ETP-KA/2020-14

Karlsruher Institut für Technologie

Modeling of distributed coordinated caching for LHC data analyses

Masterarbeit von

Tabea Feßenbecker

an der Fakultät für Physik
des Karlsruher Instituts für Technologie (KIT)
Institut für Experimentelle Teilchenphysik (ETP)

Referent:	Prof. Dr. Günter Quast
Korreferent:	Priv.-Doz. Dr. Klaus Rabbertz

Karlsruhe, 15.05.2020

Erklärung zur Selbstständigkeit

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung vom 28.05.2018 beachtet habe.

Karlsruhe, den 15.05.2020, ______________________________
Tabea Feßenbecker

Als Prüfungsexemplar genehmigt von

Karlsruhe, den 15.05.2020, ______________________________
Prof. Dr. Günter Quast

Contents

Chapter 1

Introduction

For many decades high energy physics (HEP) has pushed scientific and technological limits and contributed significantly to technological advancement. Complex experiments gathering large amounts of data have been created to examine particles and their interactions. In hopes of glimpsing what lies beyond the Standard Model of particle physics, the current focus is set on precision measurements of known phenomena and the ongoing hunt for undiscovered, extremely rare processes. To achieve this, even larger amounts of data must be recorded and analyzed pushing the borders in many fields.

With this ever-increasing amount of data, storage and data processing pose tremendous challenges and extensive infrastructures are needed to meet this demand. One of these infrastructures is the Worldwide LHC Computing Grid (WLCG) that consists of hundreds of computing centers. It provides both storage and processing capacities for the experiments studying collisions of hadrons at the Large Hadron Collider (LHC) at CERN.

The amount of events recorded by the experiments at the LHC is planned to increase significantly over the next decade due to the start of the high luminosity (HL) period. Therefore, providing the resources needed to cope with the amount of resulting data will become even more challenging. Increasing storage and processing resources are projected not to be sufficient given the expected funding, even when taking into account passive benefits from technological advances.

There are multiple approaches to meet this challenge. Besides acquiring additional resources, efforts are made to use the existing resources as efficiently as possible. For example, the necessary amount of storage space can be reduced by introducing more compact data formats. To increase the efficiency of processing resources, two approaches stand out: One approach is to make the workload, i.e. the tasks that are processed on the resources, more efficient by improving algorithms and procedures. Another approach is to optimize the management of the workload by allocating the workload's individual tasks, also referred to as *jobs*, to resources that are well suited to process them efficiently. On which resources jobs are processed efficiently is highly dependent on the job's properties and on what the job is actually doing. Therefore, the possibilities to improve the performance differs between jobs.

This thesis focuses on approaches aiming to improve the performance of jobs with high data throughput. In the following such jobs are referred to as *data-intensive*. The efficiency of such jobs suffers if they are provided with less data throughput than they can process as the actual processing must be paused until the next batch of data is transferred. Limited data throughput can occur if data needs to be transferred from other computing centers or if network connections are congested due to competing accesses. To avoid decreasing performance due to limited network throughput, efforts are made to process data on computational resources that have the most performant connection to the storage in which the processed data resides. In the WLCG this is successfully realized by distributing jobs according to the placement of the data the jobs process and by strategically placing data replicas of frequently accessed data throughout the WLCG. In this way, *data locality* — the concept of processing data close to where it is stored — is taken into account.

Achieving data locality for jobs processed on local resources at universities or research institutes is more complicated. First, not all data sets that are studied by a research group are stored locally. Instead, they have to be transferred via the network from remote storage, e.g. from the WLCG. In these cases, a direct association of computational resources providing high data locality for a job like in the design of the WLCG is not possible. Second, the heterogeneity of computing centers is increasing as well. This results in computational and storage resources not necessarily being located in close proximity as opportunistic resources such as shares of High Performance Computing (HPC) centers or Cloud resources are included. An illustration of the resulting infrastructure and its interaction with the WLCG is presented in figure 1.1. Data processed in such infrastructures has to be transferred between local and opportunistic resources as well as remote storage using the respective network connections.

To still improve the performance of data-intensive jobs on local resources, data locality can be increased by adding storage devices that are used as caches to the infrastructure. The caches hold volatile copies of data and provide these to jobs processed on resources connected to the respective cache. By reading data from a cache instead of utilizing the network, the load on the network is reduced. Consequently, the entire data transfer is accelerated and performance limitations can be avoided. Whereas this concept of caching is simple in principle, its effective implementation in distributed systems is a complex task. For example, due to the sheer size of data, not all data sets processed by a research group can be provided simultaneously by the caches. Therefore, it must be decided what data is cached for each resource and time frame. Since cached data needs to be transferred to the cache in the first place, only data that is read at least twice achieves a net benefit in the overall network load.

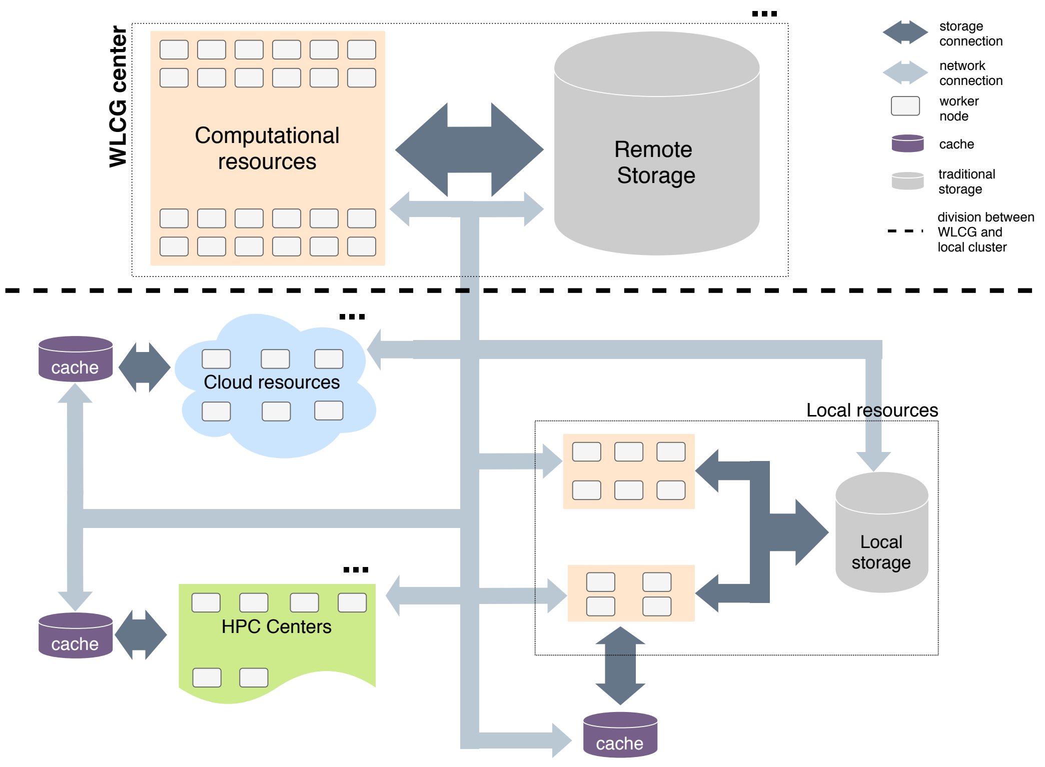


Figure 1.1: Structure of a modern computing center at a university or a research institution and its connection to the WLCG. Besides the common local resources it can consist of opportunistic resources such as cloud resources or shares at HPC centers. Unlike traditional local resources these do not include permanent and managed local storage that is shown in gray. Instead, all data processed on these resources needs to be transferred via network connections depicted as light gray arrows. As the bandwidth of all network connections is limited, bottlenecks can occur reducing the performance of jobs that require high data throughput. To prevent this, caches are distributed among such resources. Like permanent storage, these caches are accessed via a separate connection depicted as dark gray arrows. Since not all data processed on local resources is provided by the local storage, larger data sets need to be transferred from a storage in the WLCG. Therefore, it is beneficial to add caches to the local resources as well. There can be multiple WLCG centers as well as multiple opportunistic resources. All computational resources are made up of worker nodes (gray boxes) containing the actual CPU cores.

Moreover, as the computing resources in a local cluster are part of a distributed infrastructure, not all of them can share the same cache. Instead, there can be multiple caches with differing content distributed throughout the system. Therefore, it is necessary to actively coordinate jobs to the computing resources where cached data is provided to ensure that cached data is taken advantage of. This makes coordination a key point in the research on this approach. The overall concept of distributing caches throughout the system and coordinating jobs with respect to the data provided by these caches is called *distributed coordinated caching*.

In this thesis, distributed coordinated caching is examined. After supplying relevant background information in chapter 2, its benefits and challenges are discussed in chapter 3. To meet the discussed challenges, this thesis presents a two-fold approach. First, as described in chapter 4, the productive system at ETP is monitored to learn about the characteristics of jobs that are likely to benefit from cached data. The jet energy calibration workflow, a group of jobs that is particularly suitable for caching, is presented in detail in chapter 5. Second, to apply the conceptual knowledge about distributed coordinated caching a simulation tool is created as discussed in chapter 6. This simulation tool is then used to study the behavior of computing centers with distributed coordinated caching to improve our understanding of this concept. Additionally, possible mechanisms to realize coordination are developed and their impact is assessed using this simulation tool. Finally, based on the obtained insights, possibilities to further enhance the understanding of distributed coordinated caching are described and paths to realizing distributed coordinated caching are presented in chapter 7.

Chapter 2

Fundamentals

This chapter provides background information on the experiment and computing concepts discussed in this thesis.

2.1 Experimental background

In the scope of the CMS experiment collisions of particle beams accelerated by the LHC at CERN are examined to study a wide range of particle physics processes.

2.1.1 The Large Hadron Collider

The LHC is a ring collider built to accelerate and collide hadrons. Located underground near Geneva, the LHC accelerator consists of two co-located beam pipes that cross at four interaction points. During operation, the beam pipes contain two counter-rotating beams of protons or heavy ions that can be brought to collision at interaction points. With a circumference of 27 km and a center of mass energy of up to 13 TeV it is the largest and most powerful particle collider as of this writing.

Multiple experiments are associated to the LHC. The collaborations of these experiments examine particles emerging from these collisions to learn about the physical processes occurring during hadron collisions. The four major experiments, each based on a detector located at the four interaction points of the LHC, are:

- **A** **T**oroidal **L**HC **A**pparatus (ATLAS) and **C**ompact **M**uon **S**olenoid experiment (CMS). The respective detectors are general-purpose detectors, designed to study all collisions and their outcome and therefore providing measurements for an extensive physics program. While the fundamental principle of particle detection is similar in both detectors the technical implementations differ and allow for cross-checks of results [1, 2].
- **A** **L**arge **I**on **C**ollider **E**xperiment (ALICE) operates a heavy-ion detector focusing on the study of quark-gluon plasma [3].

- **L**arge **H**adron **C**ollider **b**eauty (LHCb) experiment studies charge-parity-violation [4].

Combining the focuses of the four experiments a large variety of physical processes can be observed. The first major public success was the discovery of a Higgs boson in 2012 [5, 6]. Apart from studying the properties of the Higgs boson, precision measurements of the Standard Model of particle physics are conducted and undiscovered phenomena beyond the scope of the Standard Model are probed.

Updates of the collider and the detectors are planned to further increase the experiments' performance regarding precision and observation of rare phenomena [7]. The following HL-LHC phase is scheduled to start in 2027 and the amount of registered data sets will result in significant challenges for the experiments computing infrastructures and analysis capacities [8].

2.1.2 The CMS detector

As a multipurpose detector, the CMS detector was constructed to measure as many particle types as possible with a large phase space coverage. To achieve this, the detector consists of multiple subdetectors, arranged cylindrically around the interaction point.

The name *Compact Muon Solenoid* is derived from its three main properties: With a large mass of 12.5 kt considering its external dimensions of 21.6 m in length and 14.6 m in diameter, the detector is *compact* compared to the other detectors at LHC. Besides, the detector specializes in the detection and precise measurement of *muons*. Furthermore, a strong magnetic field needs to be present in the detector's interior to bend the particles' trajectory and thereby draw conclusions about the particles' charge and transverse momentum. Therefore, a magnetic field of 3.8 T is created by a superconducting *solenoid* coil.

The overall structure of the detector is shown in figure 2.1. It consists of a central, barrel-shaped region that is closed with endcaps. Both the barrel and the endcaps consist of layers of subdetectors. Beginning with the subdetector closest to the beam line, the CMS detector contains:

- a silicon *tracker* that allows reconstructing the trajectories of charged particles passing through. Because of the magnetic field the trajectories are bent and their direction and diameter allow to determine the particles charge and transverse momentum.

- an *electromagnetic calorimeter* (ECAL) that measures the energy of electromagnetically interacting particles. Photons and electrons are stopped by the calorimeter's material and deposit their energy in this part of the detector.

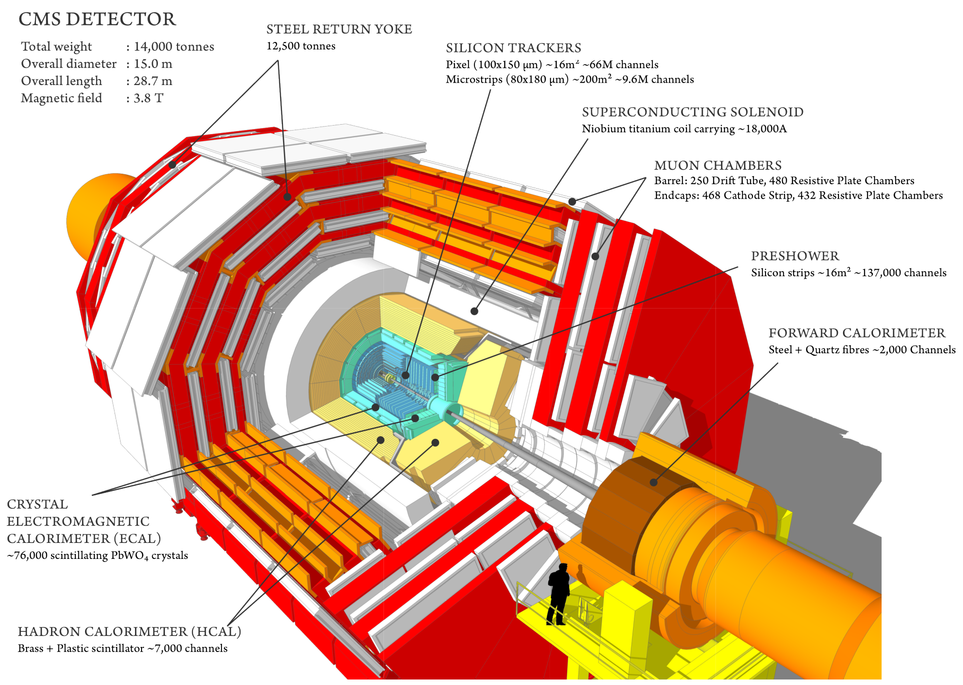

CMS DETECTOR
Total weight : 14,000 tonnes
Overall diameter : 15.0 m
Overall length : 28.7 m
Magnetic field : 3.8 T
STEEL RETURN YOKE
12,500 tonnes
SILICON TRACKERS
Pixel (100x150 μm) ~16m² ~66M channels
Microstrips (80x180 μm) ~200m² ~9.6M channels
SUPERCONDUCTING SOLENOID
Niobium titanium coil carrying ~18,000A
MUON CHAMBERS
Barrel: 250 Drift Tube, 480 Resistive Plate Chambers
Endcaps: 468 Cathode Strip, 432 Resistive Plate Chambers
PRESHOWER
Silicon strips ~16m² ~137,000 channels
FORWARD CALORIMETER
Steel + Quartz fibres ~2,000 Channels
CRYSTAL ELECTROMAGNETIC CALORIMETER (ECAL)
~76,000 scintillating PbWO₄ crystals
HADRON CALORIMETER (HCAL)
Brass + Plastic scintillator ~7,000 channels

Figure 2.1: Schematic overview of the CMS detector and its components [9].

- a *hadronic calorimeter* (HCAL) that absorbs and measures the energy of hadronic particles that passed the ECAL.
- a *muon system* to detect muons. All other particles that can be detected in this experiment are absorbed by the calorimeters or the solenoid coil located between the HCAL and the muon system. However, the muons produced in LHC collisions are typically minimally ionizing particles. Therefore, they only deposit a small amount of energy in the inner subdetectors and can be detected in the muon chambers. This subdetector is interwoven with the solenoid's return yoke and acts as a second tracker, thus enabling a more precise measurement of the muons' transverse momentum.

In the forward direction, there are additional calorimeters close to the beam pipe minimizing the area where particles can escape undetected. The number of collisions produced at the LHC exceeds the technical limit of recordable interactions per second. Besides, not every collision contains physical processes that are interesting to the experiments. Therefore, a system of triggers is used to select collisions that are likely to contain interesting processes and thereby reduce the amount of recorded data. An interaction that has passed the trigger's requirements and is recorded is called an *event*.

2.2 Jets

Hadrons consist of point-like constituents carrying color charge, quarks and gluons, that are also referred to as partons. So when hadrons collide, it is the partons that interact and are scattered. Due to the nature of the strong force, such color-charged particles can not reach the detector in unbound states and form colorless bound states. If partons in a bound state are separated from each other the force between them increases until there is enough energy to create a new quark-antiquark pair from the vacuum. Each of them binds to an original parton preserving color charge. This process can happen repeatedly. In this way, every parton produced in the original collision causes a cone-shaped shower of particles called a *jet*. The detector only detects the final constituents of the jet, not the initial partons. Since not all produced hadrons are stable, the decay of unstable hadrons produces more hadrons and leptons. Thus jets can consist of a large variety of particles.

At a hadron collider such as the LHC, nearly every event contains at least one jet. Therefore, an accurate understanding of jets is relevant to perform precise measurements and search for deviations from expected event topologies. To achieve this, jets need to be carefully reconstructed and calibrated. Jet reconstruction includes the correct clustering of all particles that stem from one parton into a jet. An overview of jet reconstruction techniques is given in [10]. Jet calibration links the reconstructed jet's energy to the energy of the original parton taking the response of the detector and used algorithms into account.

2.3 Computing for LHC experiments

With 36 PB of data recorded during Run 2 of the LHC between 2015 and 2018 by the CMS experiment alone [11], the amount of data taken by the LHC experiments is large compared to other experiments. Combined with data sets from simulations that are needed for analysis purposes, they pose a considerable challenge to any computational infrastructure. Consequently, large amounts of storage and computing resources are needed to store and process this data. Furthermore, data and resources need to be provided to a large community of researchers all over the world. As an infrastructure that can meet these challenges, the WLCG [12] was created.

2.3.1 The WLCG

The WLCG consists of various computing resources provided by the LHC experiment's member states and coordinated by the experiment's collaborations. Its purpose is to store data in various stages of processing, to make it accessible to researchers all over the world and to supply computing resources for data process-

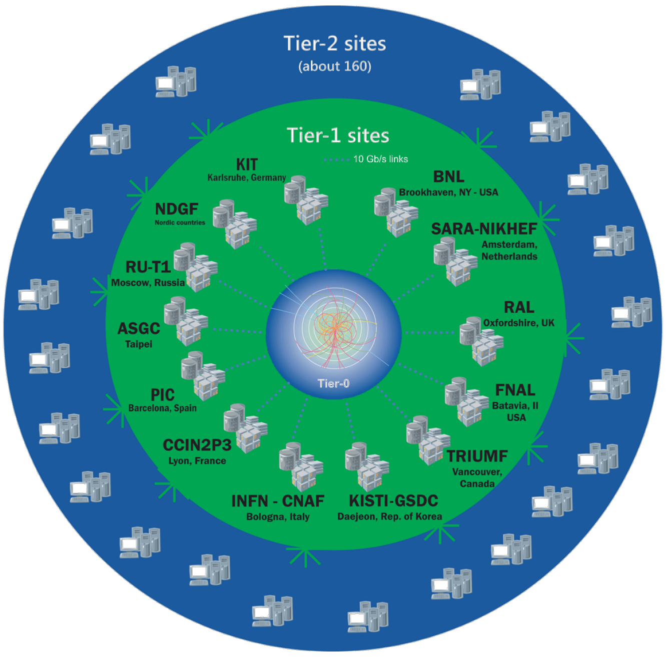


Figure 2.2: The WLCG Tier structure [13]. Computing centers of different tiers focus on different tasks like storing data taken by the LHC experiments in various stages of processing and providing computing resources for data processing and simulation.

ing. It consists of different hierarchically structured tiers, as shown in figure 2.2. Originally, each tier is specialized for a set of services [14]:

- *Tier 0* centers store the raw data of the experiments. Raw data includes all information about the detector's response. At Tier 0 centers, first reconstructions of physical objects are performed and the raw and reconstructed data are both distributed to Tier 1 centers. The sole Tier 0 computing center of the WLCG is located at CERN.

- The responsibility of *Tier 1* centers is to provide large amounts of storage for their share of raw and reconstructed data. Furthermore, they have to provide resources for data reprocessing and to store the resulting data sets. Similar to Tier 0 centers, the Tier 1 centers also coordinate the centers in the tiers beneath them by distributing data to Tier 2 centers. There are thirteen Tier 1 centers all over the world.

- *Tier 2* centers provide storage and processing resources for specific analyses. For these analyses, both data from simulations and recorded data is analyzed.

Tier 2 centers are usually associated to universities or scientific institutes and comprise a total of 160 Tier 2 centers around the world.

Today, this strict hierarchic structure is softened in favor of direct communication between all centers and more flexible distribution of tasks [15].

2.3.2 Local resources

In addition to the resources provided by the WLCG, analysis groups have local resources at their disposal. These resources are commonly referred to as *Tier 3* resources and are not formally bound to the WLCG. Local resources are usually used to process a research group's end-user analyses. Such analyses usually differ from the large-scale reprocessing tasks or simulations executed on the WLCG. For example, they are more likely to be executed frequently and tend to have shorter runtimes. As Tier 3 resources are usually used by a certain research group, they are adapted to specific needs and requirements. Changing requirements and funding usually results in Tier 3 centers being made up of different resource types. While file servers provide storage and log-in nodes are used to create, test and manage workflows, worker nodes process the analyses' actual computational workload. Users run their workflows on the worker nodes by submitting jobs to a *batch system*. This allows for the automatized and parallel execution of jobs, optimizing the worker nodes' utilization in the process.

2.3.3 Opportunistic resources

Tier 3 resources may also include so-called opportunistic resources. In HEP, the term opportunistic implies that these resources are not solely used for the workload of a certain scientific community and that they might only be temporarily available. Such resources are e.g. multidisciplinary computing clusters at universities such as the NEMO HPC cluster [16], HPC centers such as the ForHLR II at KIT [17] or Cloud resources [18]. By using technologies such as virtualization or container technologies such resources can be flexibly used for HEP purposes. With tools such as COBalD [19] and TARDIS [20] developed at KIT such resources can be easily handled and seamlessly integrated into a local batch system. In contrast to traditional local resources, opportunistic resources rarely provide permanent, managed storage. Data that is processed on these resources has to be transferred there using the network.

Concluding, the usage of opportunistic resources contributes to meet the ever-growing need for more computing resources in HEP. However, the integration of opportunistic resources results in a complex interplay of computing and storage resources.

2.4 End-user analyses

A typical end-user analysis such as the jet energy calibration workflow which is specifically considered in this thesis, consists of different stages with distinct tasks: Skimming, Analysis and Plotting [21]. These stages, also referred to as workflows, are illustrated in figure 2.3. Every stage processes an amount of input data. The output data of a stage is the input data of the following stage reducing the amount of irrelevant information with every stage. This reduction in the size of the output data sets and thus the shortening of the processing time is important as later stages have to be processed more frequently.

