Data Popularity for Cache Eviction Algorithms using Random Forests

Olga Chuchuk[1,2,*] and *Markus* Schulz[1,**]

[1]CERN, IT Department, Geneva, Switzerland
[2]Inria, Côte d'Azur University, Sophia Antipolis, France

Abstract. In the HEP community the prediction of Data Popularity is a topic that has been approached for many years. Nonetheless, while facing increasing data storage challenges, especially in the upcoming HL-LHC era, there is still the need for better predictive models to answer the questions of whether particular data should be kept, replicated, or deleted.
Caches have proven to be a convenient technique for partially automating storage management, potentially eliminating some of these questions. On the one hand, one can benefit even from simple cache eviction policies like LRU, on the other hand, we show that incorporation of knowledge about future access patterns has the potential to greatly improve cache performance.
In this paper, we study data popularity on the file level, where the special relation between files belonging to the same dataset could be used in addition to the standard attributes. We turn to Machine Learning algorithms, such as Random Forest, which is well suited to work with Big Data: it can be parallelized, is more lightweight and easier to interpret than Deep Neural Networks. Finally, we compare the results with standard cache eviction algorithms and the theoretical optimum.

1 Introduction

The advancement of scientific research heavily relies on the efficient analysis of vast amounts of data generated by experiments in various domains. In the domain of High Energy Physics (HEP), the Worldwide LHC Computing Grid (WLCG) has been the essential infrastructure for processing and analysing data at the exabyte scale that has been produced by the Large Hadron Collider (LHC) experiments [1].

Traditionally, analysis jobs on WLCG use the *local grid analysis mode* [2]. Data is replicated or transferred to the processing site before computations begin, requiring extra attention and effort to track the location of data and replicas to ensure proper handling throughout the analysis process. Instead of pre-replicating or moving data, *remote computation analysis mode* autonomously fetches required data over the WLCG network, streamlining data processing. The remote computation analysis mode enables the usage of WLCG sites without permanent data storage. These sites function as processing units with temporary storage for currently working data, which is perceived as a cache for primary storage, simplifying site operation compared to those with permanent storage while also distributing workloads.

*e-mail: olga.chuchuk@cern.ch
**e-mail: markus.schulz@cern.ch

The efficacy of this cache system depends on its size and the cache eviction policy, which is an algorithm that decides which files to remove when it reaches full capacity. Our research explores the use of Machine Learning (ML) techniques to enhance cache eviction algorithms in the WLCG context. Rather than training the ML model to directly decide which files to evict from the cache, we take a two-stage approach. In the first stage, we use ML models to forecast future file reuse patterns. Armed with these predictions, we move to the second stage, where we integrate them into our cache eviction policies as essential parameters. This ensures that we leverage the strengths of both ML predictions and cache eviction algorithms, resulting in a more refined and robust caching system. By combining Belady's algorithm [3] and the Random Forest ML model's predictive capabilities [4], we aim to develop a more efficient and adaptive cache eviction policy that aligns with our ultimate goal of optimizing data retrieval and storage efficiency within the WLCG infrastructure.

Roadmap. The remainder of this paper is structured as follows. Research motivation and previous work are outlined in sections 2 and 3, respectively. Specifics of the ML-based solution are described in section 4, including the architecture, training and integration. Results are discussed in section 5. Finally, section 6 concludes the paper.

2 Motivation

For our cache study, we focused on analyzing user read requests for physics data analysis, since these requests are the least predictable part of storage access. In the context of WLCG, this corresponds to Analysis Object Data (AOD) and Derived AOD (DAOD) files [2]. We generated a trace of data read accesses specific to the ATLAS experiment [5], covering the time period from February to April 2022, using the CERN Data Center ATLAS EOS node [6]. The trace includes 5.6e8 read accesses, 1.2e7 files, and 2.9e5 datasets[1].

