Caching for dataset-based workloads with heterogeneous file sizes

Olga Chuchuk,[a,b,*] Giovanni Neglia,[b] Markus Schulz[a] and Dirk Duellmann[a]

[a] *CERN, IT Department, Geneva, Switzerland*

[b] *Inria, Côte d'Azur University, Sophia Antipolis, France*

E-mail: olga.chuchuk@cern.ch, giovanni.neglia@inria.fr, markus.schulz@cern.ch, dirk.duellmann@cern.ch

Caching can effectively reduce the cost of serving content and improve the user experience. In this paper, we explore the benefits of caching for existing scientific workloads, taking the Worldwide LHC (Large Hadron Collider) Computing Grid as an example. It is a globally distributed system that stores and processes multiple hundred petabytes of data and serves the needs of thousands of scientists around the globe.

Scientific computation differs from other applications like video streaming as file sizes vary from a few bytes to terabytes and logical links between the files affect user access patterns. These factors profoundly influence caches' performance and, therefore, should be carefully analyzed to select which caching policy to deploy or to design new ones.

In this work, we study how the hierarchical organization of the LHC physics data into files and groups of files called datasets affects the request patterns. We then propose new caching policies that exploit dataset-specific knowledge and compare them with file-based ones. Moreover, we show that limited connectivity between the computing and storage sites leads to the delayed hits phenomenon and estimate the consequent reduction in the potential benefits of caching.

International Symposium on Grids & Clouds 2022 (ISGC 2022)
21 - 25 March, 2022
*Online, Academia Sinica Computing Centre (ASGC), Taipei, Taiwan****

*Speaker

1. Introduction

The term caching covers an array of solutions at different levels of computing/communication architectures, ranging from microprocessor caches to local disk caches, to storage clusters in geographically distributed content delivery networks. The core concept is to place separate storage media closer to the end user (either an actual user or a processing unit), to reduce the time to serve user requests and/or the amount of data flowing between the main storage and the user.

In this paper, we study cache management in large, geographically distributed systems used for scientific computation. In particular, we consider as a use-case the Large Hadron Collider (LHC) located at CERN (the European Center of Nuclear Research), which is the world's largest particle accelerator. This unique and complex facility allows scientists to study elementary particles and fundamental laws of physics while posing specific storage and computing challenges. To solve them, scientists and engineers have developed the Worldwide LHC Computing Grid (WLCG), which is a federation of large and small computing centers distributed around the globe (in total, around 170 sites located on 5 different continents). The WLCG resources are predominantly shared by four large detector-experiments: LHCb, CMS, ATLAS, and ALICE, each of which is carried out by an international consortium.

The optimal use of the WLCG is crucial for the success of the whole LHC program, especially in light of the ongoing upgrade of the accelerator toward the new High Luminosity LHC (HL-LHC). This will make it possible to obtain higher energies and a much greater number of collisions, which may lead physicists to new scientific discoveries. The consequent increase in data volume and data access rates is expected to outpace the gain from upgrading the infrastructure with next-generation hardware: improving storage resource efficiency is necessary.

Contributions. We extracted a three-month trace of user accesses of the ATLAS experiment at the CERN data center (the largest of the WLCG sites). We analyze this trace, highlighting key findings relevant to the analysis and design of caching systems. Besides a highly heterogeneous distribution of file sizes (spanning 11 orders of magnitude), we note a strong request correlation across certain groups of files (called *datasets*), which are related to the same physics analyses. We then compare the performance on this trace of widely used caching policies, like LRU, 2-LRU [1–3] and clairvoyant optimal policies [4] that minimize either the File Miss Ratio (*FMR*) or the Byte Miss Ratio (*BMR*). We do so by estimating lower bounds for the minimum *FMR* achievable by clairvoyant reactive policies with the PFOO-L algorithm [4], and for the *BMR* with the new modified algorithm we propose (PFOO-L.Bytes). We observe abundant space for improvement between the standard techniques and the optimal performance.

Motivated by these findings, we propose simple modifications to LRU that evict or prefetch files based on dataset membership information (Dataset Evict LRU and Dataset LRU). Dataset LRU achieves a significant improvement and, thanks to prefetching, can even outperform clairvoyant reactive, i.e., non-prefetching, policies. Unfortunately, the improvement comes at the cost of increased load on the links connecting the cache to the main data store, in comparison to the classic LRU-like policies.

We study then the effect of limited connectivity between the cache and the main data store, which is particularly relevant in WLCG's remote computation setting, where CPU resources are geographically decoupled from the data storage. In this scenario, the increased latency to fetch the data leads to the phenomenon called 'delayed hits' [5], or also 'virtual hits' [6]. We observe that delayed hits negatively impact all caching policies (as expected) and Dataset LRU in particular, to the point that its performance gain may disappear in small cache size and/or small bandwidth settings. We characterize experimentally the regime where the adoption of Dataset LRU is beneficial.

Roadmap. The paper is organized as follows. Section 2 describes the WLCG architecture, reviews previous work on caching, and defines metrics of interest and known bounds for the performance of caching policies. In Sec. 3 we present our algorithm to evaluate the minimum achievable BMR for reactive policies. We describe the WLCG file access trace in Sec. 4. In Sec. 5 we compare the performance of file-based and dataset-based caching policies and evaluate the influence of delayed hits. Conclusions and future research directions are discussed in Sec. 6.

2. Background and Related Work

2.1 The WLCG as a use case for scientific workloads

Since the start of the Large Hadron Collider (LHC) in 2008, the Worldwide LHC Computing Grid (WLCG) has been serving the needs of the largest LHC experiments' detectors, i.e., ATLAS, CMS, LHCb, and ALICE. Yearly, the LHC generates more than 90 PB of data, with raw data rates up to 100 GB/s. Managing such data volumes requires the best approaches in computing and storage system architectures. Today, the WLCG appears as a system of interconnected smaller and larger computing facilities, fully or partially dedicated to the storage and computing tasks of the LHC experiments. Appendix A provides additional information about the WLCG architecture.

