Report on LHC data access patterns, data uses, and intelligent caching approaches for the HL-LHC (draft)

Frank Wuerthwein, Diego Davila

Version 0.2 - 04 February, 2020

This document has been produced by the IRIS-HEP project (`http://iris-hep.org`) and supported by National Science Foundation grant OAC-1836650. Any opinions, findings, conclusions or recommendations expressed in this material are those of the project participants and do not necessarily reflect the views of the National Science Foundation.

Contents

1 Introduction

The LHC experiments will produce roughly one Exabyte of new data per year during the HL-LHC era. Taking CMS as an example, we expect roughly 0.5 Exabytes each in RAW and AOD formats, and only 50 Petabytes in MINI and 1 Petabyte in NANO formats.

In this document we start with an overview of these formats, and how data is organized and used. We then show the extend to which data within each format could reside on cheap high access latency media like tape, and what level of tape recall would be required for different disk retention policies.

We then motivate regional caches, and an IO latency tolerance target of 500-1000 miles, based on european geography.

The last two sections motivate future work, some ongoing, others not yet started, that could significantly decrease the caching disc space required in the aforementioned regional caches. The first R&D is to explore predicting reuse of files based on metadata of the associasted datasets the files belong to. The second R&D is to explore the predictability of partial file reads.

2 Overview of data organization and formats, and their science uses

All LHC experiments, as well as most of the experimental nuclear and/or particle physics community in general, use Root [1] to define their data formats and access mechanisms to the data. To understand LHC Data Access patterns it is thus worthwhile to briefly review Root-IO [2]. The following description is not meant to be comprehensive, or thoroughly accurate, but rather an indication of the essential concepts.

To understand Root-IO, we need to first acknowledge that data has a physically meaningful structure that is typically separate from the technical implementation of how the data is laid out. The physically meaningful entities are events, i.e. the collection of objects that describe all measurements (or simulations) relevant for one beam crossing. Each event contains a set of physics objects. Technically, data is laid out in files that contain "baskets". A basket contains a set of physics objects for multiple events. Which objects of an event are stored in which basket is a configuration option at file creation time. Each basket is individually compressed by a compression algorithm that is chosen as a configuration option when the file is created that contains the baskets. To read a full event thus requires uncompressing a set of baskets. Doing so generally "unpacks" multiple events at once. It is probably useful to think of the event content being split into baskets of different types because the same basket type will have the same object types for the entire file.

Simplifying a bit, one may think of events as rows and objects in events as columns. Root-IO thus allows the creation of data formats that are neither strictly row, nor strictly columnar. In fact, typically, many rows for a number of columns are stored together in a basket. The format designer chooses a trade-off between supporting maximum selectivity and thus speed in terms of number of rows read/second (fewer columns per basket) versus maximum latency tolerance during reads (many columns per basket to make sequential reads more likely because there is less jumping in file to fetch different columns for the same row).

As general practice, the LHC experiments have the concept of a dataset, a collection of files that each were created with identical configuration of basket types and compression algorithms. For an analysis to process all events of a certain physics meaning requires it to read the entire dataset. It is thus highly likely that an analysis that processes some files in a dataset will process all files of that dataset using the same configuration of the same executable. Let's refer to this as a task. In summary, a task is split into many "non-ordered" executable units that together process an entire dataset. The dataset is a collection of files, each of which store information about a set of events

in form of baskets that contain physics objects.

Let's next try to explain the objective behind this structure. Fundamentally, proton proton collisions recorded by the detectors at the LHC allow for a wide variety of different measurements to be made. Different measurements tend to depend on different subsets of the characteristics of an event, and thus use only a fraction of the physics objects that characterize an event. Moreover, most physics measurements are in one way or another counting events that fit some characteristics. The data is curated by the collaboration with the notion in mind that the majority of the datasets by volume will be reused for many measurements. A typical measurement will thus select a small fraction of the events based on characteristics of a small subset of the objects per event. For the selected events, a significantly larger fraction of the objects may then be scrutinized. Scientists thus commonly derive custom datasets from the collaboration data that are both skimmed and filtered, i.e. contain less information per event and fewer events than the curated original.

The collaborations thus curate their data in formats that allow this flexibility while at the same time aspiring to be efficient, fast, and easy to use. The scientists process those curated datasets, producing much smaller and less versatile personally derived data that they store "locally" for further processing.

Technically, these objectives (flexible, efficient, fast, and easy to use) can not all be satisfied at the same time equally well. Trade-offs are made. A good example for such a trade-off are the three main curated data formats CMS uses to analyze Run2 data, AOD, MiniAOD, and NanoAOD. Their sizes are 400, 40, and 1-2 kB/event. The largest data format is the most flexible, least user friendly, and slowest to process, while the smallest data format is the easiest, fastest, and least flexible. To set the scale, processing that starts by redoing the pattern recognition takes tens of seconds per event and requires the AOD while analysis of the NanoAOD can operate at MHz event rates. The smaller and easier to handle data formats tend to exist in multiple processing versions that reflect improvements in reconstruction and physics object definitions, i.e. less flexible formats require remaking more often to make use of improved understanding of the detector and software.

The diversity of physics measurements implies that not all simulations, and not even all detector data is necessary for all measurements. Detector data is separated into roughly a dozen or so data streams that have mutually only O(10)% overlap. This is done already as part of the trigger and data acquisition. The distinction is immutable in the sense that even future reprocessing of the data retains the original division. Events don't migrate from one stream to the next just because better calibrations and reprocessing algorithms allow for better understanding of the event. For simulations, the diversity is even more dramatic. E.g. for any given simulation release there are thousands of different datasets in CMS Run2 that range in size from 10's of GB to O(100)TB. The many small simulation datasets are typically specialized and esoteric, and thus rarely used, while the few large ones tend to be of very broad interest as input to many physics publications.

