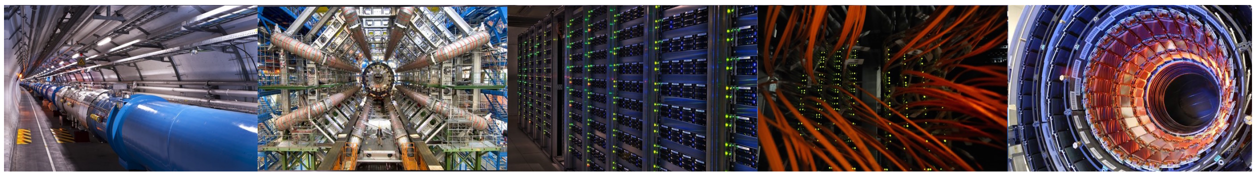


Report on cache usage on the WLCG and potential use cases and deployment scenarios for the US LHC facilities

Frank Wuerthwein, Diego Davila, Jonathan Guiang

Version 0.1 - 15 August, 2019

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 Executive Summary

Caching in both ATLAS and CMS is still in its infancy. Both collaborations have deployed their first caches that are used in production. All deployments, except for the SoCal cache for CMS are modest in size at this point. The SoCal cache comprises roughly 1 petabyte of disk space across the Tier-2 centers at Caltech and UCSD.

For ATLAS, the rational for smaller caches is that the use case is latency hiding rather than data reuse. The cache is thus used more like a processing buffer than a traditional memory cache that minimizes cache misses.

After some introductions, we present a detailed study of the SoCal cache usage for the period of late March to mid August 2019. During this period, we cached a fraction ($\sim$ 1Petabyte) of the entire MINIAOD(SIM) namespace, using $\sim$600TB of disk space in the cache. The main findings of this study are:

- The daily working set across detector data and simulation for the cached namespace was roughly 10-60 TB out of the total of 1 Petabyte of data in the cached namespace.
- The average daily file reuse ranges from 3-10 times on most days.
- Most jobs read no more than 10-20% of a file. There thus appears to be significant selective reading during analysis that deserves further detailed study.

In general, a lot more can and should be learned from analyzing data use of the SoCal cache. We thus decided to continue exploring the data about CMS data access and reuse from the SoCal cache beyond this report.

During Fall 2019, we are reconfiguring the SoCal cache for CMS to cache the entire run2 MINIAOD(SIM) namespace. Run2 comprises a total of $\sim$60 billion events at an average event size of 40kBytes. One version of all run2 data in this format thus comprises roughly 2.4 Petabytes. Accounting for multiple versions leaves us with a total of roughly 7.5 petabytes of data in MINIAOD(SIM) format today. We measured the global working set of CMS for a 4 week interval to be roughly 2.3 Petabytes for early Summer 2019. Taking all of these experiences into account, as well as what is practical, we decided to dedicate slightly more than 1 Petabyte to the SoCal cache for this fall, and configure it such that the entire run2 namespace is cached. We will then monitor the cache performance using the tools developed and increase the disk space available in the cache as needed, based on the measurements observed.

2 Global Context

The LHC community, including experts from the experiments, the software providers, and the infrastructure operators, have organized around the general theme of Data Organization Management and Access (DOMA) within WLCG in several subgroups. The most relevant group to the topic of this note is probably the DOMA Access group. This group nominally meets every other week on Tuesday's at 8:30am pacific for one hour to 90 minutes. Ilya Vukotic (US ATLAS) and Frank Wuerthwein (US CMS) are among the co-conveners of this working group, and thus bring in US perspectives in general, and IRIS-HEP concerns and contributions in particular.

Throughout the first year of IRIS-HEP, the DOMA Access working group focused on caching use cases in general, and developed a straw proposal for the meaning of a data lake infrastructure. This straw proposal was presented at the JLAB HOW meeting in Spring 2019, followed by multiple discussions during the DOMA Access meetings.

While we expect some of this to continue for year two of IRIS-HEP, DOMA Access will in addition be shifting focus a little towards DOMA relevant ideas within the "Analysis Facility" context. First presentations to that respect are scheduled for September. This is in part to prepare for a pre-meeting on this topic at CHEP 2019 in Australia.

3 A brief overview of data taking at the LHC

The LHC provides colliding beams for typically 3 years in a row, followed by a 1-2 year shutdown for repairs and upgrades. An annual data taking period commences in Spring and ends in late Fall. During the winter months there isn't sufficient cost-effective power to operate the LHC due to the need to heat houses in the Geneva area and beyond.

The beams of the LHC, made up of bunches of protons, cross every 25ns at 4 collision points around the 27km ring. During one beam crossing, many proton-proton collisions occur along a few cm in the beam direction for both ATLAS and CMS. During Run2 (2016/17/18) there were up to ~ 70 collisions per crossing, while the brighter beams during the HL-LHC era will lead to ~ 200 collisions per crossing. A trigger system down-selected nominally 1000 beam crossings per second during Run2 for processing. In addition, up to another 6kHz of crossings were recorded for deferred processing during the shutdown period after Run2. We refer to a beam crossing as an "event". Any two events are treated as independent, making the LHC physics program a perfect use case for High Throughput Computing. The large selectivity means that there is typically only one collision of interest in a crossing, the remainder just adding "noise".

4 DOMA Access Data Lake Straw Proposal Summary

As discussed in more details in the accompagnying document "LHC Data Access Patterns", ATLAS and CMS data formats and access patterns determine the tolerance of the data analysis applications to RTTs between compute nodes and data caches. The Data Lake Straw proposal envisions a mix of distributed caches that are directly accessed from compute nodes, mechanisms for handling cache misses within a data lake, and centrally organized replication & placement of data in the lakes.

Data caches may thus be regional if the type of analysis applications they serve are sufficiently tolerant of RTTs within the region. An example of such a regionally distributed cache is the US CMS MINIAOD cache deployed in Southern California, and described in the next section below.

However, in the extreme case of maximally latency intolerant applications, caches may be needed locally, or even on the compute nodes themselves to hide access latencies.

The entire US Tier-2 compute infrastructure is expected to be treated as a single data lake. Placement into it is likely from the globally distributed tape archive of each ATLAS and CMS in order to minimize disk costs.

There is thus fundamentally a trade-off between investment in tape recall capability at the Tier-1 archives, and disk space requirement for a viable data lake. It is conceivable, that different data lakes in different global regions may decide on different optimizations of recall from archive vs deployed disk space. And some data formats may only be processable after an archival recall, and may then be buffered under workload management control, and thus not replicated into the data lakes at all, but rather steered towards the exact compute locations file by file or object by object.

At this point, all of the above is largely conceptual, and more experience with pilot projects under production use is necessary in conjunction with modelling to gain a better understanding of sizing of caches and lakes, and their detailed relationship.

5 US CMS MINIAOD Use Case - A Regional Cache in Southern California

A leading example of a regionally distributed cache deployment is the "SoCal" cache infrastructure, a project currently in production with the objective of making the the collection of Run 2 MiniAOD datasets easily available from US CMS compute resources located in southern California (hence SoCal). The MiniAOD format is roughly 40kByte per event, and is used by more than 90% of

CMS analysis activity. Average event processing rate for MiniAOD is less than 10Hz. Total compute capacity for US CMS in SoCal is $\approx 15,000$ cores. So even extreme average access rates of 15k cores x 10Hz x 40kByte, or 40-50Gbps would be possible betwen the UCSD and Caltech Tier-2 centers. Moreover, the two centers have only 3ms RTT, and even a petabyte of RAW disk cache across the two centers would be less than 20% of their total disk space deployed. At the same time, petabyte/(40kB/event) $\approx$ 27 Billion events. A petabyte cache is thus sufficient to accomodate close to half of the 60 Billion Run 2 events for one version of the MINIAOD(SIM).