The typical WLCG data processing workflow can be roughly divided into reconstruction and analysis activities. The reconstruction activities are aimed at preparing the raw data coming from the LHC detectors for the physics users, and they are run through large data processing campaigns according to a precise schedule. On the contrary, analysis tasks work with the previously generated analysis files and are relatively sporadic, as they are independently triggered by different users according to their needs. In our studies, we concentrate on the analysis activities, which are less predictable and, therefore, not amenable to planned resources provisioning.

Currently, most of such analysis is performed in the so-called "local grid analysis" mode, where data expected to be processed is moved/replicated to the processing site before the computation starts. This mode requires manual intervention of the physics groups to curate the data transfer process and leads to an abundance of data replicas throughout the WLCG, which is incompatible with large data volumes generated by the future HL-LHC. For these reasons, in this paper, we evaluate a remote computation model, in which the processing unit automatically fetches data over the WLCG network based on users' requests. In this way, there is no need to couple the CPU resources with large (and expensive) permanent storage systems. Instead, working data can be placed in small-size caches, implemented through simple storage systems with a low level of service. With the introduction of such caching layer, more and more WLCG sites could be operated in this mode, reducing hardware and operation costs.

Another characteristic of the WLCG use case is that the analysis data formats have a particular hierarchical structure. The analyses process independent physics events, which are described as distinct records. Together, all the records meaningful for the same physics analysis form a logical entity called a dataset. Each dataset was divided into files based mainly on the storage and network system requirements and out of convenience for the users to process the data. As physics events are independent, records in the same file, as well as files in the same dataset, could be processed in any order, leading to embarrassingly parallel workloads.

2.2 Caching Policies

Caching policies determine which files should be stored or evicted from the cache. When the request process is stationary and file sizes are equal, LFU (Least Frequently Used) algorithm is the optimal non-anticipative reactive policy [1, 7]. At the same time, real workloads often exhibit non-stationary request processes with short-term correlations (often referred to as temporal locality [8]), which are present also in our trace (see Sec. 4). In these cases, LRU (Least Recently Used) policy often outperforms LFU. Up to this day, LRU is probably the most popular caching policy in production systems due to its low time and space complexity and significant improvement of system performance even for relatively small cache sizes [1, 2]. 2-LRU algorithm is similar to LRU, but only stores files after at least two requests, which helps to reduce cache pollution by unpopular files [1, 3]. Some algorithms that take into account both recency and frequency of file requests, for example, the Adaptive Replacement Cache policy (ARC) [9].

Caching with non-zero download delay. The actual time to retrieve files is usually discarded under the assumption that the throughput between the remote storage and the local cache is high enough to fetch each file before the next request arrives. In reality, the retrieval time might exceed the time difference between consecutive requests for the same file (the so-called reuse distance), which would cause a ‘delayed hit’ [5, 6]. Paper [5] demonstrates that the hit rate maximization does not necessarily lead to latency minimization when some hits are delayed, which shows the need for new latency-sensitive caching algorithms. In particular, they propose two LRU variants and showed how to compute analytically their relevant metrics of interest.

In our work, we evaluate the effect of delayed hits in the WLCG by including fetching delays in the performance evaluation.

Caching for heterogeneous file sizes. Existing literature on caches mostly considers the case of equal-sized files [1–3, 5–7, 9], often under the justification that files may be split into chunks with a given maximum size, and caching policies may work at the chunk level. In practice, the overhead of chunk management may be unacceptable and, even when it is not the case, chunks can still exhibit heterogeneous sizes, when file sizes vary from a few bytes to terabytes, as is the case in the WLCG. In this paper, we consider files as atomic cacheable units. LFU, LRU, and 2-LRU can also operate with heterogeneous file sizes with minor changes, but more sophisticated policies, in general, require a non-straightforward adaptation, e.g., in the case of ARC.

We dedicate part of our paper studying how to combine LRU with prefetching techniques to exploit available dataset information (Sec. 5.3). The existing policies that were specifically conceived to take into account file size, such as AdaptSize [10], GDSF [11], GD-Wheel [12]), and DynqLRU [13], could be extended in a similar manner.

Caching with prefetching. All policies described above are reactive, as they can only update the cache upon a miss by inserting the corresponding file. Existing correlations in the request process could be used to forecast which content will be requested in the near future and prefetch it.[1] This approach has been investigated especially for video streaming [14, 15], as chunks of video files are mostly read sequentially with the few exceptions of pausing, fast-forward, and rewind. As we show in Sec. 4, requests for files in the same dataset are highly correlated, but those files are read in arbitrary order, depending on the user and system software.

Caching for scientific workloads. Previous studies on caching for scientific computation often focus on the design and deployment of the caching infrastructure [16–18], rather than on the selection of well-suited caching algorithms. For example, in [16], the authors simply rely on XCache [19] and its internal implementation of the LRU policy. To the best of our knowledge, only paper [20] explores the usage of a caching technique significantly different from LRU for scientific workloads. The authors propose an adaptive caching solution that is only suitable for tasks with high re-execution rates, which are not present in the WLCG.

Alternatively, some works that explore how effectively the WLCG storage is used: at individual grid sites [21], or throughout the whole WLCG in a context of a single LHC experiment [22]. These papers also describe data access patterns potentially relevant for caching, but they do not directly investigate caching strategies.

In our paper, we also study data access patterns in the WLCG but focus on those characteristics that directly influence cache performance. Additionally, we propose new caching policies and compare their behavior with the existing ones under different scenarios (cache size, network connectivity throughput).

Metrics of interest. The standard metric used to evaluate the cache performance is the hit/miss ratio, but it fails to capture the different costs of different misses when file sizes are heterogeneous. Therefore, we distinguish the hit ratio, also called File Hit Ratio (*FHR*), and the Byte Hit Ratio (*BHR*). They can be defined as follows:

$$FHR = \frac{N_{cache}}{N}, \quad BHR = \frac{V_{cache}}{V}, \tag{1}$$

where N is the total number of requests (or the trace length), V is the total volume of the catalog, i.e., the total size in bytes of all files which have been requested at least once, N_{cache} is the number of files retrieved from the cache (the total number of hits), and V_{cache} is the total number of bytes served by the cache.