In the first stage, the *Skimming* stage, the input data consists of data sets recorded by the detector or generated by simulation. These data sets contain information about each event and the physical objects registered in the detector. When skimming the data sets, events that are unlikely to contain physical processes relevant to the current study are removed. One data set can be subject to different Skimming workflows depending on the current study resulting in multiple skimmed data sets. To ensure that the output file is several times smaller than the input file, only relevant information for events that are not rejected during Skimming are written to the output file. Skimming is executed about quarterly, whereby the runtime of this workflow is usually in the order of days.

The second stage is called *Analysis*. The output data set of one Skimming workflow can be the input to multiple Analysis stages dealing with similar physical processes as research groups often perform multiple analyses based on the same physical objects or contexts. During the Analysis stage, the physical objects' recorded properties are transformed into the relevant quantities or observables of the current study. Examples for such observables are the mass of a physical object in the final state or a physical object reconstructed from its decay products and its properties. A set of selection criteria referred to as *cuts* is applied to the extracted observables to exclude events that are unlikely to contain physical processes relevant to the current study. The resulting output data set therefore only contains information about these observables for relevant events. The Analysis workflow is executed on a weekly to monthly basis and usually takes several hours. Up to this stage, information from different events can be processed independently, and workflows can therefore be trivially parallelized by splitting the data set and processing the parts separately.

In the third and last stage of a typical end-user analysis the *Plotting* takes place. During Plotting, information can be combined across different events to extract and study the underlying distributions of observables, as well as across multiple data sets to compare simulated to recorded events. This stage lasts several minutes and is executed multiple times on an hourly basis during a workday. The resulting output files can be of visual or numerical nature.

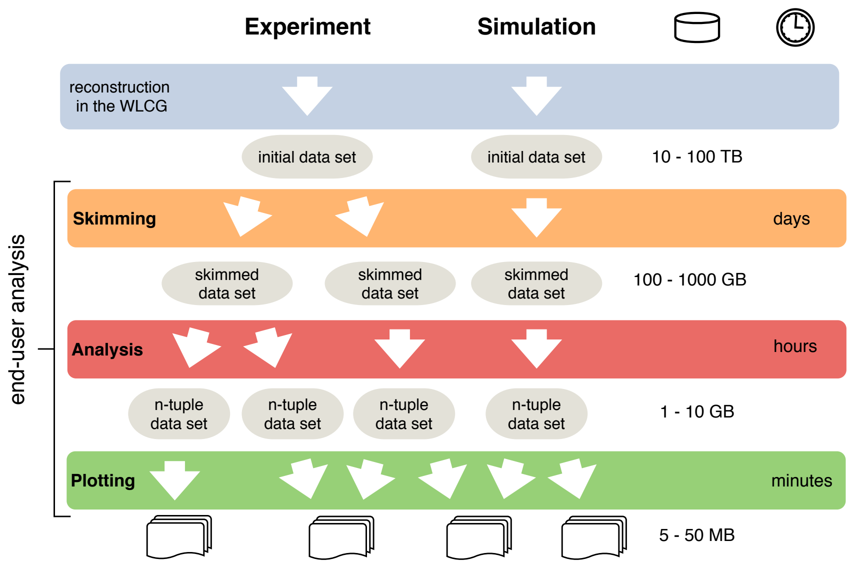


Figure 2.3: Stages that make up a typical end-user analysis in the HEP context. Typical value ranges for the jet energy calibration workflow are presented to visualize the reduction of data and processing time in each stage. Data recorded by experiments or generated by simulations is reprocessed before serving as input to the actual end-user analysis. For LHC experiments this is done in the WLCG. The size of an initial data set is usually in the range of 10 – 100 TB. End-user analyses usually consist of the three stages Skimming, Analysis and Plotting. The Skimming stage usually takes a few days to complete and performs first loose selections on the data. As the applied selection criteria can vary, multiple data sets can be skimmed from an initial data set. In the Analysis stage, additional selections are applied and the observables with relevance to the examined physical context are calculated. They are usually stored in dedicated data formats such as n-tuples and the output size is reduced from several hundred GB to several GB while the processing time for this stage is in the order of hours. As before, a skimmed data set can be processed through different Analysis stages resulting in data sets with different physical focus.

For processing, workflows are split into jobs that are independent of one another. Individual jobs therefore can be processed in parallel accelerating the total processing time of the workflow. How a workflow is split into jobs depends on the workflow's properties depending on whether it can be considered task or data parallel.

Chapter 3

Coordinated caching in distributed systems

The ever-growing amount of data recorded by experiments in HEP results in challenges for storage capacities as well as computational resources that are needed for reconstruction, simulation and analysis of data. Estimates of the CMS and ATLAS collaborations presented in figures 3.1 and 3.2 indicate that the resources available for data storage and processing will not suffice to meet the demand in the near future. Therefore, it is necessary to improve the use of available resources and acquire additional resources. The amount of additional resources that can be acquired depends on available funding and technological advance. Consequently, the efficient usage of existing resources is an important field of research.

There are multiple approaches to improve the usage of available resources. For example, existing software can be optimized and the use of concurrent programming methods can be encouraged [8]. Another approach is to optimize the workflows' processing conditions when allocating jobs to worker nodes, for example the available data throughput. In this case, workflows that process data are of particular interest. As discussed before, the performance of such workflows can be limited due to network congestion if data is not stored close to where it is processed but has to be transferred from another storage. This occurs since storage space at local resources is usually not sufficient to contain a copy of all data with relevance to the research group. Therefore, input data of many end-user analysis jobs needs to be transferred from dedicated storage located in the WLCG to where the jobs are executed. In addition, modern computing centers available to the HEP community include opportunistic resources. As discussed in section 2.3.3 such resources rarely provide suitable storage and jobs processed there rely on reading data via network connections.

The resulting load on network connections by large amounts of competing data transfers results in bottlenecks in the available network bandwidth thereby limiting the efficiency of jobs that transfer data [22]. An approach to avoiding such limitations is to increase *data locality* by bringing data storage and processing as close together as possible [23, 24]. This chapter introduces coordinated caching in distributed systems

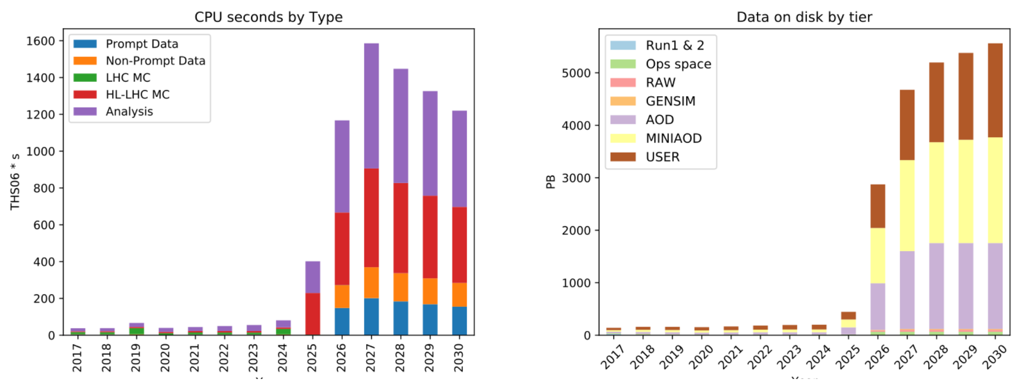


Figure 3.1: Estimate of the CPU and storage resources needed by the CMS experiment until 2030 [8]. Contributions of different computational tasks or data types are shown in different colors. Both figures show a significant increase in demand following the beginning of the HL-LHC period that was supposed to start in 2025 when this prediction was created. The amount of CPU resources is quantified as Tera HEP-SPEC06 CPU seconds with HEP-SPEC06 being a CPU benchmark [25] used in HEP to compare the performance of CPUs.

or, shortened, *distributed coordinated caching* as an approach to increase data locality for Tier 3 resources in the HEP community.

3.1 Data locality in HEP resources

Increasing data locality can be achieved by executing tasks as close as possible to the storage of their input data. In this context, a worker node executing a task is considered to be closer to a storage than another if data can be transferred more efficiently between them. This can for example be the case if the connection between storage and worker node is not as occupied or if it provides a higher maximal bandwidth. Optimal closeness is achieved if the task is executed on the same machine on which the data is stored. In this case, data can be transferred via the machine's internal connections instead of network connections.

The concept of data locality can be applied to different computing infrastructures, ranging from grid to local resources. By using computing resources according to data placement, connections prone to congestion are relieved and, therefore, freed for other transfers. This can improve the overall performance of the system. In the WLCG, data locality is already taken into account when distributing jobs to grid sites. To consider data locality at Tier 3 centers as well, approaches and prototypes to achieve this have been developed at KIT for several years.

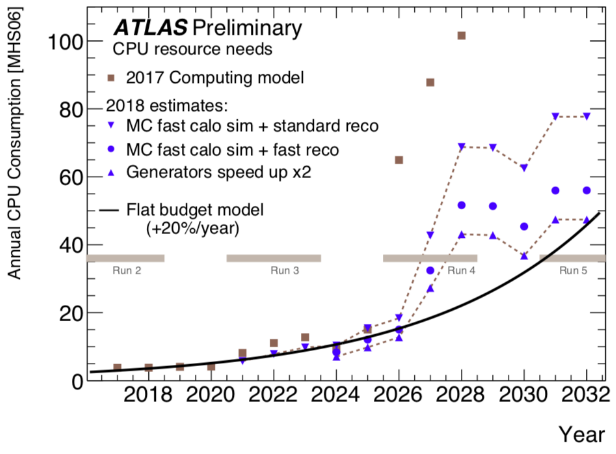
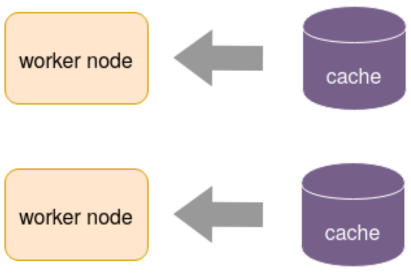


Figure 3.2: Annual amount of CPU resources required by the ATLAS experiment [26] in terms of the HEP-SPEC06 CPU benchmark [25]. The black curve corresponds to the amount of resources available assuming constant funding and a performance increase of 20 % per year. The brown points indicate the amount of required resources based on a computing model from 2017. Blue markers show the estimated demand for three scenarios that include additional development efforts. For the top curve this would include using a faster method for 75 % of Monte Carlo simulations, for the curve in the middle, event reconstruction would need to be accelerated additionally and for the bottom curve the time needed to simulate an event would have to be halved. These developments reduce the discrepancy between needed and available resources after the beginning of the HL-LHC era. Nevertheless, this illustrates that besides acquiring additional resources and software improvements, the existing resources have to be utilized as efficiently as possible to cope with the experiments computational demand.

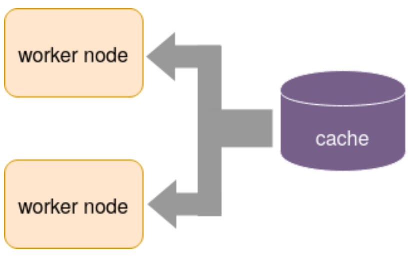
3.2 Caching for end-user data analyses

There are two ways to bring data storage and processing closer together increasing data locality. Data processing can be moved to where the data is stored or data can be moved to where processing is done.

The first option, processing end-user analysis workflows directly where the corresponding input data is stored, is rarely possible given the specific storage and processing resource setup in the WLCG. Since specific resource types are used for data storage and processing, storage nodes do not provide the processing power required for an efficient execution of a job while processing nodes do not provide the sufficient amount of performant storage. Opportunistic resources rarely provide permanent and managed storage at all and parts of the requested data might be stored



(a) Dedicated cache setup



(b) Shared cache setup

Figure 3.3: Example for a dedicated and shared cache setup on two worker nodes. In a dedicated cache setup, both worker nodes have access to separate caches while the worker nodes share a single cache in the shared cache setup.

elsewhere, e.g. in the WLCG. Such storage outside the Tier 3 centers is from now on referred to as *remote storage*.

The second option is to achieve data locality by integrating additional storage into the resources serving as *caches*. As modern Tier 3 centers are distributed infrastructures consisting of different computing resources not necessarily located in close proximity, the resulting cache infrastructure is distributed as well. This means that there are multiple caches with limited cache volume each. Each cache can only be accessed by a varying subset of the Tier 3 center's worker nodes and holds a volatile copy of a part of the data that is currently processed or will be processed in the near future. As only a fraction of the data with relevance to a research group is processed at one point in time, it can be avoided to provide large and expensive cache volumes that would be needed to store all relevant data.

Since usually multiple caches are integrated into the infrastructure of a Tier 3 center, different possibilities to place caches arise. On the one hand, caches can be placed directly on worker nodes, using the worker nodes internal connections for data transfers. In the following, this setup will be referred to as *dedicated cache setup*. On the other hand, a single cache can be connected to multiple worker nodes. This architecture will be called *shared cache setup*. The shared cache setup can be realized by combining caches placed on worker nodes into a distributed file system or by connecting an external cache to multiple worker nodes. An example illustrating a dedicated and shared cache setup for two worker nodes is shown in figure 3.3. For each setup, the load on the network is reduced if cached data is read from the cache multiple times. While the initial transfer of data to the cache uses the network connection, subsequent accesses to the same data utilize the machine's internal connection and, therefore, reduce the load on the network.

With dedicated or shared cache setup in place, jobs can only read data from caches if the caches are connected to the worker node a job is processed on. Therefore, a coordination mechanism for jobs is needed to ensure that cached data is used and that data is reasonably distributed among caches thereby utilizing the full potential of data locality. The approach combining distributed caches and job and data coordination is called *distributed coordinated caching*.

3.3 Benefits of distributed coordinated caching

Deploying shared and dedicated caches in distributed systems and applying job and data coordination as outlined in section 3.2, distributed coordinated caching allows extending the well-known benefits of caching on an individual machine to the distributed system's of Tier 3 centers and end-user analyses: First, the runtime of jobs with high data throughput, that were previously limited by network bottlenecks, can be reduced when data is read from a distributed cache instead of a remote storage. Second, jobs that do not use the cache themselves still benefit from decreased load on the network. Additionally, if jobs are processed faster for one of the two mentioned reasons, the respective computing resources become available earlier and new jobs can be processed improving the systems overall job throughput. If input data of a whole workflow is made available via caching, the workflow's total processing time can be shortened. This is beneficial to users as well as the time spent waiting for results is reduced.

In general, both job and data throughput of a computing center could be improved leading to more efficient utilization of the available resources. Especially sites with limited network bandwidth to remote storage could significantly increase performance. Also, the costs for resources with paid network connections such as Cloud resources can be reduced by using caches.

3.3.1 Expected job runtime acceleration by caching

The performance increase achieved by reading data from a cache is reflected in the acceleration of the runtime of jobs. An individual job's runtime, also called walltime, is either limited by the time needed to transfer data requested by the job or the time needed to perform calculations on these. Assuming that already transferred data can be processed while new data is fetched a job's walltime can be estimated using

$$t_{\text{walltime}} = \max\left(t_{\text{calculation}}, t_{\text{data transfer}}\right). \tag{3.1}$$

This assumption is reasonable for HEP end-user analyses. As the input data of such analyses usually consists of individual events that are processed independently, events can be processed while other events are transferred concurrently. This is enabled in frameworks used for data accesses in HEP [27]. The resulting procedure is illustrated in figure 3.4. The graphic depicts a job processing n events. Because the transfer of

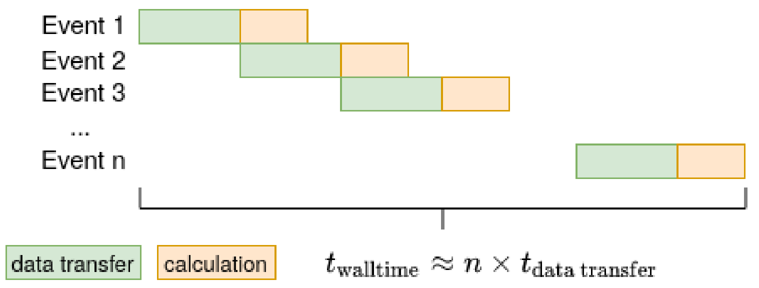

Event 1
Event 2
Event 3
...
Event n
data transfer
calculation
$t_{\text{walltime}} \approx n \times t_{\text{data transfer}}$

Figure 3.4: Visualization of the walltime of a job with input files if data can be transferred while already fetched data is processed. The depicted job's input consists of n events. As the transfer duration $t_{\text{data transfer}}$ for an event, depicted in green, is longer than the event's processing duration in yellow, the total walltime can be approximated by $n \cdot t_{\text{data transfer}}$.

each event takes longer than the actual processing, the job's walltime is dominated by data transfer.

Job processing time model

Assuming that a job's input data with volume V is transferred from an external storage that provides a bandwidth of s_{remote} to this transfer, the data transfer's duration is given by

$$t_{\text{data transfer}} = \frac{V}{s_{\text{remote}}}. \tag{3.2}$$

A job's calculation time $t_{\text{calculation}}$ can be estimated from its CPU time t_{CPU}. The CPU time corresponds to the amount of time the CPU was fully utilized during the execution of a program or a job. The CPU is not fully utilized when the job is waiting for data to be transferred or if the job's computation is not programmed efficiently. To take the latter into account, a job's calculation time is given by

$$t_{\text{calculation}} = \frac{t_{\text{CPU}}}{\epsilon_{\text{calculation}}} \tag{3.3}$$

with $\epsilon_{\text{calculation}}$ being the job's maximal achievable efficiency. Given this model, jobs are considered limited by data transfer if $t_{\text{calculation}} < t_{\text{data transfer}}$ in the context of caching. If a fraction h of the job's input data is replicated on a cache associated to the worker node the job is processed on, the job's walltime depends on h. If data can be transferred in parallel reading from the cache and via the network, equation (3.2) becomes

$$t_{\text{data transfer}}(h) = \max\left(\frac{V \cdot (1-h)}{s_{\text{remote}}}, \frac{V \cdot h}{s_{\text{cache}}}\right) \tag{3.4}$$

and equation (3.1) becomes

$$\begin{aligned} t_{\text{walltime}}(h) &= \max\left(t_{\text{calculation}},\, t^{\text{remote}}_{\text{data transfer}}(h),\, t^{\text{cache}}_{\text{data transfer}}(h)\right) \\ &= \max\left(\frac{t_{\text{CPU}}}{\epsilon_{\text{calculation}}},\, \frac{V \cdot (1-h)}{s_{\text{remote}}},\, \frac{V \cdot h}{s_{\text{cache}}}\right) \end{aligned} \tag{3.5}$$

with $t^{\text{remote}}_{\text{data transfer}}$ and $t^{\text{cache}}_{\text{data transfer}}$ being the duration of the data transfer from a remote storage and the cache respectively and s_{cache} representing the bandwidth available for this transfer from the cache. In real systems s_{remote} and s_{cache} vary over time as the available bandwidth varies if the respective connection is used for other transfers simultaneously.

Equation (3.5) is visualized in figure 3.5 for an exemplary job. The walltime's dependency on h can be split into three areas, each dominated by a different term of the equation. In the first area denoted by 1 the walltime is limited by the term corresponding to data transfer via the remote connection. With increasing amount of cached data h, data is read from the cache as opposed to from remote storage resulting in a linear decrease of the walltime until the walltime is equal to the calculation time. In the following area denoted by 2 the walltime is equal to the job's calculation time. With further increasing h the third area denoted by 3 is reached. In this area, the data transfer takes longer than the calculations and the walltime increases linearly with increasing h.

In general, a job benefits from caching if $t_{\text{walltime}}(h) < t_{\text{walltime}}(0)$ for $h > 0$. Whether this holds, depends both on the job and the network and cache connection's current utilization. Additionally, caching all data does not always minimize the walltime as $t_{\text{walltime}}(h) < t_{\text{walltime}}(1)$ for $h < 1$ holds in some setups.

3.4 Challenges of coordinated caching in distributed systems

The realization of distributed coordinated caching poses several challenges that can be broken down into three aspects: 1. data selection, eviction and placement by defining the cache's behavior and policies; 2. efforts to optimally profit from data locality by coordinating jobs and data; and 3. the choice of cache architecture.

3.4.1 Data selection, eviction and placement

The first challenge is selecting what data is cached and on which cache it is placed if multiple caches are available. This is a generalization of classical caching problems, in that the placement of data is relevant in a distributed architecture. In most applications for distributed coordinated caching in HEP, it is not reasonable to use

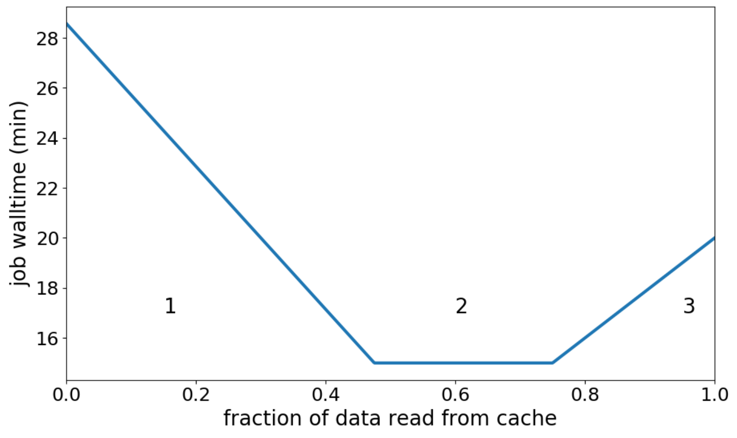


Figure 3.5: Dependency of an exemplary job's walltime on the fraction h of input data provided by a cache as stated in equation (3.5). For $h = 0$ the walltime is dominated by the time necessary to read all input data via the network connection. By increasing h the transfer duration decreases when data is transferred from the cache as well. Once the transfer duration is shorter than the job's calculation time, the walltime cannot decrease any further. As h increases, the time needed to read data from the cache increases as well until it surpasses the calculation time. From there on, the walltime increases linearly with h. This exemplary job would profit from any amount of cached data as the walltime is maximal for $h = 0$. The example also shows that not all data needs to be cached to minimize the walltime.

caches that are big enough to simultaneously hold all the data a research group works on. As shown in figure 3.5, caching all data does not necessarily result in optimal job efficiency. Additionally, at each point in time only a fraction of the total data is processed. Finally, the maximum cache volume that can be integrated into the system is limited by funding, pre-existing hardware and architecture, usage models and other factors.

Therefore, a cache's content has to be selected and updated suitably:

1. Caches should contain data that is accessed more than once to reduce the net load on the network. Reading data several times is not uncommon in HEP: sets of input data for distinct jobs or workflows can overlap, workflows are reprocessed due to user errors or executed repeatedly. As overlaps and user errors that are based on user behavior in short time ranges are hard to predict [28], efforts presented in this thesis focus on repeating workflows.

2. Data handling must respect that it serves multiple, competing workflows. Unnecessary duplication of data in the caches should be avoided. This ensures that

space for storing other data can be provided. Besides, if multiple workflows compete for cache space, input data for workflows with larger performance gain should be preferred.

3. Cache space needs to be freed reasonably taking into account whether data is likely to be reused. Especially, data that is unlikely to be reused should be removed first. Additionally, data that is more likely to be used should not be removed in favor of caching new data that is less likely to be used.

4. Optimal placement of data is an additional challenge that gains in significance when dealing with caching in distributed systems. While the throughput of the network connection to remote storage is limited, the connection to the cache is limited as well. Data placement should take into account that too many jobs reading from the same cache can create bottlenecks in cache bandwidth and thereby reduce a job's performance.

All these considerations for data selection, eviction and placement need to be kept in mind which makes defining cache behavior a challenging task.

