15] [20]. Their heavy read and write operations performed by numerous nodes can strain the back-end storage, increasing significantly the waiting time of applications to finish their I/O operations and resume their computation. This is particularly problematic for applications and workflows that regularly save their current state through checkpoints [6], resulting in bursts of writes.
One potential solution to address these I/O bottlenecks is the use of hierarchical storage, which integrates multiple storage levels, each with its distinct technologies and physical characteristics. This hierarchy commonly includes RAM, SSD-NVMe, SSDs, and HDD as backend storage.

While the purpose of this hierarchical storage is to enhance I/O speed, it presents its own set of challenges, as efficiently distributing data across the various levels is crucial to decrease latency in data access. There are two potential approaches to address this problem: smart eviction strategies whenever tiers are too full and need to be emptied, and prefetching algorithms that anticipates the application's behavior by placing in the higher tiers data that is most likely to be accessed. These algorithms range from traditional methods such as LFU [30], LRU [21] and FIFO [12] to more advanced approaches relying on machine learning. On the other hand, prefetching techniques [3], [31] consist in proactively loading data that is likely to be used by the application into higher levels. Most of these algorithms primarily analyze the application from a block perspective rather than a file perspective and this single and fine granularity significantly reduces their predictive capabilities due to the restricted valuable information. By adding a higher level of abstraction considering a file-based approach, we have a better understanding of the application behavior and simplify the tracking, organization, and management of data during migration between the tiers.
In this work, we suggest switching from a block-only paradigm to a file-based one, by evicting from the higher tiers blocks belonging to files that are most likely to be re-used. To perform this eviction policy efficiently, we introduce the concept of *File Life Cycles Events* (FLCevents) to represent the usage scheme/profile of a file which consists in the series of POSIX operations performed by a given application on it. We then use time-series forecasting methods to predict future FLCevents and rank the files according to their likelihood of being accessed in the near-future. When a data movement is required to empty the appropriate tier, we iteratively evict the least recently used blocks of the file that is predicted to be the less likely to be re-used.The main contributions of this paper are:

- The proposal of a new paradigm for data placement on heterogeneous and tierd storage based on the file re-use property using the concept of FLCevents;
- The proposal of a novel file-based algorithm for time-series prediction based on pattern matching;
- The proposal of a new eviction policy using this prediction of file access behaviors to rank files according to how likely they are to be used;
- The application of this policy on three real HPC applications and one benchmark proving its efficiency compared to LRU and LFU.

This paper is structured as follow. Section II introduces works that are similar to ours and highlights the novelty of our suggestion while section III motivates our choice of using file-based prediction instead of block-based one. Section IV presents notations and definitions followed by the methods used for time series prediction in section V. Section VI describes the experiments conducted to validate our approach. Section VII exposes and discusses the obtained results and section VIII concludes the paper giving some insight into future works.

II. Related works

Several block eviction strategies have been proposed in the literature for cache management, including simple algorithms such as Random, LRU [30], LFU [21] and FIFO [12]. More advanced techniques have also been developed to strike a better balance between recency and frequency. For example, LRFU [10] offers a range of policies between LRU and LFU, while ARC [13] divides the cache into recently and frequently accessed pages. Additionally, LIRS [16] introduces the concept of inter-reference recency to enhance efficiency. Despite the notable effectiveness of these policies, their practical implementation can be complicated due to the parameter tuning phase required prior to execution. To address this challenge, the CAR and CART [4] algorithms have been proposed, leveraging parameter auto-tuning to automatically decide which hyperparameter to select. However, both methods fail to distinguish between data that has been recently accessed but will not be accessed again and can thus be evicted from higher tiers and data that will be accessed multiple times. CART addresses this issue by introducing a temporal filter, which enables distinguishing between short-term and long-term accesses, but this policy still doesn't sufficiently differentiate between data accessed once and data reused multiple time. Recent advances in machine learning have led to the emergence of a new generation of algorithms based on reinforcement learning techniques. Among these, LeCar [33] employs a reward function to determine the relative importance of two policies, LRU and LFU. This algorithm has been extended in Cacheus [26], which introduces a novel combination of policies, namely SR-LRU (Scan-resistant LRU) and CR-LFU (Churn-resistant LFU). Furthermore, Cacheus dynamically adjusts hyperparameters using gradient-based stochastic hill climbing, enhancing its autotuning capabilities. Despite the effectiveness of these policies, they are limited in their ability to optimize systems performance, as they do not take into account the unique characteristics of each file, as we will show in section III.

Other data management solutions includes probability models [23] [34] [29] such as Markov Models (MM) that have the drawbacks of making the strong assumption that I/O requests are memory-less, which is not necessarily true in practice. Grammar based techniques allow the modelization of I/O behavior [11] to capture characteristics of the program's memory access patterns and proactively fetch data into the cache before it is requested by the processor. Other Machine Learning techniques such as decision trees have been used to predict I/O performance [19], [22].

When it comes to the prediction of I/O behavior using time series methods, [9] uses a time series model to estimate file system server load and [5] proposes a pattern matching approach for server-side access pattern detection for the HPC I/O stack. [32] and [8] use ARIMA models to capture both temporal and seasonal trends in time series data, making it a popular choice for predicting I/O performance in various settings. However, the ARIMA model is complex, computationally expensive and hard to parallelize, as highlighted in [35]. Moreover, parameter tuning can significantly affect prediction performance, making it challenging to achieve be accurate and reliable.

More generally, generic algorithms for time series prediction can be leveraged to predict I/O behavior, as they offer a variety of patterns detection techniques. We can for example cite the algorithm based on Symbolic Aggregate approXimation (SAX) that converts the original real time serie T into a sequence of symbols belonging to an alphabet, and the Piecewise Aggregate Approximation (PAA) [2] which consists in dividing a time series into fixed-size segments and approximating each segment with an aggregated value, such as the mean or the median.

III. Motivation

One of the significant limitations associated with relying solely on block accesses for data placement is the absence of contextual information regarding data usage. Unlike blocks, files at the application level typically represent logical units of information with inherent relationships and dependencies. Files can be categorized based on their access type, such as input files, checkpointing files, result files, and so on. By considering file-level usage patterns, data placement can take into account the context and semantics of the files, leveraging their inherent characteristics and access requirements. By predicting file usage at the file level, the decisions of file movements within the different storage tiers can be made much more efficiently: files which are likely to be accessed in the future can be prefetched into higher tiers, and symmetrically, files that are not likely to be accessed again can be evicted to lower tiers.

For these reasons, this paper focuses on migration policies at the file level, where blocks are moved to a lower tier on a per-file basis guided by forecasts of incoming file usage.