Respectively, File Miss Ratio (*FMR*) and Byte Miss Ratio (*BMR*) are calculated as:

$$FMR = 1 - FHR, \quad BMR = 1 - BHR. \tag{2}$$

Under different scenarios, one of these metrics can play a more important role than the other. For example, *FMR* is more relevant if the objective is to minimize the user delay and the retrieval time under a miss is almost constant (latency dominates the retrieval time). At the same time, *BMR* is more important for assessing the data volume transfer between the sites. From the definition, when the files have the same size, *FMR* and *BMR* coincide.

Lower bound for *FMR* of reactive policies. In the case of homogeneous file sizes, Belady's offline algorithm [23] achieves optimal *FMR* amongst reactive policies [2]. At each step, this algorithm evicts the file that will be requested the furthest in the future. Since it can only be

[1]Sometimes, the literature uses the expressions "caching policies" and "prefetching policies" to distinguish what we call "reactive caching policies" and "caching policies with prefetching," respectively.

calculated post-factum, practical implementation is not possible. However, in studies like ours, it can serve as a lower bound for the performance of practical reactive policies.

In the case of heterogeneous file sizes, let us denote the reactive caching policy that minimizes *FMR* (resp. *BMR*) as OPT (resp. OPT.Bytes). Both minimization problems are NP-hard [24], which means that in practice, finding the exact optimal policies is not feasible.

A simple lower bound for both FMR and BMR of reactive policies can be computed by simulating an infinite size cache [25]. This approach only quantifies cold compulsory misses, which inevitably occur when a file is requested for the first time. Instead, paper [4] proposes several algorithms to calculate lower and upper bounds for *FMR* of OPT. The flow-based offline optimal (FOO) lower and upper bounds presented by the authors are very accurate but computationally expensive; the practical FOO lower and upper bounds (PFOO-L and PFOO-U) work for hundreds of millions of requests, while still providing tight upper and lower bounds for OPT performance. Paper [26] presents yet another lower bound for *FMR* of non-anticipative policies, but only under some statistical assumptions on the request process. In our work, we use the PFOO-L lower bound and extend it to be able to compute a lower bound for OPT.Bytes' *BMR*.

3. Lower bound for *BMR* of reactive policies

While there are bounds for the optimal *FMR* of reactive policies, we are not aware of similar bounds for *BMR*. In this section, we propose a modification of PFOO-L, the PFOO-L.Bytes algorithm, which computes a lower bound for *BMR* of OPT.Bytes.

Remember that N denotes the total number of requests in the trace. The trace contains M unique files $f_1, f_2, \ldots, f_M$ with sizes $s_1, s_2, \ldots, s_M$, respectively. The i-th request can be represented by the pair $\{i, f_{j_i}\}$, where j_i is the identifier of the requested file. Let $T_{dif}[i]$ denote the reuse distance, i.e., the difference between the order of the future request for the same file and the current request. C denotes the cache capacity in bytes.

Similarly to [4], we represent the total cache resource by an initially empty rectangle with sizes N and C (the resources are limited in time and space). We can associate to each request $\{i, f_{j_i}\}$ a rectangle with height s_{j_i} and width $T_{dif}[i]$ and place it between i and $i + T_{dif}[i]$ on the time axis. Its area $s_{j_i} \times T_{dif}[i]$ corresponds to the total amount of cache resources that should be allocated to file j_i to avoid the following request (at time $i + T_{dif}[i]$) producing a miss.

The PFOO-L algorithm greedily picks the rectangles with the smallest area until all cache resources are consumed (the sum of the areas of the placed rectangles exceeds the global rectangle size, regardless of the overlaps). It, therefore, finds a lower bound for *FMR* of reactive policies, since no other reactive caching algorithm can get fewer misses using $N \times C$ total resources. In particular, the rectangles selected by PFOO-L may overlap in such a way that the required instantaneous capacity (the sum of the rectangles' heights) exceeds the constraint C.

We now describe how to adapt PFOO-L to find a lower bound for the minimum *BMR* (PFOO-L.Bytes). While in PFOO-L every selected rectangle brings a gain equal to 1 as it prevents a miss, in PFOO-L.Bytes, the rectangle corresponding to the request $\{i, f_{j_i}\}$ has associated gain s_{j_i}, i.e., equal to the bytes it prevents from downloading. This leads us to a knapsack problem, where each file is associated with a cost to store it and a potential gain. We can then lower-bound *BMR* by greedily selecting the rectangles with the best gain/cost ratio until we run out of caching resources.

The complete algorithm PFOO-L.Bytes is presented in Algorithm 1. Arrays R, T and S and initialized with the request sequence, the order or requests (or request time), and the sizes of the request files, correspondingly. The first step is to find the order of the next request $T_{next}[i]$ for the same file f_{j_i} for each request $\{i, f_{j_i}\}$ of the trace (line 7). If the file f_{j_i} is accessed for the last time, $T_{next}[i] = \infty$. Next, the reuse distance between consecutive requests $T_{dif}[i]$ to the same file f_{j_i} is calculated (line 8). We then find the rectangles' sizes (line 9), sort them by density, i.e., the gain/cost ratio (line 10), and add them to the cache while there are enough caching resources (lines 11–17). Since Alg. 1 is not an actual caching algorithm, but only finds a lower bound of the optimal performance, we allow caching fractions of the files. In the function `cache_file`, the second argument indicates the fraction of the file that needs to be cached. The final step is to add a fraction of the first rectangle that did not fit (lines 18–20). Note how we are implicitly solving the fractional knapsack problem, and then we either match or exceed the optimal solution of the original knapsack problem [27, Sec. 5.1.1].