3.4.2 Job and data coordination

As discussed in section 3.2, coordination mechanisms are necessary to utilize the full potential of increased data locality, and to enable distributed caching comparable to classical caching in the first place. Both jobs and data can be coordinated with respect to each other, yet each has different mobility. In computing infrastructures used in HEP, jobs are assigned to resources by a job scheduler, e.g. in a batch system. Therefore, direct job coordination can be realized by considering data locality in the batch systems job matching when matching jobs to available resources for execution. Potential coordination mechanisms should aim to ensure appropriate utilization of the cache, in particular maximizing the amount of jobs reading from the cache while avoiding bottlenecks in the cache bandwidth. Other aspects of the batch systems' behavior such as matching jobs without input file information should not be interfered with. The coordination mechanism should impair the batch system as little as possible, the time a job spends between submission into the batch system and execution should stay reasonable and the amount of idle resources should stay as small as possible.

In general, data can be coordinated directly to a specific cache or indirectly when a job reading this data is processed on a worker node with access to a cache that automatically places a copy of the data in the cache. Consequently, data can be automatically coordinated indirectly when a batch system is used to coordinate jobs. As data will be cached close to a worker node if a job reading the data was executed on this worker node, the placement of data can be influenced by job coordination.

Besides being complex itself, considerations regarding the coordination of jobs or data are closely interwoven with the caches' behavior. For example, the creation of unwanted file duplicates depends on the coordination mechanisms as well as on the behavior of the caches.

3.4.3 Impact of the cache architecture

There are multiple ways to set up distributed coordinated caches, the major approaches are shared cache and dedicated cache setups as outlined in section 3.2.

The chosen cache setup directly affects the number of caches, the total cache volume and the caches' placement required for optimal cache performance. The same applies to the precise technical realization: especially the caches' reading and writing speed and the cache connections' bandwidth further influence the system's overall effectiveness. In general, scalability and fault tolerance make it desirable for caches to operate in a stand-alone way. This allows for the transparent integration and removal of computing resources that contain caches. This is especially important to support dynamic systems integrating opportunistic resources.

Based on the choice of architecture, the challenges regarding coordination and cache behavior vary. For example, in a dedicated cache setup a job has to be coordinated precisely to the only worker node with access to the cache. In a shared cache setup the job can be coordinated to each worker node in a group of worker nodes. However, input data of jobs processed on multiple machines competing for the same cache can result in insufficient bandwidth of the connection to the cache degrading the performance.

3.5 Existing approaches

There are several approaches aiming at increasing data locality on different tiers of the WLCG both in the community and at KIT. Their concepts and implications for the work presented in this thesis will be discussed in this section.

3.5.1 Approaches in the community

As mentioned in section 3.1, data locality is already considered when processing workflows directly on Tier 0, Tier 1 or Tier 2 centers in the WLCG. These computing centers contain both storage elements (SEs) and Computing Elements (CEs) and data locality is taken into account by executing jobs via a CE of a computing center where the requested data is present on the SE. The information about a file's location is provided via a central data management system. For the ATLAS collaboration the central data management system is Rucio [29], in the CMS collaboration PhEDEx [30] is used. Using the job scheduling of the CMS Workflow Management

Systems [14] or the ATLAS collaborations' workflow management system PanDA [31], jobs are directly scheduled to sites where their input data is provided.

The load over the available computing resources can be evened out by creating replicas of data that is read often on additional storage. To achieve this, the dynamic data placement agent *C3PO* [32] was developed by the ATLAS collaboration. It makes use of free storage by creating replicas of data that is expected to be requested often in the near future. The prediction by C3PO is based on information about the input files of recently submitted jobs, already existing replicas, the load on network connections at different sites, and previous accesses of original files and replicas. A placement algorithm controls when and where replicas are created or deleted. The actual replication is handled by Rucio. According to tests from 2017, only 64 % of the replicas were used as the algorithm does not take into account whether the replicas are created at computing centers with available resources. The dynamic data management system Dynamo developed by the CMS collaboration serves a similar purpose [33].

Since version 4.0, the XRootD framework [27], a scalable framework for data access developed for usage in the HEP context, provides a plugin supporting basic caching functionalities. The plugin is focused on a method called *transparent caching*. If files are transferred from remote storage to a worker node with access to a cache, they can directly be copied to the cache while being transferred. The decision to cache files transparently is configurable allowing to add custom caching policies. In default mode all files are cached. If files are already present in a cache connected to a worker node, read requests from jobs running on this worker node are automatically redirected to read from the cache. Otherwise, data is read from the remote storage. If cache space is insufficient to store new files, the files that were used least recently are evicted. The decision which files to evict is not configurable as of now but is planned to become configurable in the future [34]. Based on these features, XRootD is suited as a basis for caching setups in the HEP environment.

The caching functionality of XRootD is used to operate caches in the WLCG. There are different approaches in the CMS and ATLAS community. For example, in an approach followed at LMU Munich a job's file requests can be served by the cache if it is specified that the files can read from the cache [35]. Another approach, by the CMS community, is focused on supplying data of a specific data type often used in analyses. Official CMS jobs are then explicitly steered to the Tier 2 site where the cache is located after not being scheduled for a specific time [36].

The approaches outlined in this section are created for usage in the WLCG where coordination is taken care of by the experiments workflow management systems. Directly transferring these concepts to Tier 3 centers is not suitable, since the organization of computing resources in the WLCG and Tier 3 centers differs in available storage and distance between computational and storage resources. Unlike in the WLCG, not all data is stored in storage associated to the computing resources of a

Tier 3 center. Integrating information about end-user analysis jobs and respective data locality into a WLCG workflow management system or a similar system for Tier 3 centers would prove difficult for several reasons. First, data sets processed in the different stages of end-user analyses are usually smaller, contain more files, and change more frequently than those processed in the WLCG. This might impair the performance of the workflow management systems and underlying databases. Second, supplying central management instances to heterogeneous Tier 3 infrastructures with many sites creates overhead that does not affect big, less distributed computing centers in the WLCG. Therefore, custom solutions are required to achieve distributed coordinated caching for end-user analyses in Tier 3 centers.

3.5.2 Approaches at KIT

While the other approaches in the community are focused on implementations for the WLCG, approaches to realizing distributed coordinated caching at Tier 3 centers have been studied at KIT for several years. During this time, two prototype implementations were created: the High Throughput Data Analysis middleware *HTDA* [21] and the coordination service *NaviX* [23]. For both prototypes, the behavior of the batch system HTCondor [37] is extended in two main aspects to handle jobs with specified input files.

On the one hand, the batch system's matching of jobs to resources is modified based on already cached input files. The information whether a file is cached is gathered directly from the caches instead of a central management component to avoid outdated information about the caches. On the other hand, the occurrence of jobs with input files triggers a caching decision aiming to identify and replicate files that are worthwhile caching. Both approaches aim to seamlessly integrate into the existing system without modifying or interfering with the handling of other jobs. While HTDA is limited to caching locally stored files, NaviX uses XRootD and therefore is able to cache files that are stored in the WLCG as well. The management of caches and their functionality is manually designed and implemented in HTDA. For NaviX, the cache management is based on the XRootD caching plugin discussed in section 3.5.1 for NaviX. As of now, the plugin's default configuration is used as cache behavior.

Both prototypes were successfully tested on suitable workloads in the production system at ETP. Nevertheless, both prototypes are not yet suited for long-term usage in the production system due to implementation details and, in the case of HTDA, limited range of accessible files. Additionally, in both cases, only one coordination implementation and cache behavior was tested.

3.6 Goal of this thesis

As a prerequisite to resuming the work on a prototype that enables distributed coordinated caching and is suited for long-term usage in a Tier 3 center, the challenges and requirements described in section 3.4 have to be met. As of now, only simple implementations of caching policies, coordination mechanics, and cache architectures have been tested at KIT. More complex implementations and their interrelation have yet to be systematically examined and understood. The goal of this thesis is to advance the knowledge on the realization of distributed coordinated caching in a two-fold approach.

The impact of different configurations of distributed coordinated caching depends on the workload of the examined system. Therefore, as a first step towards a more complete understanding of distributed coordinated caching, a representative system and its workload are monitored and studied regarding the suitability for caching. In the scope of this thesis the production system at ETP was examined.

The degree of complexity in systems featuring distributed coordinated caching makes it very hard to find optimized operation modes by theoretical considerations or prototyping. Instead, a flexible setup is needed to study the influence of different setups in a reproducible environment. This can not be achieved with reasonable efforts if the setup of a production or test system is modified. Instead, a simulation tool allowing to study the relevant aspects of distributed coordinated caching and their influence on the overall system was developed in the scope of this thesis.

As the different aspects of distributed coordinated caching are expected to be highly interrelated, their intrinsic behavior has to be understood separately before combining them and examining dependencies. In the scope of this thesis, the possibilities for implementing coordination mechanisms is explored by developing a set of suitable coordination configurations. These are simulated in a realistic setup to validate the functionalities and show the potential of the resulting simulator. Besides, this allows demonstrating the coordination's influence on the overall system as well as on individual jobs.

Chapter 4

Examination of a production system's suitability for caching

To understand how realistic Tier 3 center's could be affected by introducing distributed coordinated caching outlined in chapter 3, the jobs that are processed there have to be examined. This allows determining how well suited the workload of a computing center is to benefit from caching and to identify workflows that would benefit especially. This knowledge can then be taken into account to optimize cache behavior and coordination for this center and draw conclusions on general properties of cache suitable workflows. Moreover, information about the workload of a computing center that is a realistic use-case for distributed coordinated caching, is vital to perform realistic simulations.

For this thesis, the Tier 3 computing center at ETP was chosen for examination since it represents a modern Tier 3 center in infrastructure, integration of opportunistic resources and workload. This chapter first details the acquisition of monitoring data the following studies are based on. Then, the characteristics of jobs and workflows that are likely to benefit from caching are defined and used to asses the suitability for caching of the monitored workload. Lastly, individual workflows in the workload are identified and their suitability for caching is examined based on the established characteristics as well.

4.1 Monitoring workloads of end-user analyses

When monitoring the workload of a production system like the Tier 3 center at ETP, metadata about the jobs that are processed there is gathered. The metadata that has to be collected is given by the information that enables the characterization of the jobs. This section details the information needed to characterize a job and how it was acquired.

4.1.1 Metadata characterizing a job

The information that describes a job can be divided into four areas:

- Job identification: metadata that identifies the job itself and the submitter, e.g. the person or group who submitted the job, is needed. This allows distinguishing jobs and to evaluate information considering the submitter, e.g. to study jobs submitted by a specific research group or a single user. Additionally, metadata such as the path to the job's executable can be included in the monitoring, as such metadata is useful to identify similar jobs [38].

- Timing: information about the job's timing includes the timestamps of the job's submission into the batch system, the beginning of the job's processing, and the end of the job's processing. Using these timestamps, the time that the job spent in the job queue before being assigned to a worker node, also called *queue time*, and the job's *walltime* t_{walltime} can be calculated. Additionally, the job's *CPU time* t_{CPU}, is recorded. The CPU time corresponds to the total amount of time a CPU core was fully utilized while the job was processed. For example, the CPU time of a job allocating one CPU core is shorter than the walltime when the job's processing is interrupted while data is transferred. If a job uses multiple CPU cores, the CPU time is cumulated from all CPU cores and can exceed the walltime. Based on the walltime and the CPU time, the job's *CPU efficiency*
$$\epsilon_{\text{CPU}} = \frac{t_{\text{CPU}}}{n_{\text{CPU}} \cdot t_{\text{walltime}}} \tag{4.1}$$
is calculated with n_{CPU} being the number of CPU cores the job allocated to. This efficiency indicates how well the CPU cores allocated by the job were utilized.

- Resources: jobs are characterized by the amount of resources they require, namely the amount of *CPU cores*, *memory* and *disk space*. It has to be ensured that a worker node provides these necessary resources before sending a job there. The precise amount of resources the job needs is only known after it is processed. Therefore, the job's resource demand is estimated by the job's submitter before allocating the job to a worker node. This resource demand estimation is based on experience values and referred to as *requested resources*.

- Inputfiles: in the context of caching, information about a job's input data is particularly important. This includes the name, location, and size of the job's *input files*. Besides this information, the total amount of data transferred to the job, *total data input* $V_{\text{job}}^{\text{tot}}$, can be monitored. This quantity usually includes the data volume of the input files but also accounts for the data volume of all other transfers to the job, e.g. the job's execution environment that has to be transferred to the worker nodes. Combining total data input and walltime, the average data throughput provided to the job, $s_{\text{job}}^{\text{provided}}$, is given by

$$s_{\text{job}}^{\text{provided}} = \frac{V_{\text{job}}^{\text{tot}}}{t_{\text{walltime}}}. \tag{4.2}$$

4.1.2 Data acquisition

The majority of the metadata needed to describe a job can be obtained from the job log of the batch system the job was processed in. In the case of HTCondor, the batch system used at ETP, the metadata of a job is referred to as the jobs' ClassAd. The ClassAd consists of a list of attributes and is stored for a limited amount of time after a job was processed. To monitor jobs, this log of completed jobs is accessed using the command line interface of HTCondor and the relevant data is periodically collected and stored in a database. A list of the ClassAd attributes that were monitored is presented in appendix A. While the majority of the monitored ClassAd attributes are implemented in HTCondor per default, some are specific to the setup at ETP. For example, an additional attribute that contains an explicit list of the jobs' input files is added to the jobs' ClassAd attributes. This is done automatically by the job submission tool widely used at ETP, grid control [39]. Alternatively, the list can be set by the user or automatically when the job's ClassAd information is modified by the distributed coordinated caching prototype NaviX introduced in section 3.5.2.

While the filenames of input files can be obtained from the job's ClassAd attributes, the size of the input files is not directly accessible to the batch system. Therefore, this information is gathered separately for job input files that were transferred using XRootD. The size of input files of recently monitored jobs is looked up periodically and stored in the database as well. The information about file sizes can then be included in the already existing job information.

Job selection

To decrease the influence of erroneous jobs submissions, only successfully completed jobs are recorded when monitoring the production system at ETP. To further ensure that the examined jobs are described by valid information and represent the monitored system, additional selections are performed.

First, jobs that are not part of the research group's physics activity are removed. For example, this includes jobs that were used to test the functionality of XRootD or NaviX in the setup at ETP.

Second, jobs with corrupted monitoring information are removed. Some of the information about a job, that is provided by the batch system, is gathered by the docker container the job is executed in. This information is not transferred continuously but periodically, resulting in imprecise values. Additionally, out-of-bounds

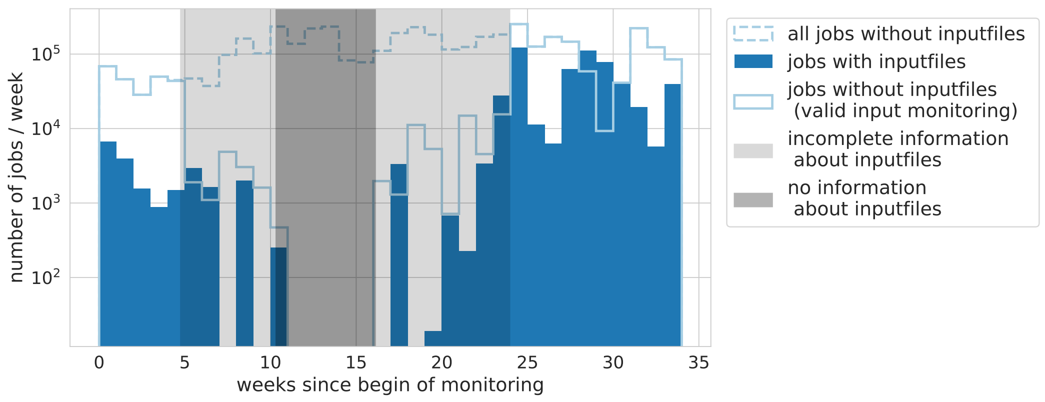


Figure 4.1: Number of monitored jobs per week during data acquisition. The number of jobs with information about input files is shown as bars while the number of jobs without input file information is shown as a line. During the time period shown in gray no information about the jobs' input files could be gathered for either all jobs (dark gray) or jobs submitted by a part of the research group (light gray). The afflicted jobs were removed from the sample. In these areas the histogram of the original amount of monitored jobs is shown with a dashed line, the amount of monitored jobs after this selection is shown with a solid line.

values can occur. To avoid distortions by incorrect values, jobs with $t_{\mathrm{CPU}} \leq 1\,\mathrm{s}$, $t_{\mathrm{CPU}} > 7\,\mathrm{d}$ and $V_{\mathrm{job}}^{\mathrm{tot}} > 2\,\mathrm{TB}$ are omitted.

Lastly, jobs requesting more than one CPU core are removed from the sample as the influence of multi-core jobs on the monitored system is small since these jobs account for less than one permille of the data sample. In addition, efforts that would be necessary to compare monitoring data from single and multi-core jobs are avoided.

Monitoring phases

The monitoring data used in this thesis was taken between May 2019 and February 2020. An overview of the number of jobs with and without input files monitored per week is shown in figure 4.1.

The acquisition of monitoring data was interrupted during the summer period as the service that added information about the jobs' input files to the jobs' ClassAd stopped working and required manual intervention. This leads to a time period without information about input files, marked in dark gray. To preserve the proportion between jobs with and without input files, all jobs that were monitored in this time period were removed. Additionally, the same service was inactive due to technical issues for a part of the research group during the period marked in light gray. Therefore, all jobs submitted by this part of the research group were removed in the respective time range as well.

The resulting number of monitored jobs without input files is depicted by a solid line while the original number of jobs without input files is shown as a dashed line. The mentioned interruptions divide the data acquisition into three phases whereby the phase from which jobs were removed separates the first and the last phase. It appears that the fraction of jobs with input files is lower in the first monitoring phase than in the final phase. On the one hand, this could be due to the activity and composition of a research group being time-dependent and fewer workflows with input files being processed in the first phase. On the other hand, in the setup at ETP it depends on the method of job submission whether information about input files is automatically added to a job's metadata or whether it has to be enabled explicitly by the users. Therefore, it is possible, that despite the efforts to ensure that information about input files is provided it was not successful for some users. Additionally, in the first monitoring phase users were encouraged to ensure that information about input data is provided for workflows that transfer data from remote storage using XRootD. In the last monitoring phase, this was extended to workflows processing locally stored input files that are transferred using POSIX.

While locally stored files are not subject to caching if only local resources are considered, this can change when the respective infrastructure contains opportunistic resources with access to caches. In the latter case, data that is stored locally at a Tier 3 center but processed on opportunistic resources without direct connection to the storage, has to be transferred via the network connection creating a use-case for caching if data is requested repeatedly. To demonstrate the potential of including these input files into caching they are listed as jobs with input files in figure 4.1. Since there is no information about the size of the files and as they only appear in the last monitoring phase they are not considered when discussing the amount of transferred data or the re-occurrence of input files. As information about locally stored files is only available for later times, it can be assumed that the fraction of jobs with input files is underestimated in the first monitoring phase.

In total, 4,894,396 jobs were monitored. After applying the selection criteria and removing jobs monitored in time periods with incomplete input file information, 2,203,798 jobs remain of which 607,762 jobs provide information on the input files. This corresponds to a fraction of 27.6 %.

4.2 Assessment of caching suitability

After monitoring the Tier 3 center as described in section 4.1, the workload is examined. The purpose of this is to determine whether the production system at ETP would benefit from caching and therefore provides a use-case to study distributed coordinated caching through simulations.

To this end, characteristics that indicate whether jobs could benefit from caching are derived in the following. The characteristics are then applied to the entirety of monitored jobs as a first indication of caching suitability. Finally, the jobs are grouped into workflows and the caching suitability of each workflow is assessed individually.

4.2.1 Characteristics of caching suitability

Jobs are likely to benefit from increased data locality if a set of properties is fulfilled. In general, jobs can only benefit from caching if they process input files and if these input files can be provided by a cache. As discussed in section 3.4.1, jobs needs to be processed repeatedly or share their input data with other jobs in order to reduce the load on the network and actively benefit from cached data. Since the content of the caches changes when new files are added as cache space is limited, files do not remain in the cache indefinitely. Therefore, the time between requests of the same file from multiple jobs has to be shorter than the lifetime of the files in the cache. Consequently, jobs that are processed repeatedly within short time intervals are more likely to benefit from caching.

As established in section 3.3.1, an estimate for the walltime of a job is given by the maximum of the time needed to transfer data, $t_{\text{data transfer}}$, and the time needed to perform calculations based on the transferred data, $t_{\text{calculation}}$. On the one hand, if the walltime of a job is dominated by $t_{\text{data transfer}}$, the job's performance is directly affected by changes in the data transfer such as bottlenecks in the network throughput. Therefore, a low CPU efficiency ϵ_{CPU} suggests that a job is limited by data transfer as the CPU cores claimed by the job are inactive while waiting for data to be transferred. In this case a job could benefit from cached data that is provided with a higher throughput making jobs with input files and low performance likely to be suitable for caching. Another indicator that a job is likely to be limited by $t_{\text{data transfer}}$ is the average data throughput required by a job,

$$s_{\text{job}}^{\text{required}} = \frac{V_{\text{job}}^{\text{tot}}}{t_{\text{CPU}}}. \tag{4.3}$$

Jobs that require high $s_{\text{job}}^{\text{required}}$ are more likely to be affected when bottlenecks exist in the network throughput that is provided to the jobs' data transfers.

On the other hand, if the walltime is dominated by $t_{\text{calculation}}$, jobs do not benefit directly as their walltimes do not shorten if the requested data are provided faster. Nevertheless, the overall system can benefit indirectly if such jobs read from the cache as this decreases the load on the network, possibly accelerating the data transfer of other jobs.

Additionally, jobs processed on local resources mainly benefit from cached data if their original data is located on remote storage. If the data is provided by local storage, creating additional copies in the cache does not increase the jobs' performance unless the connection to the cache provides more throughput than the connection to the local storage. However, if bottlenecks occur for the connection to the local storage this becomes a use case for caching as well.

Summing up, to directly benefit from caching, jobs have to repeatedly access files in intervals shorter than the lifetime of the files in the cache and the jobs have to be actively limited by data throughput. Jobs with higher $s_{\text{job}}^{\text{required}}$ are more likely to be limited by data throughput and a low CPU efficiency indicates that a job was affected by this limitation. In addition, other jobs in the system benefit indirectly if jobs read data from the cache. Even if the jobs are not accelerated directly, the load on the network is reduced and other data transfers are accelerated improving the systems overall performance.

4.2.2 Caching suitability of the overall workload

Based on the characteristics discussed in the previous section, it is examined whether the overall workload is suited for caching. Afterwards, the workload is split into workflows that are then inspected separately.

In section 4.1.2 it was already established that about quarter of the workload are jobs that process input files. As a first indicator whether there are jobs that process larger data sets than others, the amount of jobs with input files and known input file size is compared to the total data volume requested by these jobs. These quantities are compared between the subgroups of the research group at ETP as shown in figure 4.2. Subgroups are likely to contain data-intensive workflows if the fraction of data volume transferred by the jobs of a subgroup exceed the respective fraction of number of jobs of this subgroup significantly.

The categorization of the research group at ETP into subgroups is done by field of study. While all subgroups are working in the field of experimental particle physics, multiple experiments are represented. However, all monitored jobs with input files were submitted by one of three subgroups studying different physical processes as part of the CMS Collaboration. The subgroups are named Jet, Higgs and Top after the physical objects their work is mainly focused on. The *Jet group* mainly performs analyses based on Z + jet or di-jet events while the *Higgs group*, is focused on analyses of $H \to \tau\tau$ events and the *Top group*, concern themselves with processes including the top quark such as the production of a top anti-top quark pair.