In addition, the emergence of newer and better versions of the same dataset leads to gradual migration of use from old to new. Datasets thus have a lifetime that is typically measured in years.

The complexity above leads to complex data access and use patterns that have not been comprehensively studied. This document, and its sister document "Cache Usage on the WLCG" are the beginning of a more comprehensive look at these issues.

3 Study of globally aggregated data use — The tape recall problem

Both ATLAS and CMS work on the premise that archival storage is roughly 5 times less expensive per Exabyte than active storage. At present, this is based on today's tape and HDD costs. Given these large difference in cost, both collaborations are building active tape recalls into their computing models. In the following sub-sections we describe a study of hypothetical tape recall needs

given recorded data access patterns for analysis, and applying hypothetical disk retention policies based on data popularity. This study is done for CMS, but we hypothesize that the fundamental conclusions ought to apply equally well to both ATLAS and CMS for the HL-LHC era. In a way, the error made by extrapolating from Run2 to HL-LHC is probably smaller than the difference between ATLAS and CMS during the HL-LHC era.

3.1 Introduction

The data produced by the experiments (CMS and ATLAS) is distributed among the different sites that pledge resources to these collaborations. Analysis jobs are sent to those sites with the objective of processing the data. Given that the volume of the data produced by the experiments is very large, it is very costly to maintain all of it on regular hard drives or "on disk". All officially curated data is archived on tape which is cheaper but slower to access.

When a dataset stored on tape is requested it is copied from tape to disk in a process known as "tape recall". Due to limited tape recall bandwidth, this process is very expensive in terms of time and thus it is tried to be avoided as much as possible. A task that requests a dataset stored only on tape has to wait for the tape recall before it can start running hence, the overall time for a task since it is injected in to the queue until it is completed can be greatly increased if the dataset is only available in tape.

When a new dataset is created it is typically archived to tape and stored on disk, and it will remain there as long as it keeps being accessed. When a dataset is not accessed in a given period of time, known as the Retention Policy, it is removed from disk to make space for new datasets to come, this way, popular datasets are kept on disk and thus, they can be accessed faster than those which are not as popular and are kept on tape.

We make the implicit assumption here that global transfers from one disk based storage system to another are much less costly than tape recalls, and thus look at the distributed tape archive and disk storage of CMS as a single system each, i.e. in this section we look at global characteristics only.

3.2 Motivation

The objective of this study is to find the disk and tape recall needs when different Retention Policies are applied. The results of this study may be used to find a good trade off between minimizing the cost of disk needs while keeping the amount of data recalled from tape below an acceptable threshold.

By looking at the historical monitoring data of the analysis jobs it is possible to find when and which datasets were accessed in a given period of time; then an algorithm can be applied to calculate the disk space necessary and the amount of bytes recalled from tape, as well as other similar statistics, for a given Retention Policy.

We work at the level of dataset rather than files under the premise that tasks analyze full datasets, as explained in the previous section.

3.3 Input Data

The main source of data in this study comes from the historical monitoring data of the CMS analysis jobs from the period of June 1st, 2018 to May 31st, 2019. This data is produced by a monitoring system that is described in [3], from this source we obtain the name of each dataset accessed and the timestamp of the access.

Depending on the data format, CMS categorize their datasets under different data tiers. We made use of that categorization and divided the dataset accesses into 3 classes:

1. **NANO**. All datasets that belong to the NANOAOD data tier
2. **MINI**. All datasets that belong to either the MINIAOD or MINIAODSIM data tiers
3. **AOD**. All datasets that belong to the AOD or AODSIM data tier

The datasets that do not fall into any of the categories above have not been considered in this study.

From the CMS Dataset Bookkeeping System(DBS) [4] we obtain information about the datasets, specifically their name, id, size in bytes and creation timestamp.

Using the data described above the following Tables or DataFrames are created:

1. **Days**. Groups the IDs of the datasets accessed by day, it has 3 columns: day_timestamp, datasets_set and week_timestamp
2. **Weeks**. Groups the IDs of the datasets accessed by week, it has 2 columns: week_timestamp and datasets_set
3. **Datasets_sizes**. Shows the size of every dataset, it has 2 columns: dataset_id and dataset_size
4. **Datasets_creation**. Shows the week in which every dataset was created, It has 2 columns: dataset_id and week_timestamp

Apart from the DataFrames mentioned above, the algorithm utilizes the following parameters:

- ***Policy***. The Retention policy to apply, for example, Policy = 3 means that a dataset that is not accessed for 3 weeks, it is moved out from disk and into tape.
- ***DeltaT***. Represents the typical period between the creation of a dataset and its first access, at the moment it is fixed to be 7 weeks. When a dataset is created we will assume it stays on disk for a minimum of $DeltaT$ weeks, regardless whether it was accessed or not.

3.4 The Algorithm

The algorithm is divided in 3 steps. Crudely speaking, these steps are as follows. The first step calculates, for every week, the working set size which is the sum of dataset sizes for all datasets accessed in that week. This is our proxy for the minimum amount of disk space needed globally to support analysis. The second step calculates the maximum amount of bytes recalled from tape in a single day. This is any dataset accessed this day that wasn't in the previous week's working set. The third step calculates the amount of bytes recalled from tape and freed from disk each week. Freed from disk is the sum of dataset sizes in last week's working set but no longer in this week's working set.

The exact definitions of these steps are slightly more complicated to study different policies (i.e. different number of weeks duration to define the working set) and take into account an initial time $DeltaT$ after creation of a dataset during which it is not deleted from disk. and are described in detail as follows.