Algorithm 1 PFOO-L.Bytes

```
1:  $R = [], T = [], S = []$
2:  for $i = 1..N$ do
3:      append($R, f_{j_i}$)
4:      append($T, i$)
5:      append($S, s_{j_i}$)
6:  end for
7:  $T_{next} \leftarrow$ find_next_access($R$)
8:  $T_{dif} = T_{next} - T$
9:  $I = S \times T_{dif}$
10: $I \leftarrow$ sort_by_density($I, S$)
11: $i = 1$
12: $P = N \times C$
13: while $i \leq N$ and $I[i] \leq P$ do
14:     cache_file($i, 1$)
15:     $P = P - I[i]$
16:     $i = i + 1$
17: end while
18: if $i \leq N$ then
19:     cache_file($i, P/I[i]$)
20: end if
```

Both PFOO-L and PFOO-L.Bytes can be constructed through a single pass over the trace for different cache sizes. In this case, the preprocessing steps (finding the time of the next access and sorting the intervals accordingly) take $O(N \log N)$ time and $O(N)$ space, and iterating over the trace takes $O(N)$ both in time and space.

4. Workload Characteristics

In this paper, we consider the local grid analysis workflow of ATLAS, the largest LHC experiment, at the CERN data center, the largest WLCG site. From the storage logs, we generated a trace of all read file accesses for a period of three months (01/01/2020–31/03/2020). We intend to make these data publicly available later this year.

Table 1 provides an overview of this trace by comparing the analysis and the total read workloads. Analysis files represent only 25% of the files/operations in the catalog but contribute around 60% to the catalog volume and the total read workload (the sum of requested bytes).

	Total	Analysis
Number of files (N)	36,774,178	9,152,849 (24.9% of total)
Volume of files (V)	32.19 PB	19.1 PB (59.3% of total)
Number of operations	173,181,554	45,931,029 (26.5% of total)
Total read workload	91.54 PB	55.46 PB (60.6% of total)

Table 1: Comparison of the total and analysis read workloads

Figure 1 shows the file size distributions for the complete catalog and for the set of analysis files. Both distributions peak at around 1 GB and have very large support spanning 11 and 10 orders of magnitude, respectively. File sizes extend to 470.73 GB, with the median of 40.78 MB. In comparison, the maximum size of analysis files is lower (67.16 GB), but the median is higher (824.38 MB).

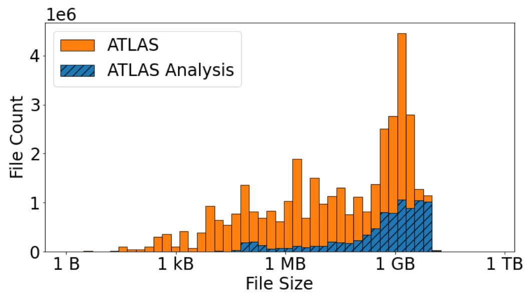


Figure 1: File size distribution.

Additionally, we plot the distribution of the analysis file popularity (the number of accesses) (Fig. 2) and estimate to which extent files with a given size contribute to the system workload (Fig. 3). By comparing them, we observe that very small files (<10 kB) tend to be accessed more often than the rest, while the system workload is almost entirely determined by the files in the range 100 MB–10 GB.

We conclude that the file size cannot be neglected: while very small files contribute the most to the number of accesses, most of the system load originates from the larger files.

Another important characteristic of the trace is the request load variability over time (Fig. 4). The byte rate averaged over one second varies from a few bytes to several terabytes per second (the mode of the distribution is around 20 GB/s), with an average request rate over the whole period equal to 9.91 GB/s.

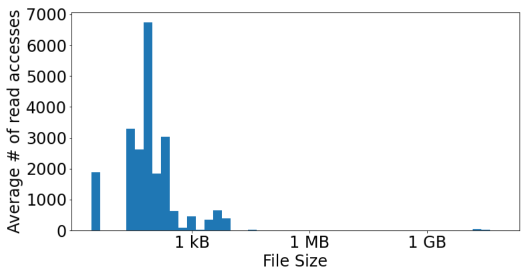


Figure 2: File popularity distribution.

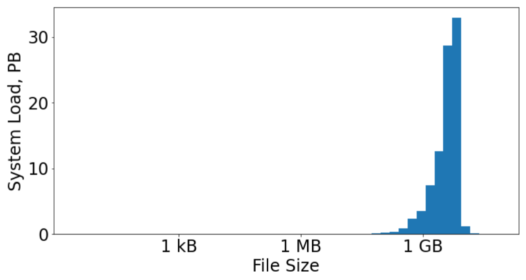


Figure 3: System load distribution.

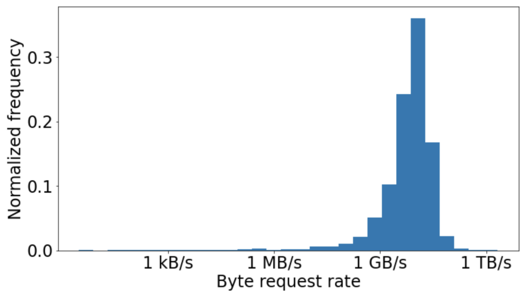


Figure 4: Distribution of byte request rate averaged per second.

The three-month trace exhibits temporal locality (Fig. 5). The distribution appears multimodal. Even though the average time between two consecutive read requests for the same file is around 3 days, 27.44% of consecutive accesses happen within a minute, and 6.47% happen within one second.

Remember that all the records meaningful for the same physics analysis are spread across different files and together form a logical entity called a dataset. We studied the properties of datasets and their access patterns. The distribution of dataset sizes is more heterogeneous than that of individual files (Fig. 6). The most common dataset size is around 1 MB, while the average is almost 80 GB. On average, there is a modest number of files in a dataset (the mean is 38, and the median is only 3).

