Advancing throughput of HEP analysis work-flows using caching concepts

Rene Caspart[1], *Max* Fischer[1], *Manuel* Giffels[1], *Christoph* Heidecker[1,*], *Eileen* Kühn[1], *Günter* Quast[1], *Martin* Sauter[1], *Matthias J.* Schnepf[1], and *R. Florian* von Cube[1,2]

[1]Karlsruhe Institute of Technology
[2]Rheinisch-Westfälische Technische Hochschule Aachen

Abstract. High throughput and short turnaround cycles are core requirements for efficient processing of data-intense end-user analyses in High Energy Physics (HEP). Together with the tremendously increasing amount of data to be processed, this leads to enormous challenges for HEP storage systems, networks and the data distribution to computing resources for end-user analyses. Bringing data close to the computing resource is a very promising approach to solve throughput limitations and improve the overall performance. However, achieving data locality by placing multiple conventional caches inside a distributed computing infrastructure leads to redundant data placement and inefficient usage of the limited cache volume. The solution is a coordinated placement of critical data on computing resources, which enables matching each process of an analysis work-flow to its most suitable worker node in terms of data locality and, thus, reduces the overall processing time. This coordinated distributed caching concept was realized at KIT by developing the coordination service NaviX that connects an XRootD cache proxy infrastructure with an HTCondor batch system. We give an overview about the coordinated distributed caching concept and experiences collected on prototype system based on NaviX.

1 Introduction

Efficient usage of computing resources is challenging for data-intensive work-flows. The performance and thus the efficiency is limited by the I/O rate. Especially within distributed computing resources, the data throughput depends highly on connections to remote servers providing data storage [1]. Bringing processing and storage resources as close as possible together can avoid limitations caused by shared connections and bottlenecks in WAN bandwidth. This *data locality* can be established on any scale such as globally within a computing community, regionally within a computing cluster, locally on a per-host basis, etc. The collaborations participating in the Worldwide LHC Computing Grid (WLCG) [2] consider data locality on scale of computing centers when distributing work-flows. Other systems focus on bringing either storage close to the processing host via distributed storage systems such as Hadoop [3], which requires high-performance storage on processing host. Or they schedule the work-flows directly on storage servers such as BEER [4], which requires sufficient processing power on storage nodes. Whereas the first concept enables data locality on the scale

*e-mail: christoph.heidecker@kit.edu

of racks and hosts, the latter one mainly profits from additional computing resources without paying any special attention to data locality. These systems suffer from inefficient resource utilization when facing the enormous amounts of data produced by future HEP experiments.

Here, caching as common concept for optimization of repeatedly processed data fits well. Storing such kind of data within high-performance caches improves access to the corresponding data, but also reduces the load on the network, which improves the overall data throughput. Our research [5–7] suggests that the granularity of the data locality should be based on individual hosts, although usually providing the best data throughput, is not always necessary or feasible. While disk or memory caches improve the throughput of single hosts, proxy servers placed within a computing cluster are able to speed up the access of a set of hosts. Since we aim at improving the overall performance, we need to determine the best feasible design suitable for the underlying infrastructure. However, conventional caches deployed on single hosts in a distributed computing infrastructure are not used efficiently when accessing huge amount of data is the limiting factor. Repeatedly processed work-flows potentially run on different hosts and, thus, the accessed data is then redundantly cached on multiple hosts and the limited cache volume is partly wasted. Furthermore, cached data is not used by work-flows running on different hosts, and thus the load on the network and remote storage systems is not reduced.

A coordination of data on distributed caches and matching of work-flows to the most suitable host in terms of data locality is therefore mandatory for boosting performance and efficiency.

2 Optimized management of distributed caches

Our coordinated distributed caching concept is based on a generalized model for data processing, that is derived from common HEP analysis processing. A HEP data analysis is usually organized in a sequence of *work-flows* that perform analysis specific tasks. Each work-flow processes a *dataset*, which is a subset of the entire data volume. For processing, the work-flow is split up into multiple *jobs*, which are instances that process independent subsets of the dataset. The jobs are sent to *worker nodes*, which are various hosts that serve as distributed computing resources for batch processing. Although the execution of jobs is distributed, all worker nodes share a single global namespace for data storage. The presented caching concept is optimized for this computing model but not strictly bound to it.