Step 1. In this step, the algorithm iterates through the weeks of the study, starting on the i-th week where $i = \max(Policy, DeltaT)$ and in each iteration it calculates the following sets:

- **recalled**. The set of datasets recalled in the current week. A dataset is accounted as recalled from tape when:
  - the period between the current and its last access is greater than $Policy$ and
  - the period between the current access and its creation is greater than $DeltaT$

- **freed**. The set of datasets moved out from disk and into tape during the current week. A dataset is considered freed when:
  - it has not been accessed in the last *Policy* weeks and
  - the time between its creation and the current week is greater than *DeltaT*
- **working_set_size**. The summation of the size in bytes of all the datasets currently in disk. A dataset is considered in disk when:
  - it has been accessed in the last *Policy* weeks or
  - it has been created within the last *DeltaT* weeks.

At each iteration the sets above are inserted in 3 independent lists: **recalled_per_week**, **freed_per_week** and **working_set_size_per_week**

Every item in these lists represent a different week and the items are sorted chronologically, for example, the first item in *recalled_per_week* represents the set of datasets recalled on the first week of the study.

Step 2. Using **recalled_per_week** and *Days*, the algorithm calculates the Maximum amount of date recalled from tape in a single day. It iterates through **recalled_per_week** and for each week:

- it calculates the day in which each of the datasets was recalled. This is done using the information about the daily accessed datasets on *Days*. Notice that the set of datasets accessed in a day is a superset of the recalled set (not all of the accessed datasets are accounted as recalled)
- using *Dataset_sizes* it calculates the number of bytes recalled per day
- if the number of bytes recalled per day in the current iteration is greater than the overall maximum of all the iterations so far, it sets the value as the new overall maximum

At the end of the iterations the overall maximum will be the "Maximum amount of date recalled from tape in a single day"

Step 3. The algorithm walks through the lists: *recalled_per_week* and *freed_per_week* and looking at the size of each dataset in *Dataset_sizes*, it calculates:

- **recalled_size_per_week**. The amount of bytes recalled every week and
- **freed_size_per_week**. The amount of bytes moved from disk to tape every week

3.5 Results

As mentioned before, we have analyzed 3 distinct categories of datasets getting accessed: NANO, MINI and AOD. For each of these categories we have executed the algorithm for 5 different Retention Policies: 1, 3, 6, 9 and 12 weeks and obtained:

- A table that shows the values for the *maximum amount of bytes recalled from tape in a single day* and the *maximum amount of bytes stored in disk per week.*
- A plot that depicts the *amount of bytes recalled from tape per week*
- A plot that depicts the *amount of bytes stored in disk per week*

3.5.1 Results for the data tier NANOAOD

Retention Policy	Max recall in a single day	Max working set size per week
1	2.21 TB	17.75 TB
3	1.34 TB	18.97 TB
6	1.21 TB	19.67 TB
9	794.03 GB	19.71 TB
12	505.85 GB	20.16 TB