It is observed that the Higgs group dominates the number of jobs with known input file size, making up 16.6 % of all monitored jobs while jobs submitted by the Jet and Top group only correspond to 4.0 % and 0.2 % respectively. In contrast, jobs submitted by the Jet group make up more than half of the cumulative amount of

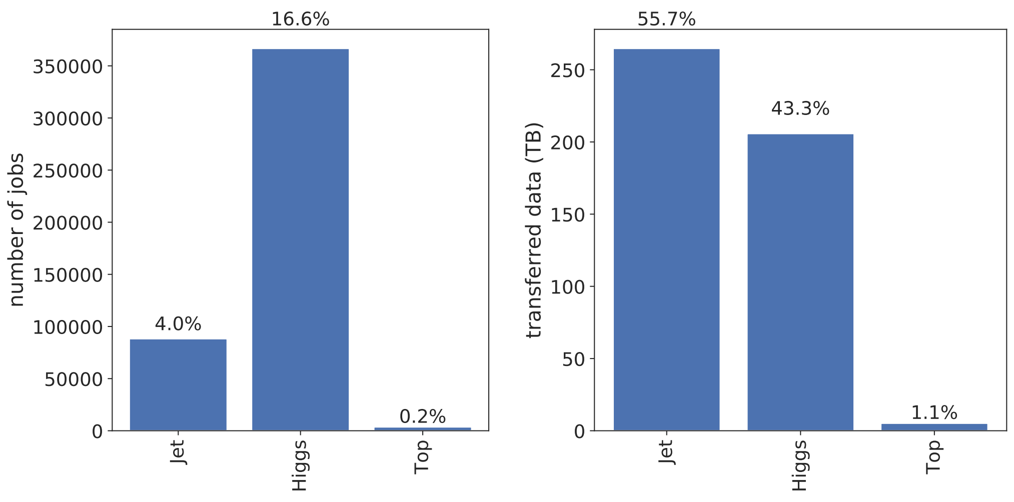


Figure 4.2: Comparison of the number of jobs with input files and known input file size to the total amount of data transferred by these jobs between research subgroups at ETP. While jobs submitted by the Jet group only make up a small fraction of all monitored jobs, these jobs transfer the majority of input data suggesting that the workload of the Jet group might contain workflows with large amounts of input data and potentially high data throughput.

requested data, followed by the Higgs group with 43.3 %. This indicates that the workload of the Jet group is likely to contain workflows that process large amounts of data. The presence of such workflows increases the chance that some workflows require high data throughput and could consequently benefit from caching. Workflows of the Higgs group should be examined as well as a similarly large amount of data as for the Jet group is requested. The workload submitted by the Top group will not be investigated further due to the low number of monitored jobs resulting in a limited conclusiveness about the jobs properties.

Repeatedly requested input files

A more explicit indicator for caching suitability is whether input files located on remote storage are requested repeatedly and the time between repeated requests of the same input file. Therefore, the timestamp of each occurrence of all input files is extracted to determine both the total number of occurrences and the time difference between each occurrence. The distributions of both quantities are shown in figure 4.3.

The distribution of the occurrence of input files on the left illustrates that input files are requested up to 21 times and that roughly half of the input files are requested more than once. There are no input files that were requested 11 to 14 times. This

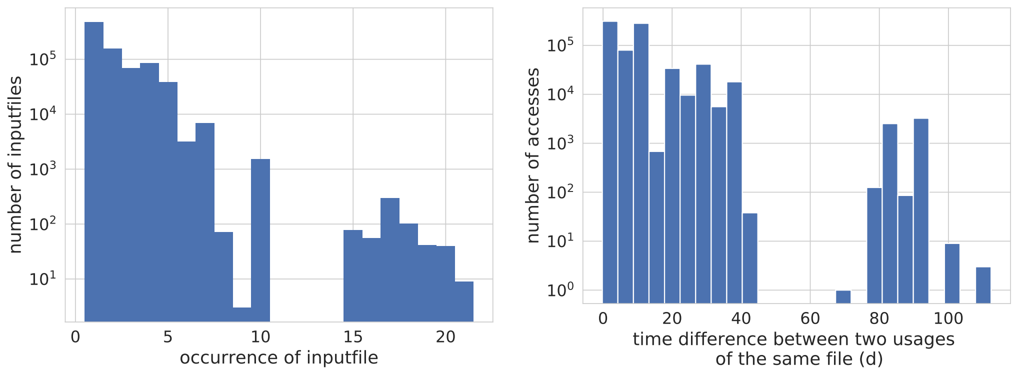


Figure 4.3: Distribution of the number of times an input files was requested (left) and of the time difference between two consecutive requests of a file in days (right).

could be an effect of workflows being repeated in different frequencies. In addition, the lack of input files with 11 to 14 repetitions could have been influenced by the time ranges with incomplete monitoring data discussed in section 4.1.2 since file requests that occurred during this period could not be recorded.

On the right of the figure, the distribution of the time difference between two consecutive accesses of a file is shown. Half of consecutive file accesses occur within less than 10 days. This suggests that the accesses of the same file do not necessarily happen due to a regularly repeated workflow but could also occur due to workflow optimization and debugging by the job's submitter or multiple jobs in a workflow requesting the same files.

Nevertheless, there are many file accesses that are separated by time differences up to 40 days and some with even larger time differences of about 80 days and more. This hints at the existence of workflows that are repeated on a monthly basis as well as workflows repeated on a quarterly basis. This observation is compatible with the stages of typical end-user analyses described in section 2.4. Workflows with a turnaround cycle of one month are likely to belong to the Analysis stage while workflows that were processed more rarely could be part of the Skimming stage.

Whether this is true will be discussed in detail after clustering the monitored jobs to workflows. In addition, the absence of consecutive accesses after 40 to 75 days can be explained by the disrupted monitoring. Even if requests separated by 40 to 75 days occurred in the monitored system, they could not have been detected as information about input files is missing for a time range of several weeks.

Eventually, the presence of jobs repeatedly requesting the same input files as presented in this section, suggests that the workload at ETP contains workflows that would benefit from caching.

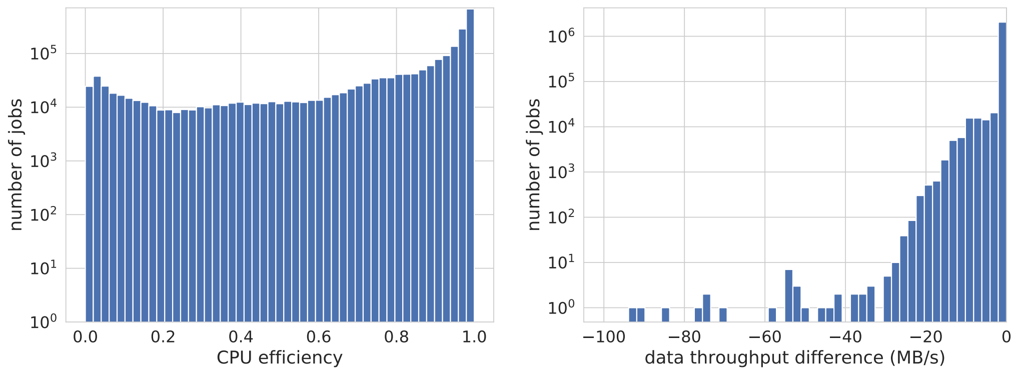


Figure 4.4: Distribution of the CPU efficiency (left) and difference between provided and required data throughput (right) of all jobs. The distribution of the CPU efficiency indicates that many jobs do not perform ideally. Similarly the data throughput provided to jobs falls short compared to the throughput that is actually required.

Job characteristics

Besides discussing whether the monitored workload contains input files that were requested repeatedly, it has to be examined whether the performance of jobs could be improved by accelerating the transfer of input data. If data transfers are limited and thereby the performance of jobs, the CPU efficiency ϵ_{CPU} of monitored jobs is low. In addition, $s_{\text{job}}^{\text{required}}$, the average data throughput that jobs require to avoid performance losses exceeds the average data throughput provided to the jobs, $s_{\text{job}}^{\text{provided}}$, if data transfer is limited. The distribution of the CPU efficiency and of the difference between the required and provided data throughput are shown in figure 4.4.

While there are many jobs with high CPU efficiency, the CPU efficiency of 40 % of the monitored jobs is below 0.9 and the number of jobs is similar for all bins between 0.2 and 0.6. The excess of jobs with $\epsilon_{\text{CPU}} < 0.1$ could be explained by underestimated values for the CPU time due to insufficiencies in the job monitoring of HTCondor as discussed in section 4.1.2 even though selection criteria were applied to minimize the impact of such jobs. In addition, it is possible that the low CPU efficiencies are due to unidentified test jobs that only read data but do not perform calculations.

The distribution of the difference between the average data throughput required by and provided to the job

$$\Delta s = V_{\text{job}}^{\text{tot}} \cdot \left(\frac{1}{t_{\text{walltime}}} - \frac{1}{t_{\text{CPU}}} \right) \tag{4.4}$$

shows that there are many jobs with insufficient data rate. For approximately 16 % of the jobs with input files the provided data rate falls at least 0.5 MB/s short.

Summing up the evidence presented in this section, it is indicated that the monitored workload is likely to contain workflows that are well suited to benefit from caching. To verify the existence of such workflows, jobs processing input files have to be clustered to workflows that can then be examined.

4.2.3 Caching suitability of individual workflows

After establishing that the workload monitored at ETP is likely to contain workflows suitable for caching, these workflows are reconstructed and reviewed. In this section the relevant workflows monitored at ETP are introduced before checking whether the individual workflows meet the expectations regarding their suitability to benefit from caching. To this end, the workflows of the Jet and Higgs group are discussed individually.

Workflow identification

To decide what workflows are well suited for caching the characteristics introduced in section 4.2.1 are applied. As jobs of one workflow usually have similar properties, the characteristics regarding suitability for caching can be generalized from jobs to entire workflows.

Only jobs that process input files are of relevance for caching. Therefore, the efforts of reconstructing what workflows the jobs belong to were focused on jobs with input files. For clustering, metadata of the job identification category were used, namely the job's submitter, the input files' path and filename, the path to the shell script that is the job's main executable and a ClassAd attribute that can be set by the user to identify similar jobs.

In this manner, four workflows submitted by the Jet group were found in the monitored time period:

- *MC production and processing*: Jobs aggregated in this workflow produce and process Monte Carlo simulations (MC) that are then used in analyses. As these simulations usually only have to be done once, this workflow is not likely to benefit from caching.

- *Skimming*: In section 2.4, Skimming was introduced as the initial stage of a typical end-user analysis. This reconstructed Skimming workflow includes all jobs submitted by the Jet group that belong to this stage. While Skimming jobs usually process large amounts of data, they are usually not well suited to benefit from caching because they are only executed a few times per year. Therefore, it is not considered to be worthwhile to block cache space over such time periods to provide cached data to Skimming jobs. Additionally, Skimming jobs can contain computationally intensive tasks making them less sensitive to delayed data transfer.

- *Di-jet analysis:* During the monitored time period a workflow of the Analysis stage focused on di-jet events was performed. Usually, large amounts of data are processed in the Analysis stage. Depending on whether the calculations that are performed on this data are computationally intensive, the performance of an Analysis can be impaired by bottlenecks in network bandwidth. While this workflow is not necessarily performed repeatedly it can still benefit from caching as it can take several iterations to develop a new workflow.

- *Jet Energy Calibration (JEC) analysis*: The JEC analysis workflow belongs to the Analysis stage as well. This workflow is expected to be particularly well suited to benefit from caching because it has to be processed repeatedly as it belongs to the multi-staged process of calibrating jets recorded by the CMS detector. In every stage a correction factor is derived and the process of adjusting all stages' correction factors has to be repeated multiple times until the calibration is complete leading to repeated processing of the same input data with varying correction factors.

In the workload submitted by the Higgs group, three workflows were identified:

- *MC production and processing*: As for the Jet group Monte Carlo simulations are produced and processed in the Higgs group as well. Jobs that contribute to such simulations make up this workflow. Similar to MC production and processing in the Jet group, this workflow is not expected to benefit from cached data.

- *Skimming*: Skimming is performed in the Higgs group as well. Again, this workflow is not expected to benefit from cached data either.

- *$H \rightarrow \tau\tau$ analysis*: The last workflow, short Higgs analysis, is an analysis based on events which are likely to contain Higgs bosons decaying into pairs of τ leptons. This is the only workflow in the Higgs group that is likely to benefit from caching. While it is not repetitive in the way that the jet energy calibration is repeated regularly, this workflow is performed often nonetheless as it makes up a large portion of the Higgs groups physics activity and contains multiple sub analyses processing the same input data.

Despite the efforts to match all jobs to workflows, some jobs could not be matched to the these workflows. Such jobs are marked *other* in the following and occur in the Jet group as well as in the Higgs group.

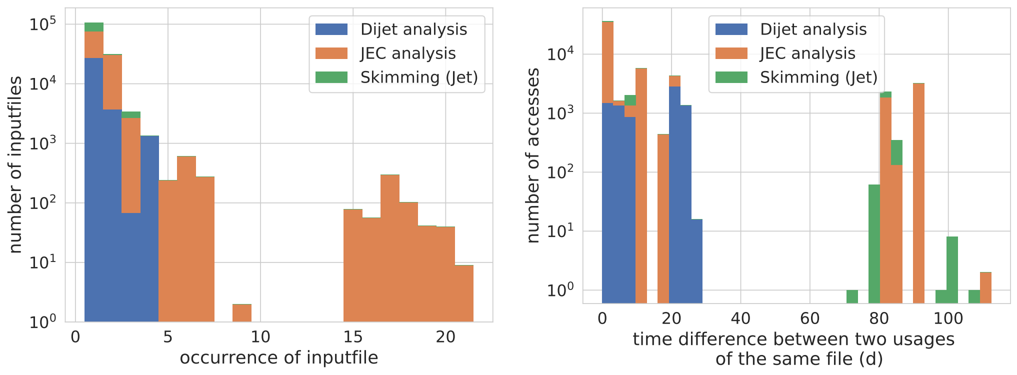


Figure 4.5: Distribution of the number of times an input file was requestd and distribution of the time difference between two consecutive requests of the same file of workflows in the Jet group.

Workflows of the Jet group

To assess the caching suitability of the workflows of the Jet group, file requests as well as quantities tied to the jobs' performance are studied. The distribution of the number of times distinct input files were requested is shown in figure 4.5 along with the distribution of the time difference between two consecutive requests.

As jobs of the MC production and processing workflow did not read any input files from remote storage they do not appear in these distributions. Input files of Skimming jobs are only requested a few times, in this case either once or thrice, while the time differences between accesses is either in the range of about a week or of several months. This fits the expected behavior of Skimming being repeated rarely and in long intervals. The shorter time difference is likely to be caused by different jobs processing the same input files or re-submission of jobs because they were not configured or processed correctly on first try.

The input files of the Di-jet analysis occur up to four times and are requested within a week or a month. This represents ongoing work on the workflow with the larger time differences likely being due to other activities taking place or holiday periods. This indicates that caching the input files would have been beneficial as the time differences between accesses is reasonable.

The access pattern of input files of the JEC analysis agrees with the expectation as well as the input files are requested up to 21 times and across a wide range of time differences. The disruption between 30 and 80 days is most likely due to the disruption in the monitoring process discussed before or general dynamics within the research group.

To determine whether the performance of jobs of the workflows in the Jet group with repeatedly requested input files could be improved the CPU efficiencies of the

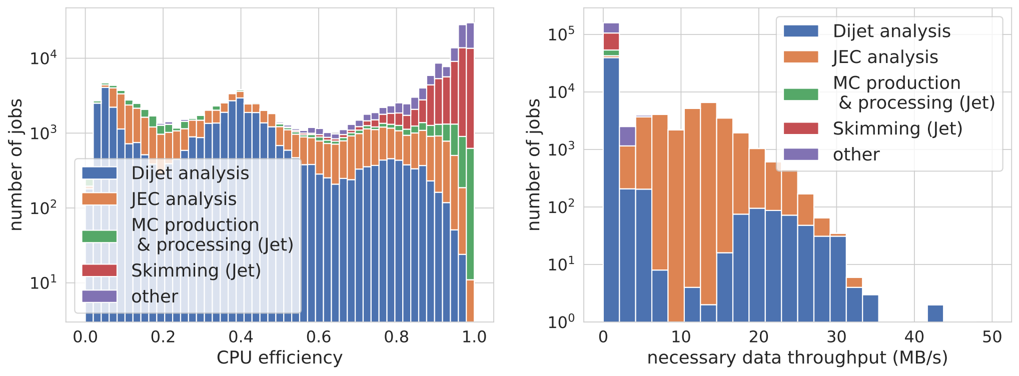


Figure 4.6: Distribution of the CPU efficiency and required data throughput for workflows of the Jet group.

workflows are studied along with the required data throughput $s_{\text{job}}^{\text{required}}$ that allows to deduce which jobs require high data throughput. The distributions of these quantities are illustrated in figure 4.6.

While the CPU efficiency of Skimming jobs is high, the CPU efficiency of the workflows that process larger amounts of data, namely the Di-jet and JEC analysis workflow, is distributed across a wide range. It is unlikely that the CPU efficiency of similar jobs is this wide-spread without depending on external conditions in the system, for example bottlenecks in data transfer. This suggests that the performance of these workflows is likely to be limited. The distribution of $s_{\text{job}}^{\text{required}}$ further motivates this. While almost all jobs that do not belong to these workflows are located in the lowest bin of $s_{\text{job}}^{\text{required}} < 3\,\text{MB/s}$, the required throughput of the Analysis workflows range up to $40\,\text{MB/s}$ making them sensitive to insufficient data throughput.

Combining the results from examining the file access pattern and the jobs' performance it can be concluded that all workflows submitted by the Jet group agree with the expectations concerning caching suitability. The Di-jet analysis could benefit from caching as it was executed repeatedly during its development. In contrast, the JEC analysis workflow is well suited to benefit from cached data due to its repetitive nature and indications of performance limitations. To understand in detail why this workflow shows the mentioned properties, it is discussed in chapter 5 in greater detail. The jobs of the remaining workflows are not likely to benefit from caching as expected.

Workflows in the Higgs group

The caching suitability of workflows submitted by the Higgs group is investigated analogously to that of the Jet group. As only about 5000 jobs of the Skimming

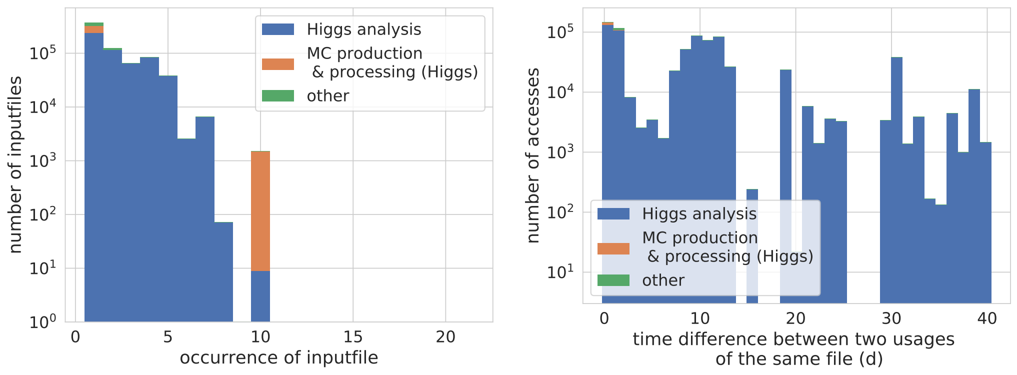


Figure 4.7: Distribution of the number of times an input file was requested and distribution of the time difference between two consecutive requests of the same file of workflows in the Higgs group.

workflow were monitored, this workflow is not investigated further in the following due to the resulting limited possibility to draw valid conclusions for this workflow. The distribution of the number of times distinct input files were requested and of the time difference between two consecutive requests of the same input files are shown in figure 4.7.

As expected, the majority of repeatedly requested input files belong to jobs of the Higgs analysis workflow. Input files of this workflow are requested up to ten times. All other workflows except for the MC production and processing workflow do not have input files that are requested more than once.

The input files of the MC production and processing workflow are requested either once or ten times. This could be due to the production and processing consisting of multiple stages with each stage taking the output of the previous stage as input. Therefore, this behavior could be caused by each output file of one stage being the input to multiple different jobs of the next stage. Alternatively, a part of this workflows jobs could have been processed repeatedly, e.g. for testing purposes. However, suitability for caching does not follow directly as the time difference between these accesses is small indicating that repeated accesses are not due to repeated processing of the same workflow. Instead, repetition most likely appear due to testing or multiple jobs accessing the same input files. In contrast, the time difference for input file accesses of the Higgs analysis is spread across a time range of up to 40 days. This corresponds to a time range that input files could realistically remain in a cache.

The distributions of the CPU efficiency and $s_{\mathrm{job}}^{\mathrm{required}}$ are shown in figure 4.8. As for the Jet group CPU efficiencies across the whole range were measured and low efficiency values were observed for data processing workflows or unidentified workflows while the amount of jobs with low CPU efficiency decreases for other workflows.

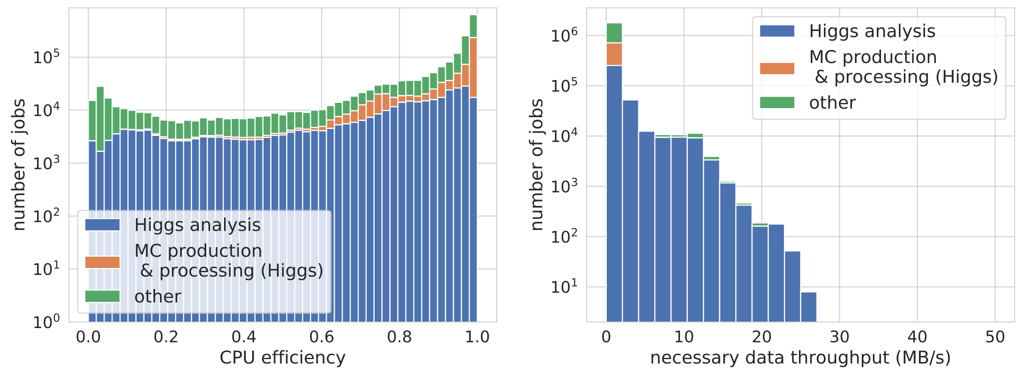


Figure 4.8: Distribution of the CPU efficiency and required data throughput for workflows of the Higgs group.

High amounts of required data throughput are also mainly populated with jobs of the Higgs analysis workflow or unclassified jobs. This suggests that the set of unclassified jobs still contains jobs that could belong to either the Higgs analysis workflow or a similar workflow. Despite this, the behavior of the identifiable workflow submitted by the Higgs group regarding caching suitability matches the expectations formulated in section 4.2.3 as well.

4.3 Conclusion

In the scope of this thesis, a system for monitoring the workload of the production system at ETP was developed and successfully utilized to collect monitoring data. Despite a relatively short data acquisition period of less than a year and time periods that had to be excluded due to incomplete monitoring data, insights into the composition and characteristics of the workload as a whole were gained. Individual workflows that make up this workload were identified and studied as well.

To identify workflows that are well suited to benefit from caching, characteristics were defined and compared to the extracted workflows. After establishing that the overall workload is likely to contain jobs suitable for caching, the expectations towards the individual workflow's caching suitability were confirmed. The JEC analysis workflow, the Higgs analysis workflow and the Di-jet analysis workflow were identified as well suited for caching. Among these workflows, the JEC analysis workflow and the Higgs analysis workflow are classified as particularly well suited for caching as they are actively repeated while the Di-jet analysis workflow is mainly repeated due to testing and development. Based on these insights, the monitored jobs can be used to represent the workload of a real system with a performance that could benefit from cached data in the simulations exploring distributed coordinated caching that were performed in the scope of this thesis.

Furthermore, the monitored workload can be made subject to future research investigating the workloads properties especially as monitoring data is continued to be taken. In the scope of facing the challenges of distributed coordinated caching as introduced in section 3.4 a thorough examination of existing workflows could be necessary to decide what files should be cached based on the jobs' properties. Finally, experience gained with this monitoring could be used to examine performance improvements once a prototype for handling distributed coordinated caching is implemented.