There are two main challenges that need to be overcome within a coordinated distributed caching system: On the one hand, we need to select the most relevant data for caching to optimize the overall data throughput. On the other hand, we have to match the data-intensive tasks to the most suitable computing resource that provides locality of the input data. Although both challenges can not be answered separately, the presented concept distinguishes them to simplify optimization and to enable scalability. As caches are distributed within the computing infrastructure and adjusted to the subsystem they serve, each cache is treated independently. Within this subsystem, the scheduling of data needs to be adopted to the characteristics of the cache volume, the surrounding infrastructure and the accessing worker nodes. We assume that all worker nodes within the computing infrastructure are managed by an overlay batch system that matches jobs to worker nodes. The overlay batch system, however, can make improved decisions of job placement by integrating the meta-data about data distribution in caches into the match making. This allows sending jobs to the computing resource, which is most suitable also in terms of data locality. By matching jobs to most suitable worker nodes, the batch system indirectly influences data access and, thus, takes care of the data placement on caches. It can, therefore, also be used to coordinate data to caches by sending jobs to specific computing resources, which triggers caching there.

2.1 Data selection and placement

The coordinated distributed caching concept aims at optimizing a fraction of work-flows that access huge amounts of input data stored on Grid storage systems and are affected by throughput limitations mentioned above. We expect a mixture of jobs from different kind of work-flows to be processed. Computing resources that are blocked by processing of data-intensive work-flows are being freed faster, since these jobs directly benefit from cached input files and their processing time is reduced. Although, work-flows that do not directly benefit from caching, benefit from overall reduced processing time, since resources are freed earlier. Work-flows that only read a minor part of the accessed file or do a sophisticated processing of accessed data can be excluded from caching. In addition, work-flows that process data only once do not benefit from caching and should also be excluded. This reduces the effective data volume, so that also small sized caches can achieve acceptable cache hit rates and low cache trashing. Hence, also commercial cloud systems can profit from coordinating small-sized caches.

For coordinating the data selection and placement, each cache can either make decisions on its own or use a central decision-logic that coordinates all caches within the computing infrastructure. Since coordinating jobs to computing resources already influences data placement, an additional centralized management for all caches is not explicitly required. Hence, we focus on improving a local cache decision-handling, which does not require synchronization of meta-data as it is necessary for a centralized approach. We assume, that each cache can make sufficiently accurate decisions with locally available information. Here, different local cache decision approaches can be used to optimize data selection and placement.

2.2 Optimization of overall data throughput

We aim at increasing the overall data throughput of the computing infrastructure and thus not focus on boosting the performance of isolated jobs, but all work-flows together. Caches reduce concurrent data access on remote storage systems by jobs and reduce congestion on data sources and network infrastructure. These caches serve as additional data sources that not necessarily have to be faster than data transfer via network, which is usually shared. Bandwidths provided by SSD and HDD caches $r_{\text{cache}} \approx 1\,\text{Gbit/s}$ to $5\,\text{Gbit/s}$ are sufficient, compared to those of shared network connections $r_{\text{network}} \approx 1\,\text{Gbit/s}$ to $10\,\text{Gbit/s}$. Combining the performance of multiple SSDs or HDDs allows to deliver the required read rate per job even for an arbitrary large number of parallel data accesses. The theoretical prediction of the improvement of work-flows in terms of processing time $t_{\text{processing}}$ that relies on reading performance is calculated by Equation 1.

$$t_{\text{processing}} = max\left(\frac{(1 - x_{\text{cached}}) \cdot V_{\text{data}}^{\text{total}}}{r_{\text{network}}}, \quad \frac{x_{\text{cached}} \cdot V_{\text{data}}^{\text{total}}}{r_{\text{cache}} \cdot n_{\text{worker nodes}}}, \quad \frac{V_{\text{data}}^{\text{total}}}{r_{\text{work-flow}} \cdot n_{\text{slots}}^{\text{total}}} \right) \tag{1}$$