Our workloads and grid infrastructure are distinct from many existing caching studies, which usually cover content delivery networks (CDNs) and web environments (Web). Specifically, in our case, we deal with large-scale file sizes (see figure 1), a smaller number of users, and, consequently, profoundly different user access patterns (see figure 2). These unique characteristics indicate the necessity for the development of cache eviction policies tailored to our specific workload requirements.

Another key distinction lies in the optimization goal. While most of the scientific research papers deal with cached objects of the same size and, therefore, focus on optimizing File Miss Ratio (FMR) [7], our data involves considerably large file sizes with a wide distribution (figure 1). Hence, we are primarily interested in optimizing Byte Miss Ratio (BMR) - a metric that takes into account the impact of file size on cache performance. It is determined by dividing the total number of bytes that result in cache misses by the total number of bytes requested from the cache.

3 Previous work

LRU (Least Recently Used) is a commonly implemented and lightweight cache eviction policy [8]. In figure 3, we observe how BMR changes for LRU depending on the cache sizes, which are measured as a percentage of the total volume of unique files seen in the trace. As the cache size increases, the BMR metric decreases rapidly and eventually reaches an asymptote, representing the best possible performance any cache eviction policy can achieve on the given trace. This asymptote is determined by the number of cold misses, which indicates

[1] In ATLAS, analysis files are organized into datasets representing meaningful sets of data.

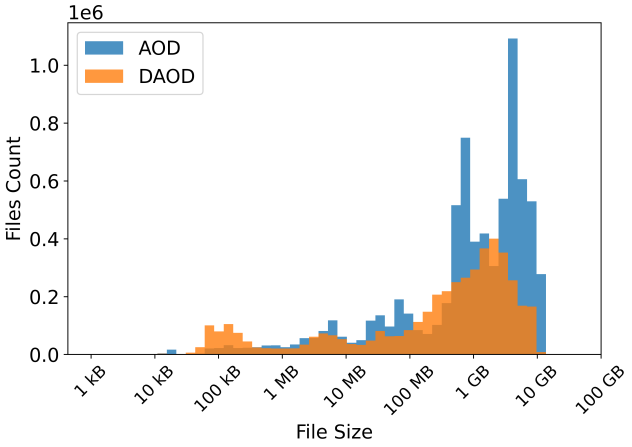

Figure 1. File size distribution. Average file size: 2.13GB. Maximum file size: 51.30GB

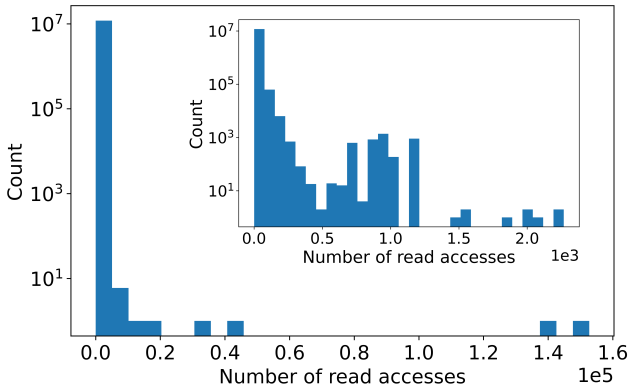

Figure 2. File popularity (excluding files with zero accesses). Maximum access count: 152,742. The inset graph zooms in on the 1 to 2,500 range.

the proportion of unique volume to the total requested volume, effectively representing the performance of an infinite-size cache.

LRU demonstrates commendable performance with favourable ease of implementation and maintenance. However, it raises the question of whether a better BMR can be achieved through the adoption of alternative cache eviction policies.