The remainder of this section describes the deployed infrastructure, and a variety of measurements of production use.

5.1 XRootD: Overview and Nomenclature

In simple terms, XRootD is a modular, fast, low-latency, and scalable file server which can serve any kind of data. Moreover, XRootD servers themselves can be configured to perform a variety of services. For example, an XRootD server that serves data is called a Data Server. Conversely, an XRootD server that does not serve data but instead communicates with other XRootD servers to locate a requested file is called a Redirector. A Redirector can then be put in contact with many Data Servers to form a more dynamic and scalable infrastructure. Additionally, such a network of XRootD servers is simply more suitable to serve distributed data. Consider a simple file request: first, an XRootD client will request a file by querying a Redirector which in turn will query all of its Data Servers to find the desired file; once the file is found in one of the servers, the Redirector will instruct the client to contact that Data Server (see Fig. 1).

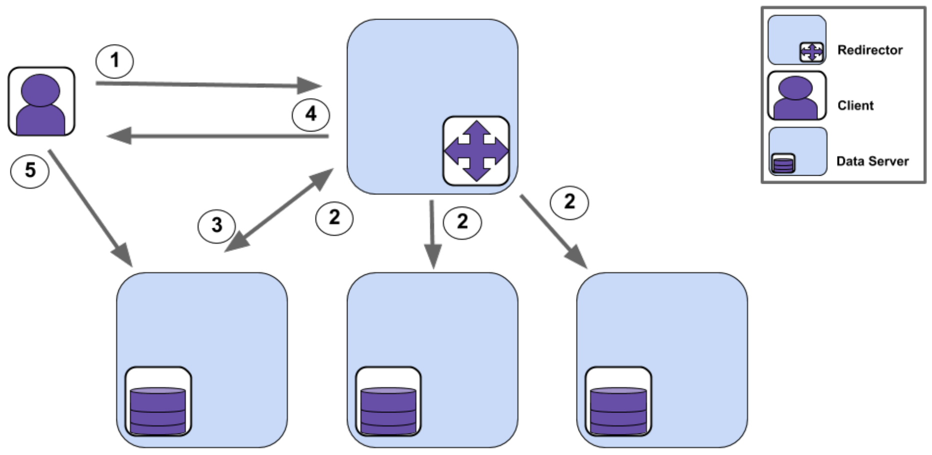


Figure 1: File request/serve workflow: 1. The Client request a file, 2. The Redirector queries its Data Servers for the requested file, 3. A Data Server answers affirmatively for the requested file, 4. The Redirector instructs the client to contact the Data Server that has the file, 5. The Client opens the file remotely from the Data Server

Redirectors can be connected in arbitrarily complex tree-structures. A given redirector communicates to one and only one upstream redirector, but can receive connections from many downstream redirectors. This way, if a Redirector cannot find a file in its downstream network, it could query its upstream Redirector in the tree hierarchy (see Fig. 2). Such a configuration is the fundamental organization of an XRootD Data Federation, where multiple Redirectors and their corresponding Data Servers work together to make distributed data accessible to any client contacting any Redirector in the federation. The CMS Data Federation provides remote data access to analysis jobs through XRootD as described, which allows for a job to run anywhere and read its input data from anywhere it is available with, of course, an overhead due to the network latency.

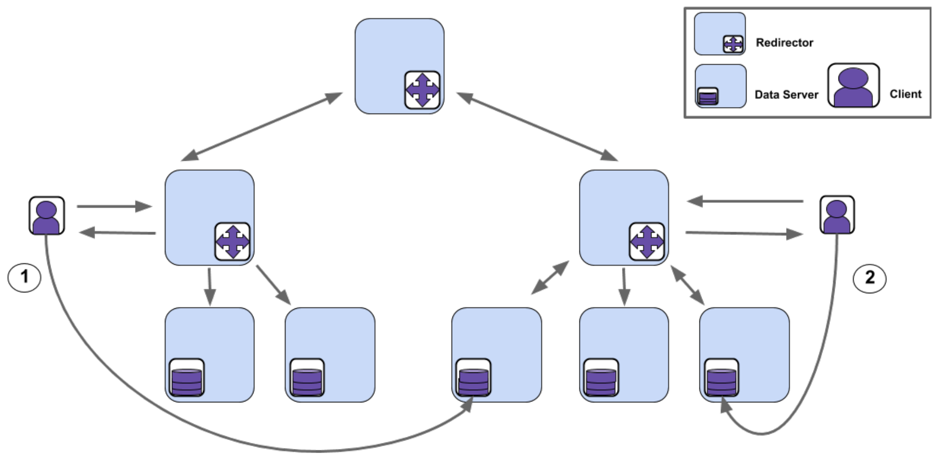


Figure 2: A diagram illustrating a Redirector tree or "Data Federation". 1. A file getting served from a remote Redirector, 2. A file getting served from a local Redirector

XCache is yet another special configuration of XRootD that features a caching mechanism. Under this scheme, when a Redirector requests a given file from a XCache Data Server and that file does not exist in the local storage of that Data server, it (the XCache Data Server) will act as an XRootD client itself, triggering a data request to a remote Redirector - typically over the WAN, then it will cache the data such that when a subsequent client requests the same data, it is served from the local storage of the XCache server. This reduces the overall WAN traffic and, at the same time, makes the processing job that requested the data more efficient by reducing its I/O wait time for remote data.

5.2 The SoCal Cache

The SoCal cache is an XRootD system that integrates several XCache servers from two different US CMS Tier-2 centers in southern California (UCSD and Caltech), forming a single, larger overall cache infrastructure. While it is a common practice to merge several servers into a single cache, this setup is unique in that it is formed by caches in two separate geographical locations acting as one. Moreover, they use the same XRootD Redirector, as in the XRootD configuration described previously. When a file is requested by a user, the Redirector will query all of the cache servers connected to it, once found, it instructs the client to contact the server that has the file. If the requested data is not found, one of the caches will request it from the CMS Data Federation. (see Fig. 3). The SoCal cache thus dynamically moves popular data closer to the resources used to process it, making this data more accessible to CMS users running jobs on the UCSD and Caltech sites. At the same time the SoCal cache enablas these sites to receive a wider variety of jobs. Overall, the SoCal cache is composed of 13 servers and features a total storage capacity of 624TB. The exact specifications of the different servers are recorded in Table 1.