Traditional time series modeling approaches like ARIMA, AR, and MA are often time-consuming and require complex parameter tuning, making them impractical for production applications. Moreover, their sensitivity to parameters and potential placement errors can adversely affect system performance. To address these limitations, this study employs pattern matching algorithms for forecasting file usage, which offer faster results with limited modeling requirements.

By identifying the most similar pattern to the current activity that has been observed in the past, we make predictions about the future activity of each file, assuming that historical patterns tend to repeat themselves. These predictions enable us to establish a priority ranking for the files. The file predicted to exhibit the least activity or deemed less important based on the observed pattern is then selected for migration to a lower tier with higher priority. This approach allows us to make informed decisions about which file to migrate while considering their predicted future activity, rather than simply relying on recency or frequency of accesses at the block level.

IV. Definitions and notations

- **Time Serie:** A time serie $\mathbf{T} = (\mathbf{t_1}, \mathbf{t_2}, ..., \mathbf{t_n})$ of length n is a sequence of n real-valued observations, where each t_i represents a value observed at a specific time point.
- **Subsequence:** Let T be a time series of length $\mathbf{n}$. A subsequence $\mathbf{S_{i,l}}$ is a contiguous time series of length $\mathbf{l}$ with $\mathbf{1 \leq i \leq n}$ and $\mathbf{1 \leq i + l \leq n}$. Specifically, $\mathbf{S_{i,l}}$ is defined as $(\mathbf{t_i}, \mathbf{t_{i+1}}, \ldots, \mathbf{t_{i+l-1}})$.
- **Trivial Match:** Two subsequences $S_{i;l}$ and $S_{j;l}$ of the same time serie T overlap if they share at least half of their common positions in T, where l is the length of the subsequences. The condition for overlap is that the index j of the second subsequence should be between $i - l/2$ and $i + l/2$.
- **FLC**: a temporal series of FLCevents $(o_t)_{1 \leq t \leq T}$ performed on a given file, with o_t an operation in the POSIX set $\{$READ, WRITE, OPEN, CLOSE$\}$.
- **Pattern**: a pattern refers to a recurring subsequence that repeats at regular intervals in the FLC.

V. Methods

The main objective of our approach is to anticipate the appropriate data placement on the tiers by establishing a priority files ranking based on their likelihood of being reused or not in the near future. By accurately predicting file usage patterns, a priority ranking of files can be established at runtime. This ranking guides the decision-making process that determine which files to evict first from the higher tiers of the hierarchical storage.

A. Time serie modeling

For each File Life Cycle (FLC), we apply a binarization which is a common technique used to enhance predictions, as displayed in figure 1. It involves dividing the time serie into fixed-sized intervals, reducing the data variability. The bin size, denoted as w, serves as an algorithm hyper-parameter and determines the width of these intervals. By carefully selecting an appropriate bin size, we can gain a better understanding of the time serie and achieve more accurate predictions. This binarization divides the time series into a set of bins denoted $I = I_1, I_2, \ldots, I_n$, where I_i represents the i^{th} interval of the life cycle. The total number of bins, n, is determined by the ratio between the total duration of the application and the bin size. Function f counts the number of requests within interval i and $f(I_i)$ represents the file access frequency at each interval i, calculated for READ and for WRITE operations. The life cycle is then converted to a time serie, associating the number of requests to a temporal interval of the life cycle. The analyzed time serie can be denoted T = $\{f(I_1), f(I_2), \ldots, f(I_n)\}$

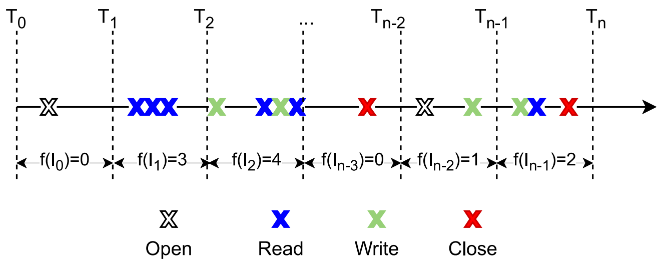

Fig. 1: Conversion of FLC into binned time serie

B. Tested distances

In KNN regression, a distance is a measure of similarity between two time series sub-sequences. We choose to focus on the Euclidean distance for its computational advantages instead of *Dynamic Time Warping* (DTW) [17]

Dynamic Time Warping measures the similarity between two temporal sequences, even when they have different lengths or exhibit temporal distortions, as it searches for the optimal alignment between the two sequences by warping and stretching them in the time dimension. A distance measure is computed between the two sequences by creating a matrix that represents the accumulated cost of aligning each pair of elements in the sequences. The elements of this matrix correspond to the partial alignments of the sequences at different time points.

The formula for computing the DTW distance between two sequences X and Y of lengths n and m respectively is as follows:

$$\text{DTW}(X, Y) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} d(x_i, y_j)}$$

where $d(x_i, y_j)$ is the local distance measure between elements x_i and y_j in the sequences X and Y respectively. We use here the Euclidean distance.

The DTW algorithm then finds the optimal alignment by iteratively computing the accumulated cost matrix and back-tracking through it to determine the optimal path. The resulting optimal path represents the alignment between the sequences.

C. Suggested eviction algorithm

When applied to FLC prediction, this pattern matching algorithm leverages the assumption that by identifying the most similar observed FLCevents patterns, we can predict the future behavior of the application at the file level, assumption validated by works on the periodicity of HPC applications [28] [14]. The FLC timeseries is then browsed with a sliding time window that moves through the timeseries and examines subsequences, as displayed in figure 2. This window should not be confused with the binning window w mentioned earlier. It corresponds to the pattern containing h windows w. The distance calculation function is a callback parameter $dist$ to test both the DTW (Dynamic Time Warping) and ED (Euclidean Distance) calculation algorithms. Once the best matching pattern is found, it serves as the basis for prediction as described in algorithm 1. We can thus predict how the file will behave in the future based on its historical observed patterns and anticipate data placement according to the predicted behavior.

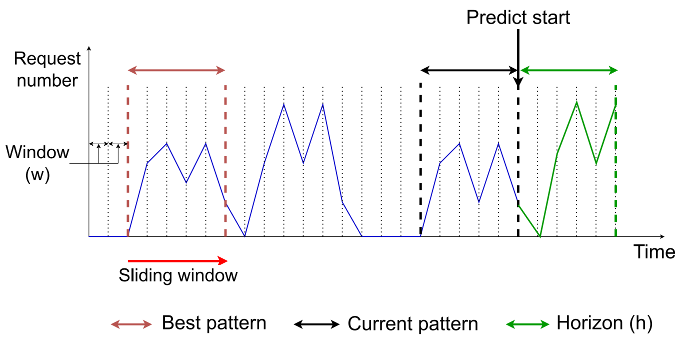

Fig. 2: Schematic representation of file prediction process

Once the file with the least likelihood of being reused selected, it is also necessary to choose the specific blocks to be evicted within that file. The blocks will be evicted in the LRU (Least Recently Used) order, meaning that the least recently accessed blocks within the file will be evicted first. Our aim is just to free enough room in the tier to achieve our anticipation and load its corresponding data. As the prior file for eviction is also in use (even with the lowest reuse score), we adopt a conservative strategy and we evict only blocks to free enough space for potential data to be accessed in the future according to our prediction.