It depends on the total amount of input data $V_{\text{data}}^{\text{total}}$ accessed by the work-flow, fraction of cached data x_{cached}, the number of processing worker nodes $n_{\text{worker nodes}}$ and the slots in total $n_{\text{slots}}^{\text{total}}$. Since the limiting factor of the work-flow itself $r_{work-flow}$ highly depends on the type and the implementation of the analysis, is can be neglected for pure cache performance comparisons. Nevertheless, Equation 1 has a minimum processing time at a certain fraction of cached data. At this point, we achieve the maximum data throughput combining the available bandwidths from cache and remote storage system. Hence, we need to exclude a certain fraction of each dataset from caching in order to achieve the minimal overall processing time.

3 Realization of the coordinated distributed caching concept

Based on former studies and experiences [5–7], we developed a coordinated distributed caching system that is based on software components that are commonly used in HEP: The HTCondor batch system [8] schedules jobs to computing resources in a highly scalable manner. Additionally, we use the XRootD data transfer protocol [9] for handling storage or cache access. This allows us to stream block-based data efficiently and it also provides basic caching functionalities. Bringing both systems together allows for matching jobs to the most suitable computing resources in terms of cached input data. Furthermore, it allows for coordinating data on distributed caches to shrink replication of data and optimize data placement.

3.1 Basic prerequisites of the implementation

We use the HTCondor batch system for distributing jobs to various different computing resources. These computing resources are typically clustered into computing clusters, which can reach from few to hundreds of similar worker nodes. For the coordination of jobs to the cached input data, we need to include the required information into the HTCondor job scheduling. Therefore, HTCondor allows various so-called *hooks* to manipulate submitted jobs and, thus, influence the job to resource matching. For the implementation of the job to data coordination, we use three types of hooks: A *translate hook* is called for each job after submission, whereas an *update hook* allows to periodically update the job when waiting for available resources. We use the translate hook to integrate matching information into the scheduling cycle. The update hook continuously updates the coordination-information. Finished jobs may invoke *finalize hooks*, which are currently used to monitor job and caching performance.

The data transfer protocol XRootD provides data management services, such as *data servers*, *management servers*, and *proxy servers*. XRootD data servers enable access to a connected storage volume. Management servers combine multiple data servers within a hierarchical structure to provide a global namespace. Hence, an XRootD client request to the top-level management server is redirected to a data server providing the requested file. An intermediate XRootD proxy server that forwards data access requests can also serve as *cache proxy server* [10] when connected to storage volume. These cache proxy servers can easily be placed at any kind of computing resource that provides temporary storage volume and network connection, which enables a flexible provisioning of caches. Hence, we can adjust the number of caches according to the required data throughput of a computing cluster or a single worker node to the hardware and the infrastructure.

The coordinated distributed caching concept requires adjustable cache decision-logic to influence the data selection and placement, and support of various kind of storage systems to enable distributed caches on opportunistic computing resources. Since XRootD cache proxy servers provide both features and can be managed within the above-mentioned hierarchical structure, they fit perfectly into the concept. XRootD system configurations allow to automatically redirect XRootD clients version 4.7 and higher to a pre-configured proxy server. Lower versions need to be manually adjusted to find the proxy.

3.2 Conceptual design of NaviX coordination service

The coordinated distributed caching requires meta-data of both systems, HTCondor and XRootD, to be combined for matching jobs to most suitable worker nodes in terms of cached files. This includes information about cache contents, jobs, and the surrounding computing infrastructure. For calculating the coordination decision, we developed a specialized coordination service called NaviX [11], which intermediates between XRootD and HTCondor. The general structure of NaviX is shown in Figure 1. Triggered by a special HTCondor translate hook, it calculates a score for each job cache combination that describes how well each cache fits to the job. It integrates this information into the HTCondor job description to influence the job to resource matching. For the score calculation, NaviX extracts information such as