5.3 The CMS Computing Model: Overview and Nomenclature

The design of the CMS computing model is fundamentally derived from the sheer scale of the LHC. Put simply, the CMS detector, just one of two all-purpose detectors within the LHC, produces a *lot* of data - a direct response to the statistical demands of a particle physics experiment. In total,

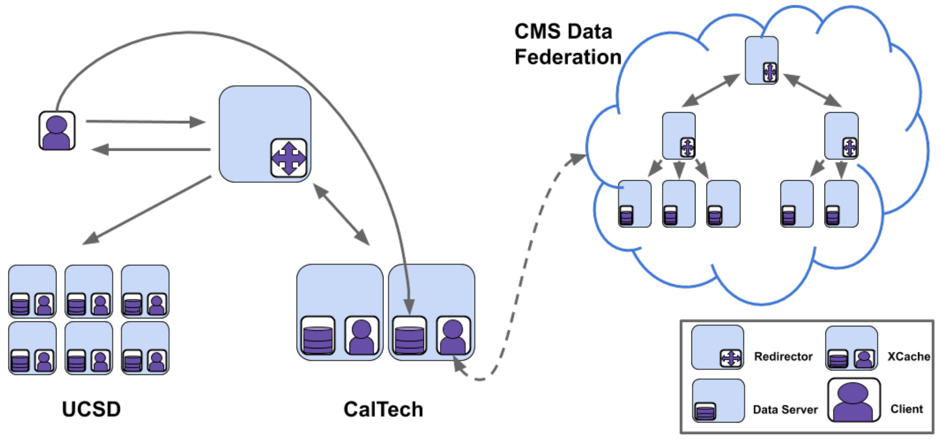


Figure 3: A diagram illustrating the essential workflow of the SoCal cache. When a file is not found within the cache that is supposed to have it, a request is made from the XCache server to the CMS Data Federation

Site	Nodes	Disk Capacity Node	Network Card Speed	Total Capacity
UCSD	11	24 TB	10 Gbps	264 TB
CalTech	2	180 TB	40 Gbps	360 TB

Table 1: Technical specs for the UCSD and Caltech XCache infrastructure for the ”SoCal cache”

Run2 comprises roughly 60 Billion events from collisions and simulations combined, not including the 6kHz trigger rate of parked data.

During Run2, the CMS experiment ran each year with a different major release and even somewhat differing detector configurations, requiring simulations to match. The nominal ratio of events produced per year from beam collisions vs simulations is 1:2, though some years it ends up more like 1:3 in practice due to large numbers of ”fast simulations” that require little CPU but still lead to the same output data size per event in MiniAOD. Fast simulations thus affect storage but not CPU needs of the experiment.

In order to meet the challenges previously described, the CMS computing model was made to be a *distributed* system: the data and the computational resources used to process it are spread amongst different sites, namely institutes that collaborate with CMS. These sites are accessed through the World Wide LHC Computing Grid (WLCG) and their processing resources are put together in a HTCondor “Global pool” so that they can be managed centrally. This pool also controls the queues where the analysis jobs (those that will process the data produced by the LHC) are injected. From there, the pool decides which, where, and when jobs get to run. The computing resources available through the Global pool are provisioned on demand using a pilot-based system (GlideinWMS). The Frontend of GlideinWMS will periodically monitor the job queues of the Global pool and request that pilots be sent to the sites where the jobs are willing to run. When a pilot lands in a given site, it will allocate a certain amount of cores and memory and make them available to the Global pool so that the jobs can start flowing to the site (see Fig. 4).

Given that the data is distributed over many different sites, it is logical to conclude that a job requiring a specific dataset should be sent where the data is available. Thus, initially, jobs only ran where their required data was. However, a typical problem that arises under this model is that fluctuations in data popularity lead to long queues in some sites while compute resources remain unused in others. To overcome this problem, the Overflow Mechanism was designed – it is currently

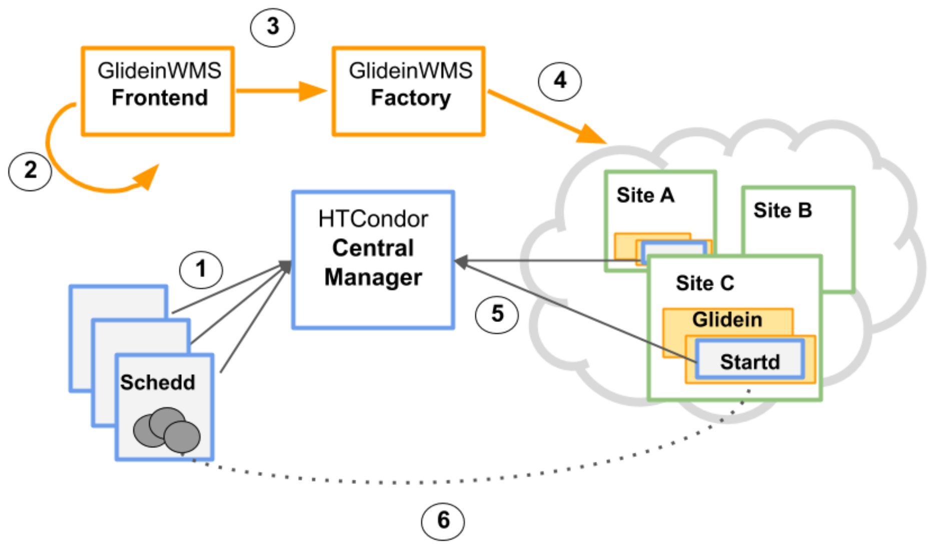


Figure 4: A diagram illustrating the essential workflow of the Global pool: 1. A job, requesting to run in a given site(s), gets queued, 2. The Frontend queries the queues for idle jobs, 3. The Frontend requests pilot for the Idle job noted, 4. The pilot is sent to the requested site, 4. Once the pilot starts executing on the site, it connects back to the Global pool, 5. The Central Manager assigns the job to the pilot

implemented within the GlideinWMS Frontend – to function as follows: when a site "A" is busy and thus the jobs targeting "A" are waiting idle in the queues, special "overflow-pilots" are sent to a nearby site "B", where they target the idle jobs at "A" and move them to "B". Then, they will read their input data remotely from site "A". Now, this system operates within a few constraints:

1. The jobs are willing to overflow (not all of the jobs are suitable for overflowing)
2. The jobs have been idle for too long (typically quantified as more than 1 hour)
3. Both sites are close together (within the same region or country). This is done in order to reduce the overhead of reading the data remotely instead of locally.

Another important aspect of the CMS computing model that is relevant for this use case is the fact that all the data shared by the collaboration has the same namespace, regardless where it is stored. In other words, a specific file will always have the same name, no matter in which site it is located. Given that it is not possible for all the sites to have the same namespace, a mapping between the Logical File Name (name of the file relative to the CMS namespace) and Physical File Name (site-specific name and location of a given file) is required. The Trivial File Catalog (TFC) specifies this mapping. Every site has its own TFC and it used by any job running on it. Moreover, when a job is looking for a file that is not present in the filesystem of the worker node running the job, the TFC will give the job a service URL that can provide the requested file, one example of this is URL of a specific XRootD Redirector within the CMS Data Federation.

5.4 The CMS Computing Model and the SoCal Cache

The SoCal caching infrastructure is configured to cache all MiniAOD(SIM) for Run2. It is configured to allow accesses from any US CMS compute resources in southern california (presently the Tier-2s at Caltech and UCSD, and the Tier-3 at UC Riverside). To steer jobs from the global pool that require any of the Run2 MiniAOD(SIM) to these compute resources, and then access the caching infrastructure, two features of the CMS computing model are used.