Chapter 5

Jet energy calibration as a typical end-user analysis workflow

In chapter 4, the Analysis stage of the jet energy calibration workflow was identified as a workflow that could benefit from caching. To perform high precision analyses or to enable a consistent comparison of results between different analyses and experiments, data taken with the CMS detector needs to be carefully calibrated. Fully calibrating the CMS detector is a huge effort requiring both the calibration of detector components and reconstructed objects. Many research groups in the CMS collaboration contribute different parts of the overall calibration. The research group at ETP contributes in different ways, one being part of the calibration of jet energies. As discussed in section 2.2, jets are complex objects made of multiple particles. During calibration a reconstructed jet's energy is related to the energy of the particle the jet originated from. Correctly calibrating jets is of great importance, as they occur in most collisions and play a significant role in many analyses conducted by the CMS collaboration. To further motivate why it is essential to process this workflow efficiently, this chapter discusses the approach for calibrating the jet energy employed by the CMS collaboration and the part carried out at ETP using $Z\,(\rightarrow \mu\mu/ee) + \text{jet}$ events.

5.1 Jet energy calibration at CMS

The calibration process is split into three major stages [40]. Each stage addresses a different effect contributing to the deviation between the jet's measured energy and the energy of the original parton. An overview of the stages is shown in figure 5.1.

- **Pileup corrections:** The first correction stage, also known as L1 stage, corrects for particles that were falsely identified as contributing to the jet. These particles stem from additional interactions besides the hard scattering process within a proton-proton collision such as underlying event or pileup. The underlying event includes all particles present from the current collision that are not part of the hard scattering process [41] while particles that stem from previous or simultaneous collisions and their remnants in the detector are summarized as pileup [42].

- **Response corrections based on Monte Carlo simulation:** The second stage corrections, called L2L3 corrections, are fully based on simulated events and relate the energy of the reconstructed jet to that of the original parton. To achieve this, events containing jets are simulated on parton level and propagated through the detector's response simulation. The resulting corrections are derived from the difference between the energies of the original particle and the reconstructed jet. They are determined in bins of p_T and η of the jet and applied to simulated and recorded events.

- **Residual corrections:** Residual corrections are determined to correct for the remaining differences between simulated and observed detector responses. To achieve this, the properties of a jet and a reference object are compared. There are two types of residual corrections: **Relative** corrections (L2Res) ensure a correct response across all detector regions. Therefore, a jet detected inside the detector's central region is balanced against one in a different detector region. **Absolute** corrections (L3Res) ensure a correct detector response across a wide p_T range and are determined by balancing jets in the central region of the detector against Z bosons, photons or other, already well-calibrated jets.

- There are further calibration stages that can optionally take place after residual corrections. As they do not affect the previous stages, they are considered in this thesis but are described in [43].

The research group at ETP contributes to the calibration process by determining absolute residual corrections using Z ($\rightarrow ee/\mu\mu$) + jet events. Other research groups in the CMS collaboration examine the remaining channels and the complete absolute residual corrections are determined in a global fit.

5.2 Absolute residual jet energy corrections using $Z\,(\rightarrow \mu\mu/ee) +$ jet events

To determine absolute residual corrections to the jet energy as described in section 5.1, Z bosons are chosen as reference object. They are a suitable choice because their properties are well-known. Additionally, they can decay into a pair of electrons or muons. The CMS detector is able to precisely measure these particles making it possible to derive correction factors in both channels and combine them.

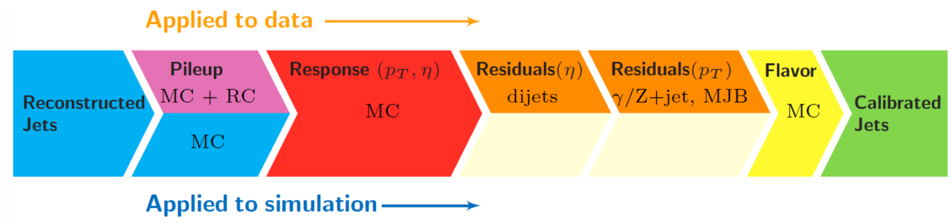


Figure 5.1: Visualization of the different stages of the jet energy calibration performed by the CMS collaboration. Reconstructed jets are first corrected for effects such as pileup, using simulation based (MC) and data driven methods like the Random Cone method (RC) [43]. Second, corrections based on the detector's response and derived from simulations are applied. Lastly, the remaining discrepancies between the response of recorded and simulated data sets are compensated by residual corrections based on simulations and recorded data of different channels. Before obtaining fully calibrated jets, optional adjustments like flavor calibrations can be made [43].

5.2.1 Balancing methods

There are two calibration methods utilizing the balancing of a jet against a reference object, in this case a Z boson. In each method a jet response R is determined that relates the energies of the jet and of the reference object. A correction factor is obtained by calculating the ratio of R for recorded and simulated data sets.

Method 1: p_T-balance method

The p_T-balance method is based on the assumption of an ideal event topology. In this case the reference object's transverse momentum is equal to that of the parton from which the jet originated. Therefore, the jet response R_balance is given by

$$R_\mathrm{balance} = \frac{p_\mathrm{T}^\mathrm{rec\ jet}}{p_\mathrm{T}^\mathrm{Z}} \tag{5.1}$$

and is equal to unity in the calibrated case. This method is robust against pileup since only the Z boson and it's decay products need to be well understood and identified. Biases by other objects are avoided, but there can be biases due to final state radiation and similar effects. Due to these phenomena, a fraction of the parton's energy is carried away in additional jets. Therefore, the event's topology deviates from the ideal event topology.

Method 2: MPF method

The missing transverse energy projection fraction (MPF) method is based on the assumption that in an ideal event consisting of a jet and a Z boson, all energy results either from the jet or the Z boson and that there is no intrinsic MET present in the event:

$$\vec{p}_{\mathrm{T}}^{\mathrm{Z}} + \vec{p}_{\mathrm{T}}^{\mathrm{jet}} = 0. \tag{5.2}$$

Additional jets are neglected. Therefore, assuming that the Z boson is reconstructed correctly, every MET in the recorded event stems from incorrect measurements of the jet's energy:

$$-\vec{E}_{\mathrm{T}}^{\mathrm{miss}} = \vec{p}_{\mathrm{T}}^{\mathrm{Z}} + R_{\mathrm{jet}}\vec{p}_{\mathrm{T}}^{\mathrm{jet}}. \tag{5.3}$$

Combining equation (5.2) and (5.3), the MPF methods jet response R_{MPF} corresponds to the jet's response factor R_{jet}:

$$R_{\mathrm{jet}} = 1 + \frac{\vec{E}_{\mathrm{T}}^{\mathrm{miss}} \cdot \vec{p}_{\mathrm{T}}^{\mathrm{Z}}}{(p_{\mathrm{T}}^{\mathrm{Z}})^2} \equiv R_{\mathrm{MPF}} \tag{5.4}$$

This method is robust with respect to imperfect event topologies as it does not depend on the measured properties of the balanced jet. However, unlike the p_{T}-balance method, this method is not robust against pileup. Transverse momentum of particles that do not stem from the hard scatter process might not be balanced and contribute to the event's intrinsic MET. The preferred balancing method in the CMS collaboration is the MPF method as it delivers better results. The p_{T}-balance method is used for cross-checks [40, 44]. More in-depth explanations for both methods can be found in [43].

Topology extrapolation

Events taken by the CMS detector rarely match the ideal event topology of a single jet recoiling against a Z boson. Instead, there are multiple jets due to pileup, final state radiation, underlying event or deviation between reclustered and actual jets. The amount of such *second jet activity* present in an event is quantified by

$$\alpha = \frac{p_{\mathrm{T}}^{\mathrm{jet2}}}{p_{\mathrm{T}}^{\mathrm{Z}}}, \tag{5.5}$$

where $p_{\mathrm{T}}^{\mathrm{jet2}}$ is the transverse momentum of the jet with the second highest transverse momentum. Higher values of α correspond to increased additional jet as energy carried away from the original jet activity can manifest as second jet. As the response of the p_{T}-balance method is proportional to the original jets energy, the former is expected to decrease with α while the MPF response is expected to be less dependent on α because it is less sensitive to additional jet activity.

To calculate the jet response value for an ideal topology ($\alpha = 0$), MPF and p_{T}-balance are determined in bins of α and an extrapolation to $\alpha = 0$ is performed. This extrapolation is shown in figure 5.2 for $Z\,(\rightarrow \mu\mu) + \mathrm{jet}$ events from the CMS

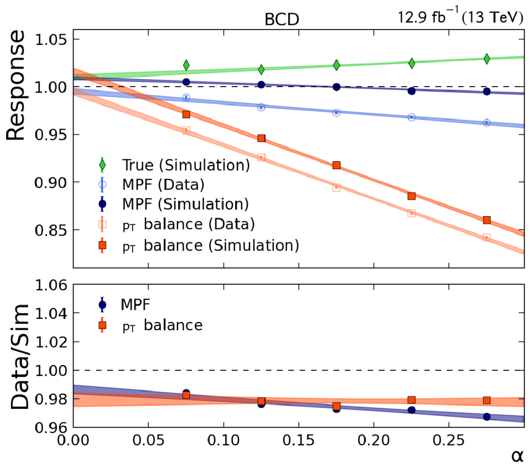


Figure 5.2: Extrapolation of jet responses to an ideal topology for the CMS Runs B, C and D in 2016 in the $Z\,(\rightarrow \mu\mu) + \text{jet}$ channel. In order to derive the jet responses (upper figure) and correction factors (lower figure), an extrapolation of the second jet activity α to $\alpha = 0$ is performed. In this case, the ideal event topology, jet responses of the p_T-balance and MPF method as well as the true response agree within their uncertainties. The correction factors, defined as the ratio of jet responses of recorded and simulated events, are compatible for the p_T-balance and MPF methods [45].

runs B, C and D in 2016. The upper figure shows the jet responses' dependency on α for simulated and recorded data. Both jet responses show the expected behavior and agree for $\alpha = 0$ within their uncertainties for data and simulations respectively. In addition, the true response is shown as a cross-check. The true response can only be derived from simulated events and is defined as the ratio of the events most energetic jet's transverse momentum true value and reconstructed value. For $\alpha = 0$, the true response agrees with the p_T-balance and MPF jet responses of simulated events, showing that all methods give consistent results for ideal event topologies. The correction factors are the ratio of the jet responses derived from data and simulation. The results of both methods are consistent and agree within uncertainties.

Combination of absolute residual corrections

To determine absolute residual corrections that are less dependent on a single channel's properties and therefore more reliable, the results of all examined channels and both methods for deriving the jet responses are combined by a *global fit*. As stated in section 5.1, γ + Jet, Z + Jet and channels with multiple jets are used. For all channels, the jet responses are determined and extrapolated in bins of p_T. The resulting global fit for runs B, C and D in 2016 is shown in figure 5.3 as an example. By

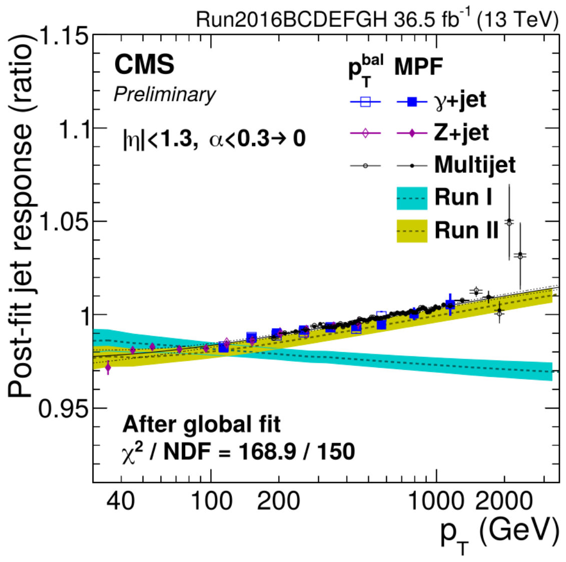


Figure 5.3: Preliminary global fit determining the absolute residual corrections for all CMS runs in 2016. For all channels and both jet responses MPF and p_T-balance, the ratio of the responses extracted from data and simulation is determined in bins of p_T and extrapolated to the ideal event topology. The resulting fit's error band is shown in yellow, the correction factors from LHC Run I are shown with a blue error band for comparison [46].

combining all channels, the correction factor, shown with a yellow uncertainty band, is determined for a large range of p_T. Correction factors for jets detected outside the central region ($|\eta^{jet}| \geq 1.3$) are determined by combining these results with relative residual corrections.

5.3 The jet energy calibration workflow at ETP

The determination of absolute residual jet energy corrections at ETP is performed in an end-user analysis workflow as described in section 2.4. In this section this workflow is described in more detail and its suitability for caching is discussed.

5.3.1 Workflow stages

The workflow consists of three subsequent stages, Skimming, Analysis and Plotting that can be executed separately and using the batch system.

Skimming

In a first stage, Skimming, the *Karlsruhe Package for Physics Analyses (KAPPA)* [47] is used to perform a loose selection of events. In addition, jets in selected events are reclustered to extend the data set for calibration to low $p_{\mathrm{T}}^{\mathrm{jet}}$. To derive absolute residual corrections in the $Z\,(\rightarrow \mu\mu)$ + jet channel for data taken in 2018 12 TB of recorded and 4.26 TB of simulated data sets are skimmed and reduced to 3.4 TB and 1.24 TB respectively. This corresponds to a reduction by factor 3.5. This stage is processed once and repeated when the initial data sets change due to new data being recorded or existing data being reprocessed. Therefore, this stage is usually executed only a few times per year.

Analysis

In the second stage, the Analysis, the events remaining after Skimming are processed using *Excalibur* [48], an analysis tool for studying Z+jet events recorded by the CMS detector based on the analysis framework *Artus* [49]. This includes a more strict event selection as well as the calculation of observables specific to residual corrections such as p_{T}-balance and MPF. During event selection all events that do not contain $Z\,(\rightarrow \mu\mu/ee)$ + jet processes and do not match the described event topology are rejected. Suitable selection criteria to identify such events are listed in the following.

- Events have to contain at least two well identified muons (electrons) that were scattered into the detector region in which leptons can be detected reliably ($|\eta| < 2.3$). The leptons transverse momentum needs to be high enough to produce valid traces in the detector's tracker and therefore allow correct reconstruction.
- The properties of the Z boson candidate are reconstructed from the observed leptons. The resulting reconstructed mass has to be close to the actual mass of a Z boson and its transverse momentum has to be large enough to allow for correct reconstruction. Furthermore, it has to be scattered in the opposite direction of the event's most energetic jet.
- At least one jet needs to be present in every event and the transverse momentum of the most energetic jet has to exceed a threshold that can vary between analyses. Furthermore, the most energetic jet needs to be located in the detector's central region as discussed in 5.1 and the events second jet activity α needs to be lower than 0.3.

A more explicit list of selection criteria and corresponding threshold values used to create the extrapolation in figure 5.2 is presented in appendix B. In case of the $Z\,(\rightarrow \mu\mu)$ + jet channel of 2018 this stage reduces 4.6 TB of input data to 34 GB of output data. This stage is processed more often than Skimming as it needs to be redone whenever a correction factor from previous calibration stages change. Therefore, this stage can be executed as often as once a week in periods when the jet energy calibration is actively worked on.

Plotting

Plotting, the last stage, is performed using the *Karma* software package [50]. In this stage, distributions of the p_T-balance and MPF jet responses are created as well as a variety of control plots used to verify the agreement of simulated and recorded data sets and detect anomalies. The amount of output data for this stage is of the order of MB, depending on the amount of plots that are created. For example, the size of the files serving as input for the $(Z \to \mu\mu)$ + jet channel in the global fit of 2018 was 70 MB.

5.3.2 Suitability for caching

The jet energy calibration is processed at regular intervals for two reasons. On the one hand, the calibration needs to be carried out when new data becomes available. On the other hand, the calibration is usually not completed with a single iteration of all stages. Whenever results and cross-checks show anomalies or effects that are not yet understood, discrepancies and sources of error are identified and removed and all calibration stages affected by them need to be redone. For example, the simulation might not represent the detector's functionality due to aging detector parts.

Additionally, the update of one or more jet energy calibration stages tend to be time-critical. First, the results of one stage are needed to perform the next stage and long workflow processing durations might impair schedules of other research groups. Second, the jet energy corrections are needed for every analysis containing jets. A delay in the determination of new correction factors can therefore delay many analyses. Based on these attributes, the jet energy calibration workflow is a promising choice for workflows that could benefit from caching.

Nevertheless, not all workflow stages are suited for caching as shown in section 4.2.3. Although large amounts of data are processed during Skimming, this stage is not suited for caching as it is only performed a few times per year. Additionally, the jet reclustering is computationally intensive, resulting in job runtimes dominated by data processing instead of data transfer and thereby benefiting little from cached data. In contrast, the Analysis stage is performed more frequently. Furthermore, the amount of data processed in this stage is still large while data processing is less computationally intensive than Skimming. This makes the Analysis stage exemplary for a workflow suitable for caching. While the Plotting stage is processed most frequently of all stages, this stage does not rely on being executed using the batch system as its input data is usually still manageable for local resources. For example, no Plotting jobs were identified in the monitoring presented in chapter 4. Considerations concerning the use of caching are therefore focused on the Analysis stage.

Chapter 6

Simulating Distributed Coordinated Caching

As discussed in chapter 3, distributed coordinated caching for Tier 3 resources is complex to realize. Multiple aspects such as the physical setup and the software behavior of the caches as well as the actual coordination mechanism need to complement each other and match the infrastructure of resources and caches as well as the workload of the computing center they are applied to. While suitable realizations can be found by testing multiple implementations of distributed coordinated caching, this approach can easily become time-intensive and expensive. In addition, optimizations that are made by testing implementations in real systems are limited to the infrastructure and workload available in the computing center at one point in time. Consequently, implementations may only be suitable for a limited amount of time and can not necessarily be transferred to other systems. To avoid this, simulations are a promising approach as they allow to study the influence of the aspects of distributed coordinated caching and to test implementations on computing centers with varying infrastructure and workload. This allows to gain a better understanding of distributed coordinated caching and to develop suitable implementations that are robust against changes in the computing centers infrastructure or workload.

The general approach followed in this thesis is to create a simulation tool that is able to mimic the behavior of a real system and its relevant mechanics. This allows directly transferring findings from simulation to real systems. In this chapter the requirements for and development of such a simulator as well as the actual implementation that was done in the scope of this thesis are discussed. Using this simulator, the possibilities to coordinate jobs to cached data are examined.

As discussed in section 3.4, there are multiple aspects to realizing distributed coordinated caching that have to be examined. While the developed simulation tool allows studying all of them, coordination mechanisms are the focus of this thesis. Suitable choices for coordination mechanisms that can be integrated into the matchmaking process of the batch system as proposed in section 3.4.2 are deduced and evaluated using the simulation tool. Besides this thesis, work on the cache eviction policy was performed in the research group and is presented in [51]. The caching policy has not yet been examined in detail and remains an interesting topic for future studies. Basic considerations on the topic are presented in appendix D.

6.1 Design of a simulation tool for distributed coordinated caching

There are multiple aspects to be taken into account when designing and developing a simulation tool suited to study distributed coordinated caching. On the one hand, there is a set of requirements that needs to be met as different realizations for cache behavior and coordination mechanisms are simulated. On the other hand, configurations that perform well in simulations need to be transferable to the computing environment of real systems. Therefore, the computing environment of Tier 3 centers has to be taken into account as well.

6.1.1 Requirements

The simulation approach pursued in this thesis mirrors the real system with a resolution of individual jobs. This means that for each job the submission into a batch system, the matching to adequate resources, as well as the job's processing, down to file transfers from caches and remote storage, are simulated. This means that jobs, worker nodes, a scheduler performing the matchmaking of jobs and worker nodes, caches and remote storage as well as their respective connections need to be simulated.

To face the challenges of distributed coordinated caching presented in 3.4, the properties and the behavior of the mentioned components need to be configurable as many systems with varying properties and structure have to be simulated. For worker nodes different resource types such as homogeneous or heterogeneous resources and different system occupancies have to be realizable while the simulation's job input has to be able to represent different workload compositions. To reproduce the behavior of caches their placement throughout the system, the storage space they provide, the worker nodes by which they can be accessed, and the bandwidth provided by the connections between worker nodes and caches need to be modifiable. In addition to the cache connection, bandwidth of the connection between worker nodes and one or multiple remote storages has to be configurable. Apart from providing configurability for the setup of the simulated system, the caches' behavior including cache population policies and cache eviction policies need to be exchangeable and the scheduler needs to allow for modification to include different coordination mechanisms.

Based on the requirements discussed in this section, a simulator is created that allows studying distributed coordinated caching in various setups.

6.1.2 Computing environment for distributed coordinated caching at a Tier 3 center

To make sure that configurations performing well in simulations can be implemented in real systems the simulator needs to be modelled after software used in the real system for job scheduling and the operation of caches. At ETP, HTCondor is used as a batch system while XRootD is used for file transfers and cache management. Therefore, the respective components of the simulator should be modelled to mimic these implementations.

Batch System: HTCondor

HTCondor is an open-source batch system developed at the Center for High Throughput Computing at University Wisconsin-Madison. It is used in the context of High Throughput Computing (HTC) in both science and industry as it is designed to optimize the throughput of jobs over long periods of time. In contrast to batch systems for HPC, it is not focused on providing large amounts of resources for large jobs across multiple machines. Instead, HTCondor maximizes the amount of resources available for execution of independent jobs. It especially allows combining resources with different owners while upholding usage priorities between them. Furthermore, HTCondor allows for job execution in complex distributed systems as it is able to transfer a job's input and output data and does not rely on storage being accessible by all resources. Additionally, mechanisms such as flocking, allow combining different HTCondor systems for even more wide-ranged pooling of resources. These properties make it a common choice of batch system to manage distributed resources in the HEP context.

An HTCondor system is made up of multiple services which are distributed across all machines of an HTCondor cluster. A usual set up that is also used at ETP consists of worker nodes to process the actual workloads, entry points for job submission called submit nodes and a central instance to collect information and match jobs to resources. The functionality of HTCondor is handled by four main services: Startd, Schedd, Collector and Negotiator. While information about the status of worker nodes is collected regularly by the Collector, the gathering of information about submitted jobs and their management is performed by a Schedd on each submit node. The matchmaking between jobs and resources is done by the Negotiator and a Startd handles the execution of jobs on each worker node.

The communication of job and worker node requirements and preferences is handled through Classified Advertisements, short *ClassAds* [52]. Each job and each resource is described by pairs of attributes and corresponding expressions combining all relevant metadata. Expressions are either primitive terms, such as numbers, or multiple terms linked by operators. Individual terms can be constants or contain references to another object's ClassAd. By evaluating one ClassAd against another these references are resolved. This mechanism is used for example to evaluate the compatibility of jobs and worker nodes during the matching of jobs to resources.

The ClassAd of jobs can be modified to introduce new or update existing metadata after the job is submitted. For example, this can be used to provide information about data locality. Information in the job's ClassAd about which caches provide a job's input files is determined after submission and can be updated each time the job is attempted to be matched. To achieve this, two ways to add or update ClassAd information are provided. First, this can be achieved by using external scripts, so-called hooks, that are executed by HTCondor in case certain events occur, for example the submission of a new job. Second, the ClassAd information of jobs in the job queue can be edited explicitly, both by a user or a separate service.

Negotiation in HTCondor

Jobs submitted into the batch system are stored in a job queue. In regular intervals Δt_{neg} of the order of minutes, the negotiator starts a new negotiation cycle during which jobs in the job queue are matched to resources.