Table 1: Tape recall and Disk requirements for the data tier NANOAOD

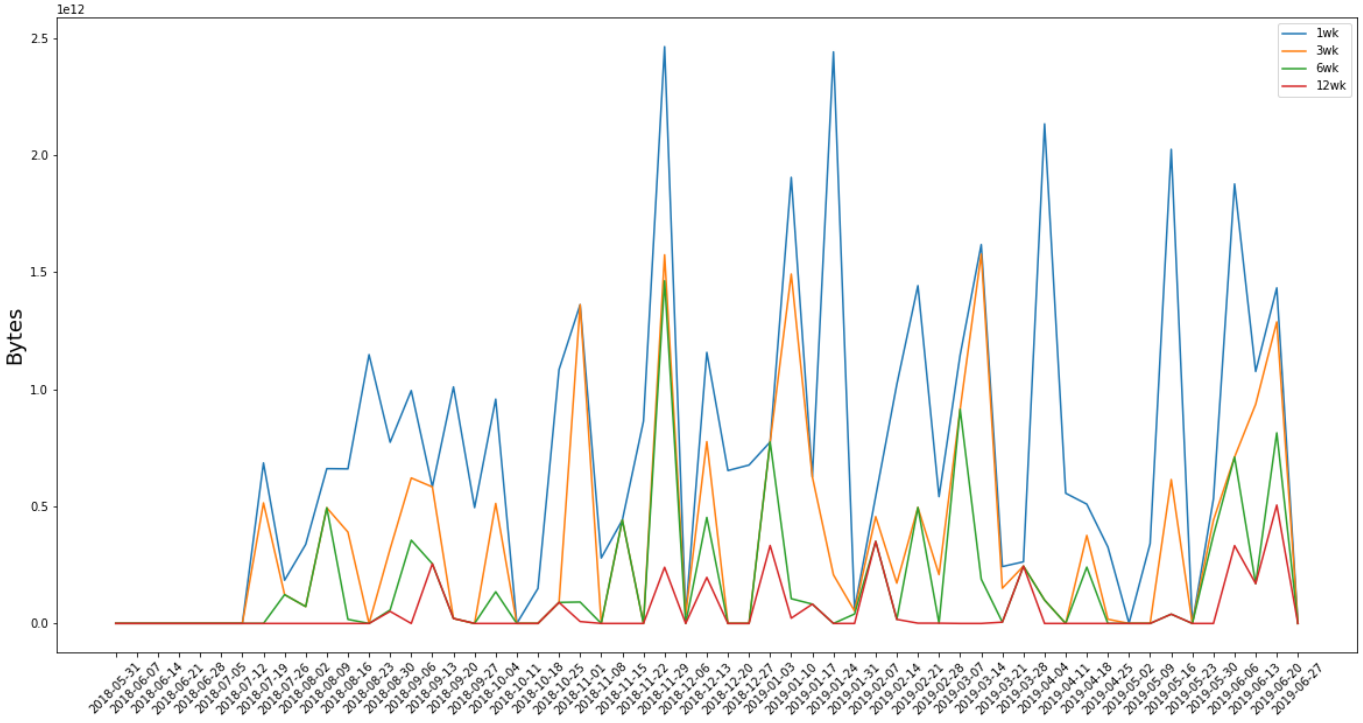

Figure 1: Amount of data recalled per week for the data tier NANOAOD

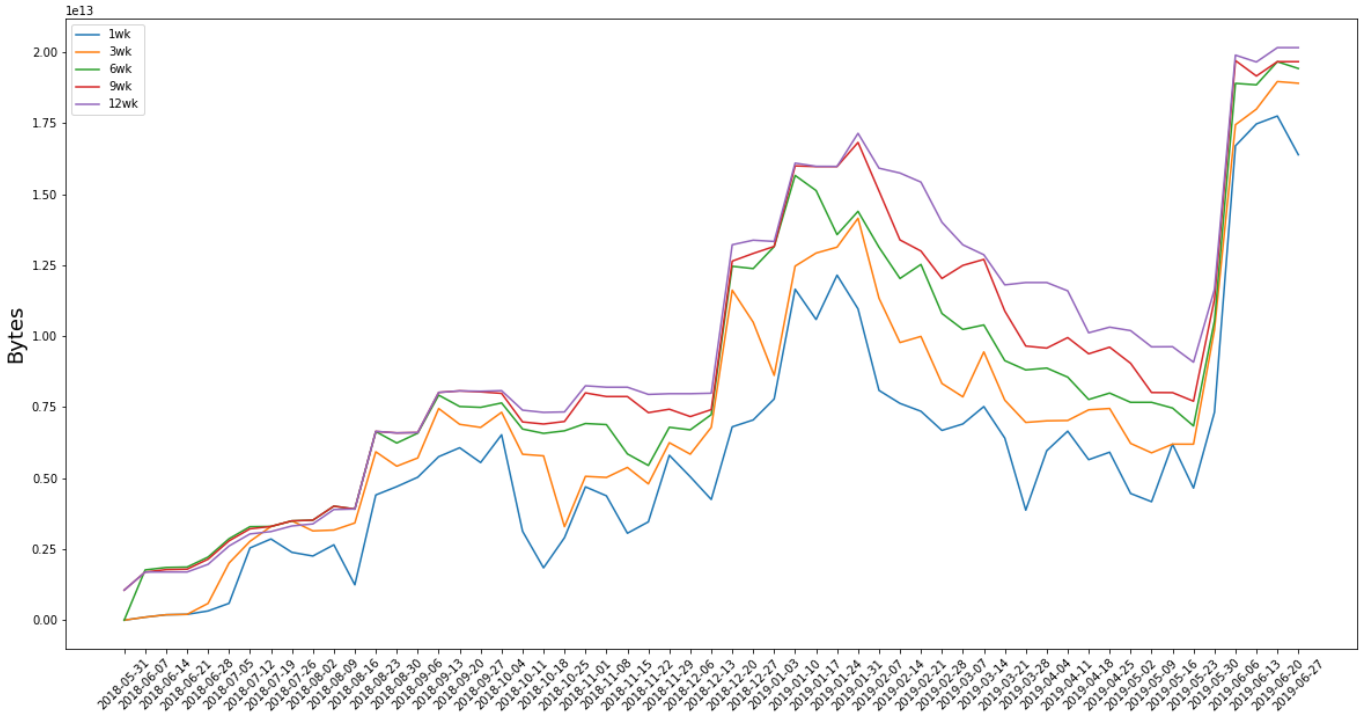

Figure 2: Amount of data stored on Disk per week for the data tier NANOAOD

3.5.2 Results for the data tiers MINIAOD and MINIAODSIM

Retention Policy	Max recall in a single day	Max working set size per week
1	187.27 TB	2.13 PB
3	82.87 TB	2.60 PB
6	52.59 TB	2.95 PB
9	44.29 TB	3.26 PB
12	43.33 TB	3.52 PB

Table 2: Tape recall and Disk requirements for the data tiers MINIAOD and MINIAODSIM