First, we define a special overflow mechanism in the Frontend such that any job that is allowed to run at any US Tier-2 and requires data from the cached namespace, i.e. any of Run2 MiniAOD(SIM), may run at UCSD or Caltech in addition to whatever sites the user may have chosen upon submission of the job via CRAB. Second, we modified the Caltech and UCSD TFCs such that any accesses to the cached namespace are directed to the redirector for the SoCal caching infrastructure, instead of trying to read directly from disk.

5.5 CMS XCache Monitoring Dashboards

US CMS chose an XCache monitoring strategy that logs data accesses to the caches locally, and then sends the aggregated access information to the CERN MONIT infrastructure to be stored in HDFS. There are two types of accesses, single byte ranges, and lists of byte ranges (referred to by the XRootD software as vectors of byte ranges). An application authenticates for a ”session”, and the data accessed is locally aggregated by session. The monitoring system described later on this section uses those aggregations, in particular the following data fields:

- server_host. This is used to filter the XCache servers that are interesting for us (currently only the SoCal Data Servers)
- file_lfn: The Logical File Name of the accessed file
- app_info: This field uniquely identifies the job that made the request
- file_size: The size in bytes of the file accessed
- read_bytes: The total amount of bytes read during the session, including both single byte and vector byte operations

The field app_info attached to each session log allows correlation with the HTCondor ClassAds recorded for each job running in the global pool.

In order to understand XCache use, and provide ongoing monitoring that populates an XCache dashboard, we have developed software to mine both the XCache session logs and the HTCondor ClassAds. Trying to populate a dashboard in realtime from the logs in HDFS has proven too compute intensive to be viable. Instead, we decided to define a set of metrics that are aggregated daily, and thus display very fast. We refer to this hereafter as “Monicron”.

Fig. 5 shows an example dashboard for the Monicron data, allowing the operations team for an XCache installation to monitor the health of a cache at a glance. In addition, the data has been made to be readily accessible for further analysis. Finally, Monicron is capable of providing its services for any cache over any monitoring source stored on HDFS at CERN, where all of the long-term monitoring data is stored for CMS, and new aggregations can be added on the fly via a simple plugin system. This feature will be heavily used in the future, as Monicron is currently only configured for the SoCal cache.

Specifically, Monicron’s workflow is as follows. First, monitoring data is pulled from HDFS records at CERN. Then, meaningful aggregations of this data are calculated and stored on a database managed by the MONIT team at CERN. This is done on a daily basis, such that each day, all monitoring records created over the course of the previous day are fetched, then aggregated as described. Aggregations over longer periods of time (e.g. weekly, monthly, yearly) are then made by pulling and aggregating Monicron’s daily aggregations from MONIT’s database. The monitoring data that Monicron pulls is currently taken from two sources: XRootD monitoring data sent from the XRootd Servers to MONIT and ClassAds data sent to MONIT from a monitoring system that queries the Global pool queues or “HTCondor Schedulers.” These two in particular are used to form a total picture of the SoCal cache’s health, where XRootD and ClassAds monitoring data

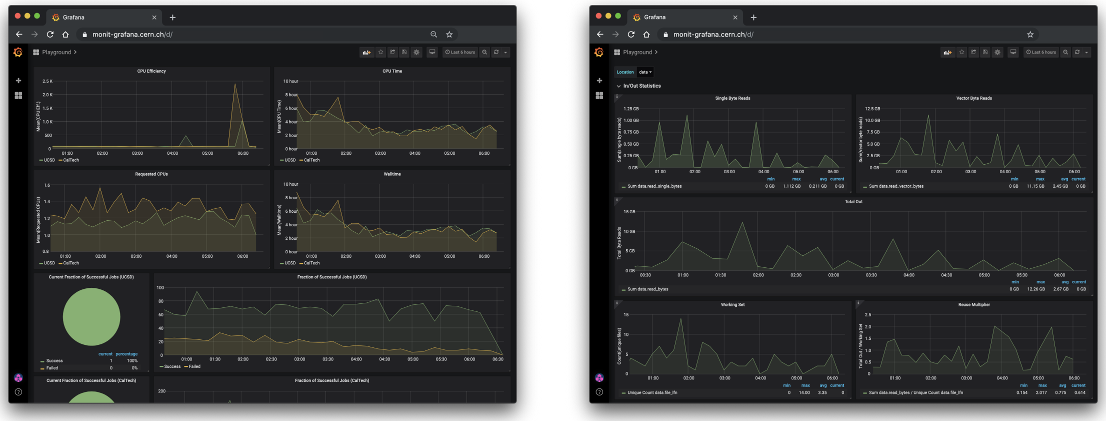


Figure 5: A prototype dashboard produced by the Grafana service, which will be used to visualize Monicrons output. It shows statistics about the performance of the jobs using the SoCal cache. (Left) Plots made using cache monitoring data. (Right) Plots made using job monitoring data.

provides metrics relevant to serverside and clientside performance respectively. The two can also be cross-referenced via a unique job id.

A complete list of the currently available metrics from Monicron is given in Table 2.

5.6 Discussion of Selected Caching Metrics

The SoCal MiniAOD cache has been reporting session logs to MONIT@CERN via the OSG RabbitMQ bus since late March 2019. In the following we show some of the metrics defined in Monicron for the time period late March to mid August 2019, and explain what we think we learn from them.

We made two types of plots, one that shows the time history of a metric, and a second that shows the distribution of a metric during the time interval analyzed. The latter is useful for understanding the distribution of typical values during the time period, while the former allows us to spot trends versus time.

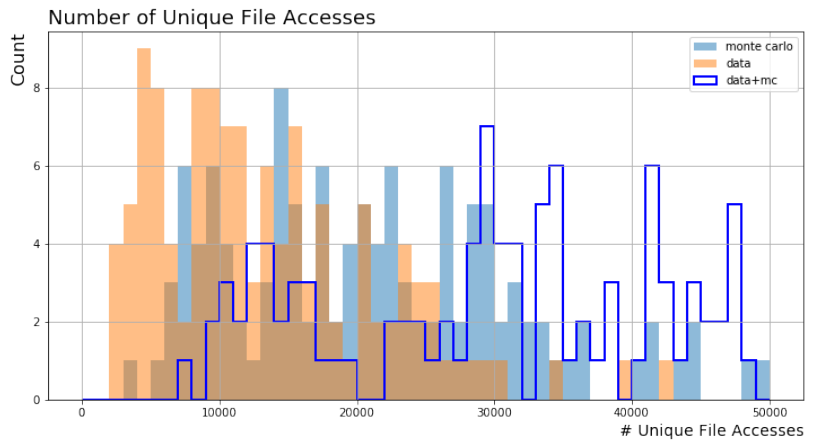


Figure 6: 1D histogram showing the distribution of the daily number of unique files accessed on the SoCal cache.

We start with the number of accesses and number of files accessed. Figure 6 shows the distribution of daily aggregates for simulations, labelled here as ”Monte Carlo”, and detector data, labelled here as ”data”, and the total. Simulation is accessed more often than detector data. We hypothesize that this is simply a reflection of CMS having more simulated than detector data.