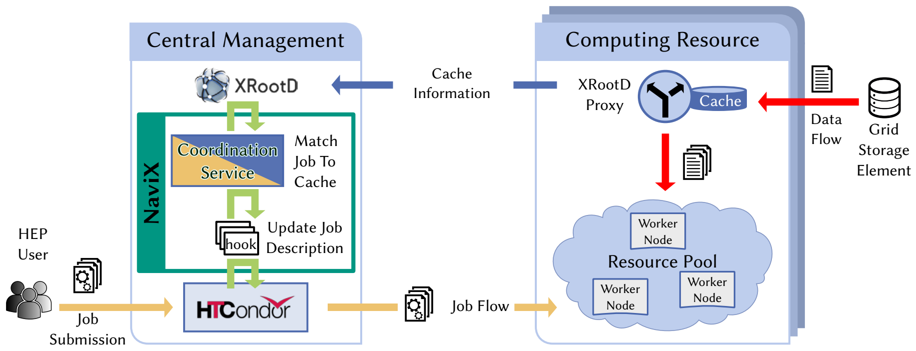

Figure 1. The NaviX coordination service connects the XRootD based cache proxy system with the HTCondor batch system.

the location of cached files using the XRootD hierarchy. Meta-data requests to the top-level XRootD manager are propagated down to the cache proxy servers and the information is cached by all XRootD managers for repeated usage. This allows an efficient and scalable aggregation of the required information. In our studies, we observed that meta-data of the caches can be outdated to some extent, which does not have a major impact, since access to data deleted from the cache is automatically redirected to the original storage. Overall, we have a time delay between cache matching and job processing caused by aggregation and processing of meta-data, job scheduling, job start-up, and the redirection of data accesses within the XRootD infrastructure. Since conventional HEP jobs usually run for several hours, this time delay in the order of minutes can be neglected and therefore has little effect on data throughput.

NaviX itself gets the meta-data of the job via the HTCondor hook, which also needs to contain a list of files that will be processed. Since it is not possible to automatically retrieve the catalog of files, needed for a specific job, user input is required during the creation of the jobs. After extracting and comparing job and cache meta-data, NaviX calculates a ranking of suitable caches and changes the job meta-data accordingly. This forces HTCondor to match the job to the most suitable computing resource with the highest amount of cached files. On the one hand, the pre-scheduling of jobs to certain caches done by NaviX reduces the load on the HTCondor system, since it can be calculated in parallel separately from the HTCondor decision. On the other hand, this pre-scheduling can lead to inconsistencies in the HTCondor decision, which must then be intercepted. The logic of NaviX decision-making can easily be changed and extended, so all kind of information can be included for optimization. Including for example information about the load of each cache or the utilization of the computing resources may improve the overall efficiency. After beeing pre-scheduled by NaviX via the HTCondor translate hooks, jobs waiting for resources to become free are repeatedly updated using update hooks. This allows NaviX to change the resource matching, when new files are available within caches or intercept when above-mentioned inconsistencies are detected. Furthermore, it allows NaviX to release job requirements, so that jobs can run on different kind of computing resources with a lower amount of cached files. However, NaviX has to balance high data throughput using the most suitable cache against long waiting times. Reducing the waiting times and advancing the NaviX decision-making, is one of the future fields of research we will address. After finishing the processing, each job is analyzed using a finalize hook. NaviX as central connection of HTCondor and XRootD allows the aggregation of information about job and cache performance that is used for monitoring and fine-tuning of the system.

4 Prototype system for coordinated distributed caching via NaviX

A first prototype of the coordinated distributed caching system using the NaviX coordination service is installed at the Institute of Experimental Particle Physics at KIT.

4.1 Testing environment and setup

The prototype system includes a host that provides an HTCondor management instance, a top-level XRootD manager, and the NaviX coordination service. Furthermore, three hosts serve as worker nodes processing jobs and providing XRootD cache proxy servers. For the benchmarks presented in this paper, we used a basic HTCondor and XRootD setup as well as a simple job-cache matching-logic that prefers computing resources with a high fraction of cached files implemented into NaviX. For that purpose, we aggregate the location of cached files and calculate a score, which enforces HTCondor to match jobs to the corresponding worker nodes. All caching proxy servers use the basic XRootD caching-logic that caches files considering the time since last access. This means that all accessed files are cached, and when the cache volume reaches its limit, old files are deleted from the cache. Each cache proxy server is connected to a software RAID 0 of SSDs that provides about 2 TB cache volume with a measured mean read rate of 9.6 Gbit/s per worker node. All hosts are treated as separated systems, so that access to files is restricted to the own cache proxy. Whereas the interconnection between the hosts is limited to 10 Gbit/s, all hosts share a connection to remote storage systems, which reaches about 6 Gbit/s. Datasets used for benchmarking are stored on common WLCG storage systems and offer a typical reading performance of about 100 Mbit/s in average per file in the context of our benchmarks with up to 60 parallel read accesses.