To understand if files within the same dataset tend to be accessed together, we performed the following analysis. First, we looked at the fraction of each dataset (in terms of the number of files) accessed over the whole three-month period. For 75% of the datasets, all files were accessed; on average this fraction is 83%. Moreover, we quantified the variability of the number of accesses for files within the same dataset. For most datasets (92% of them), the standard deviation of the

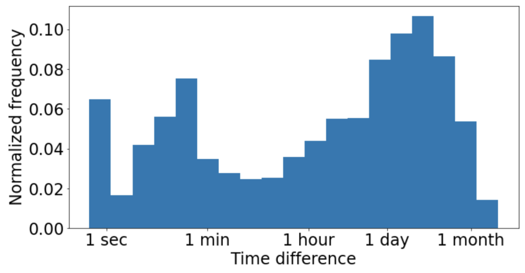


Figure 5: Distribution of time intervals between consecutive requests for the same file.

number of file accesses is at most 1. We conclude that when a user accesses a dataset file, (s)he is also likely to access all other files in the same dataset.

The order of the individual file accesses is difficult to reconstruct since log files only show the aggregate request process, where multiple users may access the same dataset simultaneously, and the same user can access different parts of the same dataset in parallel (for example, when the data being processed is coordinated by Rucio, the distributed data management system widely used in the WLCG [28]).

From all of the above, we conclude that when a dataset is accessed, most of its files are read, while not necessarily in any specific order.

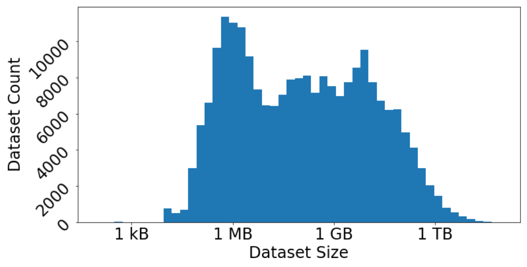


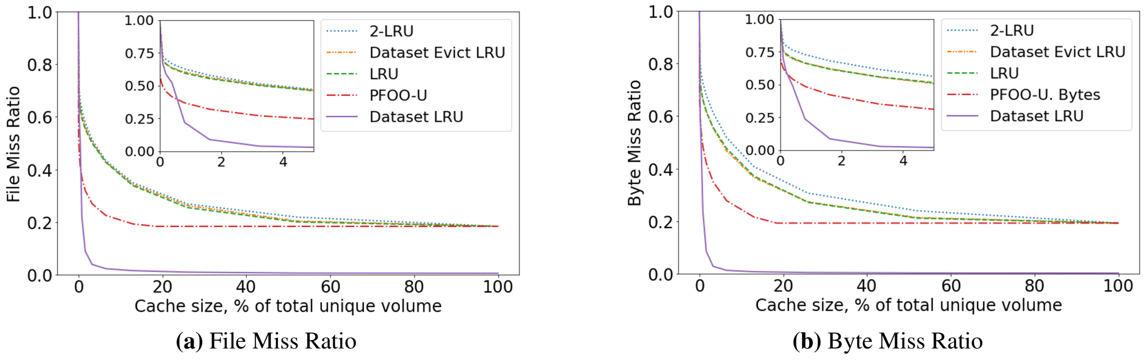
Figure 6: Dataset size distribution.

5. Caching for the WLCG

5.1 Caching policies

In this section, we evaluate the performance (*FMR* and *BMR*) of an array of different caching policies. We consider the classic LRU, as well as its 2-LRU variant, which has been shown to outperform LRU in many cases [1, 29]. Both policies can profit from the high level of temporal locality present in the trace (Fig. 5).

In addition, we propose and evaluate new caching policies that take advantage of some specifics of the WLCG workload. In particular, these policies rely on the dataset membership information, which was reconstructed for each file using an additional data source, Rucio metadata. As datasets are mostly read entirely, if some files in the dataset are currently in use, one may expect that the other files within the same dataset will be accessed in the near future. In particular, we propose the following policies, which preserve LRU's low complexity and are then particularly suited to serve high-rate request processes.



(a) File Miss Ratio **(b)** Byte Miss Ratio

Figure 7: Miss ratios vs cache size for different caching policies.

Dataset Evict LRU/MRU. These policies insert a file into the cache only upon a miss for that file (as LRU does). They maintain information about the last access to a dataset and, when the space is needed, they start evicting files belonging to the least recently accessed dataset. Among the files within this dataset, Dataset Evict LRU first evicts the least recently accessed files, while Dataset Evict MRU evicts the most recently accessed ones. We stress that both policies operate on the file level.

Dataset LRU. This policy relies on prefetching, as, upon a file miss, all files belonging to the accessed dataset are retrieved from the remote server and stored in the cache. Similarly, when cache space is needed, all files of the least recently accessed dataset are evicted. In short, this policy operates on individual files (remember that datasets are only logical entities unknown to the underlying file system), but practically behaves as LRU would if datasets were the atomic cacheable units.

We have also tested a variant of Dataset LRU that is less aggressive upon eviction. This policy does not evict entirely the least recently accessed dataset, but only as many of its files as needed. While this variant better uses the available storage space and achieves smaller *FMR* and *BMR*, the improvement is negligible (in the order of 10^{-4}).

5.2 Performance comparison

We evaluated the performance of the caching policies by simulating the cache behavior using the same three-month trace reflecting the request process. In each case, the initial state of the system is an empty cache.

Figures 7a and 7b compare how *FMR* and *BMR* change depending on the cache size, which is measured as a percentage of the total volume of unique files seen in the trace. PFOO-U and PFOO-U.Bytes algorithms provide lower bounds for the performance of any reactive caching policy. Dataset Evict LRU and Dataset Evict MRU show a negligible difference in performance in the order of 10^{-3}, so we show results only for the first of them.

These plots show that among the reactive techniques we considered, LRU remains the best option. Differently from what was observed in many previous studies [1, 29], 2-LRU results in significantly worse *BMR* and almost the same *FMR*. Curves for LRU and Dataset Evict LRU almost coincide. As most of the files in the same dataset are accessed consecutively, files in the least recently accessed dataset are to a large extent also the least recently accessed files. Cache states are then almost identical for LRU and Dataset Evict LRU.