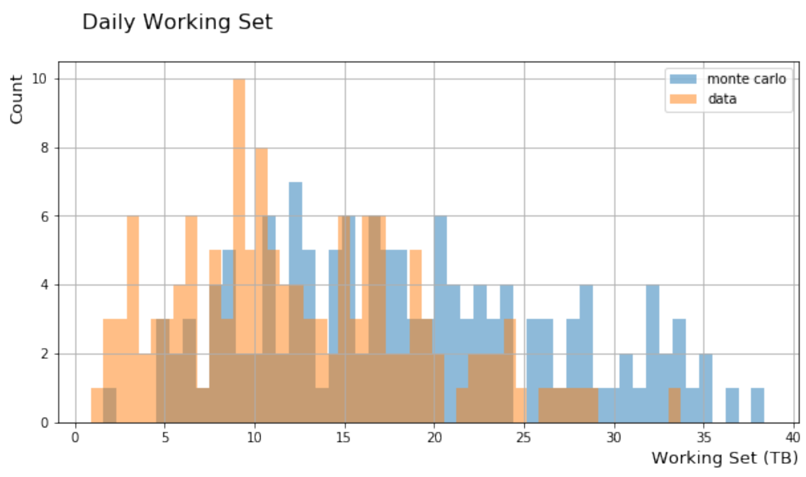


Figure 7: 1D histogram showing the distribution of the daily working sets on the SoCal cache.

Figure 7 shows the distribution of daily working sets for both types of data. On any day, the working set has always been less than ∼40TB. At 40kB/event on average, this is implies that up to 1 billion events were accessed in a given day, with more normal values being half that size each for the two types of data. This should be compared to ∼ 60 Billion events total for Run2 for a given version of the MiniAOD processing.

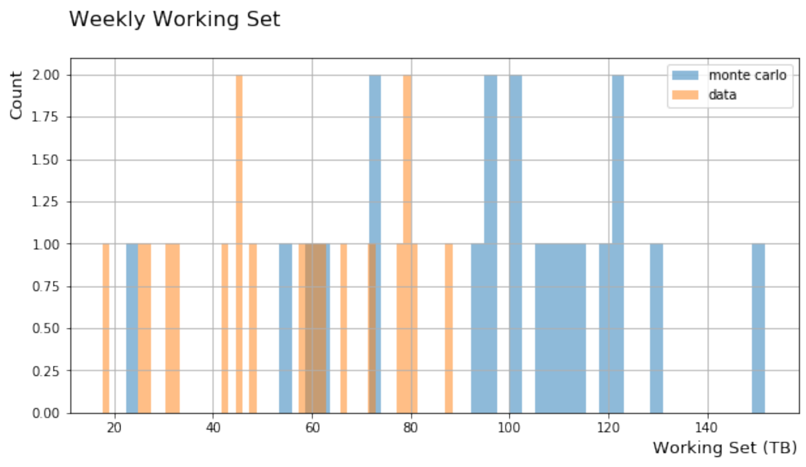


Figure 8: 1D histogram showing the distribution of the weekly working sets on the SoCal cache.

Figures 8 and 9 show the distribution of weekly and monthly working sets for both types of data. These plots give us a way to estimate whether the size of our cache is good or not. Figure 9 tell us that the SoCal cache should be able to hold a month worth of data before starting to replace files.

Figure 10 shows the time history of the daily working set. We note two things, first, the working

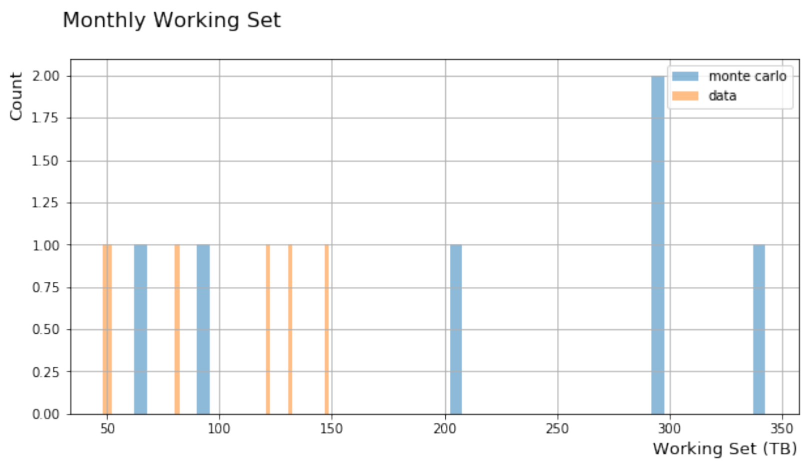


Figure 9: 1D histogram showing the distribution of the monthly working sets on the SoCal cache.

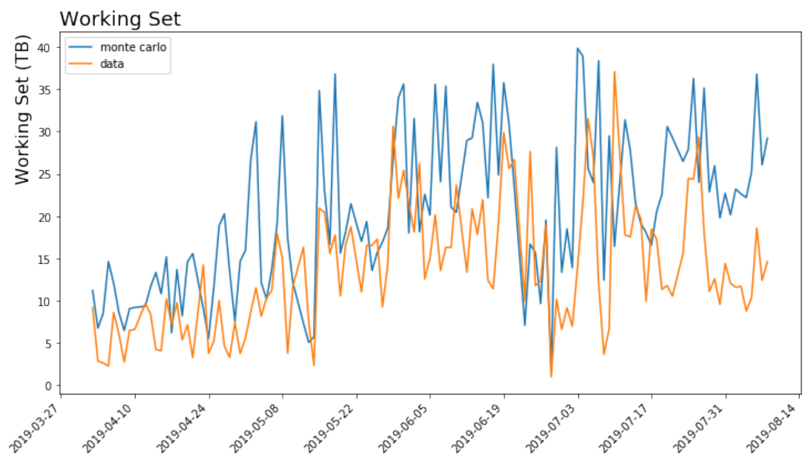


Figure 10: Time series plot of the daily working sets on the SoCal cache.

set size fluctuates from day to day by about a factor 2, from about 20-40TB when both types of data are added. Second, there is a small trend towards increasing working set size. We hypothesize that this is a result of a new version of the MiniAOD starting to see more use, while the previous version is still in use.

Figures 11, 12 and 13 show the Reuse Multiplier $\mathcal{R}_2$ over time calculated daily, weekly and monthly respectively. This metric shows the relation between the amount of data accessed and the working set, for example, the daily $\mathcal{R}_2$ is calculated summing up the size of all the files accessed in a given day and divided by the sum of the size of the unique set of files accessed in that day. The longer the time range, this is, month ¿ week ¿ day the larger the is going to be the working set(denominator) but we can see that the reuse of the data is even larger than that.

Looking at figure 11 we wanted to see how was the distribution of file reuse, i.e. how many times a file is accessed, within 2 apparently different periods of time. One that looks average with respect to figure 11 and another that features a moderated peak of $\mathcal{R}_2$, figures 14 and 15 show that respectively. These plots tell us that if we could predict what files are in the last couple bins, we would be able to massively increase the average reuse fraction by deleting the files in the first

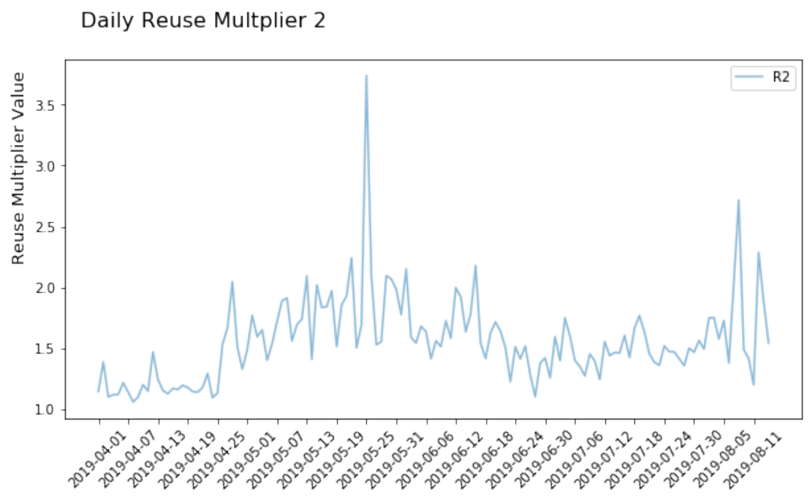


Figure 11: Time series plot of the daily Reuse Multiplier $\mathcal{R}_2$ on the SoCal cache.

couple bins rather than the others.