4.2 Benchmarking data throughput

Two different types of work-flows are used to benchmark the prototype system. To measure the maximum achievable performance, the test jobs only read data without further processing. Furthermore, we use a data-intensive CMS jet energy calibration work-flow that performs selection and pre-processing step on the data to estimate the improvement for typical HEP analysis work-flows. To test the functionality of coordinating jobs to data, we repeatedly process these test jobs and increase the fraction of files to be cached with each run. This allows us to monitor the mismatch between jobs and caches during the coordination process and the stability of the implementation when processing multiple jobs in parallel. Repeated processing of jobs on different worker nodes would lead to repeated caching of files on different caches. Over multiple test runs, this would end up in increasing file duplication. Since we observe that no file is cached more than once during all test runs, the basic coordination-logic appears to succeed in matching jobs to their most suitable caches.

The results of the performance benchmarks in Figure 2 (left) show that the prototype system improves the data throughput performance and thus the processing time of the jobs to a

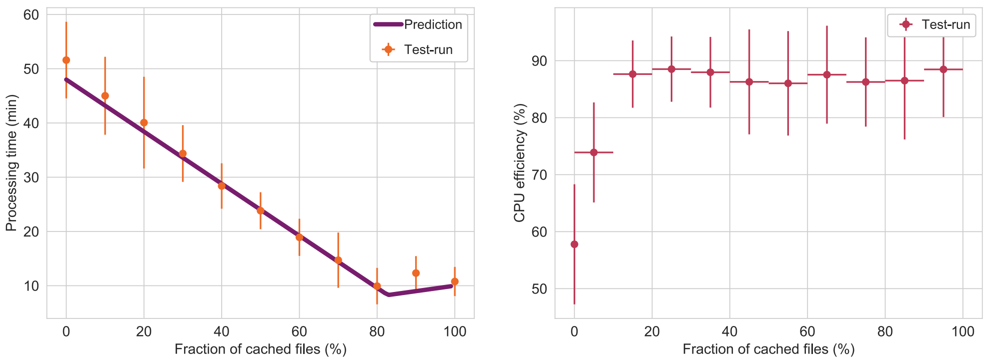

Figure 2. Performance benchmarks allow to scan for optimal working point (left), whereas a CMS jet energy calibration work-flow allows to estimate the real improvement of a typical HEP analysis. Here, data points show the mean value, error bars the corresponding standard deviation.

factor of 5. With increasing fraction of cached files, the processing time decreases and reaches its minimum, and thus maximum data throughput, at about 80%. Beyond this minimum, the processing time increases slightly caused by reading the data from the cache itself and ignoring the available network bandwidth. Hence, we observe an optimal working point, which we can also calculate using Equation 1 neglecting limitations caused by the test work-flow itself. Comparing predictions with the performance benchmarks, the data points show good agreement within the variance extracted of the set of benchmarks.

Additionally, the CPU efficiency of typical HEP analysis jobs was measured as shown in Figure 2 (right). Increasing the amount of cached files and thus the data throughput, we are able to increase the CPU efficiency of the CMS jet energy calibration work-flow up to 89% by caching about 20% of the input files. A further increase of the CPU efficiency is blocked by the data throughput limitation of the work-flow itself. As described in Section 2.2, the optimal working point highly depends on the specific cache environment and the work-flow itself. For further improving the setup, we will research on how to individually consider the optimal working point of each work-flow within the cache decision-logic.