Even though we have not found a reactive technique that performs better than LRU, the comparison with the optimal bounds suggests that there is a large room for potential improvement. For example, for the 5–10% cache sizes, the miss ratios could be reduced by a factor of 1.5.

As expected, all the reactive techniques (LRU, 2-LRU, Dataset Evict LRU, PFOO-U, PFOO-U.Bytes) provide the same performance when the cache can store the whole catalog. In these cases, the misses occur only the first time a file is requested (cold misses) and are not due to space constraints. In contrast, thanks to prefetching, Dataset LRU can avoid most of the cold misses. It results in the lower miss ratios in general (starting from the cache size around 1%), and specifically for the 100% cache size. Dataset LRU performs better than the lower bounds of reactive policies performance even for 3–5% cache sizes; this suggests that the total volume of the active datasets at any particular moment in time is low. At the same time, the miss ratio reduction comes at the price of increased accumulated transfer volume (Fig. 8).

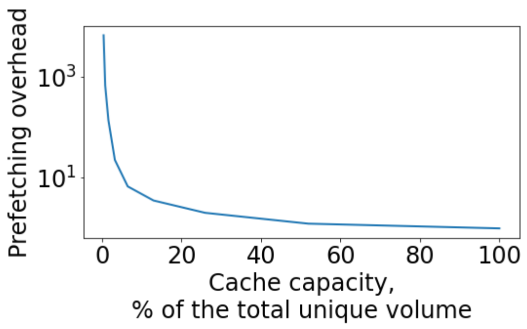


Figure 8: Ratio between accumulated transferred volumes of Dataset LRU and LRU (prefetching overhead).

5.3 Limited connectivity throughput

So far, we have been assuming that the throughput between the processing grid site and the data source is large enough to make the content retrieval time negligible in comparison to the inter-request arrival times. In reality, as described in Sec. 2.2, limited connectivity could lead to delayed hits and significantly influence the cache performance. To estimate this effect, we conducted an array of experiments that emulate the retrieval process under throughput constraints and compare the performance of LRU, as the best reactive policy, and Dataset LRU, which showed the best performance previously. We expect delayed hits to penalize Dataset LRU more, as prefetching leads to retrieving larger volumes of data.

The backhaul link and the main storage are modeled as a single-server FIFO queue [30] with a constant service rate (corresponding to the throughput) and customers in the queue are the objects to be retrieved (a file in the case of LRU, a dataset in the case of Dataset LRU). We refer to such a queue as the 'loading queue.' Upon a miss for an object, the retrieving job is added to the queue if not already present, i.e., if there is not already a pending request for that object. Objects are inserted into the cache when their service is completed.

Figure 9 shows how the size of the queue changes over time under LRU and a 100 Gbit/s network throughput. With this level of connectivity, the network is rarely congested: the queue occupancy is close to zero most of the time, with some peaks appearing only for small cache sizes like those reported in Fig. 9. Such peaks correspond to clusters of misses which require the retrieval of large volumes of data.

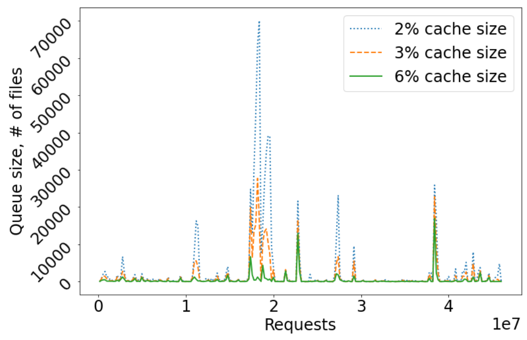


Figure 9: Queue size over time (requests) for 100 Gbit/s throughput and LRU.

As we decrease the throughput or switch to Dataset LRU, there are settings for which requests would require to retrieving content at a rate that constantly exceeds the connectivity throughput: the queue occupancy keeps increasing over time. In these pathological cases, the numerical values for *FMR* and *BMR* are not representative (e.g., they heavily depend on the trace length) and are then represented through dashed lines in the *BMR* plots (Fig. 11a and 11b).

Figure 10 shows how the hit rate changes over time for different cache sizes. The squares on the curve designate the first time instant when the cache is full. Lower throughput and larger cache sizes shift these points further to the right. In order to eliminate the effect of the initial transient, we considered the first 40% of the trace as a warm-up period and evaluated the metrics of interest only on the final 60%. In this manner, we guarantee that the cache is full for all the considered set-ups (different throughputs, cache sizes, caching policies), but for the cache sizes closely approaching 100%.

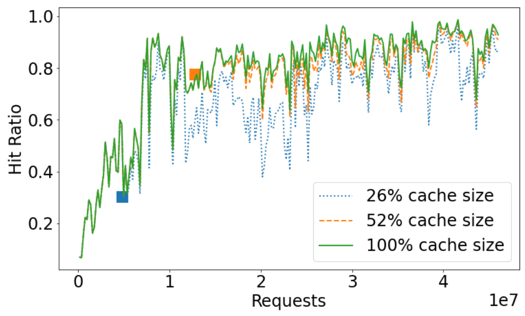
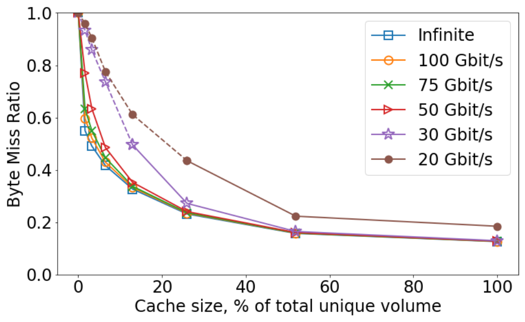
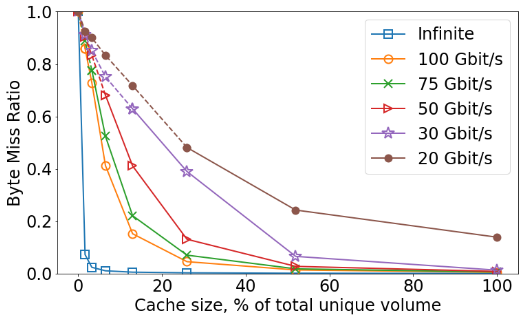


Figure 10: Hit ratio over time (requests) for 30 Gbit/s throughput and LRU. A square indicates the first time instant when the cache is full.