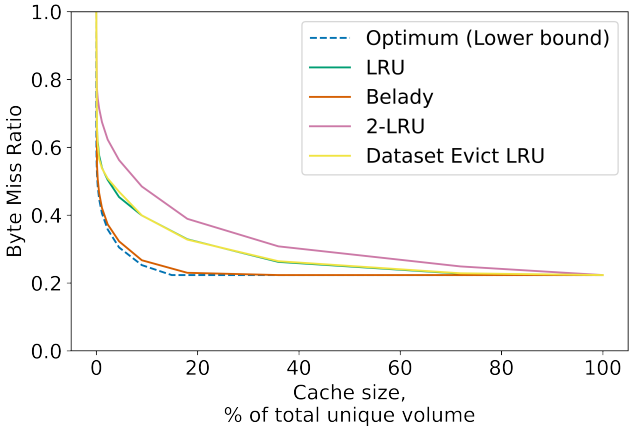

Figure 3. Visual comparison of non-ML cache eviction policies: LRU, 2-LRU [8], Belady's [3], Dataset Evict LRU, and PFOO-U.Bytes (a lower bound of the theoretical optimum) [7].

To explore the performance boundaries, we examine PFOO-U.Bytes [7], representing a tighter lower bound for BHR than an infinite-size cache [9]. In figure 3 it is marked as "Optimum (Lower bound)". Comparing PFOO-U.Bytes to LRU, we observe a noticeable scope for improvement, particularly for smaller cache sizes (less than 30% of the total volume).

In an attempt to close this gap, we have previously explored existing state-of-the-art policies, such as 2-LRU [8], MRU [10], GDSF [11], and a modification of AdaptSize [12] suitable for our case. Moreover, we have tailored cache eviction policies specific to our types of workloads, such as Dataset Evict LRU and Dataset Evict MRU [7], which leverage the observation that files within a dataset are often read together during analysis jobs. Despite our extensive exploration and efforts, we were unable to surpass the performance of LRU in previous attempts (see "2-LRU" and "Dataset Evict LRU" on figure 3 as an example).

4 ML-based solution

4.1 Architecture

A theoretical policy capable of predicting future file read accesses with certainty would be able to identify the optimal file for eviction at each step. When files have the same size, this policy is straightforward - it selects the file that will be accessed farthest in the future (known as Belady's algorithm [3]). However, when file sizes are different, determining the best eviction strategy becomes more difficult and is an NP-hard problem.

Intriguingly, on our trace, we discovered that Belady's algorithm behaves nearly optimally for our data, as depicted in figure 3. It is important to note that this algorithm is purely theoretical and requires knowledge of future file accesses, making it impossible for real-world implementations. For our caching solution, we decided to combine the nearly optimal performance of Belady's algorithm with the potential of Machine Learning models. This led us to adopt a two-stage architecture:

Firstly, we train a Random Forest ML model (RF) [4] using historical data to forecast future file read accesses. Secondly, we use the predicted access information to guide cache eviction decisions. Specifically, we incorporate the predicted time (or probability) of file reuse into our caching model. This integration enables us to make informed choices about cache eviction, prioritizing the removal of files less likely to be needed shortly.

Contrasting with existing efforts within the WLCG community, such as those employing Reinforcement Learning for caching tasks ([13]), our approach adopts the RF model. We favoured RF due to its ability to effectively handle both classification and regression tasks, along with its capability to be parallelized. Additionally, this algorithm provides interpretable insights into feature importance, making it a valuable tool for our analysis.

4.2 Training Random Forests

For constructing the features and target, we partitioned the data based on a time threshold, dividing the trace into approximately 75 and 15 days (corresponding to 79.33% and 20.67% of all read accesses, respectively). We tailored 18 features, including file and dataset sizes, frequencies, recency of file accesses, read access durations, and dataset-related characteristics. We opted to predict the logarithm of the reuse time, rather than the raw value, to facilitate interpretation and distinguish between files that will be reused within different time thresholds. The computational task remained nearly the same. The feature-target selection and the use of features not present for all files resulted in a reduction of the available data for training. Starting with over 10 million files, this filtering process resulted in 342,878 files (only 2.89% of the original number) containing all the necessary features for training the ML model.