Figure 16 shows the the distributions for reuse multipliers $\mathcal{R}_2$ and $\mathcal{R}_3$. Given the way the caches are configured, $\mathcal{R}_2$ effectively measures space reuse. If a client reads even just one byte from a file, that entire file will get cached eventually. The idea here is that different users and applications may read different parts of a file, but any part of a file that is accessed will eventually be read by somebody, and is thus worth caching speculatively. For $\mathcal{R}_2$ the numerator counts the full filesize of a file for every job that accesses anything in that file, while the denominator counts the working set. In contrast, $\mathcal{R}_3$ counts only the total bytes read from all files that were accessed that day in the numerator. The ratio of $\mathcal{R}_3/\mathcal{R}_2$ measures the fraction of a file that is read on average by a job that accesses that file. Comparing the distributions for $\mathcal{R}_2$ and $\mathcal{R}_3$ indicates that only O(10)% of a file is read by a typical job, and this fraction is roughly a factor 2 smaller for detector data than simulation. The same can be learned from Figure 17. Here we show the total naive and actual reads, i.e. the numerators of $\mathcal{R}_2$ and $\mathcal{R}_3$.

Finally, Figure 18 shows the actual daily reads versus time, i.e. the numerator of $\mathcal{R}_3$ versus time. We notice a striking peak around July 21st or so. Figure 19 shows the walltime consumed by jobs from HTCondor ClassAds. We note that while there is a similarly striking peak on July 7-9, the two peaks do not coincide in even the same week, and thus must be considered uncorrelated.

Obviously, this data deserves further analysis. However, as is, it already shows that there is merit in caching because reuse of files is high even on the same day. Figure 20 shows the naive and actual replication factor, i.e. $\mathcal{R}_2$ and $\mathcal{R}_3$, for a one week period. As expected, both increase as we choose a larger time window. I.e. the working set grows slower as we increase the aggregation time than the total read volume.

We think that the most interesting question to pursue further is whether or not partial reads are predictable, and what their structure is. This is discussed in more detail in the document "LHC Data Access Patterns". We think future R&D work should be focused on that question, after completion of the Monicron dashboard.

6 US CMS NANOAOD Use Case

As next step in the deployment of XRootd caching in US CMS, the ops program decided to ask the seven US CMS T2s for volunteers to deploy a 100TB cache to host all of the NANOAOD. The

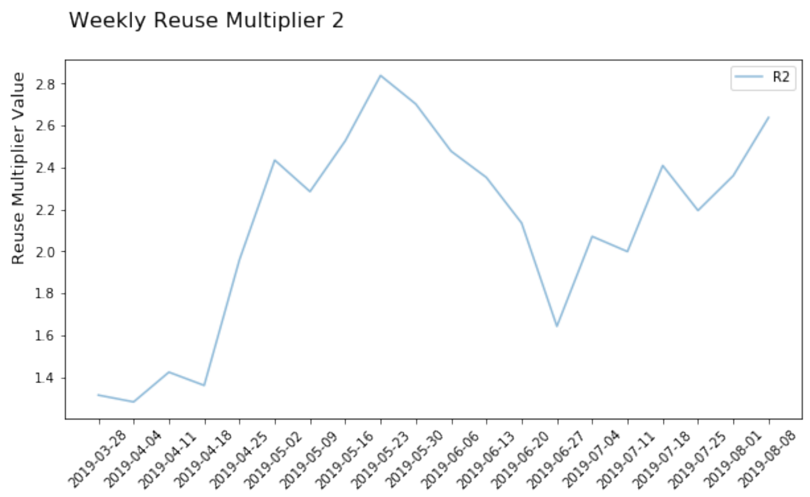


Figure 12: Time series plot of the weekly Reuse Multiplier $\mathcal{R}_2$ on the SoCal cache.

main purpose of this is threefold. First to allow the participating T2s to familiarize themselves with the technology. Second, to have multiple T2s provide feedback on the deployment and operations support of this technology via OSG-LHC in IRIS-HEP, and third, to make the NANOAOD widely accessible, and thus motivate a larger community to transition to its use.

The NANOAOD use case will build on the experience with the MINIAOD use case, including the "dashboard" development described in the previous section. Initial deployment was completed at Purdue in August 2019. Integration into the global pool, and the dashboard infrastructure via MONIT is ongoing.

Among the planned caches is one that engages ESnet via a distributed NANOAOD cache across ESnet at Sunnyvale, Caltech, and UCSD. The rational here is to engage ESnet on a project with storage in the network backbone in a similar way as OSG has engaged Internet2. In the case of OSG, this has led to 3 deployed caches at POPs in Kansas City, Chicago, and Manhattan, with three more planned (Amsterdam as an extension of a heavily used cache at University of Amsterdam, Houston, and the Bay Area).

7 US ATLAS Use Case

ATLAS run2 data use is significantly different than CMS. The equivalent of the MINIAOD in CMS is called XAOD in ATLAS. However, while there is only one MINIAOD for the entire CMS collaboration across all physics groups, there are many version of the XAOD in ATLAS. CMS thus has successfully accomplished a feat of social engineering (convincing the collaboration in its entirety to use the same data format) that ATLAS is planning to achieve for HL-LHC.

As a result, reuse of data is much less prevalent in run2 for ATLAS than CMS, simply because in CMS everybody uses the same data, while in ATLAS different groups within the experiment use completely different data formats.

In addition, the ATLAS IO stack is more latency sensitive than in CMS. While CMS makes extensive use of remote reading during reconstruction, as well as otherwise, ATLAS benefits from hiding data access latencies through caching. The primary use case for caching in ATLAS for Run2 is thus latency hiding rather than data reuse, while for CMS it is data reuse.

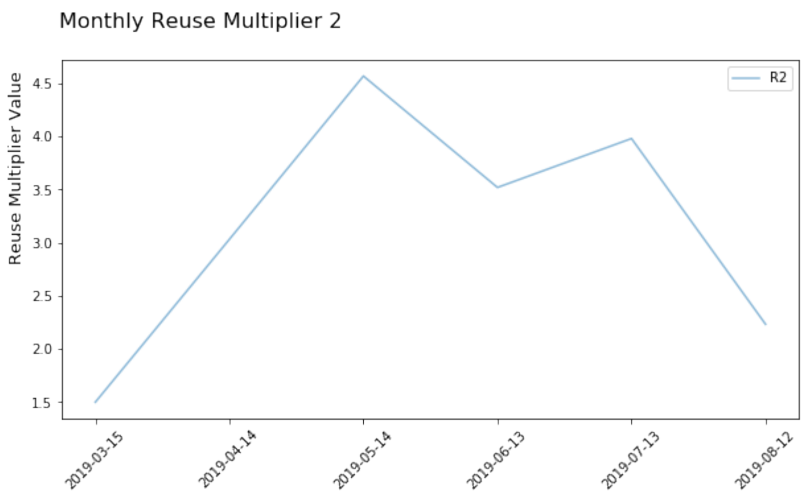


Figure 13: Time series plot of the monthly Reuse Multiplier $\mathcal{R}_2$ on the SoCal cache.