D. Complexity

Let m be the number of manipulated files, p the number of patterns, FLC the file lifecycle time serie, and n = $|FLC|$, where $|FLC|$ denotes the cardinality of FLC in terms of number of windows w. The complexity of this algorithm is $O(n \times m \times p)$ when the selected distance metric is euclidean and $O(n^2 \times m \times p)$ in the case of DTW metric. There is an optimized version of the DTW algorithm that uses a subsampling strategy to reduce the time complexity to $O(n \times \log(n) \times m \times p)$ at the cost of a slight loss in precision due to the resolution reduction. Despite that faster computation time, we adopted the original DTW distance rather than fast DTW for the sake of precision. Additionally, our solution is highly parallelizable since the files behaviors are independently predicted and the comparison between patterns are independently computed. This means that a subset of opened files can be considered by our prediction in parallel (subject to the number of cores, parallel execution strategy, etc), which will be an important feature when porting our work to production.

VI. Validation

We evaluated our data management propositon to a Burst Buffer IO-accelerator with a heterogenous hierarchical storage that will be detailed further in this section. We organize the validation of our prediction strategy based on pattern matching coupled with a file-based management policy of the tiered storage in two steps: we first validate the quality of the file re-use prediction ranking by running the prediction algorithm every 10 bins, and evaluate the prediction once the actual data, which will act as a ground truth, is available. In the second phase, we evaluate and compare the performance of our eviction policy with standard policies. To measure the effectiveness of our policy, we utilize representative metrics such as the hit ratio. By comparing the hit ratio of our eviction policy to that of standard policies, we can assess the efficiency and effectiveness of our approach.

A. Selected applications

Four applications have been selected to evaluate the performance of our algorithm: three HPC applications and a synthetic I/O benchmark. A summary of the I/O behavior in terms of file manipulation of each application is available in table I.

1) NAMD:
NAMD [25] is a parallel molecular dynamics code designed for high-performance simulation of large bio-molecular systems. It has the particularity of being very dependent on the storage hardware, due to its large I/O bursts, and is thus a good use-case.
For our experiment, we use the Satellite Tobacco Mosaic Virus (STMV-28M) configuration. This is a 3x3x3 replication of the original STMV dataset from the official NAMD site, containing roughly 28 million atoms.
NAMD execution goes through 50 steps corresponding to the number of simulation time steps to achieve. Another parameter defines the number of steps after which a checkpoint is performed that is set to 5 to obtain ten checkpoints per run for a significant I/O activity. In this configuration, a total of 2 GB of data is read and 5 GB of data is written, resulting in a total of 10226 read operations and 1079 write operations. These operations involve a total of 14 files, 10 of which are heavily re-used.

2) NEMO: NEMO [27] (*Nucleus for European Modeling of the Ocean*) is a state-of-the-art modeling framework for research activities in ocean and climate sciences. It is characterized by a significant file re-use, highlighting the importance of a custom file placement policy in the hierarchical storage

Algorithm 1 Pattern matching algorithm with time series prediction

```
Require: $time\_series, h, dist$
 1: procedure PATTERNMATCHINGPREDICTION($time\_series, h$)
 2:     $current\_pattern \leftarrow$ get_curr_pattern($time\_series, h$)      ▷ Get the current pattern of size$h$
 3:     $closest\_pattern \leftarrow$ None
 4:     $min\_distance \leftarrow \infty$
 5:     for $i \leftarrow 0$ to len($time\_serie$) $- h - 1$ do
 6:         $pattern \leftarrow time\_serie[i : i + h]$      ▷ Extract the pattern
 7:         $distance \leftarrow$ dist($current\_pattern, pattern$)      ▷ Calculate the distance
 8:         if $distance < min\_distance$ then
 9:             $min\_distance \leftarrow distance$
10:             $closest\_pattern \leftarrow pattern$
11:     $prediction \leftarrow$ predict_from_pattern($closest\_pattern$)      ▷ Predict from closest pattern
12:     return $prediction$      ▷ Return the prediction
```

to keep the most accessed files in the most efficient level. Additionally, we can see that NEMO constantly manipulates a certain number of files, whether in read or write mode, which offers rich patterns for both READ and WRITE operations. It presents a higher activity in reading at the beginning of the application and a higher activity in writing at the end of the application, which is explained by the reading of input files at the beginning and the writing of output files at the end of the application.

For our experiment, we use the GYRE configuration, which simulates the seasonal cycle of a double-gyre box model, and which is often used for I/O benchmarking purpose as it is very simple to increase grid resolution and does not require any input file. In our case, the grid resolution is set to 5 and the number of MPI processes to 32 to increase the I/O activity. In this configuration, a total of 65 GB of data is read and 50 GB of data is written, resulting in a total of 17,816 read operations and 17,816 write operations. These operations involve a total of 184 files, among which 21 are heavily reused.

3) LQCD: Lattice QCD is a well-established non-perturbative approach for solving the quantum chromodynamics (QCD) theory of quarks and gluons. It is a lattice gauge theory formulated on a grid or lattice of points in space and time. When the size of the lattice is taken infinitely large and its sites infinitesimally close to each other, the continuum QCD is recovered [7].

In this lattice quantum chromodynamics (LQCD) simulation, the parameters are set as follows: the quark mass (qmass) which represents the mass of the quark is set to 0.02, and the gauge coupling strength (beta) which represents the strength of the interactions between quarks and gluons in the lattice simulation is set to 0.2. The simulation is performed with a spatial extent of 16 lattice units and a temporal extent of 36 lattice units. The simulation is divided into a total of 8 tasks in the temporal direction, 4 tasks in the spatial direction, 4 tasks in the y-direction, and 4 tasks in the x-direction, resulting in a total of 512 tasks.

In this configuration, a total of 27 GB of data is read and 40 GB of data is written, resulting in a total of 1673538 read operations and 3335740 write operations. These operations involve a total of 284 files, among which 21 are heavily reused.

4) Benchmarking application: We design this synthetic application using a generic benchmarking I/O tool in such a way that the read phases gradually lengthen, as can be seen in figure 8. Except for the beginning and end of the application, the activity in write in relatively low as shown in figure 7. In this configuration, a total of 42 GB of data is read and 1 GB of data is written, resulting in a total of 36137 read operations and 1240 write operations. These operations involve a total of 9 files and all of them are heavily reused. We have chosen this specific bench to visually demonstrate the difference in patterns when they expand, compare the effectiveness of DTW and ED as distance metrics, and observe a profile characterized by significant file reuse.