In addition to the regression task, we reformulated the prediction objective as a binary classification task - determining whether a file would be read in the next 15 days. We used the same RF model architecture and features as in the regression task. After a slightly different filtering stage and an additional stage of balancing the prediction classes, the classification dataset comprised approximately 686,942 files, accounting for 5.79% of the original number.

We employed Root Mean Squared Error (RMSE) as the evaluation metric for regression, and both RMSE and Area Under the Receiver Operating Characteristic Curve (AUC-ROC) for classification. The use of RMSE as a regression-specific metric was justified by the caching model's utilization of the exact predicted probability, rather than the class, which we will demonstrate in subsection 4.3. Conversely, AUC-ROC is a standard classification-specific metric, indicating the quality of the True Positive Rate (TPR) vs. False Positive Rate (FPR) trade-off.

Through model tweaking and hyperparameter tuning, we achieved encouraging results, summarized in table 1 ("Previous Results"). The proximity of the training and test results indicate minimal overfitting. These compelling results validate the effectiveness of the ML approach and pave the way for enhancing our cache eviction policies.

Table 1: ML model performance comparison

	Previous Results	**Updated Results**
Regression		
RMSE on the training data (70%)	0.29	0.43
RMSE on the test data (30%)	0.34	0.46
Classification		
RMSE on the training data (70%)	0.11	0.19
RMSE on the test data (30%)	0.12	0.19
AreadUnderROC on the training data	0.99	0.98
AreadUnderROC on the test data	0.99	0.98

To gain insights into the Random Forest models' workings, we examined the feature importance, a valuable attribute that RF provides. Figure 4 demonstrates the top 5 features based on their importance for each ML model. This metric is derived from how frequently a feature is chosen for a node split and the resulting decrease in impurity at the decision tree nodes. The higher scores indicate a more substantial impact on the model's predictions.

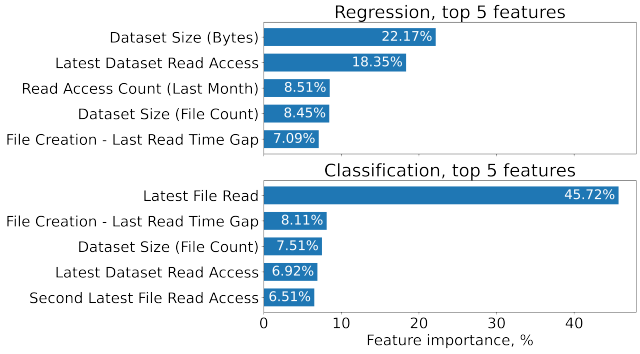

Figure 4. The top 5 features ranked by their feature importance for regression and classification models.

We made several interesting observations from the feature importance analysis. First, dataset-related features hold significant importance in both models. This suggests that the

dataset a file belongs to substantially influences its likelihood of being reused. Additionally, recency-based features, such as the last file and dataset read access times, play a critical role in models. They indicate a file's relevance and potential future reuse.

4.3 Integration into the caching policies

To extend the applicability of our ML models to the entire read access trace, we retrained them to operate with a reduced set of features: file size, time of last read access, duration of last read access, dataset size (volume), dataset size (number of files), and time of last dataset read access. Despite the reduced feature set, the adapted models still demonstrated robust performance in both regression and classification tasks (see table 1, "Updated Results"). The minimal decrease in performance indicates the resilience and flexibility of the RF models, allowing them to effectively handle the limited number of available features.

To incorporate predictive capabilities into the cache eviction algorithm, we implemented the "watermarks" method, which allows the cache's size to fluctuate within adjustable thresholds - a high watermark (set at 95%) and a low watermark (set at 80%). The high watermark ensures sufficient cache space for incoming files during the prediction process, preventing potential disruptions. Meanwhile, the low watermark serves as the limit for the cache purging process. To select watermark levels, we consider the rate at which incoming requests arrive to maintain adequate space between cleanups and the time required to complete predictions, requiring careful balance to prevent limiting the cache size.