To make this process more efficient, the negotiation process is applied to groups of similar jobs, called *autoclusters*, instead of repeating identical calculations for many jobs. Jobs are clustered by submitters, being either individual users or groups of users. The order in which the job clusters are treated is determined by a priority value that reflects how many resources a submitter may currently receive. Jobs of submitters with high priority are matched first. The sequence in which jobs of one submitter are matched is determined by a user-defined priority and, for identical or undefined priorities, in chronological order of their submission. The matching of jobs to resources is then performed sequentially. The steps of this process are visualized in figure 6.1.

For each job all worker nodes are considered regardless if they are claimed by other jobs or not. This provides an opportunity to free machines for jobs that suit them better or are deemed more relevant. The corresponding mechanism is called preemption. Currently, preemption is not used for end-user analysis jobs at ETP and whether jobs should be preempted in favor of other jobs based on data locality is beyond the scope of this thesis but may be subject to future studies.

Resources that are identical with respect to all ClassAd attributes taken into account in the following process are grouped into autoclusters as well and are from now on are handled together. Incompatible matches between jobs and resources are excluded by evaluating the `Requirement` attribute in the job's and the machine's ClassAd. Preferences regarding the remaining resources are determined in three steps.

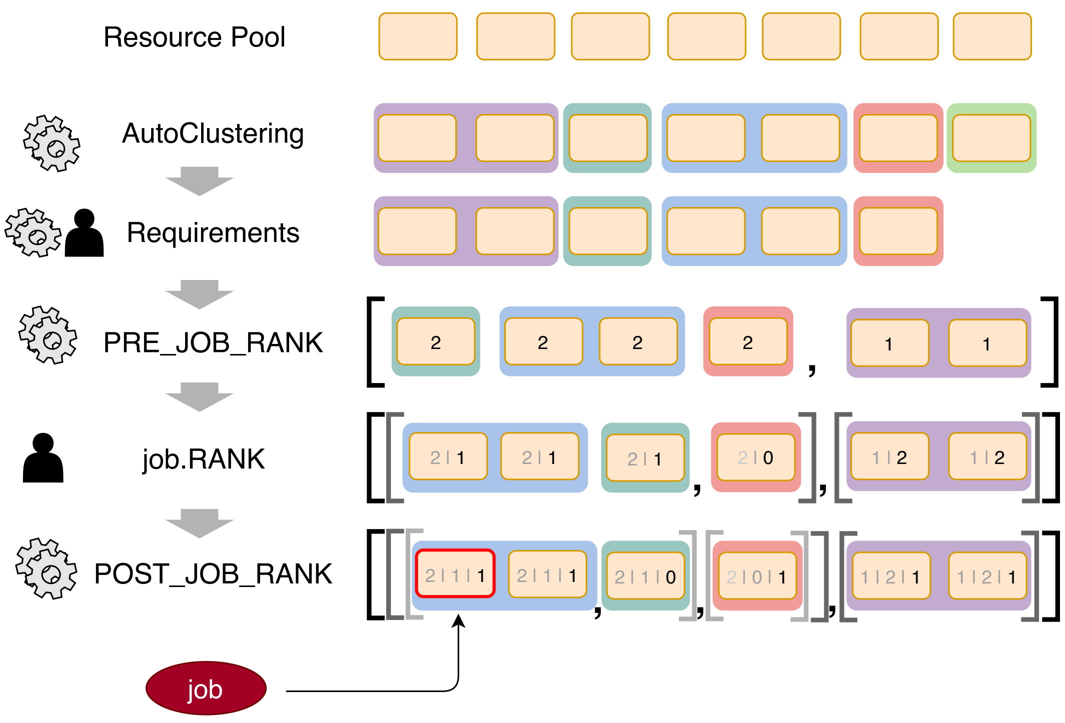


Figure 6.1: Schematic representation of the stages in a job's matchmaking process in HTCondor. The symbols on the left-hand side indicate whether the mechanism of the current stage is defined by the user that submitted the job (black user symbol) or system-wide (gray gears). In general, jobs are considered sequentially, for each job all available resources in the resource pool are considered. To improve the performance of the matchmaking processes, resources that can be considered identical in the following stages are combined into *autoclusters* in the first stage. After autoclustering both the considered job's and the resources' compelling requirements are evaluated. There are default requirements defined on system level but additional requirements can be defined by the user. After evaluating all requirements and dropping resources that do not fulfill these, the remaining resources are ranked in a three-stage process. The graphic shows an exemplary evaluation of the ranks. The current rank is shown in black while previous ranks are shown in gray. The first rank that is evaluated is the `NEGOTIATOR_PRE_JOB_RANK` or, shortened, `PRE_JOB_RANK`. It enforces the systems overall job distribution policy. The second rank, the `job.RANK` provides a possibility for the user to include preferences regarding the resources the job is processed on. Finally, the `NEGOTIATOR_POST_JOB_RANK` or `POST_JOB_RANK` is evaluated. In HTCondor's default configuration this rank prefers resources with faster CPUs to ensure that resources more suitable for faster job execution are utilized. For job execution, among the resources with the highest `PRE_JOB_RANK` the resources with the highest `job.RANK` and among them resources with the highest `POST_JOB_RANK` are chosen.

First, the NEGOTIATOR_PRE_JOB_RANK, short PRE_JOB_RANK, attribute is evaluated for every machine. Its expression is the same throughout the cluster and allows deploying general job distribution policies. For example, the default configuration in HTCondor version 8.8 [53] is:

$$\begin{aligned}\texttt{PRE\_JOB\_RANK} = {} & (10^6 \cdot \texttt{My.Rank}) \\ & + (10^5 \cdot (\texttt{RemoteOwner} =?= \texttt{UNDEFINED})) \\ & - (10^4 \cdot \texttt{Cpus}) - \texttt{Memory}.\end{aligned} \tag{6.1}$$

The weights of the summands allow for a graded relevance and separate the individual effects of terms. The highest weight is assigned to the Rank attribute of worker nodes. The default is Rank = 0 but it can be set per worker node to prefer specific matches. The second term checks whether the machine is claimed by anyone and prefers machines that are currently not running other jobs, thereby avoiding preemption. To avoid fragmentation of large chunks of free CPUs or memory, which could e.g. be used to process multicore jobs, the two remaining terms favor machines with little free CPUs or memory.

Second, the Rank attribute of the job is evaluated to distinguish between autoclusters of resources with the same PRE_JOB_RANK. The job.RANK defaults to zero but can be defined by the owner of the job. It is evaluated against the remaining autoclusters and represents the users preferences. Lastly, the NEGOTIATOR_POST_JOB_RANK or POST_JOB_RANK is applied to choose between resources that rank equally considering both the systems' general policy and the users' preferences. For example, the default rank prefers slots with faster cores over slower cores.

The job will be processed on a resource with the highest PRE_JOB_RANK, job job.RANK and POST_JOB_RANK value in lexicographical order [54]. This negotiation mechanism based on ClassAds can be utilized to coordinate jobs for distributed coordinated caching.

Data transfer and cache management: XRootD

Data transfers to jobs in HEP are commonly performed using the XRootD framework [27]. This framework was developed with focus on handling root files, a commonly used data type in HEP. It allows to create a common name space for files across multiple storage resources allowing to access files on remote as well as local storage. Additionally, it provides a plugin that can be used to manage caches with basic caching functionality by employing transparent caching as described in section 3.5.1.

6.2 LAPIS the adaptive, performant and interactive scheduling simulator

Instead of creating an entirely new simulator that fulfills the requirements discussed in section 6.1, LAPIS [55], a simulation tool developed at KIT, was extended. LAPIS, an adaptive, performant and interactive scheduling simulator, is an event based simulator written in *python3.7* and is based on the python simulation framework *μSim* [56]. It was developed to study job scheduling in heterogeneous systems featuring opportunistic resources. A schematic overview of its functionalities including the extensions made in the scope of this thesis is shown in figure 6.2. The original setup of LAPIS and the extensions are discussed in this section.

6.2.1 LAPIS base functionality

The original setup of LAPIS [57] forms the basis for the caching extension developed in this thesis. It allows to simulate the submission, scheduling and execution of jobs in a user-defined system of computational resources.

These computational resources, worker nodes, are characterized by the amount of resources they provide for the execution of jobs. The definition of what resource types are taken into account is flexible but usually CPU cores, memory and disk space are chosen. While the CPU cores differ in speed in reality and lead to different job walltimes if the same job is processed on different CPU cores, in the scope of this simulation a job's runtime is assumed to be identical on all CPU cores.

Two types of computational resources can be simulated using LAPIS: static resources and dynamic resources. While the capacities of static resources do not change throughout the simulation, the capacities of dynamic resources can change, e.g. additional worker nodes can become available or unavailable according to the current resource demand. This feature allows studying opportunistic resources that can be flexibly integrated into and removed from the system using LAPIS. Since coordination decisions are based on a system's current state, which is static for any certain time frame, only static resources are examined in the scope of this thesis. Therefore, general insights attained in these studies can be transferred to dynamic resources.

Simulated jobs are characterized by their metadata. To simulate a job, information about the job's walltime, resource requirements and submission time are necessary. The walltime determines the time frame for which the resources the job needs for processing are allocated. Resource requirements define the amount of CPU cores, memory and disk space that the job needs to be processed. They are divided into requested resources and used resources. Requested resources correspond to the amount of resources that are allocated where the job is processed while used resources refer to the resources the job actually needs. If the job's used resources exceed the requested resources, the job can be terminated. Lastly, the submission time denotes

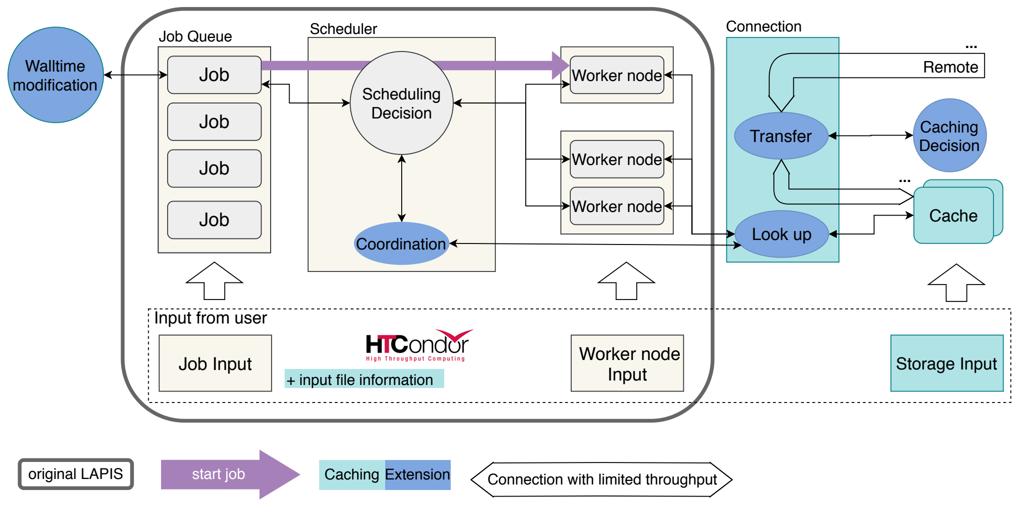


Figure 6.2: Schematic overview of the LAPIS simulator including the caching extension. Components that make up the original structure of the simulator are shown in gray while components added to simulate distributed coordinated caching are shown in blue. The main purpose of LAPIS is to simulate the distribution of jobs in a job queue across resources on which the jobs are then processed. The input characterizing both jobs and worker nodes is provided by the user. Information provided by HTCondor can be used to create this input. Jobs are characterized by their metadata containing information about a job's walltime and demand for resources such as number of CPU cores, memory and disk space. Worker nodes are characterized by the resources they provide. The matching of jobs to available worker nodes is done by the scheduling decision of the scheduler module. To simulate the execution of a job, the resources demanded by the job are allocated on the worker node for the job's walltime. When extending the simulation to take caching into account, the job's walltime needs to be modified based on the time needed to transfer data requested by the job. To simulate this the jobs' input is extended by information about input data, objects representing caches and network connections with limited throughput. There can be multiple caches with associated connections as well as multiple connections to simulate data transfers from remote storage. Data transfers are coordinated by the connection module that contains functionality to look up whether requested data is already cached and that can trigger a caching policy if data is not cached yet. To study the effects of coordination the scheduling decision is extended to include information about data locality that is provided by the connection module.

the point in time when a job is appended to the batch system's job queue. The metadata of jobs and worker nodes can be imported from HTCondor job monitoring and resource information. This allows to represent workloads and systems existing in reality without extra effort.

The scheduling process during which jobs in the job queue are matched to available worker nodes and scheduled to be processed is started in regular intervals. The length of this interval is configurable and is set to one minute as default. Jobs in the job queue are handled sequentially. For each job the worker node best suited to process the job is determined. What worker node is best suited is given by the scheduling decision. The criteria applied in this decision can be configured. Per default, worker nodes that provide exactly the amount of resources the job requested are preferred to avoid fragmentation of resources. If a suitable worker node is found the job is removed from the job queue and its requested resources are allocated for the job's walltime. If there is no well suited worker node available, the job remains in the job queue and is reconsidered in the next scheduling interval.

Information about the activity and state of jobs, worker nodes and job queue is provided by a monitoring module. Thereby, the simulation's progress can be tracked similarly to how a real system is monitored.

6.2.2 Caching Extension for LAPIS

Simulating distributed coordinated caching is not supported by the original LAPIS simulator. As part of this thesis, extensions have been designed and developed to support caching functionality. They are described in this section. In the following, the term LAPIS refers to the LAPIS simulator including the new features unless it is stated otherwise.

When effects of caching are taken into account, a job's walltime can no longer be considered to be independent of the worker node it is processed on. Instead, it depends on data locality at the worker node the job is executed on and on the current network utilization. As introduced in equation (3.1) the walltime is estimated by the maximum of the time needed to transfer data the job requested and the time that is needed to perform calculations based on this data. The latter can be estimated from the job's CPU time and calculation efficiency using equation (3.3). Since $t_{\text{data transfer}}$ potentially depends on multiple jobs running concurrently, it cannot be statically calculated but has to be simulated. To do this, several adjustments have to be made:

1. Information about requested data and CPU time needs to be included into the jobs' metadata. While the CPU time is provided by the batch system, information about the job's input data has to be acquired and provided separately.

2. Objects representing caches are included into the worker node setup. There can be multiple caches. Their size, connection bandwidth and accessibility by worker nodes are configured by the user. Caches provide a list of files that they contain and functionality to add and remove files.

3. Network connections and their usage by many jobs need to be simulated. Each shared connection is modelled as one entity with a fixed total throughput, which is equally divided between any transfers using this connection. Any transfer that attempts to use more throughput than its share is proportionally delayed. This corresponds to a throttling of the throughput provided for each transfer when network is congested.

The transfer time $t_{\text{data transfer}}$ is determined by the simulated time that passes until the transfer is complete. There can be multiple remote connections and there is one cache connection for every cache.

The simulation of a job's transfer is handled by the connection module. It provides the functionality to determine whether the requested data is provided by a cache and triggers the transfer from either a cache or remote storage. If data is not provided by the cache a configurable caching policy determines whether it will be cached. Caching new data is done transparently without an additional transfer from remote storage. A configurable cache eviction algorithm frees cache space if necessary.

The cache operation mode in which a cache has a limited size and information about the files it contains is called *file based caching.* This operation mode highly depends on the caching policy and cache eviction policy and is, therefore, well suited to study cache behavior configurations. Besides file based caching, the simulator has a second operation mode called *hitrate based caching.* This approach bypasses the influence of the cache's actual configuration. Instead of defining which data it provides, the job's information about requested data is extended by a cache hitrate h for every cache available in the system. This cache hitrate corresponds to the probability the data is cached when it is requested by the job. Using this operation mode allows to study the effects of coordination in different architectures without introducing further complexity by the choice of cache behavior as cache eviction policy and cache sizes do not have to be specified. This corresponds to assuming that caching policy, cache eviction policy and cache sizes are chosen optimally to produce the given cache hitrates.

Regardless of the operation mode the following assumptions are made for jobs with input data: First, a job's input data consists of files. Each file is read exactly once. The validity of this assumption is discussed in appendix C.1. Second, if there are multiple input files they are read sequentially. This assumption is reasonable as multiple input files in end-user data analyses usually stem from data sets that were split into multiple input files as described in section 2.4. These data subsets are then processed sequentially. Third, the total throughput of all network connections is static. Only the transfer of the jobs' input data is simulated as the jobs' execution environment is significantly smaller than the input data in the use cases observed at ETP. Fourth, if hitrate based caching is used, either all input files of a job are provided by the cache or none since a file usually can not be read from a cache and a remote storage simultaneously. This assumption remains valid even if the job has multiple input files. As these input files usually provide similar metadata to the cache eviction algorithm, they are likely to be evicted together.

Benchmark recreation using the caching functionality

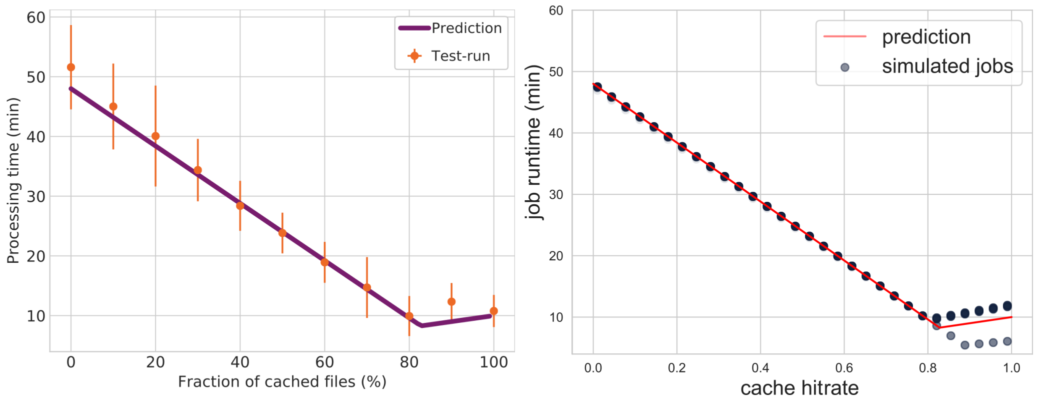
To illustrate the potential of LAPIS with caching functionality, it was used to recreate a benchmark measurement made in the production system at ETP [58]. This benchmark was initially used to demonstrate the effects of increased data locality on the walltime of jobs and to show that NaviX, introduced in section 3.5.2, is feasible.

For the original benchmark, 60 jobs with 10 input files each, were processed on a set of three worker nodes optimized for high data throughput and referred to in the following as supermachines. In total, the supermachines provide 60 CPU cores with two worker nodes providing 24 CPU cores and one providing 12 CPU cores. All worker nodes share a network connection to remote storage with a measured bandwidth of 6 GBit/s. Additionally, each supermachine is equipped with a cache made up of a 1 TB SSD and a connection to this cache providing 1.2 GBit/s. The transferred input files were of similar size and totaled to approximately 36 GB per job. Since the supermachines were idle before job submission and the jobs were submitted simultaneously it can be assumed that the jobs were processed approximately simultaneously. As the jobs only read the input files and do not perform calculations on the transferred data, according to equation (3.5) the jobs' walltime is expected to be

$$t_{\text{walltime}}(h) = \max\left(\frac{V\cdot(1-h)}{\frac{s_{\text{remote}}}{n_{\text{jobs}}}}, \frac{V\cdot h}{\frac{s_{\text{cache}}}{n_{\text{jobs}}}}\right). \tag{6.2}$$

with $V = 36\,\text{GB}$ being the average amount of transferred data per job, s_{remote} and s_{cache} denoting the total bandwidth of the network connection and cache connections respectively which are equally distributed across a number of $n_{\text{jobs}} = 60$ jobs. The cache hitrate h was varied by caching a fraction of a job's input files.

The jobs' walltime was measured for multiple h. The average values and variations are shown in figure 6.3a along with the expected walltime expressed in equation (6.2). The walltimes measured in the benchmark agree with the expectation within their variation. Multiple effects contribute to the variation in the measured job walltimes. For example, the bandwidth of remote and cache connections is not constant but can vary in time due to competing transfers and the available bandwidth is not necessarily split equally between transfers. Furthermore, while the assumption that



(a) Original benchmark taken from [58]. **(b)** Benchmark recreation using LAPIS.

Figure 6.3: Comparison between a benchmark of jobs with a large amount of input data (a) and its simulated recreation (b). In both cases the jobs' walltimes were determined depending on the fraction of files that were read from a cache instead of remote storage, also called cache hitrate. In case of the original benchmark the average walltime and its variance are shown while the spread of the measured walltimes is shown for the simulation. Up to a cache hitrate of 0.8 the simulated and measured values agree well with the prediction formulated in equation (6.2). For higher values the expectation for the walltime does not hold because the worker nodes have a differing amount of CPU cores. This effect cannot be seen in the original benchmark as the measured walltimes are subject to variations due to the real system's complexity.

jobs are scheduled simultaneously is valid for many jobs, deviations for some jobs can still occur due to the internal functioning of the batch system. Lastly, the measured walltimes are extracted from the batch systems monitoring that is not always exact, especially for jobs with short walltimes that are processed in docker containers. In this case a part of the monitoring information is provided by the container but only during the container's lifetime. The information is collected periodically and therefore has a limited resolution in time.

By gathering the necessary metadata information about the jobs and configuring the worker node setup based on the parameters discussed above, the benchmark is recreated using LAPIS. The hitrate based caching mode is used as it allows a simple setup for studying the effects of varying h, and matches the setup of the experiment. Figure 6.3b shows the simulated jobs' walltime depending on h along with the expected walltime given by equation (6.2). Instead of calculating an average value, the walltime of every job is represented by an opaque dot. Up to $h = 0.8$ the walltime values are similar as no spread can be seen but for higher values of h the simulated walltimes split in two. The reason for this is that while all cache connections have the same bandwidth the number of CPU cores on the worker nodes differs. In one case, 24 CPU cores share a cache connection. In the other case, only 12 CPU cores share the cache connection resulting in higher bandwidth per CPU core and, therefore, $t^{\text{cache}}_{\text{data transfer}}$ shortens. Therefore, the walltime is shorter when $t^{\text{cache}}_{\text{data transfer}}$ dominates t_{walltime} for jobs running on this worker node. As the expectation for the walltime, given by equation (6.2), does not distinguish between the worker nodes, it is in between the simulated walltimes.

Besides demonstrating the possibilities LAPIS provides, this benchmark recreation is exemplary for the advantages but also potential shortcomings when using simulation concepts to deepen the understanding of real systems. On the one hand, simulations allow flexible configurations and parameter scans that would be difficult to realize in reality. For example, more close-knit values for h can be chosen in the simulation while measurements in reality are limited by the composition of the actual input files. Additionally, interrelations and dependencies can be studied without distortions of other aspects of the system. However, this also means that it is challenging to simulate real systems in their full complexity, since they require very complex simulation. Nevertheless, simulations can provide sufficiently good approximations of real systems with explicitly chosen level of detail that allows reasoning about the results. For example, to study the impact of network being shared with third-party users, variations could be introduced to the simulation discussed in this section when it becomes necessary to take them into account. Additional cross-checks of the simulator can be found in appendix C.2.

6.2.3 Coordination Extension for LAPIS

Within the context of Tier 3 centers jobs are coordinated by modifying the batch systems matchmaking process. To obtain results that can be transferred from simulated to real systems, the matchmaking process needs to be simulated realistically. Therefore, a scheduler that performs this matchmaking of jobs to resources in a way that closely resembles HTCondor's negotiation process described in section 6.1.2 was implemented. When extending the matchmaking process to include information about input files provided by a cache, this information is made available using functionalities provided by the connection module.