B. Evaluation metrics

To evaluate the performance of our prediction algorithms, we calculate standard multi-class supervised learning metrics in Machine Learning: accuracy, precision, and MCC for Matthews Correlation Coefficient [5]. We subdivide the order of file prediction into four quartiles, and treat each of these as a specific category. We define for each quartile:

- True Positives (TP) : number of correctly predicted instances for a specific quartile
- True Negatives (TN): number of correctly predicted instances for all other classes excluding the specific quartile
- False Positives (FP) : number of incorrectly predicted instances as the specific class
- False Negatives (FN) : number of instances belonging to the specific class but incorrectly predicted as other classes.

the different metrics are calculated and averaged for quartile:

- Accuracy = $\frac{1}{4} \times \sum_{i=1}^{4} \frac{TP_i+TN_i}{TP_i+FP_i+FN_i+TN_i}$
- Precision = $\frac{1}{4} \times \sum_{i=1}^{4} \frac{TP_i}{TP_i+FP_i}$
- MCC = $\frac{1}{4} \times \sum_{i=1}^{4} \frac{TP_i \cdot TN_i - FP_i \cdot FN_i}{\sqrt{(TP_i+FP_i)(TP_i+FN_i)(TN_i+FP_i)(TN_i+FN_i)}}$

TABLE I: I/O behavior of the four applications

Application	Total files	File re-used	Number of read	Number of write	READ volume (Gb)	WRITE volume (Gb)
NEMO	184	21	17816	12867	65	50
NAMD	14	1¡0	1226	1079	2	5
LQCD	284	121	1673538	3335740	27	40
BENCH	9	9	36137	1240	42	1

For example, an Accuracy of 100% means that every file was predicted in its right quartile. We check the quality of our prediction via these metrics each $w \times 10ns$, then average the results over the whole application execution.

We also consider the data placement management quality by considering the hit ratio at the first tier (the one with the lowest access latency) for each application and compare it to three of the most popular cache management algorithms: LRU, LFU, F-LRU.

C. *Tested hyperparameters*

Our prediction also requires several hyperparameters. One of these hyperparameters is the window size (w), which is used to determine the number of IO requests to consider when creating the time serie. The window size determines the granularity of the time serie, i.e., the frequency at which FLCevents are aggregated to form the observation points where each observation $f(I_n)$ refers to the number of FLCevents made within that specific interval I_n. The length of the subsequence that describes the pattern is also a hyperparameter. The number of predicted values is denoted by the horizon (h). We select for our experimentation campaign a value of 40 as a trade-off between the quality of the prediction (the closer the prediction the easier it is to predict accurately) and the number of predicted points (the more information we have, the more efficiently we can predict data placement). The window size (w) corresponds to the width of the pooling interval to transform: in production, it should be carefully chosen based on the shape of the bursts of the application, as it impacts the shape of the timeseries and subsequently affects the quality of predictions. For this work, we select (w) based on the execution time of the policy, to have 1000 data point in total within the time series. The value is then rounded to the nearest power of 10. A summary of tested hyperparameters is available in table II.

TABLE II: Tested hyperparameters per application

Application	h	w
LQCD	40	10 000 000
NEMO	40	1 000 000
NAMD	40	100 000
Synthetic App	40	100 000

D. *Hardware and software*

To capture their I/O behavior, each application is run on a single compute node, with 134GiB memory, an AMD EPYC 7H12 processor with 64 cores.
Our selected particular use-case of hierarchical storage is a burst buffer, a fast intermediate layer located between compute nodes and the slower backend storage, as displayed in figure 3. Their purpose is to allocate a temporary cache positioned between the computing processes and the permanent storage system. Burst of I/O can be redirected to this high bandwidth cache to avoid I/O bottlenecks during intensive I/O phases, such as checkpointing phases. The burst buffer absorbs burst of I/O and asynchronously performs the I/O on the permanent storage backend to reduce the job's stalling time and accelerate its execution. In our particular implementation, two layers of intermediate faster storage are available: RAM and NVME. We have selected three different setups in terms of NVME and RAM size, as described in table III.

TABLE III: Selected hierarchical storage characteristics for our experiments

Setup	RAM (Level 1) Size	NVME (Level 2) Size
1	1 GB	5 GB
2	10 GB	50 GB
3	100 GB	500 GB

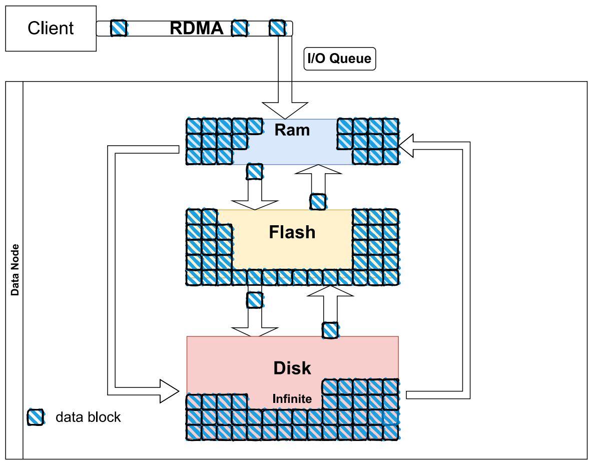

Fig. 3: Burst Buffer architecture

The FLC of each application is extracted using the FiLiP software, presented by Khelili and al. in [18], which allows us to extract on a file-per-file basis the type of each operation and its timestamp, as shown in listing 1.

E. *Implementation*

For quick simulation and comparison of our data movement policy to the state of the art, we have developed a simulator that processes requests as they arrive and simulates the execution of the eviction policy. We implemented LRU and LFU in their most optimal version, O(1), for all operations to allow fast experimentations. Our file-based eviction policy

Listing 1: Example of FileLifecycle

```
"file_1": [
    {
    "timestamp": 4096,
    "type": "WRITE"
    },
    {
    "timestamp": 4098,
    "type": "READ"
    },
]
```

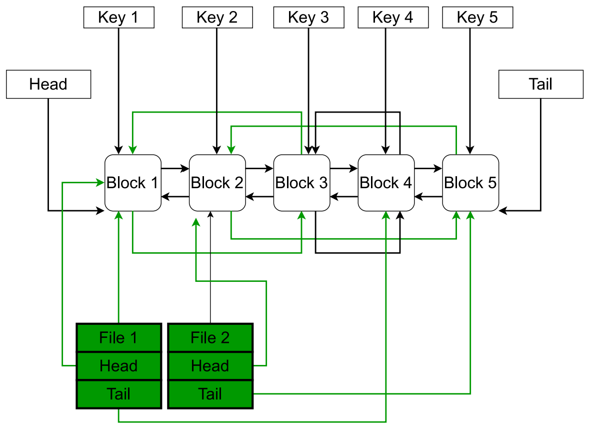

Fig. 4: File-based eviction data structure

also required the creation of a specific data structure to achieve O(1) execution for all operations.