Employing the watermarks approach for cache management effectively reduces CPU overhead by decreasing the frequency of cache cleanups (see Table 2) while minimally impacting system performance (see Figure 5). Furthermore, the watermark-based model conveniently integrates predictive models. Machine learning predictions are only triggered when the high watermark is reached, thus improving caching system performance. At this point, the predictive process runs for all cached files, which are subsequently sorted based on predicted reuse. For regression, sorting is done by predicted reuse distance, and for classification, by reuse probability within two weeks. Files are then purged until the low watermark is reached.

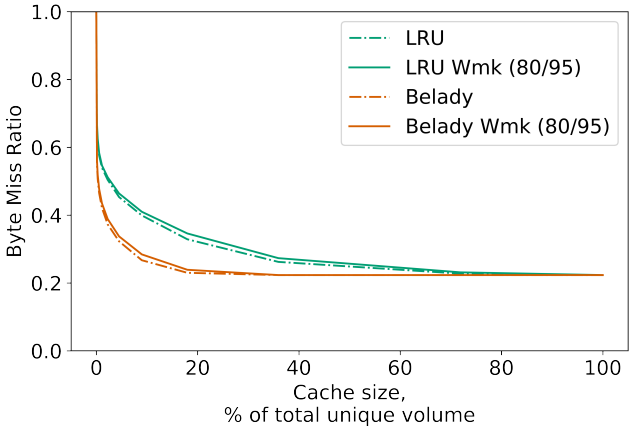

Figure 5. Visual comparison of the basic LRU and Belady cache eviction policies with their watermark-based counterparts (Wmk).

5 Experimental Results and Discussion

The results depicted in figure 6 reveal that the classification model performs somewhat better than the regression model, but still falls short of outperforming LRU. One possible explanation for the classification model's superior performance could be attributed to the fact that

Table 2: Watermarks implementation (Classification model)

Cache Size	Number of Cache Cleanups	Simulation Time (min)	Average RMSE
100%	0	13	-
72%	4	16	1.61
36%	18	20	1.79
18%	53	25	1.94
9%	136	36	1.99
4%	319	57	2.12
2%	724	101	2.39

with small cache sizes (2-5%), the distinction between files likely to be reused within several days versus several weeks or more becomes more crucial. In this context, the classification approach is better equipped to handle such distinctions.

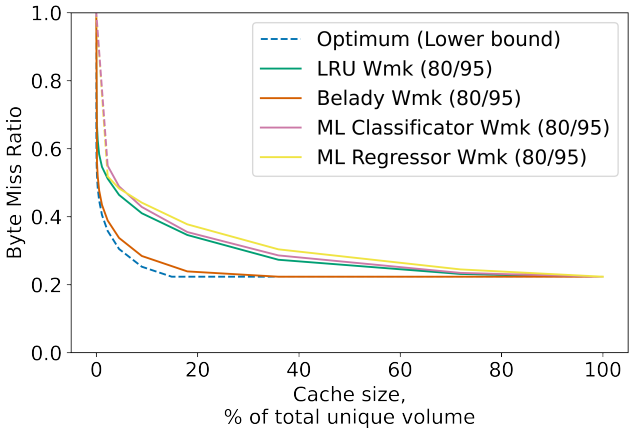

Figure 6. Visual comparison of LRU and Belady watermark-based models, and models integrating regression and classification predictions, along with the theoretical optimum lower bound.

Additionally, we conducted an analysis to understand why predictors with favourable scores did not lead to significant improvements in the cache eviction policy. It became evident that the performance of the trained models on the actual full trace was considerably worse than on the test data. Specifically, the average RMSE of the regression model for each cache size was notably higher than the RMSE of 0.46 obtained on the test data (as demonstrated in Table 2).

Several factors contribute to this discrepancy. Firstly, the model's training exclusively relied on files created within the restricted time frame. Moreover, a significant portion of files within this short time window were used only once (approximately 60%), a scenario that a regression model alone cannot adequately capture. Additionally, the training data underwent specific feature/label cuts, diminishing the diversity of training entries, and the prediction was constrained to a 15-day timeframe.