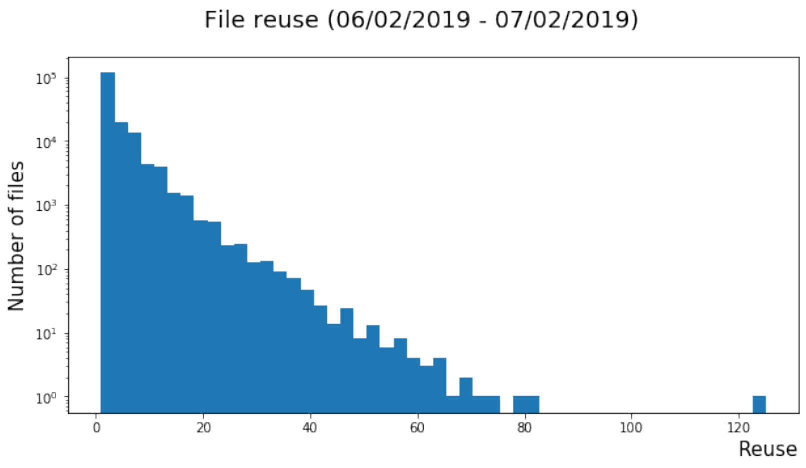


Figure 14: 1D histogram showing the distribution of file reuse.

Name	Calculation	Description
Number of Unique Files	$\mathcal{N}_{files}$	The number of files that were accessed
Working Set	$w = \sum_{\mathcal{N}_{files}} size(f_i)$	Sum of the file sizes of all unique files that were read
Total "Naive" Reads	$r_{naive} = \sum_i size(f_i)$	Sum of the file sizes for all files read
Total "Actual" Reads	$r_{actual} = \sum_i bytes(f_i)$	Sum of the bytes read from all files that were read
Number of Unique File Accesses	$\mathcal{N}_{access} = \sum_{\mathcal{N}_f} \mathcal{N}_{jobs}(f_i)$	Where "unique file accesses" are defined as the number of unique jobs that accessed a particular file
Reuse Multiplier 1	$\mathcal{R}_1 = \frac{\mathcal{N}_{access}}{\mathcal{N}_{files}}$	"Reuse" of cache quantified in terms of accesses: if number of accesses > 1 for most files in working set, cache is effective
Reuse Multiplier 2	$\mathcal{R}_2 = \frac{r_{naive}}{w}$	"Reuse" of cache quantified in terms of disk space: if sum of size of files used > working set, cache is effective
Reuse Multiplier 3	$\mathcal{R}_3 = \frac{r_{actual}}{w}$	"Reuse" of cache quantified in terms of the disk space that is actually used: if total bytes read > working set, cache is effective
Number of Unique Jobs	$\mathcal{N}_{jobs}$	Number of unique jobs
Number of Unique Users	$\mathcal{N}_{users}$	Number of unique users
Total Walltime	$W_{tot} = \sum_i walltime(j_i)$	Sum of the walltime of all recorded jobs
Total CPU Time	$C_{tot} = \sum_i cputime(j_i)$	Sum of the CPU time of all recorded jobs
CPU Efficiency	$\epsilon = \frac{C_{tot}}{\sum_i (walltime(j_i) \times N_{cores}(j_i))}$	CPU efficiency calculated over all recorded jobs
Fraction of Jobs w/ Exit Code 0	$F = \frac{N_{success}}{N_{jobs}}$	Total number of jobs with exit code zero divided by the total number of recorded jobs

Table 2: All available metrics from Monicron

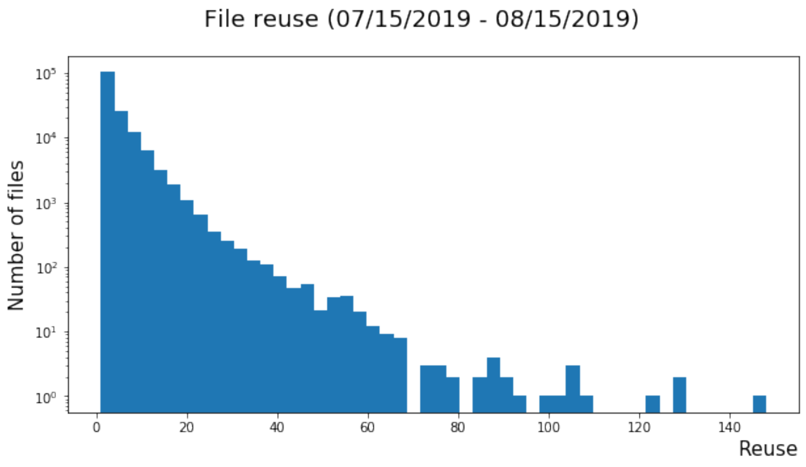


Figure 15: 1D histogram showing the distribution of file reuse.

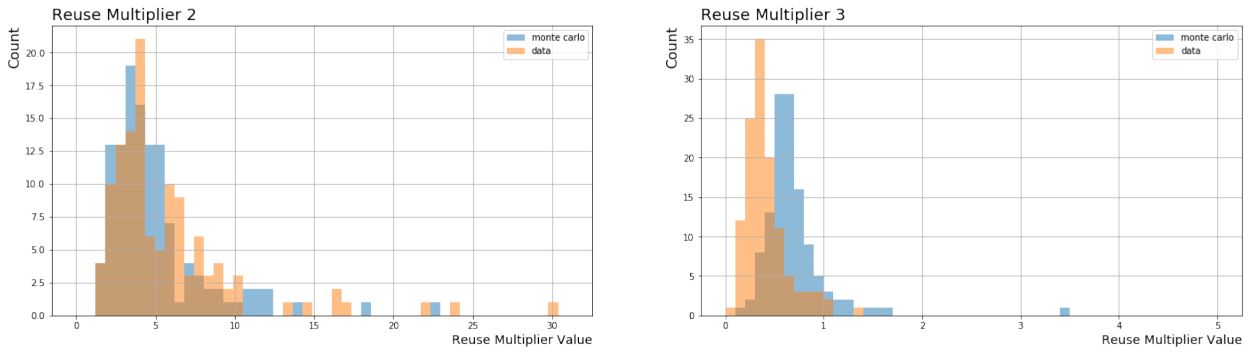


Figure 16: 1D histogram showing the distribution of the daily values of the reuse multipliers $\mathcal{R}_2$ (left) and $\mathcal{R}_3$ (right) for the SoCal cache.