Our data structure figure 4 is built upon the foundation of the LRU data structure (in black), incorporating a combination of a linked list and a hash map. This design allows an efficient insertion, deletion, and update operations with optimal time complexity. The linked list component is responsible for organizing the blocks based on their recent usage. It maintains the order of blocks, placing the most recently accessed block at the front of the list and the least recently accessed block at the back. This arrangement ensures that the migration of the least recently used blocks to a lower storage tier can be performed in constant time by simply removing it from the tail of the linked list.

To further enhance our file-level policy, we recognized that updating and deleting blocks belonging to the same file might result in a time complexity of O(n), where n represents the number of blocks. To address this concern, we introduced an additional level (green links) of the linked list that connects blocks belonging to the same file. This allows us to efficiently update or delete blocks associated with a specific file by traversing only the linked list related to that file, resulting in improved performance. The simulator takes as input the files lifecycles of an application, and returns the number of hits for the different tested policies.
The capacity of every tier of the hierarchical storage configuration can be adjusted, as well as the block size.

VII. Results and discussion

Figures 5 and 6 represent the FLC's of the three studied scientific applications. Each color in these figures represents a different FLC. In these figures, each color represents the FLC of a specific file and for each application we observe its prediction quality using accuracy, precision, and MCC metrics.

A. NAMD

The results on table IV show 100% accuracy on read operations prediction corresponding to a perfect prediction. The prediction accuracy is lower for writing operations compared to reading ones because the read activity is consistently low for the majority of the time with 2Gb of read data concentrated at the beginning of the application and no activity after that making it more predictable compared to the write activity. As shown in figure 5, after an initial reading phase corresponding to the application launch, the files are negligibly reused in subsequent reads. In the case of write operations, the profile exhibits higher levels of activity with 5Gb distributed throughout the entire execution of the application, presenting a greater challenge for accurate prediction. However, the accuracy remains relatively high, reaching 81% because of the contextual nature of file access. The files are accessed in a specific context, such as being used as input or output files, or being accessed periodically as can be seen in the same figure 6b: there are recurring peaks for the brown-colored file, indicating regular access. Additionally, the yellow file shows a single access towards the end of the execution corresponding to the results write.

TABLE IV: NAMD evaluation

	Metrics	NAMD		
		Accuracy	Precision	MCC
READ	Euclidean distance	100 %	100 %	100 %
	DTW distance	100%	100 %	100%
WRITE	Euclidean distance	81%	60 %	61%
	DTW distance	81 %	60 %	61 %

B. NEMO

The results in tables V show that NEMO's I/O patterns can be predicted with a good accuracy 77% on READ and 68% on write using both Euclidean and DTW distances. This quality of prediction correlation is likely due to the fact that NEMO is a data-intensive application that frequently accesses the same data files in a similar way.

In the case of WRITE operations, the accuracy, precision, and MCC are slightly better with the Euclidian distance despite its lower calculation time. The fact that the Euclidean distance produces better results than DTW suggests that the patterns do not undergo significant expansion or contraction that would provide an advantage to DTW. The results also highlight the importance of custom file placement policies for applications like NEMO, where file reuse is a significant factor. The predicted I/O patterns can be used to inform file placement decisions, such as placing frequently accessed files in the

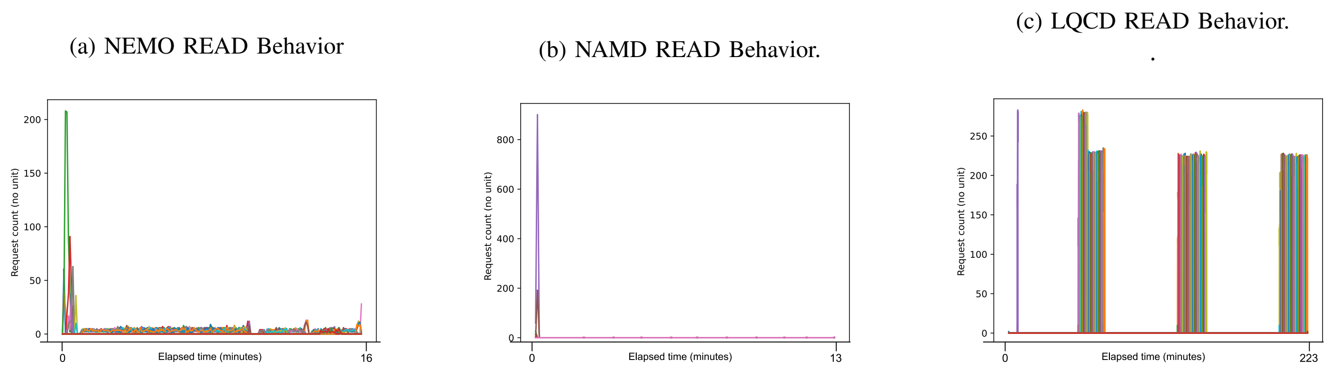

Fig. 5: FLC for each application filtered by read FLCevents. Each color represents a manipulated file.

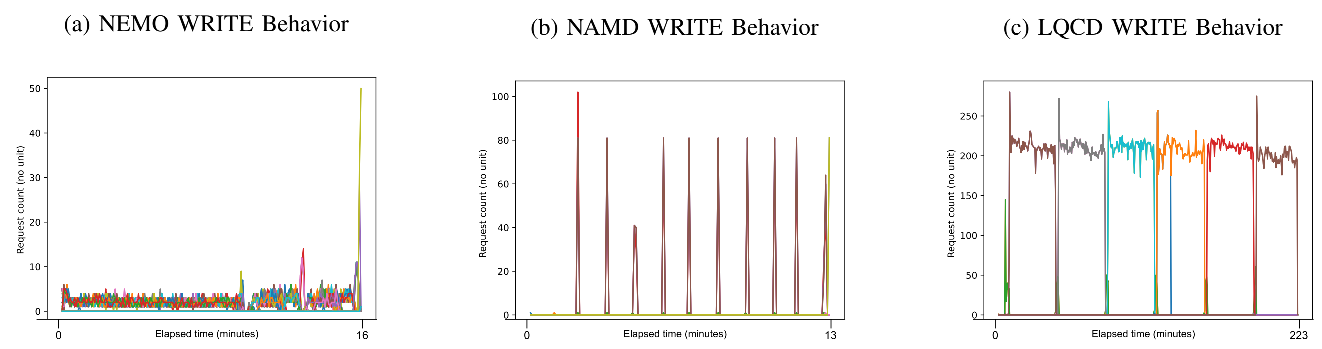

Fig. 6: FLC for each application filtered by write FLCevents

most efficient storage level using data pre-fetching or data eviction. Overall, the high prediction accuracy and the insights gained from analyzing NEMO's I/O patterns provide valuable information for improving the performance.