HTCondor based scheduler

To simulate a scheduler that recreates both the functionalities and usage of the scheduling and negotiation process of HTCondor, the ClassAd mechanism described above was translated to python [59]. This allows to use ClassAd syntax to implement ranks and requirements for jobs and worker nodes. Using this, a scheduler that models the behavior of an HTCondor negotiator was created in the research group. This scheduler includes the following adjustments to keep it as simple and efficient as possible while keeping features necessary for the simulations performed in the scope of the thesis:

1. Jobs and resources are not represented by autoclusters. Autoclusters are an optimization of the lexicographic sorting of HTCondor's ranks described in section 6.1.2, which is only beneficial when most job to resource matches are exchangeable and have the same rank. Since coordination ideally matches only specific pairs of jobs and resources, this does not hold and a flat lexicographic sort is equivalent and more performant.

2. Jobs are matched only in chronological order of their submission. Submitter priorities are omitted.

3. As there is no distinction made between the speed of different worker nodes in LAPIS, the `POST_JOB_RANK` is omitted.

4. The evaluation of the worker nodes `Requirements` is done after the ranking. This does not change the outcome of the matchmaking process.

6.3 Coordination mechanisms for HTCondor

As established in chapter 3, by distributing caches throughout the computational resources of a Tier 3 center data locality can be increased. To optimally benefit from this, jobs have to be coordinated to worker nodes where the data that they request is cached. Propositions discussed in this section are focused on the functionality of HTCondor but the overall concepts and considerations can be transferred to other batch systems as well. In section 6.2.3 the HTCondor based negotiator model used for simulations in this thesis is presented. In this negotiation process, *requirements* for both jobs and worker nodes express hard conditions while *ranks* are used to express preferences.

Possibilities to modify these ranks and requirements to benefit from data locality and optimize the system's job and data throughput are discussed in this section . The performance of these possible configurations can be subsequently evaluated by simulation.

6.3.1 Ranks

In the process of choosing which of the available worker nodes is suited best to process a job, ranks are used to express preferences. To benefit from increased data locality an additional rank expression, the `CACHE_RANK`, is introduced. The definition of its expression and its placement in the negotiation process are discussed in this section.

Rank expressions

The `CACHE_RANK` is supposed to reflect how well a job could benefit from data locality at a worker node and thereby allow to compare the suitability and potential performance gain of available worker nodes. There are various possibilities to form `CACHE_RANK` expressions suited to achieve this. Three approaches are discussed in the following.

One approach is focused on data locality. The more data requested by the job is cached at a worker node, the more preferable this worker node is. This leads to a simple expression for the `CACHE_RANK` of a worker node. In a real system or a simulation in file based caching mode it is given by the amount of cached data:

$$\texttt{CACHE\_RANK} = V^{\text{cached}} = \sum_{\text{files}} V_{\text{file}}^{\text{cached}}. \tag{6.3}$$

If the hitrate based approach is used in a simulation it is given by the expectation value of the amount of cached data

$$\texttt{CACHE\_RANK} = \sum_{\text{files}} h_{\text{file}} \cdot V_{\text{file}} \tag{6.4}$$

with $h(\text{file})$ being the files' cache hitrate.

Besides the amount of cached data, the bandwidth of the connection between cache and worker node affects the duration of the data transfer and thereby the job's performance. The bandwidth available for the transfer is not known precisely as it depends on the whole system. Nevertheless, the fraction of the theoretical bandwidth of the cache connection s_C^{theo} and the number of CPU cores that can process jobs that read from the cache, n_{cores}, yields an estimate of the bandwidth that theoretically can be provided for a job. This can be used to compare potential performance gains between worker nodes:

$$\texttt{CACHE\_RANK} = \frac{s_C^{\text{theo}}}{n_{\text{cores}}}. \tag{6.5}$$

In this context, the theoretical or design value for the bandwidth of the cache connection s_C^{theo} is used as the real bandwidth can differ due to other transfers outside the scope of caching. Hence, worker nodes with fast connections to the cache are preferred.

If too many jobs read from a cache simultaneously, bottlenecks in the cache connection bandwidth can compromise job performance as seen in figure 3.5. Therefore, the demand on a cache needs to be balanced and coordinating jobs to worker nodes with fully utilized or congested cache connections has to be avoided. A metric to represent the current utilization of the cache is the amount of data transfers $n_{\text{transfers}}$

that are currently using it. In simulation, this quantity corresponds to the pipes' throttling factor τ resulting in

$$\texttt{CACHE\_RANK} = \tau = \frac{1}{n_{\text{transfers}}}. \tag{6.6}$$

with $n_{\text{transfers}}$ corresponding to the number of jobs that are currently reading from the cache.

In real systems it is challenging to provide advanced information about the cache connection to the batch system and use it there. To still avoid bottlenecks, other interrelations need to be used which are easier to observe but indirectly connected. For example, the correlation between a job's CPU efficiency and its average data throughput as discussed in [60] could be exploited. If a job is reading from the cache its data throughput is limited by the current demand on the connection to the cache. Therefore, the CPU efficiencies of jobs that are reading from a cache give an indication about the current demand on the cache. Determining how this could integrate into a `CACHE_RANK` expression is beyond the scope of this thesis but is an interesting topic for future studies.

To form a `CACHE_RANK` that takes all discussed aspects into account, the individual `CACHE_RANK` expressions are combined.

Rank placement

Besides the definition of the `CACHE_RANK` its influence is determined by the rank's placement in the negotiation process of HTCondor. As discussed in section 6.1.2, worker nodes are ranked by the `PRE_JOB_RANK`, `job.RANK` and `POST_JOB_RANK` consecutively. For the presented simulation the `POST_JOB_RANK` is dropped. A `CACHE_RANK` expression could therefore be added to one or both of the remaining ranks.

On the one hand, extending the expression of the `job.RANK` seems suitable as in many systems the `job.RANK` is not set by default. Therefore, this rank can be modified without interfering with its original purpose. On the other hand, the cache rank's influence is limited as it only allows to differ between worker nodes with the highest `PRE_JOB_RANK`. To avoid this, a `CACHE_RANK` expression can be added to the `PRE_JOB_RANK`. This seems intuitive as the `PRE_JOB_RANK` enforces the system's overall job distribution policy. Preferring worker nodes where requested data is already cached, affects the job distribution and therefore should be part of the job distribution policy. However, the default `PRE_JOB_RANK` as given in equation (6.1) is complex making suitable integration and scaling of `CACHE_RANK` expressions a challenging task.

Apparently, the placement of the `CACHE_RANK` greatly affects the system's functionality. Furthermore, the `CACHE_RANK`'s placement depends on its definition and

		job.RANK	
		0	CACHE_RANK
PRE_JOB_RANK	0	●	●
	0 + CACHE_RANK	–	–
	default	●	●
	default + CACHE_RANK	●	●

Table 6.1: Overview of possible CACHE_RANK placements. Scenarios that should be studied are marked by a black dot. Two scenarios are marked with dashes. They are redundant since they are identical to the scenario in the upper right cell of the table. If the default PRE_JOB_RANK expression is not used, it does not matter to which the cache rank is added. The CACHE_RANK can be combined with the PRE_JOB_RANK, the job.RANK or both. For every scenario a reference scenario without CACHE_RANK has to be studied. Additionally, the system with PRE_JOB_RANK = 0 should be studied.

is difficult to choose based on initial considerations only. Instead, a determination process for suitable CACHE_RANK expressions and placements is using simulations. A system without CACHE_RANK is thereby included as reference. An overview of CACHE_RANK placements that should be studied is shown in table 6.1. Black dots mark distinct scenarios while dashes mark redundant scenarios.

6.3.2 Requirements

During matchmaking the requirements of job and worker nodes are evaluated. If they are not fulfilled, the worker nodes are not considered in the matchmaking of the job. While the main efforts to optimally benefit from increased data locality are focused on ranks, extending the jobs' requirement by a CACHE_REQUIREMENT could be beneficial: If the computational resources are highly utilized only few worker nodes accept new jobs in every scheduling interval and the probability that a worker node with connection to a cache containing a jobs requested data is small. In such a case, a rank can only chose between several unsuited resources. Therefore, requirements could be introduced to delay the matching until a more suitable resource becomes available where the job would benefit from caching. The behavior for other jobs should remain unchanged by this.

However, the scheduling of jobs should not be held back indefinitely for two reasons. First, if the amount of time that resources are idle is longer than the speed up gains from jobs that are processed on a worker node with cached data, the systems overall job throughput decreases. As the purpose of introducing distributed coordinated caching is to increase the system's job throughput and not to optimize the performance of isolated jobs this should be avoided. Second, the time a job spends waiting in the job queue before being processed should be small compared to the walltime. Delaying the processing of a job until a specific resource becomes available can delay the completion of the jobs' workflow, preventing timely job completion that is expected by users. Therefore, requirements need to allow the job to be scheduled and executed normally if no well-suited worker node is found in a reasonable time frame. The definition of a reasonable time frame is derived from the job's expected walltime, since it is the only timescale known ahead of running the job. For example, if a job usually has an expected walltime of twelve hours it is beneficial to let it wait for suitable worker nodes for one hour if this allows to decrease the walltime by about an hour or more and reduce the demand on the network connection in the process, from which jobs running in parallel benefit as well.

A requirement expression chosen to avoid excessive waiting times is

$$\begin{aligned}\texttt{CACHE\_REQUIREMENT} = \texttt{IfThenElse}(&V^{\text{input}},\\ &V^{\text{cached}}\ \texttt{or}\ (n_{\text{cycles}} > N_{\text{lim}}\ \texttt{or}\ (t - t_{\text{sub}}) > t_{\text{lim}}),\\ &\texttt{True}).\end{aligned} \tag{6.7}$$

This requirement enforces that if the job requests input data $\left(V^{\text{input}} > 0\right)$ the job is not scheduled to any worker node unless it provides cached data during the first N_{lim} scheduling cycles. After N_{lim} scheduling cycles or if t_{lim} has passed since the job's submission time t_{sub} the job is scheduled normally. The possibilities to implement requirements like this depend on available metadata and require a suitable choice for thresholds such as N_{lim} or t_{lim}. To chose beneficial requirements the performance of possible implementations can be studied using simulations.

6.4 Simulation setup

The simulations performed in the scope of this thesis are focused on studying coordination mechanisms. To reduce dependencies on other aspects of distributed coordinated caching, coordination has to be studied independently of the caches' behavior and the examined system has to be kept simple and comprehensible. The resulting simulation setup and the assumptions it is based on are described in this section.

6.4.1 Caching mode and data transfer

To study coordination mechanisms, hitrate based caching is chosen as simulation mode. Additionally, it is assumed that the chosen caching policy suppresses file replication. This means that when a job is processed on a worker node with no access to the job's cached data no caching policy is invoked and no additional copy is produced. These seemingly hard constraints chosen for the studied scenario provide a controlled and comparable caching mode.

Using this caching mode, the performance of different coordination mechanisms can be studied depending on h. Similarly, the cache's behavior can be studied independently to determine how certain hitrates h can be provided without taking coordination into account. After studying both aspects separately the results can be combined and the interrelation between coordination and cache behavior can be studied to achieve a more complete understanding of distributed coordinated caching.

For data transfers, the assumptions described in section 6.2.2 apply. Indeed, it is assumed that input data dominate the amount of data transferred during a job's processing and therefore only these transfers are simulated. Besides, all input files are read exactly once and if there are multiple input files they are transferred sequentially. Additionally, either all input files requested by a job are cached or none. The probability for a job's input files to be cached, h_{job}, is calculated as a sum of the distinct files' cache hitrates h_{file} weighted with the files' sizes V_{file}:

$$h_{\text{job}} = \frac{\sum_{\text{files}} h_{\text{file}} \cdot V_{\text{file}}}{\sum_{\text{files}} V_{\text{file}}}. \tag{6.8}$$

Lastly, the bandwidth of both cache connections and remote connections is constant omitting transfers besides the requested data.

6.4.2 Resources

The chosen setup of worker nodes specifically selected for these studies is designed for simplicity and comprehensibility. Caches and remote storage are characterized by the corresponding connection bandwidths. To simplify the matchmaking process and to avoid distortions in the system's utilization that could overlap with effects of job coordination it is assumed that all worker nodes provide enough memory and disk space for a job if the job fits the number of CPU cores available on the worker node. Information about required memory and disk space is therefore neglected for worker nodes and jobs. This is justified as additional memory and disk space can be upgraded to necessary amounts.

While complex systems such as the infrastructure of computational resources at ETP can be simulated using LAPIS, an idealized and uniform setup is chosen for this simulation to exclude effects unrelated to caching and coordination. It consists of a group of worker nodes with access to caches, and a separate group of worker nodes without access to caches, from now on referred to as *caching cluster* and *basic cluster* respectively. Both clusters consist of worker nodes providing 24 CPU cores each in order to avoid bias from the default `PRE_JOB_RANK` preferring worker nodes with fewer available CPUs.

The caching cluster's infrastructure is modelled according to a cluster at ETP that is designed to process jobs with high data throughput. The caching cluster is made up of three worker nodes with a total of 72 CPU cores and a dedicated or a shared cache setup. The bandwidths of remote and cache connections are based on this cluster's respective bandwidths as measured in [58]. Consequently, the worker nodes share a remote connection providing a bandwidth of 6 GBit/s. In the shared cache setup the worker nodes share one connection to the cache providing 4.8 GBit/s, while in the dedicated cache setup the connection between every worker node and its associated cache provides 1.2 GBit/s.

The walltime of jobs executed on the caching cluster is adapted as described in section 6.2.2. While the size of the caching cluster is constant, the size of the basic cluster is varied to simulate different degrees of system utilization. The amount of CPU cores $n_{\mathrm{CPU}}^{\mathrm{basic}}$ necessary to simulate a fully utilized system is given by

$$n_{\mathrm{CPU}}^{\mathrm{basic}} = n_{\mathrm{CPU}}^{\mathrm{total}} - n_{\mathrm{CPU}}^{\mathrm{caching}} = \frac{\sum_{\mathrm{jobs}} t_{\mathrm{walltime}}}{\Delta t_{\mathrm{sim}}} - n_{\mathrm{CPU}}^{\mathrm{caching}} \tag{6.9}$$

with $\sum_{\mathrm{jobs}} t_{\mathrm{walltime}}$ being the sum of the original walltimes of all jobs, Δt_{sim} being the desired simulated time frame and $n_{\mathrm{CPU}}^{\mathrm{caching}} = 72$ being the number of worker nodes in the caching cluster. The size of the basic cluster in simulations of less utilized or overloaded systems is derived from $n_{\mathrm{CPU}}^{\mathrm{basic}}$. As the basic cluster does not provide caching functionality, the walltime of jobs processed on this cluster is not modified.

6.4.3 Job input

Besides the cache's behavior and the resource setup, the workload of jobs processed during the simulation is important as it has a large impact on the behavior of the simulated system. As discussed in section 6.2.1, jobs are simulated based on their metadata extracted from the batch system's monitoring information. To create a job input representative for the workload of the system aimed to optimize, jobs that were executed at ETP are used to create the simulation's job input. The information about such jobs is taken from the monitoring described in chapter 4. Taking the discussion of the validity of the monitoring data in section 4.1.2 into account, a pool of jobs that were completed during a period of eleven weeks between 20.11.2019 and 05.02.2020 is used to create the simulation's job input.

The resulting job input has to fulfill a set of requirements. First, the runtime of the simulation processing the job input has to be reasonable. This constrains the amount of simulated jobs and thereby the simulated time for a given amount of resources. For this simulation, the simulated time Δt_{sim} is chosen to be one week. Second, it has to be ensured that the job input only contains jobs with valid monitoring data. Lastly, the job input needs to represent the workload at ETP to enable a realistic comparability to a typically utilized cluster. This includes that the method used to create the job input for one week of simulated time yields similar results if executed multiple times. In particular, the number of jobs, number of jobs processing input data and processed data as well as the workflow composition should be similar in every generated job input. Additionally, the method has to preserve temporal interrelations between jobs that are submitted in the same time frame. The method chosen to achieve this and the resulting job input samples are described in the following.

Job selection

To create a job input sample suitable for simulation and representative for the system we want to simulate, appropriate selections are necessary. In addition to the selection criteria applied to the monitored jobs described in section 4.1.2, jobs are omitted if the size of their input files could not be reconstructed. Removing jobs requesting than one CPU is especially important as in a full system slots with multiple cores might only infrequently be available distorting the simulation duration or resource utilization. In real systems this problem is solved by periodically draining the system or by allocating worker nodes for the execution

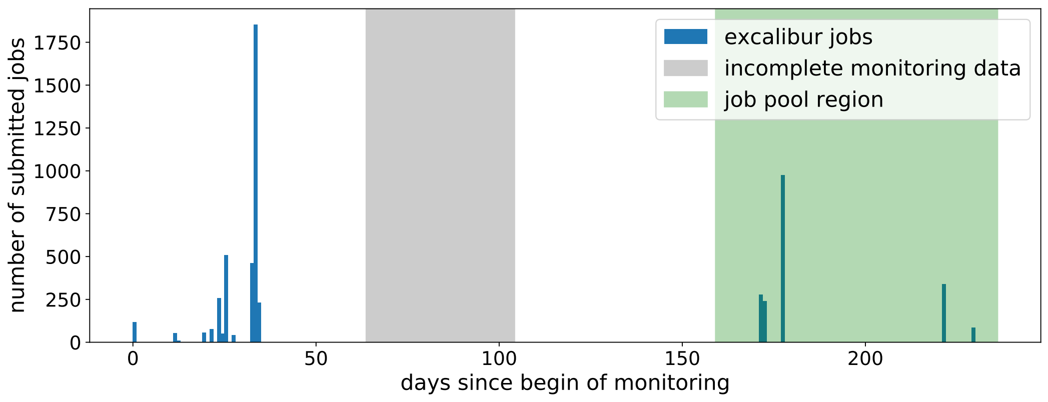


Figure 6.4: Histogram of the submission timestamps in days since the monitoring's start of jobs from the jet energy calibration workflow during the monitored time period. The time range in which the monitoring was inactive for jobs of this workflow is marked in gray, the time range from which the job input for simulations is sampled is marked green.

of these jobs but these methods are not studied in the scope of the simulation setup used in this thesis.

During the chosen monitoring phase no updates on absolute residual corrections of the jet energy calibration in CMS were requested or made. Therefore, only few jobs of the jet energy calibration workflow were monitored in this period. But as discussed in chapter 4 the jet energy calibration workflow at ETP is well suited for caching and therefore should be part of the job input sample of the simulation.

Figure 6.4 depicts the occurrence of jobs of the jet energy calibration workflow throughout the monitored time period. As there are jobs of this workflow earlier in the monitoring where the monitoring data for this workflow are valid as well, these additional jobs are integrated into the job pool from which the simulations job input sample is created. The jobs are inserted four times to ensure that the resulting job input sample contains about as many jobs as processed when updating the absolute residual corrections. To preserve the chronological connection of the jobs' submission time the sequence of submission times is shifted into the job pool time frame by adding a randomized constant for each of the four replications. The resulting distribution of jet energy calibration jobs in the job pool is shown in figure 6.5.

Job resampling method

To create a job input sample corresponding to one week of simulated time an appropriate amount of jobs is sampled from the job pool. Randomly selecting jobs is suited well as it reduces the effects of time-dependence in the research group's activity such as day-night cycles, weekends or holidays [28]. This allows to create a job input sample with a uniform workload that is suitable for examining changes in the system's utilization. As the simula-

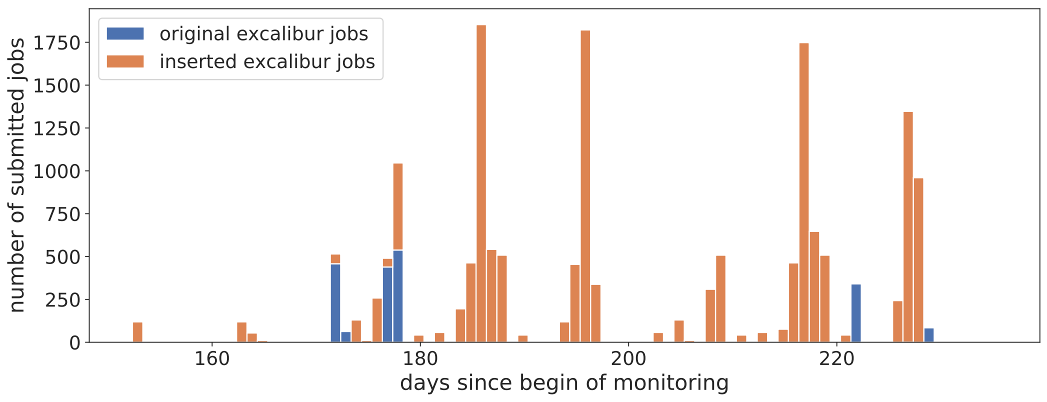


Figure 6.5: Occurrence of jobs from the jet energy calibration workflow after integrating jobs from earlier monitoring phases. The amount of jobs is plotted against the job's submission time stamp in days since the start of the monitoring. Jobs that were monitored in this time range are shown in blue. To increase their occurrence, a cluster of jet energy calibration from earlier monitoring phases is replicated four times into the depicted time frame, shown in orange.

tion's time frame is chosen to be smaller than the time frame the jobs are sampled from, multiple job input samples can be created and compared.

However, in reality as shown in figure 6.6, there is an interrelation between jobs that are submitted together. For example, if the processing of a data set or a workflow is split into multiple jobs these jobs have similar properties such as being well-suited for caching and are submitted simultaneously or within a short interval. This interrelation needs to be preserved to properly represent the system. This makes random selection of jobs an unsuitable technique to create the job input sample.

Nevertheless, a method that respects these interrelations while retaining the benefits from randomly sampling jobs can be found. Instead of randomly sampling individual jobs the simulated time period $\mathit{\Delta} t_{\text{sim}}$ is split into small time windows of duration $\mathit{\Delta} t_{\text{sample}}$. Each of i time windows is filled by randomly selecting a timestamp t_i in the time range covered by the job pool and selecting all jobs that were submitted within the interval $\mathit{\Delta} t_i$ given by

$$\mathit{\Delta} t_i = \left[\left(t_i - \frac{\mathit{\Delta} t_{\text{sample}}}{2} \right), \left(t_i + \frac{\mathit{\Delta} t_{\text{sample}}}{2} \right) \right]. \tag{6.10}$$

The submission timestamps of the selected jobs are then shifted to cover time window i. For this simulation $\mathit{\Delta} t_{\text{sample}} = 120\,\text{s}$ is chosen as the amount of jobs submitted in larger intervals decreases as shown in figure 6.6.

Cross-Checks

To check the stability of the method described in the previous section, 30 different job input samples are created and their composition is examined to ensure the job input samples represent the job pool and that the job input samples are similar.

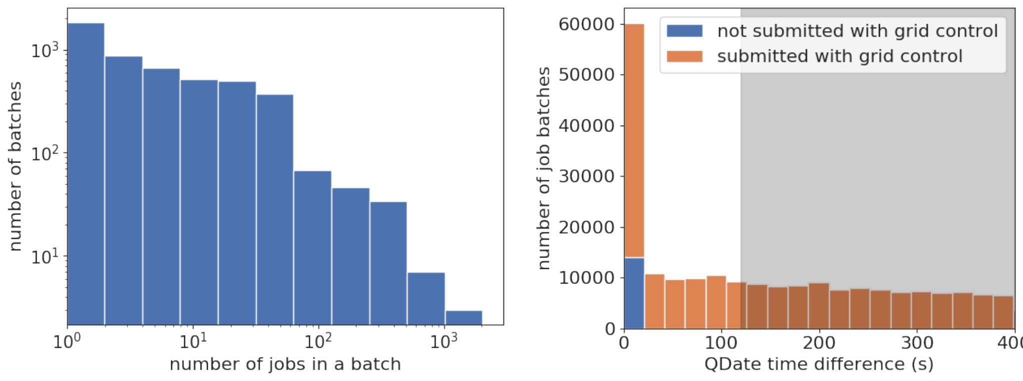


Figure 6.6: Examination of the chronological connection in job submission to determine whether jobs that are submitted in a short interval are similar. Jobs are considered to be similar or part of the same batch if they were submitted by the same user and if the script they are executing and the `JobBatchName` identifier is the same. The plot on the left shows the number of jobs in one batch. It indicates that there are many batches containing multiple jobs. To determine whether the jobs were submitted simultaneously, for every job in a batch except the one with the earliest submission date (QDate) the difference between this job and the first job's submission date is calculated. The plot on the right shows a histogram of these time differences. It shows a peak in the first bin indicating that many of the jobs in a batch were submitted simultaneously or within a few seconds. Besides the peak, there is a uniform tail across a wide range of time differences. This tail can be traced back to the resubmission of failed jobs that are done by grid control [39], the tool used for automated submission of jobs at ETP. The area shaded in gray indicates that in the following, a time window of $\Delta t_{\text{sample}} = 120\,\text{s}$ is chosen to select jobs that were submitted together.