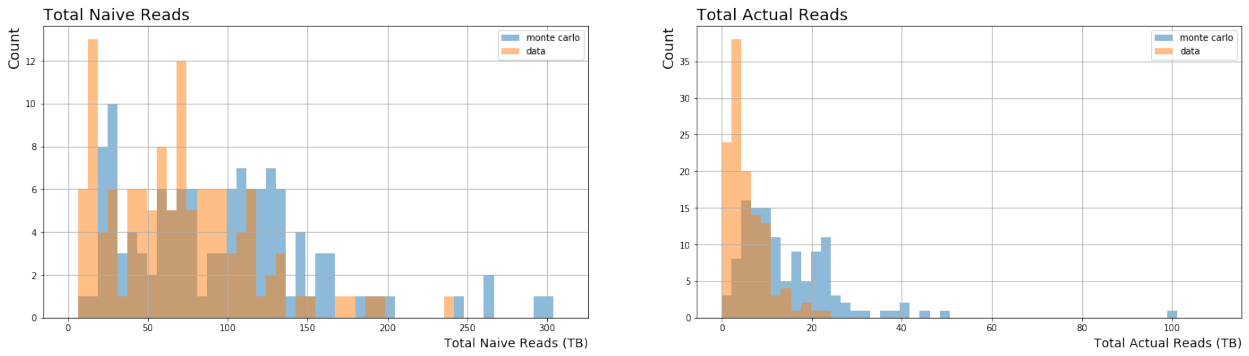


Figure 17: 1D histogram showing the distribution of the daily "naive" reads (left) and "actual" reads (right) for the SoCal cache.

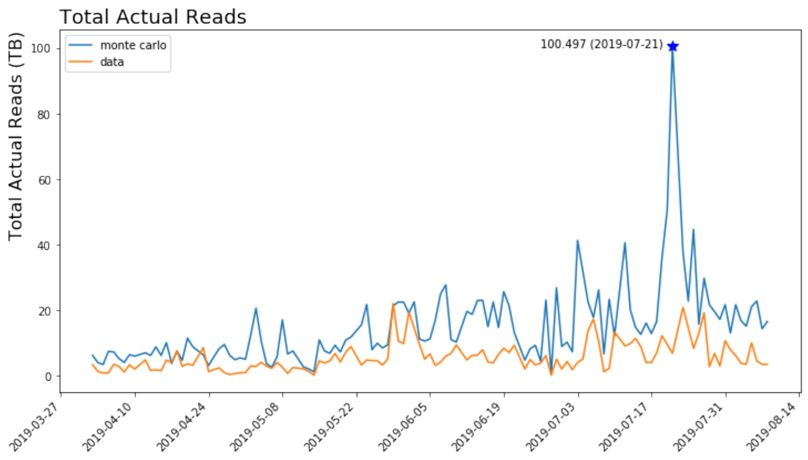


Figure 18: Time series plot showing the daily number of "actual" reads for the SoCal cache. The maximal point of the significant peak in the middle of July is marked by a star and labeled with the value and date of that point.

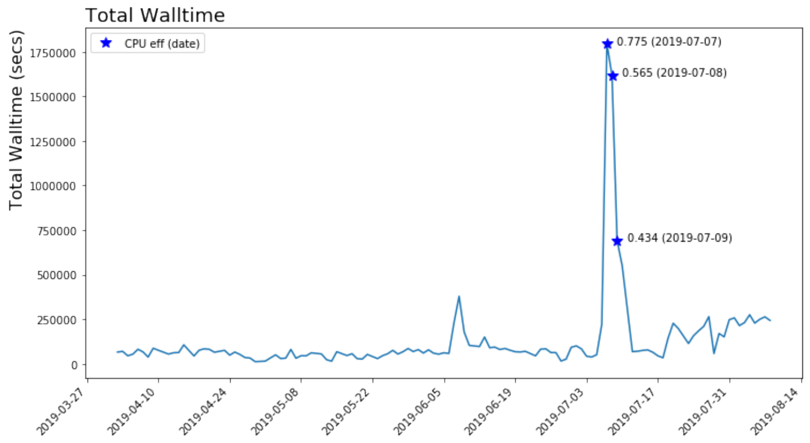
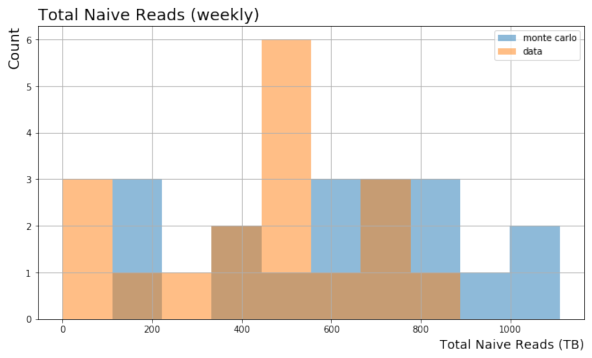


Figure 19: Time series plot showing the daily total walltime for jobs accessing the SoCal cache. Some maximal points of the significant peak in the beginning of July is marked by stars and labeled with the values and dates of those points.



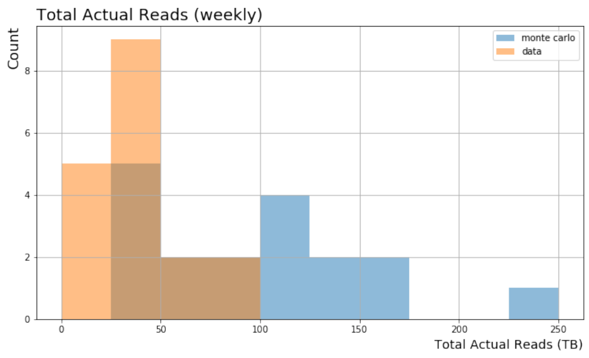
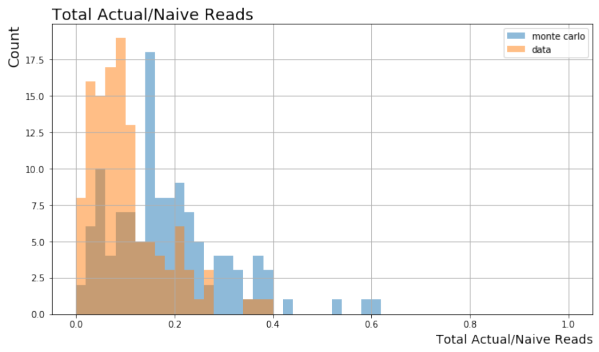


Figure 20: 1D histogram showing the distribution of the weekly "naive" reads (left) and "actual" reads (right) for the SoCal cache.



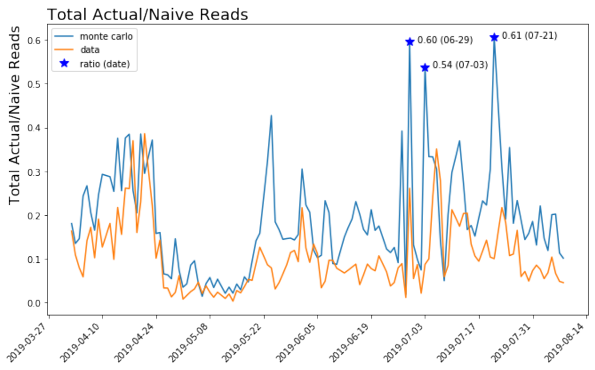


Figure 21: Plots of the ratio of the daily "actual" and "naive" reads, i.e. $R = r_{actual}/r_{naive}$. (Left) 1D histogram showing the distribution of R. (Right) Time series plot of R where maximal points above $R = 0.5$ for simulation (Monte Carlo) file reads are marked by a star and labeled with the date and value at that point.