TABLE V: Nemo evaluation

	Metrics	NEMO		
		Accuracy	Precision	MCC
READ	Euclidean distance	77 %	53%	60%
	DTW distance	77 %	53%	60%
WRITE	Euclidean distance	68%	36 %	44%
	DTW distance	66%	33 %	41%

C. LQCD

Unlike NAMD, the activity in this case is almost similar for read (65GB) and write (50GB) operations, and they are both evenly distributed throughout the application execution . The results table VI shows that the prediction is relatively high in READ and WRITE operations with an accuracy reaching 96% for WRITE cases and 96% for READ cases, with a MCC of 93% . In the case of WRITE prediction,we can see that DTW performs slightly better than ED in terms of accuracy and precision, indicating that the patterns vary slightly during the execution of the application. While it may not be completely evident by observing figure 8, we can hypothesize that this difference is related to a narrowing/widening of patterns over time.

TABLE VI: LQCD evaluation

	Metrics	LQCD		
		Accuracy	Precision	MCC
READ	Euclidean distance	96 %	92 %	93%
	DTW distance	96%	92 %	93%
WRITE	Euclidean distance	95%	91 %	93%
	DTW distance	96 %	92 %	93%

D. Synthetic Benchmark

In terms of application profile, this case is inversely similar to NAMD. In this case, the read operations exhibit high and evenly distributed activity (42GB), while the write operations have low activity (1GB) only at the beginning of the application. In the figure 7 representing the FLC's filtered by READ, we can see that files represented by different colors exhibit patterns that widen over time. This observation suggests that the behavior of files varies or evolves as they are used. When I/O phases expand or contract as shown in figure 7, the DTW distance surpasses the Euclidean distance (ED) with a 86% accuracy against 84% for the ED and 73% againt 71% for precision, because it is designed to adapt to such patterns, making it more suitable for scenarios where I/O phases vary in length. Its ability to align and compare sequences with varying lengths allows it to capture temporal variations more effectively than the Euclidean distance. This adaptability makes DTW a preferred choice when dealing with time series data that exhibit variable patterns or temporal shifts.

TABLE VII: Benchmarking App evaluation

	Metrics	Benchmarking App		
		Accuracy	Precision	MCC
READ	Euclidean distance	84 %	71%	75%
	DTW distance	86%	73 %	77%
WRITE	Euclidean distance	100 %	100%	100%
	DTW distance	100%	100 %	100%

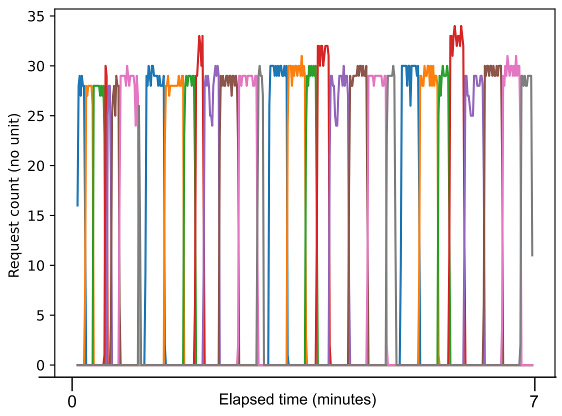

Fig. 7: FLC's of the synthetic application filtered by read FLC_Events. Each color represents a manipulated file.

E. Prediction time

In figure 9, we highlight the difference in execution time between the different algorithms, and we can observe that in both cases of Euclidean distance and DTW, the execution remains lower than ARIMA. It is worth noting that the execution times of the differentiation algorithms, required when using ARIMA in the case of non-stationary time serie are not included in these recorded times. Comparing our performance to existing methods in figure 9, we can see that our algorithm is faster than the reference time serie prediction method (ARIMA) on figure 9. Additionally, using the Euclidean distance metric reduces the execution time significantly compared to using the DTW distance metric. Because our window has a fixed size, we see no major improvement in performance between DTW and ED, and we recommend using ED for speed gain when the window has a fixed size.

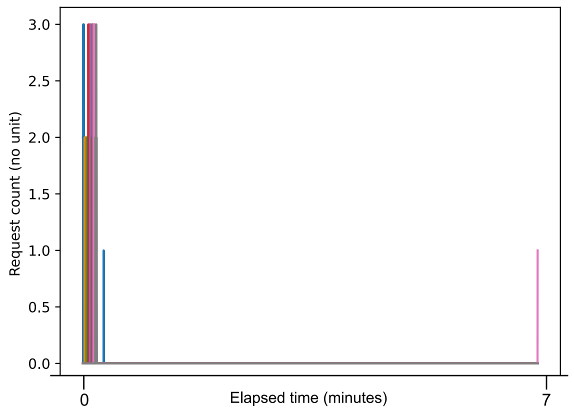

Fig. 8: FLC's of the synthetic application filtered by write FLC_Events. Each color represents a manipulated file.

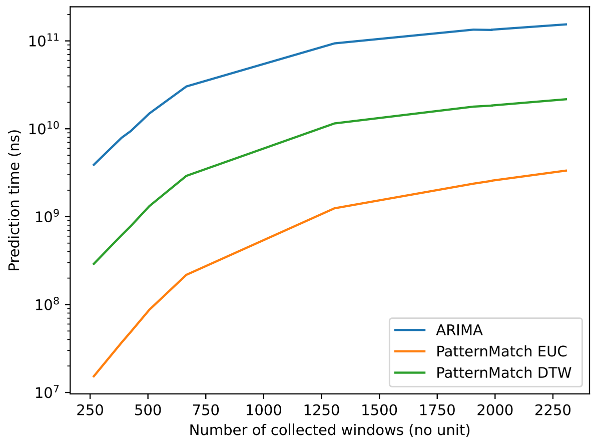

Fig. 9: Execution time of the algorithm as a function of the size of the time serie

F. Hit ratio and anticipated data placement

We compare here the hit ratio on the highest level of the storage hierarchy of the *Burst Buffer* using our File-based migration strategy (F-LRU) against the literature cache eviction policies: LRU and LFU. All of them were simulated using our developed tool.