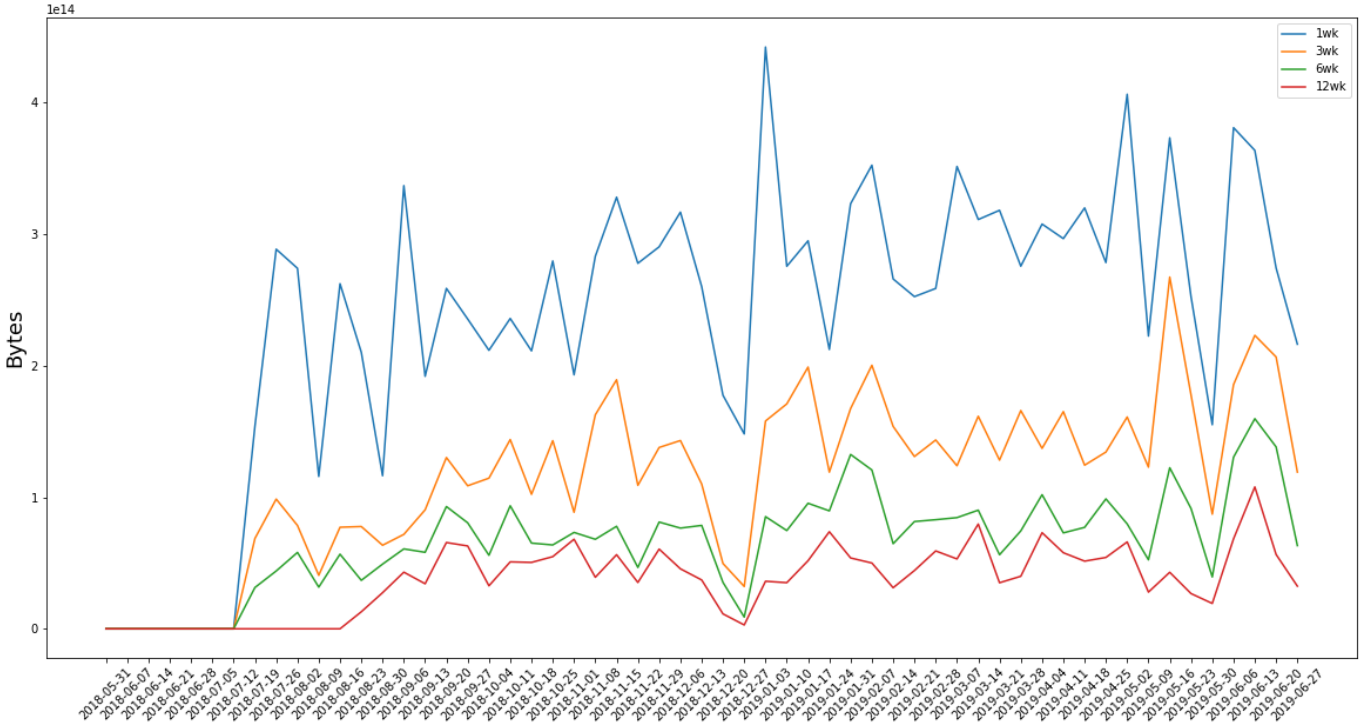

Figure 3: Amount of data recalled per week for the data tiers MINIAOD and MINIAODSIM

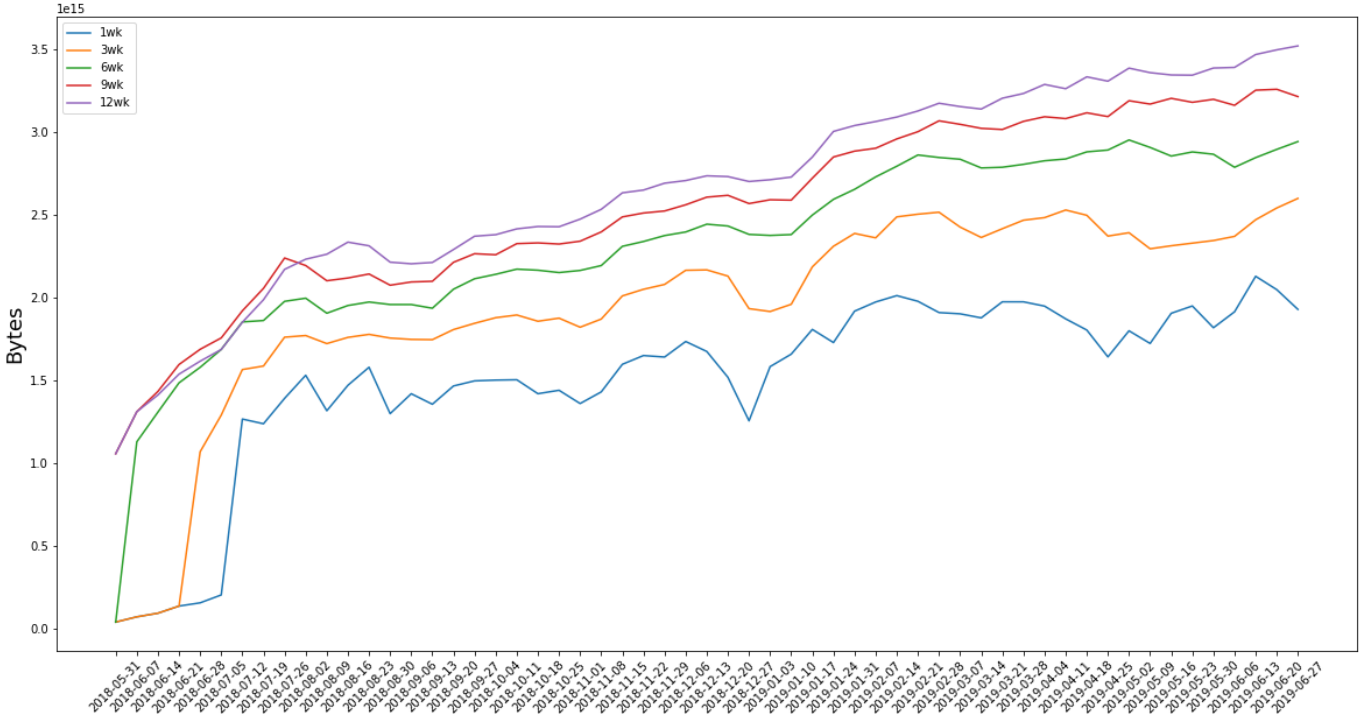

Figure 4: Amount of data stored on Disk per week for the data tiers MINIAOD and MINIAODSIM

3.5.3 Results for the data tier AOD

Retention Policy	Max recall in a single day	Max working set size per week
1	1.49 PB	7.59 PB
3	1.38 PB	9.14 PB
6	1.33 PB	10.17 PB
9	1.33 PB	10.63 PB
12	1.33 PB	11.61 PB