The effect of throughput constraints on LRU performance is illustrated in Fig. 11a. We observe that as far as the system provides a throughput of at least 50 Gbit/s, delayed hits have little effect on *BMR*, which is very close to the ideal case of infinite throughput. At the same time, the throughput of 30 Gbit/s shows a significant difference in performance with small cache sizes (20% and less). For 20 Gbit/s the miss ratio is larger for all cache sizes.



(a) *BMR* for LRU



(b) *BMR* for Dataset LRU

Figure 11: *BMR* vs cache size for different throughput values (miss ratios are computed on the final 60% of the trace). Points connected by dashed lines correspond to the settings where the loading queue is constantly growing.

In contrast, the effect of the limited throughput on the performance of Dataset LRU—the best policy in the infinite throughput regime (Fig. 7b)—is more profound (Fig. 11b). In fact, upon a miss, Dataset LRU retrieves a much larger amount of data than LRU (compare dataset sizes to file sizes, Fig. 1 and 6). With a 13% cache size, even a 100 Gbit/s throughput, which had virtually no influence on LRU performance, leads Dataset LRU to experience 0.15 *BMR* from just 0.01 for the infinite throughput case. Dataset LRU still outperforms LRU in this scenario, but the relative performance improvement is significantly reduced. For even smaller throughputs and/or smaller cache sizes, the effect of delayed hits annihilates the advantage from prefetching, and Dataset LRU starts performing worse than LRU.

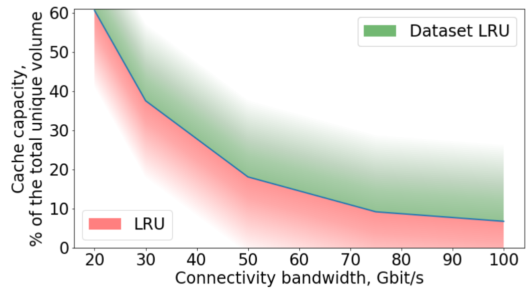


Figure 12: Caching policy minimizing *BMR* for different throughputs and cache size values.

From these results, we conclude that the throughput must be taken into account when developing cache eviction policies for the remote computations model. The limited throughput worsens the performance of cache algorithms, not proportional to the results under the infinite throughput assumption. Figure 12 shows which caching policy should be preferred, depending on the throughput and the cache size.

6. Conclusions and Future Work

In this work, we studied the existing scientific workloads of the CERN data center in the context of caching. Specifically, we analyzed the access patterns of a three-month trace of ATLAS, the largest of the LHC experiments. We focused on the user read requests for analysis data, as they correspond to the most unpredictable part of the storage accesses. We explored request rates, time locality, file popularity, and system load in relation to the file size, as well as intra-dataset access patterns.

Trace analysis motivated the selection of existing caching policies (LRU, 2-LRU) that can take advantage of temporal locality characteristics and operate with heterogeneous file sizes. We compared them to the new cache policies specifically tailored for dataset-based workloads (Dataset Evict LRU and Dataset LRU). We evaluated the cache performance with respect to *FMR* and *BMR*, and proposed the PFOO.Bytes algorithm to compute a lower bound for the Byte Miss Ratio for any reactive caching policy.

Dataset LRU heavily relies on prefetching and provides significant improvements in terms of both *FMR* and *BMR* compared to the other policies but raised our concerns about the possible high number of delayed hits. To estimate their influence, we ran simulations of LRU and Dataset LRU policies with network throughput values from 20 to 100 Gbit/s. We showed that Dataset LRU loses its advantage over LRU when the throughput is not sufficient to sustain the higher data volumes retrieved by Dataset LRU. In the future, we plan to study a new variant of Dataset LRU where the amount of prefetching is dynamically adapted as a function of the observed retrieval time.

Our research investigates a three-month period, which is enough to capture the lifespan of individual files but is not representative of yearly data access from the physics community (for example, massive preparation for big physics conferences, end-of-the-year closure of CERN, etc.). Notably, a tangible change in data access patterns happens when the LHC starts running, and the WLCG enters a so-called ‘data-taking’ period, the next of which is foreseen to start in late spring/summer of 2022. We plan to move to these data as soon as they become available. Moreover, on a longer time horizon, we want to develop a model to evaluate the global cost of caching (storage, latency, network) and inform future storage architectural design and deployment.

A. The WLCG Architecture

The WLCG is a three-tier hierarchical global federation of computing centres. Tier 0 (or T0), is the CERN data center—the largest facility within the network. Tier 1 consists of 13 computer centres located in Europe, Asia, and North America. Each of them has a direct high-capacity connection to T0 using the LHCOPN (LHC Optical Private Network) [31]. T0 and T1 provide data stewardship and processing. There are about 155 Tier 2 sites around the globe. They vary in size from a dozen of servers to data centres. T2 and T1 sites are connected using public networks and LHCONE (LHC Open Network Environment) [31], which provides dynamic point-to-point virtual connections. In this way, each T1 is connected to T0, and each T2 is connected to one or more T1s, but there are also connections between some sites belonging to the same Tier. Roughly 1/5th of the resources are at the T0, 1/5th at the T1 level, and 3/5th are provided by the T2. Furthermore, there are additional compute resources, which range from small clusters to large national analysis facilities, that are not curated by the WLCG community. These are often referred to as Tier 3 and are almost exclusively used for physics analysis.