We considered the three scenarios as detailed in table III with three different setups for the hierarchy of the *Burst Buffer*. We run the simulation using as input the traces generated by the four aforementioned applications. For comparison purpose, we generated a baseline scenario where the capacity of the highest tier is large enough to capture all the references. So, the fit ratio is the maximum and reaches its upper bound of each application under the assumption of an infinite highest tier. We normalize the number of hits on each tier by this maximum number of hits to obtain the percentage of successful hits out of the possible hits in an ideal configuration with an infinite first tier.

1) First Scenario: We can see on table VIII that the number of hits is improved using our policy for both NEMO and the synthetic benchmark application we developed. Specifically for NEMO, our File-based policy (F-LRU) outperforms both LRU and LFU with a better hit rate at the highest level of the hierarchy with the lowest latency. Note that, LFU performs better than LRU for the case of NEMO but conversely LRU takes the advantage in the case of NAMD. This is justified by the fact that in such a setup, recently accessed blocks are more reused than frequently ones for NAMD and vice versa for NEMO.

We observe also that the suitability of LRU or LFU is more dependent of the application whereas our policy remains stable and offers a better hit rate for both NEMO and Bench. This is because it does not consider only the blocks recency or frequency, but also the higher level via file reuse property which gives more information on the contextual usage of the file.

TABLE VIII: First setup hit comparison to LRU and LFU

		Policies		
	Levels	LRU	LFU	F-LRU
NEMO	**level1**	41%	43%	47%
	level2	57%	54%	52%
LQCD	**level1**	41%	41%	41%
	level2	0%	0%	0%
NAMD	**level1**	87%	88%	87%
	level2	12%	11%	12%
BENCH	**level1**	0%	0%	0%
	level2	0%	0%	4%

2) Second scenario: This scenario corresponds to a larger capacity for both the two highest tiers of the *burst Buffer* to represent a configuration with less restrictions on resources compared to the first scenario. In this case, we can observe a significant change for NEMO which idealy performs for the three policies because the capacity of the highest tier is large enough to accommodate more data, resulting in a higher hit rate (98% to 99%). On the contrary, for LQCD, we observe that LRU and F-LRU, with a hit rate of 97%, outperform LFU, which has a hit rate of only 50% because in this setup LQCD recently accessed blocks are more reaccessed than frequently ones. Regarding NAMD, all its data fit in the highest tier leading to a 100% hit rate. Finally, for the synthetic benchmark, our F-LRU policy performs better than the other two ones with a hit rate of 75% against 71% for LRU and LFU respectively. So, F-LRU performs at least as well as the best of LRU and LFU for all applications.

3) Third scenario: This scenario represents a configuration almost without restrictions on resources as the tiers storage capacities are significantly larger than the precedent scenarios. Consequently, the highest tier can capture all the IO traffic rising its hit rate to 100% and No data migration is needed.

TABLE IX: Second setup hit comparison to LRU and LFU

		Policies		
	Levels	LRU	LFU	F-LRU
NEMO	**level1**	99%	98%	99%
	level2	1%	2%	1%
LQCD	**level1**	97%	50%	97%
	level2	3%	49%	3%
NAMD	**level1**	100%	100%	100%
	level2	0%	0%	0%
BENCH	**level1**	71%	71%	75%
	level2	28%	28%	24%

TABLE X: Third setup hit comparison to LRU and LFU

		Policies		
	Levels	LRU	LFU	F-LRU
NEMO	**level1**	100%	100%	100%
	level2	0%	0%	0%
LQCD	**level1**	100%	100%	100%
	level2	0%	0%	0%
NAMD	**level1**	100%	100%	100%
	level2	0%	0%	0%
BENCH	**level1**	100%	100%	100%
	level2	0%	0%	0%

VIII. Conclusion and further works

The file-based approach shows great promise in accurately predicting I/O patterns, especially in terms of anticipating file reuse probabilities: we have shown that in terms of accuracy, the file-based approach achieves result ranging from 77% accuracy and 53% precision in the most difficult scenarios, to a perfect 100% accuracy and precision in cases where file behavior is easier to predict. Whenever comparing cache hits, the results we obtained are promising, demonstrating that our proposed F-LRU (File-based Least Recently Used) is at least as effective as traditional LRU and LFU and can increase the hit rate by a factor of 1.94 against LFU and 1.06 against LRU for LQCD and IO-Bench respectively.

The experimental evaluation results demonstrate that our algorithm achieves high prediction accuracy for file-level I/O patterns compared to existing techniques. Additionally, the algorithm's complexity is competitive with state-of-the-art methods, making it a practical and efficient solution for real-world scenarios.

In the near future, we plan to merge the patterns from different files, and calculate the weighted average of the nearest neighbors, which has been shown to yield interesting results in the literature [24] and can improve the data placement. We also want to consider inter-file and intra-file comparisons rather than only intra-file to obtain better prediction. The desired effect is to capture the inter-files similarity and thus improve prediction accuracy.

REFERENCES

[1] Daniel A. Reed and Jack Dongarra. Exascale computing and big data. In *Communications of the ACM*, volume 58, pages 56–68, 2015. DOI : 10.1145/2699414.

[2] Zaher Al Aghbari and Ayoub Al-Hamadi. Finding k most significant motifs in big time series data. *Procedia Computer Science*, 170:595–601, 2020. DOI: 10.1016/j.procs.2020.03.131.

[3] Grant Ayers, Heiner Litz, Christos Kozyrakis, and Parthasarathy Ranganathan. Classifying memory access patterns for prefetching. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 513–526, 2020. DOI: 10.1145/3373376.3378498.

[4] Sorav Bansal and Dharmendra S Modha. CAR: Clock with Adaptive Replacement. *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, page 15, 2004.

[5] Francieli Zanon Boito, Ramon Nou, Laércio Lima Pilla, Jean Luca Bez, Jean-François Méhaut, Toni Cortes, and Philippe OA Navaux. On server-side file access pattern matching. In *2019 International Conference on High Performance Computing & Simulation (HPCS)*, pages 217–224. IEEE, 2019. DOI:10.1109/HPCS48598.2019.9188092.

[6] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kalé, Bill Kramer, and Marc Snir. Toward exascale resilience. In *IJHPCA*, volume 23, pages 374–388, 2009. DOI : 10.1177/1094342009347767.

[7] Christine TH Davies, E Follana, A Gray, GP Lepage, Q Mason, M Nobes, J Shigemitsu, HD Trottier, M Wingate, C Aubin, et al. High-precision lattice qcd confronts experiment. *Physical Review Letters*, 92(2):022001, 2004. DOI :10.1103/PhysRevLett.92.022001.

[8] K Lalitha Devi and S Valli. Time series-based workload prediction using the statistical hybrid model for the cloud environment. *Computing*, 105(2):353–374, 2023. DOI: 10.1007/s00607-022-01129-7.