Table 3: Tape recall and Disk requirements for the data tier AOD

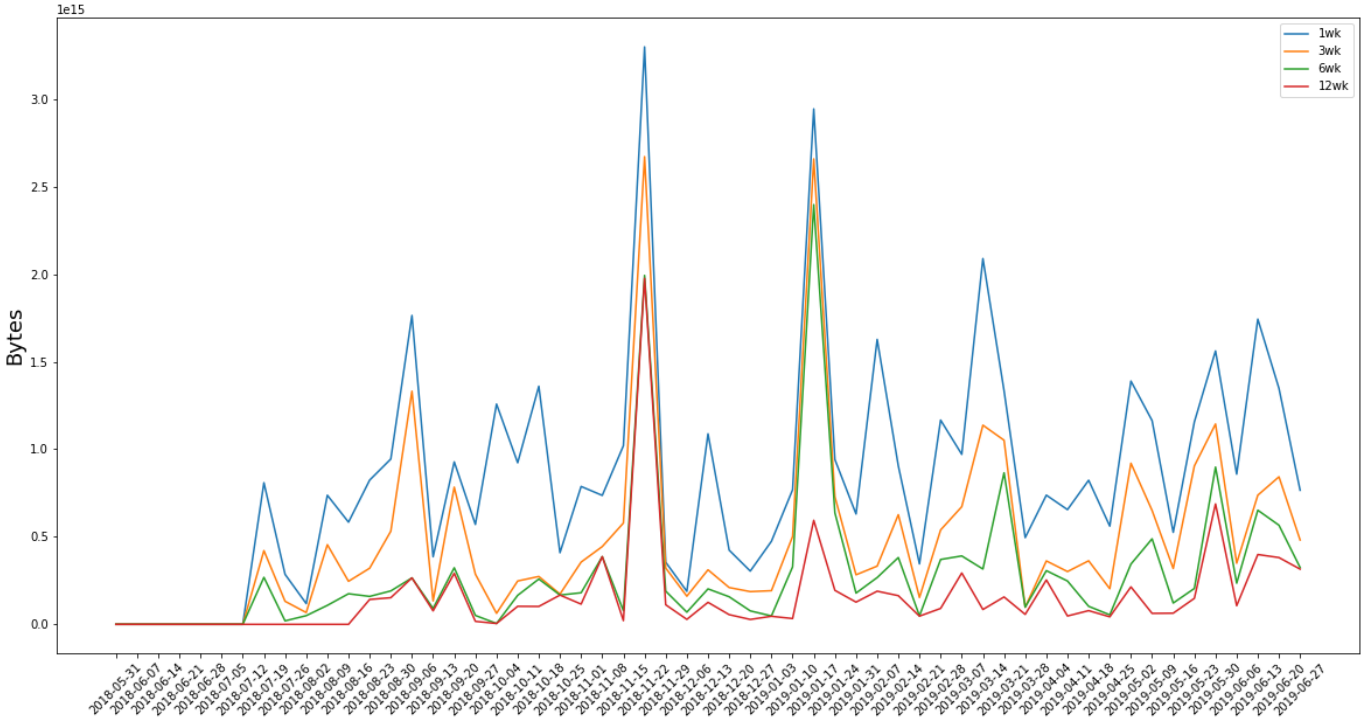

Figure 5: Amount of data recalled per week for the data tier AOD

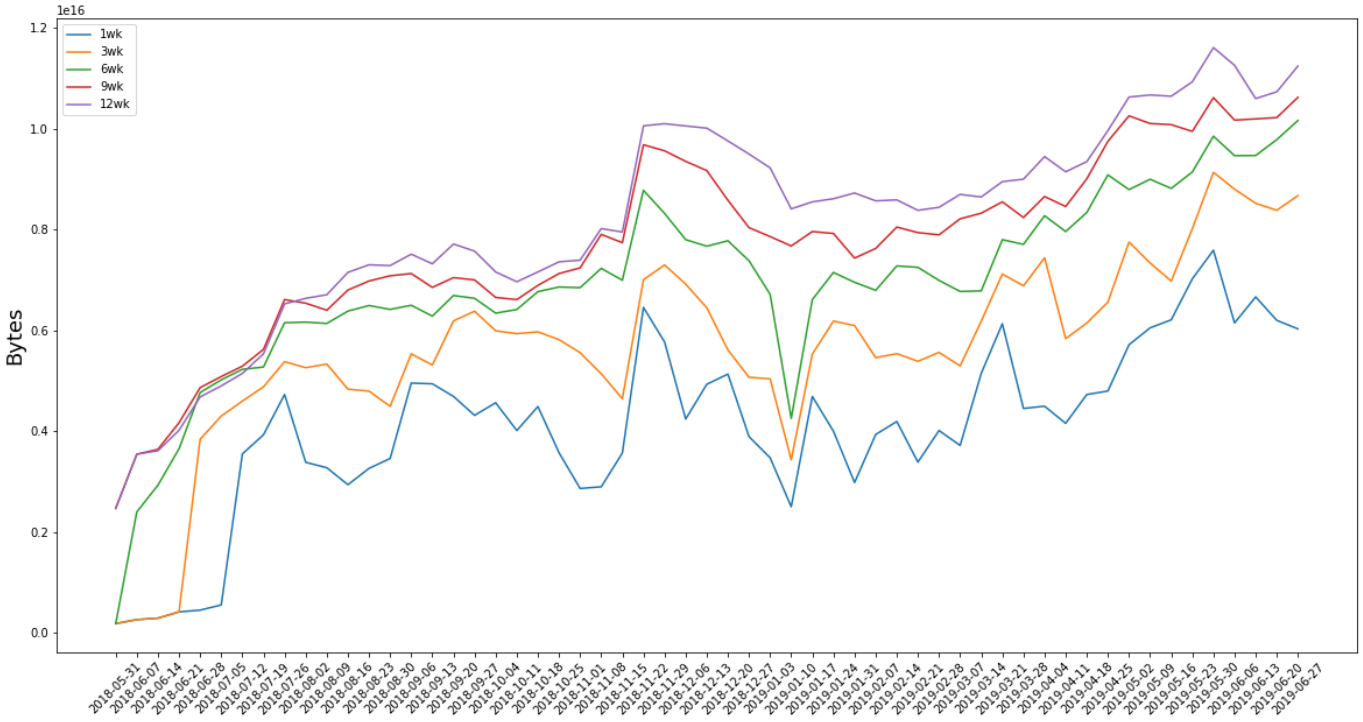

Figure 6: Amount of data stored on Disk per week for the data tier AOD

3.6 Remarks

As it is expected we can see, in the 3 categories, that the Retention Policy is directly proportional to the *maximum amount of data store in disk per week* and inversely proportional to the *maximum amount of bytes recalled from tape in a single day*, i.e. the longer we leave datasets in disk, the more disk is required but the lower are the chances for a dataset to be recalled from tape.