Data serving and computing tasks are distributed within the whole WLCG network, depending on the specific needs of each of the large LHC experiments. However, only T0 and T1 sites are responsible for data archiving: T0 maintains one copy of the data and all T1 sites together provide a second copy. T0 and T1s share the load of operating the centralized coordination services at a high service level.

References

[1] M. Garetto, E. Leonardi and V. Martina, *A unified approach to the performance analysis of caching systems*, *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* **1** (2016) 1.

[2] R.L. Mattson, J. Gecsei, D.R. Slutz and I.L. Traiger, *Evaluation techniques for storage hierarchies*, *IBM Systems journal* **9** (1970) 78.

[3] A. Montazeri, N.R. Beaton and D. Makaroff, *LRU-2 vs 2-LRU: An Analytical Study*, in *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pp. 571–579, IEEE, 2018.

[4] D.S. Berger, N. Beckmann and M. Harchol-Balter, *Practical bounds on optimal caching with variable object sizes*, *Proceedings of the ACM on Measurement and Analysis of Computing Systems* **2** (2018) 1.

[5] N. Atre, J. Sherry, W. Wang and D.S. Berger, *Caching with delayed hits*, in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 495–513, 2020.

[6] H. Dai, B. Liu, H. Yuan, P. Crowley and J. Lu, *Analysis of tandem PIT and CS with non-zero download delay*, in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pp. 1–9, IEEE, 2017.

[7] D. Lee, J. Choi, J.-H. Kim, S.H. Noh, S.L. Min, Y. Cho et al., *LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies*, *IEEE transactions on Computers* **50** (2001) 1352.

[8] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi and S. Niccolini, *Temporal locality in today's content caching: Why it matters and how to model it*, *ACM SIGCOMM Computer Communication Review* **43** (2013) 5.

[9] N. Megiddo and D.S. Modha, *{ARC}: A {Self-Tuning}, Low Overhead Replacement Cache*, in *2nd USENIX Conference on File and Storage Technologies (FAST 03)*, 2003.

[10] D.S. Berger, R.K. Sitaraman and 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)*, pp. 483–498, 2017.

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

[12] C. Li and A.L. Cox, *Gd-wheel: a cost-aware replacement policy for key-value stores*, in *Proceedings of the Tenth European Conference on Computer Systems*, pp. 1–15, 2015.

[13] G. Neglia, D. Carra and P. Michiardi, *Cache policies for linear utility maximization*, *IEEE/ACM Transactions on Networking* **26** (2018) 302.

[14] G. Rossini, D. Rossi, M. Garetto and E. Leonardi, *Multi-terabyte and multi-gbps information centric routers*, in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 181–189, IEEE, 2014.

[15] E.W. Biersack, A. Jean-Marie and P. Nain, *Open-loop video distribution with support of VCR functionality*, *Performance Evaluation* **49** (2002) 411.

[16] D. Weitzel, B. Bockelman, D.A. Brown, P. Couvares, F. Würthwein and E.F. Hernandez, *Data Access for LIGO on the OSG*, in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, pp. 1–6, 2017.

[17] T. Li, R. Currie and A. Washbrook, *A data caching model for Tier 2 WLCG computing centres using XCache*, in *EPJ Web of Conferences*, vol. 214, p. 04047, EDP Sciences, 2019.

[18] A. Alekseev, S. Jezequel, A. Kiryanov, A. Klimentov, T. Korchuganova, V. Mitsyn et al., *Evaluation of the Impact of Various Local Data Caching Configurations on Tier2/Tier3 WLCG Sites*, in *CEUR Workshop Proceedings*, vol. 2679, pp. 1–10, RWTH Aahen University, 2020.

[19] L. Bauerdick, K. Bloom, B. Bockelman, D. Bradley, S. Dasu, J. Dost et al., *XRootd, disk-based, caching proxy for optimization of data access, data placement and data replication*, *Journal of Physics: Conference Series* **513** (2014) 042044.

[20] G. Heidsieck, D. De Oliveira, E. Pacitti, C. Pradal, F. Tardieu and P. Valduriez, *Efficient execution of scientific workflows in the cloud through adaptive caching*, in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLIV*, pp. 41–66, Springer (2020).

[21] O. Chuchuk and D. Duellmann, *Access Pattern Analysis in the EOS Storage System at CERN*, in *CEUR Workshop Proceedings*, pp. 22–31, 2020.

[22] T. Beermann, O. Chuchuk, A. Di Girolamo, M. Grigorieva, A. Klimentov, M. Lassnig et al., *Methods of Data Popularity Evaluation in the ATLAS Experiment at the LHC*, in *EPJ Web of Conferences*, vol. 251, p. 02013, EDP Sciences, 2021.

[23] L.A. Belady, *A study of replacement algorithms for a virtual-storage computer*, *IBM Systems journal* **5** (1966) 78.

[24] M. Chrobak, G.J. Woeginger, K. Makino and H. Xu, *Caching is hard—even in the fault model*, *Algorithmica* **63** (2012) 781.

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

[26] N.K. Panigrahy, P. Nain, G. Neglia and D. Towsley, *A New Upper Bound on Cache Hit Probability for Non-anticipative Caching Policies*, *ACM SIGMETRICS Performance Evaluation Review* **48** (2021) 138.

[27] M.T. Goodrich and R. Tamassia, *Algorithm design and applications*, Wiley Hoboken (2015).

[28] M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron et al., *Rucio: Scientific data management*, *Computing and Software for Big Science* **3** (2019) 1.

[29] G.I. Ricardo, A. Tuholukova, G. Neglia and T. Spyropoulos, *Caching policies for delay minimization in small cell networks with coordinated multi-point joint transmissions*, *IEEE/ACM Transactions on Networking* **29** (2021) 1105.

[30] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*, Cambridge University Press (2013).

[31] E. Martelli and S. Stancu, *LHCOPN and LHCONE: status and future evolution*, *Journal of Physics: Conference Series* **664** (2015) 052025.