These findings reveal the limitations of our current implementation while highlighting several directions for further enhancing the ML-based solution, such as expanding the training dataset and optimizing model hyperparameters. This would allow the models to learn from more diverse and extensive data, which overall can lead to better predictive capabilities. Additionally, by exploring different ML model combinations, one could potentially find more effective ways to leverage the predictive power of ML for cache management.

6 Conclusions

After analysing cache eviction policies in the WLCG framework, the Least Recently Used (LRU) policy emerged as the most robust and efficient choice, despite exploring modern Machine Learning models for prediction. The feature importance distributions of the ML models revealed the significance of file recency and popularity, supporting the efficacy of recency-based eviction policies. Additionally, dedicating resources to optimizing eviction policies for minuscule cache sizes may not be practical in highly loaded distributed systems.

Despite not surpassing LRU's performance, the predictive model coupled with watermark-based implementation showcases a promising approach for cache eviction in the context of WLCG. By leveraging the predictive power of ML models and optimizing the cache cleanup process, our approach demonstrates the potential to enhance caching efficiency and reduce unnecessary cache evictions, ultimately benefiting the overall performance of the caching system. Furthermore, the model's capacity for generalization over extended periods would allow for a comprehensive understanding of global access patterns. Its lightweight nature also permits periodic retraining, ensuring its adaptability and relevance over time.

References

[1] Tech. rep., CERN, Geneva (2022), `https://cds.cern.ch/record/2802918`

[2] I. Bird, F. Carminati, R. Mount, B. Panzer-Steindel, J. Harvey, I. Fisk, B. Kersevan, P. Clarke, M. Girone, P. Buncic et al., Tech. rep. (2014)

[3] L.A. Belady, IBM Systems journal **5**, 78 (1966)

[4] L. Breiman, Machine learning **45**, 5 (2001)

[5] A. Collaboration, G. Aad, E. Abat, J. Abdallah, A. Abdelalim, A. Abdesselam, O. Abdinov, B. Abi, M. Abolins, H. Abramowicz et al., *The atlas experiment at the cern large hadron collider* (2008)

[6] G. Adde, B. Chan, D. Duellmann, X. Espinal, A. Fiorot, J. Iven, L. Janyst, M. Lamanna, L. Mascetti, J.M.P. Rocha et al., *Latest evolution of EOS filesystem*, in *Journal of Physics: Conference Series* (IOP Publishing, 2015), Vol. 608, p. 012009

[7] O. Chuchuk, G. Neglia, M. Schulz, D. Duellmann, *Caching for dataset-based workloads with heterogeneous file sizes*, in *ISGC 2022-International Symposium on Grids & Clouds 2022* (2022)

[8] V. Martina, M. Garetto, E. Leonardi, *A unified approach to the performance analysis of caching systems*, in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications* (IEEE, 2014), pp. 2040–2048

[9] Q. Huang, K. Birman, R. Van Renesse, W. Lloyd, S. Kumar, H.C. Li, *An analysis of Facebook photo caching*, in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), pp. 167–181

[10] N. Guan, M. Lv, W. Yi, G. Yu, ACM Transactions on Embedded Computing Systems (TECS) **13**, 1 (2014)

[11] L. Cherkasova, *Improving WWW proxies performance with greedy-dual-size-frequency caching policy* (Hewlett-Packard Laboratories Palo Alto, CA, USA, 1998)

[12] D.S. Berger, R.K. Sitaraman, M. Harchol-Balter, {*AdaptSize*}*: Orchestrating the Hot Object Memory Cache in a Content Delivery Network*, in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017), pp. 483–498

[13] T. Tedeschi, M. Baioletti, D. Ciangottini, V. Poggioni, D. Spiga, L. Storchi, M. Tracolli, Journal of Grid Computing **21**, 42 (2023)