Something that can be noted is the fact that the working set keeps increasing with time. This makes us think that the range of popular datasets grows with the time but other factors, like the size of new datasets compared to older ones, are needed to be considered to prove it.

Finally, something that we believe is worth to investigate is the spiked behavior shown in the plots for the recalled data.

This study is a work in progress and there is still a more refined analysis to be done in order to understand the full spectrum of the of the resulting data.

4 Tier-2 Network of Processing Centers

For largely social reasons, the LHC collaborations analyze their curated data across a network of dozens of regional centers. In essence, the distribution of computing resources roughly matches the distribution of human resources. This has lead to a bifurcation of the total disk space available to each collaboration, and thus replication of data and/or bifurcation of processing power to the extend that only locally accessible data is processed.

In principle, the XRootD data federations [5] that have emerged during Run2 allow remote file open/read operations via the XRoot protocol [6]. In practice, such operations are efficient only if the applications are latency tolerant, i.e. applications are compute intensive and IO is dominated by large reads. Different file formats and their uses will thus be processed efficiently, i.e. not

dominated by file access latency, for different RTTs of the remote file reads.

As resource distribution is known today, we can set a target for the latency tolerance of the applications based on typical distances between centers, and the social organizations they belong to. Figure 7 shows a map of Europe with arrows indicating distances of 500 miles. Distances like that are interesting because they allow for integration of disk space at a national scale for several countries that provide significant resources each to ATLAS and/or CMS. Geography thus motivates a goal of latency tolerance for 500-1000 miles remote reads.

Figure 7: Examples of 500 miles distance in different European regions.

Details on measured reuse of data at the level regional level can be found in [7].

5 Dataset Popularity

Figure 8 shows the distribution of file reuse within one month for the SoCal cache discussed in [7]. It clearly shows that the most likely reuse is zero. However, the distribution has a tail that is more than exponential, i.e. flatter than a straight line on this log plot. If re-use of files was predictable then a caching algorithm could be constructed that would be much more effective than the currently implemented LRU cache management. This might lead to significant cost savings because much less disk space would be needed to support the same analysis pattern for the same network utilization due to cache misses.

Anecdotally, we know that the physics content of simulation datasets is a good predictor of reuse. Same is true for reconstruction version number, and some other dataset metadata. This

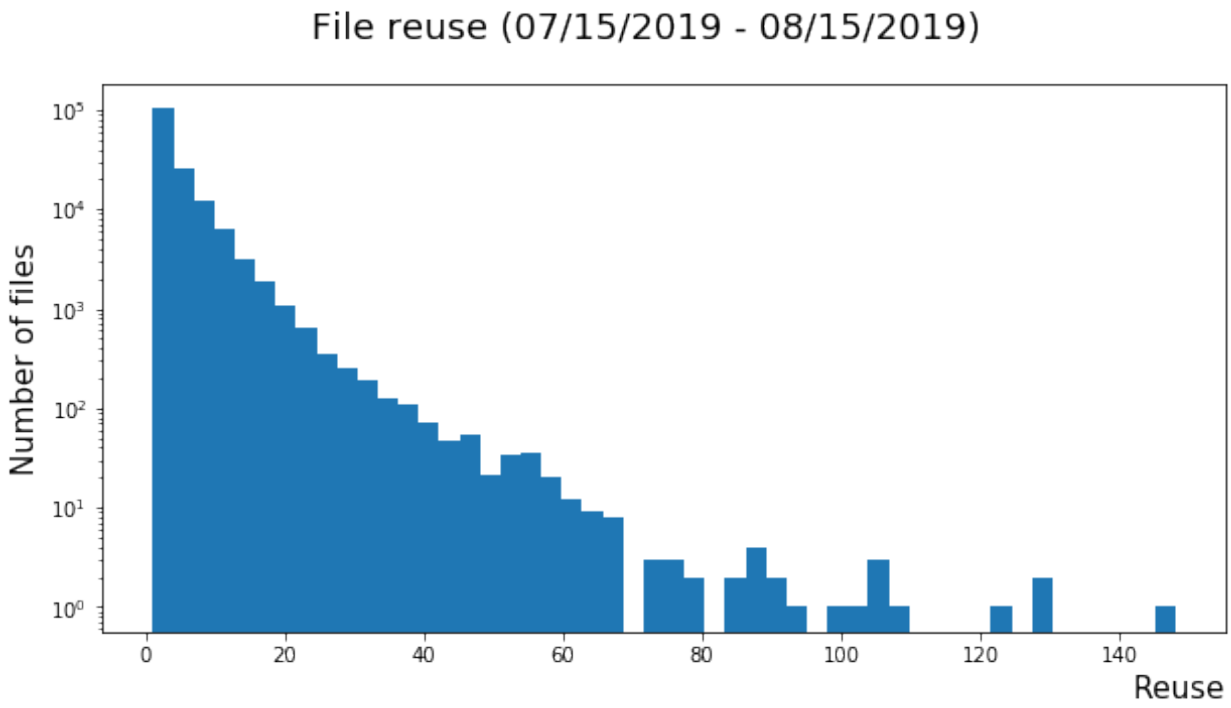

Figure 8: Number of accesses to each file within a cache.

R&D direction is pursued by [8].