[9] Bin Dong, Xiuqiao Li, Qimeng Wu, Limin Xiao, and Li Ruan. A dynamic and adaptive load balancing strategy for parallel file system with large-scale i/o servers. *Journal of Parallel and distributed computing*, 72(10):1254–1268, 2012. DOI: 10.1016/j.jpdc.2012.05.006.

[10] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, S.H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12):1352–1361, December 2001. DOI: 10.1109/TC.2001.970573.

[11] Matthieu Dorier, Shadi Ibrahim, Gabriel Antoniu, and Rob Ross. Omnisc'io: a grammar-based approach to spatial and temporal i/o patterns prediction. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 623–634. IEEE, 2014. DOI :10.1109/SC.2014.56.

[12] Ohad Eytan, Danny Harnik, Effi Ofer, and Roy Friedman. It's Time to Revisit LRU vs. FIFO. *HotStorage'20: Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File System*, page 7, 2020.

[13] San Francisco. Proceedings of FAST '03: 2nd USENIX Conference on File and Storage Technologies. *Adaptive Replacement Cache (ARC)*, page 17, 2023.

[14] Felix Freitag, Julita Corbalan, and Jesus Labarta. A dynamic periodicity detector: Application to speedup computation. In *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, pages 6–pp. IEEE, 2001. DOI: DOI:10.1109/IPDPS.2001.924928.

[15] Al Geist and Robert Lucas. Major computer science challenges at exascale. *The International Journal of High Performance Computing Applications*, 23(4):427–436, 2009. DOI: 10.1177/1094342009347445.

[16] Song Jiang and Xiaodong Zhang. LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance. *ACM SIGMETRICS Performance Evaluation Review*, page 12, 2002. DOI: 10.1145/511399.511340.

[17] Rohit J Kate. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, 30:283–312, 2016.

[18] Adrian Khelili, Sophie Robert, and Soraya Zertal. Filip: A file lifecycle-based profiler for hierarchical storage. *INFOCOMMUNICATIONS JOURNAL*, 14(4):26–33, 2022. DOI: 10.36244/ICJ.2022.4.4.

[19] Julian Kunkel, Michaela Zimmer, and Eugen Betke. Predicting performance of non-contiguous i/o with machine learning. In *High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings 30*, pages 257–273. Springer, 2015. DOI: 10.1007/978-3-319-20119-1_19.

[20] Jay Lofstead, Ivo Jimenez, Carlos Maltzahn, Quincey Koziol, John Bent, and Eric Barton. Daos and friends: a proposal for an exascale storage system. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 585–596. IEEE, 2016. DOI: 10.1109/SC.2016.49.

[21] Dhruv Matani, Ketan Shah, and Anirban Mitra. An O(1) algorithm for implementing the LFU cache eviction scheme. Technical Report arXiv:2110.11602, arXiv, October 2021. DOI: 10.48550/arXiv.2110.11602.

[22] Ryan McKenna, Stephen Herbein, Adam Moody, Todd Gamblin, and Michela Taufer. Machine learning predictions of runtime and io traffic on high-end clusters. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 255–258. IEEE, 2016. DOI:10.1109/CLUSTER.2016.58.

[23] James Oly and Daniel A Reed. Markov model prediction of i/o requests for scientific applications. In *Proceedings of the 16th international conference on Supercomputing*, pages 147–155, 2002. DOI: 10.1145/514191.514214.

[24] R Keith Oswald, William T Scherer, and Brian L Smith. Traffic flow forecasting using approximate nearest neighbor nonparametric regression. Technical report, 2000.

[25] J. C. Phillips, D. J. Hardy, J. D. C. Maia, J. E. Stone, J. V. Ribeiro, R. C. Bernardi, R. Buch, G. Fiorin, J. Henin, W. Jiang, R. McGreevy, M. C. R. Melo, B. K. Radak, R. D. Skeel, A. Singharoy, Y. Wang, B. Roux, A. Aksimentiev, Z. Luthey-Schulten, L. V. Kale, K. Schulten, C. Chipot, and E. Tajkhorshid. Scalable molecular dynamics on CPU and GPU architectures with NAMD. *Journal of Chemical Physics*, 2020. DOI : 10.1063/5.0014475.

[26] Liana V Rodriguez, Farzana Yusuf, Steven Lyons, Eysler Paz, Raju Rangaswami, Jason Liu, and Giri Narasimhan. Learning Cache Replacement with Cacheus. *FAST '21*, page 15, 2021.

[27] F. Sevault, S. Somot, and J. Beuvier. A regional version of the NEMO ocean engine on the Mediterranean Sea: NEMOMED8 user's guide. Technical report, Meteo-France, CNRM, 2009.

[28] Mathieu Stoffel, François Broquedis, Frédéric Desprez, and Abdelhafid Mazouz. Phase-ta: Periodicity detection and characterization for hpc applications. In *HPCS 2020-18th IEEE International Conference on High Performance Computing and Simulation*, pages 1–12. IEEE, 2021.

[29] Pradeep Subedi, Philip Davis, Shaohua Duan, Scott Klasky, Hemanth Kolla, and Manish Parashar. Stacker: an autonomic data movement engine for extreme-scale data staging-based in-situ workflows. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 920–930. IEEE, 2018.

[30] Andrew S. Tanenbaum. *Modern operating systems*. Pearson, Boston, fourth edition edition, 2015.

[31] Houjun Tang, Xiaocheng Zou, John Jenkins, David A Boyuka, Stephen Ranshous, Dries Kimpe, Scott Klasky, and Nagiza F Samatova. Improving read performance with online access pattern analysis and prefetching. In *Euro-Par 2014 Parallel Processing: 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings 20*, pages 246–257. Springer, 2014. DOI : 10.1007/978-3-319-09873-9_21.

[32] Nancy Tran and Daniel A Reed. Arima time series modeling and forecasting for adaptive i/o prefetching. In *Proceedings of the 15th international conference on Supercomputing*, pages 473–485, 2001. DOI: 10.1145/377792.377905.

[33] Giuseppe Vietri, Liana V Rodriguez, Wendy A Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. Driving Cache Replacement with ML-based LeCaR. *HotStorage'18: Proceedings of the 10th USENIX Conference on Hot Topics in Storage and File Systems*, page 6, 2018.

[34] Lipeng Wan, Matthew Wolf, Feiyi Wang, Jong Youl Choi, George Ostrouchov, and Scott Klasky. Analysis and modeling of the end-to-end i/o performance on olcf's titan supercomputer. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1–9. IEEE, 2017. DOI: 10.1109/HPCC-SmartCity-DSS.2017.1.

[35] Xiaoqian Wang, Yanfei Kang, Rob J Hyndman, and Feng Li. Distributed arima models for ultra-long time series. *International Journal of Forecasting*, 2022. DOI : 10.1016/j.ijforecast.2022.05.001.