5 Conclusion

The coordinated distributed caching concept optimizes the overall data throughput of data-intensive work-flows by caching critical data on distributed caches. Coordinating data to caches enables the efficient use of limited cache volumes by reducing replication of data on distributed caches. Furthermore, scheduling each job to the most suitable cache in terms of cached files optimizes overall processing efficiency. We build a prototype system based on common HEP tools, the batch system HTCondor and the data transfer protocol XRootD, to test the advantages of a coordinated distributed caching. We developed the central coordination service NaviX, which interacts between both parts to match jobs to the most suitable computing resource in terms of data locality. This also allows to coordinate the content of caches in order to reduce replication of data and optimize data placement on distributed caches. Here, a simple caching- and coordination-logic already leads to a large improvement of data throughput and a more efficient utilization of the computing resources compared to directly processing remote data. The coordinated caching system thus releases pressure from the entire infrastructure and accelerates the processing of all kind of HEP work-flows by freeing resources earlier.

6 Outlook

The presented implementation of the coordinated distributed caching concept is set up with a simple, but easily extendable decision-logic. Hence, further development will test the benefits of advancing the caching decision-logic of the XRootD cache proxy servers and the matching-logic of the NaviX coordination service. Here, we need to balance both scalability of the approach and optimizations achieved by the advanced decision-making. In addition, the overall processing time can be improved by extrapolating future data accesses from previous evolution of the jobs and prefilling the caches accordingly. In the future, the coordination service NaviX should tune itself automatically and adjust the matching-decision dynamically to a constantly changing infrastructure as it is required for efficiently handling of opportunistic resources.

References

[1] Schnepf M J, von Cube R F, Fischer M, Giffels M, Heidecker C, Heiss A, Kuehn E, Petzold A, Quast G, Sauter M *Dynamic Integration and Management of Opportunistic Resources for HEP*, EPJ Web of Conferences **Proceedings of CHEP 2018** (to be published) (2018)

[2] Eck C et al., *LHC computing Grid : Technical Design Report*, Technical Design Report LCG **URL** https://cds.cern.ch/record/840543 [Accessed 2018-10-07] (2005)

[3] The Hadoop project homepage, **URL** http://hadoop.apache.org [Accessed 2018] (2018)

[4] Jones B, Kirianov A, Lamanna M, Mascetti L, McCance G, Rousseau H, Schulz M and Smith D, *Sharing server nodes for storage and compute*, Journal of Physics: Conference Series, **URL** https://indico.cern.ch/event/587955/contributions/2937728 [Accessed 2018-10-07] (to be published)

[5] Fischer M, Metzlaff C, Kühn E, Giffels M, Quast G, Jung C and Hauth T, *High Performance Data Analysis via Coordinated Caches*, Journal of Physics: Conference Series **664** 9 092008 (2015)

[6] Fischer M, Heidecker C, Kuehn E, Quast G, Giffels M, Schnepf M, Heiss A, and Petzold A, *Opportunistic data locality for end user data analysis*, Journal of Physics: Conference Series **898** 5 052034 (2017)

[7] Heidecker C, Schnepf M J, Kuehn E, Fischer M, Giffels M, and Quast G, *Provisioning of data locality for HEP analysis workflows*, Journal of Physics: Conference Series **1085** 3 032005 (2018)

[8] Thain D, Tannenbaum T and Livny M *Distributed computing in practice: the Condor experience*, Concurrency and Computation: Practice and Experience, **17** 2-4 323–56 (2005)

[9] Dorigo A, Elmer P and Furano F and Hanushevsky A *XROOTD/TXNetFile: A Highly Scalable Architecture for Data Access in the ROOT Environment*, **Proceedings of TELE-INFO'05** 46:1–46:6, (2005)

[10] Bauerdick L A T, Bloom K, Bockelman B , Bradley D C, Dasu S, Dost J M, Sfiligoi I, Tadel A, Tadel M, Wuerthwein F, Yagil A, and the CMS collaboration *XRootd, disk-based, caching proxy for optimization of data access, data placement and data replication*, Journal of Physics: Conference Series **513** 4 042044

[11] Heidecker C and Sauter M, *NaviX*, **URL** https://gitlab.ekp.kit.edu/ETP-HTC/NaviX [Accessed 2018-10-22] (2018)