6 Data Access within files

Figure 9 is reproduced from [7]. It shows that only a small fraction of the content of a file is typically read by the applications. In principle, this might be exploitable to reduce the working set required to be cached if the subset of the file that is read was predictable. No R&D along these lines is presently pursued, to the best of our knowledge. If partial file reads were predictable then future functionality like that of ServiceX [9] might be used to reformat the data and drop unnecessary data from files. As the potential savings in cache disk space are large we think that R&D in this direction should be pursued.

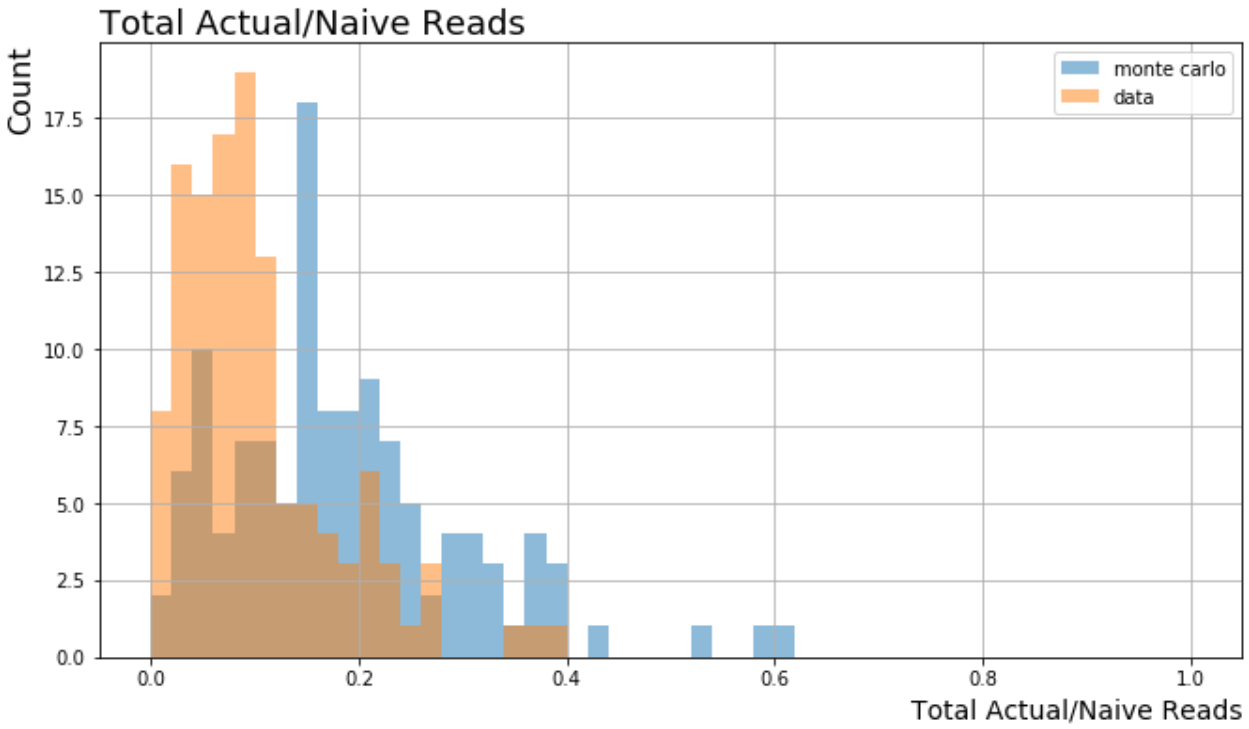

Figure 9: Distribution of the percentage of bytes being read from each file on a cache.

References

[1] Fons Rademakers and Rene Brun. ROOT: an object-oriented data analysis framework. *Linux J.*, page 6.

[2] Philippe Canal, Brian Bockelman, and Rene Brun. ROOT I/O: The Fast and Furious. *Journal of Physics: Conference Series*, 331(4):042005, 2011.

[3] Cms htcondor monitoring system. `https://github.com/dmwm/cms-htcondor-es/blob/master/README.md`.

[4] A Afaq, Andrew Dolgert, Yanrui Guo, C Jones, S Kosyakov, Valentin Kuznetsov, L Lueking, D Riley, and Vansh Sekhri. The cms dataset bookkeeping service. *Journal of Physics: Conference Series*, 119:072001, 07 2008.

[5] L Bauerdick, D Benjamin, K Bloom, B Bockelman, D Bradley, S Dasu, M Ernst, R Gardner, A Hanushevsky, H Ito, D Lesny, P McGuigan, S McKee, O Rind, H Severini, I Sfiligoi, M Tadel, I Vukotic, S Williams, F Wuerthwein, A Yagil, and W Yang. Using Xrootd to Federate Regional Storage. *Journal of Physics: Conference Series*, 396(4):042009, 2012.

[6] A Dorigo, P Elmer, F Furano, and A Hanushevsky. XROOTD - A highly scalable architecture for data access. *WSEAS Transactions on Computers*, 4.3, 2005.

[7] Frank Wuerthwein, Diego Davila, and Jonathan Guiang. Report on cache usage on the WLCG and potential use cases and deployment scenarios for the US LHC facilities. 02 2019.

[8] Valentin Kuznetsov, Tommaso Boccali, Daniele Cesini, Marco Baioletti, Valentina Poggioni, Diego Ciangottini, and Mirco Tracolli. Smart caching at CMS: Applying AI to XCache edge services. International Conference on Computing in High Energy and Nuclear Physics, 2019.

[9] B. Galewsky, R. Gardner, L. Gray, M. Neubauer, J. Pivarski, M. Proffitt, I. Vukotic, G. Watts, and M. Weinberg. Service X: A distributed, caching, columnar data delivery service. International Conference on Computing in High Energy and Nuclear Physics, 2019.