First, the distribution of the number of jobs in a job input sample is compared to the job pool's average number of jobs in one week. This distribution is shown in figure 6.7 along with the distribution of the number of jobs with input files and the number of jobs without input files as well as the respective average values. For all three quantities the distributions are scattered around the average value in the same order of magnitude. The spread is smaller for the number of jobs with input files than for the number of jobs without input files. Nevertheless, in all three cases the number of jobs can be considered stable.

Second, the workflow composition of the job input samples is examined. The distribution of the number of jobs belonging to the workflows processing input data is shown in figure 6.8. The definition and identification of the five workflows are discussed in chapter 4. While the order of magnitude of the number of jobs differs between workflows, the number of jobs in different job input sample is similar.

Lastly, the distribution of the total amount of transferred data i.e. the sum of the input file size of all jobs in a job input sample is shown in figure 6.9. The amount of transferred data is scattered between 30 and 60 TB. This scattering range is expected considering that

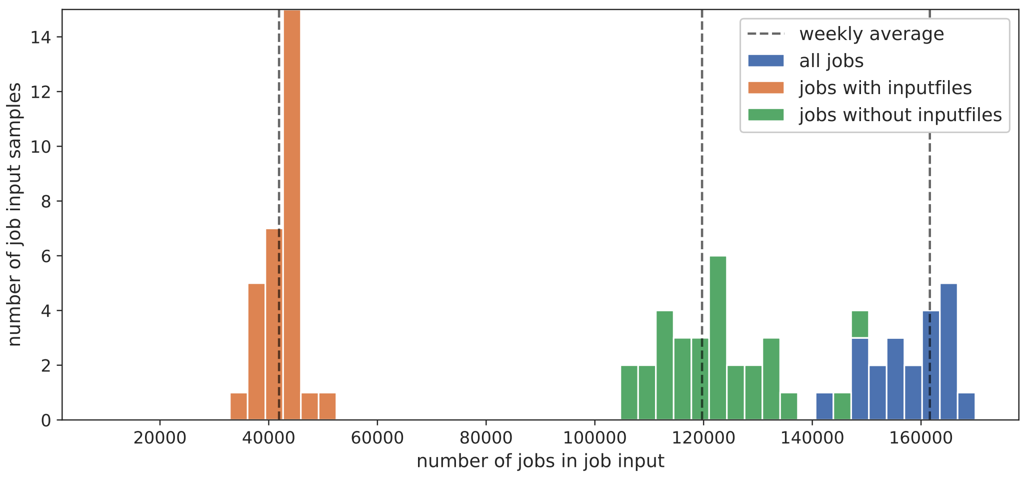


Figure 6.7: Distributions of the total number of jobs, the number of jobs processing input files and the number of jobs that are not processing input files in 30 different job input samples. All three quantities are of the same order of magnitude and spread around the respective average value.

the size of a job's input files varies between workflows and between jobs of a workflow but is similar for jobs submitted at the same time. Nevertheless, the amount of transferred data of different job input samples is of the same order of magnitude. Further cross-check concerning the amount of input data and its distribution among workflows is discussed in appendix C.3.

Summarizing, the method outlined in section 6.4.3 produces similar job input sample with relevant quantities being of similar or of the same orders of magnitude. Therefore, general relations can be determined by means of simulation using only one job input sample as job input in the simulations. However, multiple job input samples that cover different job compositions have to be used to examine more detailed relations.

6.5 Simulation and evaluation

Based on the coordination mechanisms discussed in section 6.3 the simulation setup described in section 6.4 is used to study the fundamental principles of coordination and to show that they are compatible with the expectations. Besides, the procedure provides a proof of concept for simulation based studies of distributed coordinated caching. In this section, the simulated scenarios and the derived results are described in detail.

6.5.1 Scenarios

After defining the simulation setup and its assumptions, some parameter choices or configuration details remain undefined. These specify the simulation scenario and include the

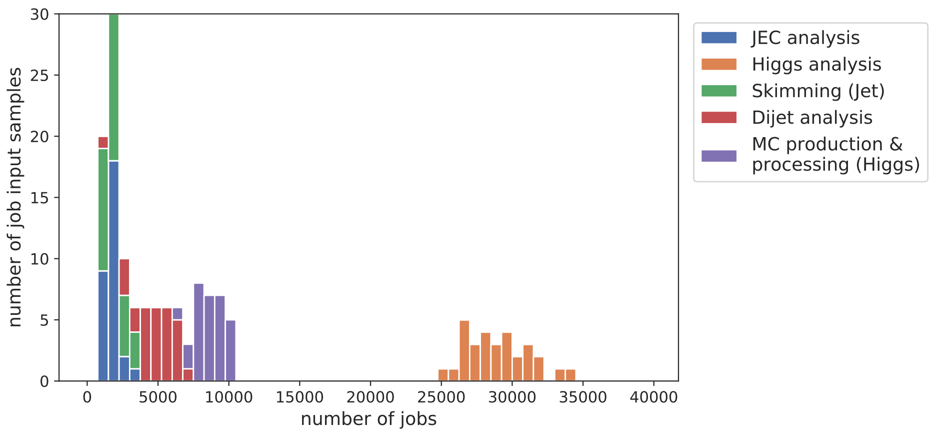


Figure 6.8: Workflow composition of 30 different job input samples. While the number of jobs differs between workflows the number of jobs of one workflow is of the same order of magnitude for different job input samples.

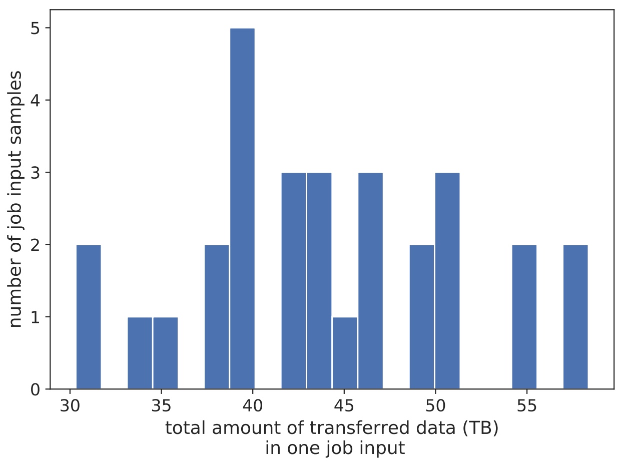


Figure 6.9: Distribution of the total amount of transferred data of 30 different job input samples. While being scattered, the amount of transferred data remains in the same order of magnitude.

ID	number of jobs	CPUh	data volume (TB)	$n_{\text{CPU}}^{\text{total}}$
1	76446	81113	48.36	480
2	76651	87837	55.88	522
3	76954	89463	42.07	532

Table 6.2: Properties of the job input samples used as job input in simulations including the number of jobs, the jobs' cumulative walltime in CPUh and the cumulative transferred data volume in TB. $n_{\text{CPU}}^{\text{total}}$ corresponds to the number of CPUs necessary to provide enough CPUh to process all jobs within one week and is calculated using equation (6.9).

choice of job input, the number of worker nodes in the simulated system, the chosen cache setup, the cache hitrates of jobs and the expressions for `PRE_JOB_RANK` and `job.RANK`.

Job inputs and resource size

As the properties of the job input samples such as the amount of transferred data differ, multiple job input samples are simulated to estimate how different workloads affect the performance of different coordination mechanisms. Thus, a simulation for three different job input samples is presented. To make these differences more visible, the fraction of jobs with input files is increased by removing 75 % of the jobs without input files from the original job inputs. The resulting job inputs' characteristics such as the number of jobs, the sum over the walltime of all jobs and the data volume of all input files are listed in table 6.2. Additionally, the number of CPU cores necessary to process all jobs in one week, $n_{\text{CPU}}^{\text{total}}$ of every job input sample, is listed there. As all job input samples are processed in the same setup of worker nodes, the number of CPU cores necessary to simulate a fully utilized system for job input sample 1, $n_{\text{CPU}}^{\text{total}} = 480$, is used as a size of the fully utilized cluster in the following simulations.

Alongside the fully utilized system, an overcrowded, and two underutilized systems are simulated to examine the influence of system utilization on the performance of caching and coordination. The corresponding cluster sizes are defined in relation to the fully utilized system and are listed in table 6.3. Hence, three job input samples for four cluster sizes result in twelve different scenarios.

The calculation efficiency is chosen globally to be $\epsilon_{\text{calculation}} = 0.99$ based on the experience that jobs usually do not achieve maximum efficiency. For some workflows the calculation efficiency might be smaller in reality. But as this simulation is focused on the differences in the performance of coordination methods, overestimating $\epsilon_{\text{calculation}}$ is less harmful than underestimating it because overestimating $\epsilon_{\text{calculation}}$ corresponds to making the job's walltime more susceptible to the amount of cached data.

Cache setup and cache hitrate

In all scenarios a shared cache setup is examined. The setup consists of one cache that is connected to all three worker nodes supporting caching functionality. The cache connection's

relative size	100 %	80 %	150 %	200 %
$n_{\text{CPU}}^{\text{total}}$	480	384	720	960
$n_{\text{CPU}}^{\text{caching}}$	72	72	72	72
$n_{\text{CPU}}^{\text{basic}}$	408	312	648	888

Table 6.3: Composition and total number of CPU cores of the different systems of worker nodes simulated in the following. To simulate overly utilized, fully utilized and underutilized systems, four different cluster sizes are chosen relatively to the fully utilized systems size $n_{\text{CPU}}^{\text{total}} = 480$ that is calculated from the job input samples' properties using equation (6.9). As the size of the caching cluster, $n_{\text{CPU}}^{\text{caching}}$, remains constant only the basic cluster's size, $n_{\text{CPU}}^{\text{basic}}$, compensates for changes in the total number of CPU cores in the system.

bandwidth of 4.8 GBit/s remains constant during the simulation. For this simulation, the cache hitrate h_{file} is assumed to be identical and constant for files belonging to jobs of workflows that are likely to benefit from caching as established in section 4.2.3, i.e. the Jet energy calibration and Higgs analysis workflow. This cache hitrate is denoted h_{caching}. For the input files of all other workflows the cache hitrate is denoted $h_{\text{file}}^{\text{other}}$ and is constant as well. By modifying both cache hitrates the amount of data provided by the cache during the simulation can be varied. For the majority of the following simulations $h_{\text{file}}^{\text{caching}} = 1$ and $h_{\text{file}}^{\text{other}} = 0$ are chosen as these simulations provide a proof of concept for the mechanics of coordination mechanisms. This choice of cache hitrates corresponds to an ideal cache that provides all input files of workflows well-suited for caching and avoids blocking cache space by storing files of other workflows.

Coordination scenarios

With a shared cache setup consisting of one cache, the `CACHE_RANK` expressions proposed in section 6.3.1 can be simplified. If there is only one cache, the `CACHE_RANK` expression in equation (6.5) that is supposed to assess the bandwidths of different caches becomes redundant. Similarly, comparing the occupancy of individual caches as enforced by the `CACHE_RANK` expression in equation (6.6) becomes superfluous. In this scenario, both simplify to an indicator whether a resource is connected to a cache or not. Nevertheless, a condition based on the throttling factor τ of the cache connection can be used to avoid cache limitations by not preferring the caching cluster if more than n jobs are currently reading from the cache. In a shared cache setup, the `CACHE_RANK` expression of equation (6.4) can still be used to coordinate jobs with input files that can be provided by the cache to the worker nodes of the caching cluster.

Including the reference scenario without CACHE_RANK expression this results in three CACHE_RANK expressions studied in the following simulations:

$$1.\quad \texttt{CACHE\_RANK} = 0 \tag{6.11}$$

$$2.\quad \texttt{CACHE\_RANK} = \sum_{\text{files}} h_{\text{file}} \cdot V_{\text{file}} \tag{6.12}$$

$$3.\quad \texttt{CACHE\_RANK} = \sum_{\text{files}} h_{\text{file}} \cdot V_{\text{file}} \cdot (\frac{1}{\tau} < n), \tag{6.13}$$

$$n \in \{0.25, 0.5, 0.75\} \cdot n_{\text{CPU}}^{\text{caching}}.$$

Theoretically, there are two possibilities to place the CACHE_RANK in the negotiator process. However, the current implementation of the HTCondor based scheduler used in LAPIS does not support to make information about the job such as the amount of cached data available to the PRE_JOB_RANK. Therefore, the CACHE_RANK expression cannot be added to the PRE_JOB_RANK resulting in two possible expressions for the PRE_JOB_RANK: 1) no PRE_JOB_RANK to study the system's behavior with only the CACHE_RANK; and 2) the default implementation. As there is no machine Rank set, only available slots are included and the worker nodes' memory is not taken into account, therefore, the PRE_JOB_RANK expressions of equation (6.1) become

$$1.\quad \texttt{PRE\_JOB\_RANK} = 0 \tag{6.14}$$

$$2.\quad \texttt{PRE\_JOB\_RANK} = 10^5 - 10^4 \cdot \texttt{Cpus} \tag{6.15}$$

where Cpus is the number of unoccupied CPU cores of the worker node. As the influence of requirements is left to be subject of further research, the discussed expressions for CACHE_RANK and PRE_JOB_RANK result in six different coordination scenarios that are studied in the following.

The simulation presented in this thesis is supposed to provide a proof of concept demonstrating the principles of coordination. To examine the effects of caching and coordination, the following analysis strategy is chosen: First, the system is simulated without any coordination and without the default PRE_JOB_RANK implementation. This allows to study the general impact of caching on the system because even without coordination some jobs with input files will be processed on the caching cluster and, therefore, benefit from caching. Avoiding prioritization by the PRE_JOB_RANK allows to perform functionality cross-checks that require a random distribution of jobs. Second, the effect of coordination without PRE_JOB_RANK is studied. This is equivalent to treating data locality with the highest priority and suitable usage of the cache with the highest priority. Additionally, excluding the PRE_JOB_RANK allows to study the CACHE_RANK's full effect and potential. Besides, assessing the performance of different CACHE_RANK expressions becomes easier without distorting effects introduced by the PRE_JOB_RANK's earlier selection. Lastly, after examining the system without the PRE_JOB_RANK, the default PRE_JOB_RANK is used to study how it affects the CACHE_RANK's impact. This allows to determine whether using the CACHE_RANK as job.RANK alone allows to coordinate jobs.

6.5.2 Results

The simulation scenarios described in section 6.5.1 were processed on the Tier 3 center at ETP. The goal of creating a simulation setup that allows for reasonable runtimes was achieved as the runtime of one weeks worth of simulated time was of the order of minutes or, at most, a few hours. This section details the analysis of the performed simulations. To this end, metrics suited to quantify the effect of both caching and coordination are introduced and the results for the simulation scenarios described above are discussed.

Metrics

There are multiple quantities that can be used to compare the performance of coordination mechanisms implemented in the simulation scenarios. These metrics can be divided into metrics suitable to compare between simulations that process the same job input and metrics that allow comparing the performance on different job input samples.

In case of the former, the number of jobs processed on the caching cluster can indicate job acceleration due to the caching functionality. The performance of a coordination mechanism can be assessed by comparing the number of jobs that actually used the cache, that is the number of cache hits, as this number increases when jobs are coordinated to the cache. Therefore, a higher number of cache hits indicates that the coordination process succeeded frequently. For the same reason, the amount of data that is actually transferred using the cache connection is a good indicator as well.

Metrics that allow comparing multiple job input samples and worker node setups are more complex as they have to stay representative for varying job compositions and potentials of cache utilization. A metric based on the number of cache hits is given by

$$\epsilon_{\text{cachehits}} = \frac{n^{\text{scenario}}_{\text{cache hits}} - n^{\text{no coordination}}_{\text{cache hits}}}{E(n_{\text{cacheable jobs}})}. \tag{6.16}$$

In this case, the performance of a coordination scenario is quantified by determining the gain in cache hits compared to no coordination $n^{\text{scenario}}_{\text{cache hits}} - n^{\text{no coordination}}_{\text{cache hits}}$. The excess in cache hits is then divided by the job input samples expectation value of number of jobs that could use caching $E(n_{\text{cacheable jobs}}) = \sum_{\text{jobs}} h_{\text{job}}$. This compensates for the differences in numbers of jobs that could actually use caching between job input samples. Analogous to this metric, a metric based on the amount of data read from the cache can be defined.

In the following, the number of cache hits will be used to quantify the impact of coordination as the focus is on the performance of coordination mechanisms within the job input samples. Discussions of the impact of coordination using other metrics can be found in appendix C.4.

No coordination: cross-checks & effects of caching

Before examining the impact of coordination mechanisms, the system's behavior without coordination has to be inspected and understood. In this scenario, the `job.RANK` and the `PRE_JOB_RANK` are omitted resulting in jobs being distributed randomly among available worker nodes.

To verify that the jobs are indeed distributed randomly, the number of jobs processed on the caching cluster, $n_{\text{jobs}}^{\text{caching}}$, is examined. Neglecting the effects of caching, the number of jobs that ran on the caching cluster is expected to be

$$n_{\text{jobs}}^{\text{caching}} = n_{\text{jobs}}^{\text{total}} \cdot \frac{n_{\text{CPU}}^{\text{caching}}}{n_{\text{CPU}}^{\text{total}}} \tag{6.17}$$

with $n_{\text{jobs}}^{\text{total}}$ being the total number of jobs in the job input sample. Figure 6.10 shows $n_{\text{jobs}}^{\text{caching}}$ in blue for the inspected cluster sizes and job input sample 1, defined in table 6.2. In addition, the expected number of jobs on the caching cluster, given by equation (6.17), is marked by a horizontal line. The number of jobs processed on the caching cluster decreases with increasing cluster size since the size of the caching cluster remains constant and jobs are distributed across unoccupied worker nodes randomly following a uniform distribution. However, for all cluster sizes the observed number of jobs processed on the caching cluster exceeds the expectation. The same applies when the other job input samples are used. The respective plots can be found in appendix C.4.

As the simulation's time frame and job input are fixed, observing more jobs on the caching cluster than expected implies that many jobs with a short walltime were processed there. There are two ways to explain this. On the one hand, the excess could be an effect of the intrinsic structure of the job input sample resulting in many jobs with a short walltime being assigned to worker nodes of the caching cluster. On the other hand, this could be a manifestation of the impact of caching on the system. Even without coordination some jobs of workflows well suited for caching are processed on the caching cluster and their input data is provided by the cache. This results in shortening the jobs' walltime for jobs that benefit directly from improved data throughput and thus an increased number of jobs processed on the caching cluster.

To determine whether the excess is caused by accelerated job processing due to caching or the job input sample's intrinsic structure, more simulation are executed using modified job input samples. First, a simulation is performed without caching functionality but using the same job input sample. Second, simulations are performed using a new job input sample of identical jobs with every job's walltime being the average walltime of all jobs. In figure 6.10 the resulting number of jobs processed on the caching cluster, $n_{\text{jobs}}^{\text{caching}}$, is shown in green and orange respectively. For identical jobs, $n_{\text{jobs}}^{\text{caching}}$ is compatible with the expectations while it scatters around the expectation for the original job input sample simulated without caching functionality. This implies that the excess in the original simulation is an effect of caching as disabling the caching functionality decreased the excess. The job input sample's intrinsic structure can also cause deviations from the expected value but these effects are smaller.

In addition to the number of jobs processed on the caching cluster, the amount of jobs that actually read from the cache, also referred to as cache hits, is inspected. For a random job distribution the amount of cache hits is expected to be

$$n_{\text{cache hits}} = n_{\text{jobs}}^{\text{caching}} \cdot \frac{n_{\text{jobs}}^{\text{cachable}}}{n_{\text{jobs}}^{\text{total}}} \tag{6.18}$$

with $n_{\text{jobs}}^{\text{cachable}}$ being the number of jobs that could read from the cache. The number of cache hits that was observed in simulations using job input sample 1 is shown in figure 6.11.

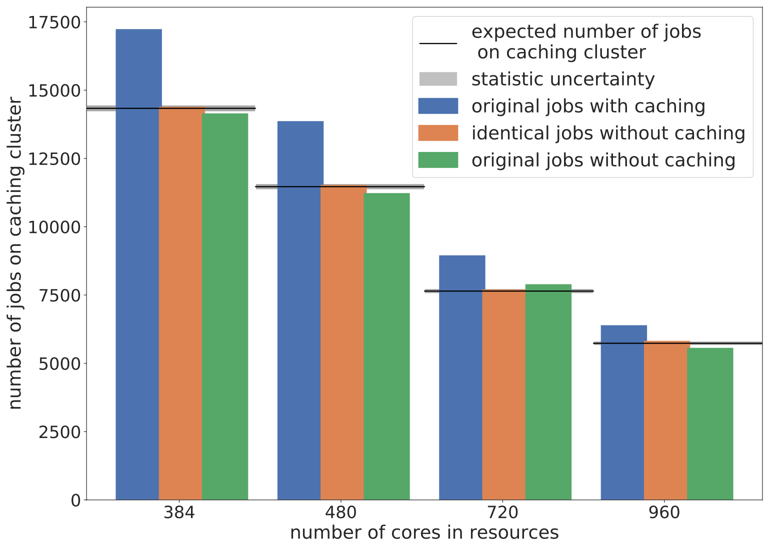


Figure 6.10: Number of jobs processed on the caching cluster without coordination depending on the cluster size. The jobs are distributed randomly between available resources and only the basic cluster increases in size while the size of the caching cluster remains constant. Consequently, the amount of jobs executed on the caching cluster decreases with increasing cluster size. The observed number of jobs exceeds the expectation as jobs of workflows that are well suited for caching can be accelerated when being processed on the caching cluster. Additional simulations without caching functionality (green) and with identical jobs (orange) verify, that this effect is not caused by the intrinsic structure of the job input sample. Without caching functionality, the excess decreases implying that it is caused by caching. Statistical uncertainties on the expected number of jobs on the caching clusters are shown as a gray band.

The amount of cache hits decreases with increasing cluster size as fewer jobs are processed on the caching cluster. The observed number of cache hits exceeds the naive expectation for every cluster size. For job input samples 2 and 3, similar results are obtained again. As the increased amount of jobs processed on the caching cluster is already accounted for in equation (6.18), additional simulations are performed to determine whether this excess is caused by the jobs' intrinsic structure, namely the order and submission times of jobs in the job input sample. Therefore, the jobs in the corresponding job input sample are shuffled resulting in a random order of jobs. Further, submission dates are assigned following a uniform distribution. Using this job input sample the simulation is repeated. For this modified input sample the number of cache hits is compatible with the expectation implying that the excess in cache hits is due to the order of the jobs, probably because similar jobs are submitted together. As different job input samples are created using the same job pool and preserving the temporal link between jobs submitted in short intervals it is conclusive that these effects manifest similarly in simulations with different job input samples.

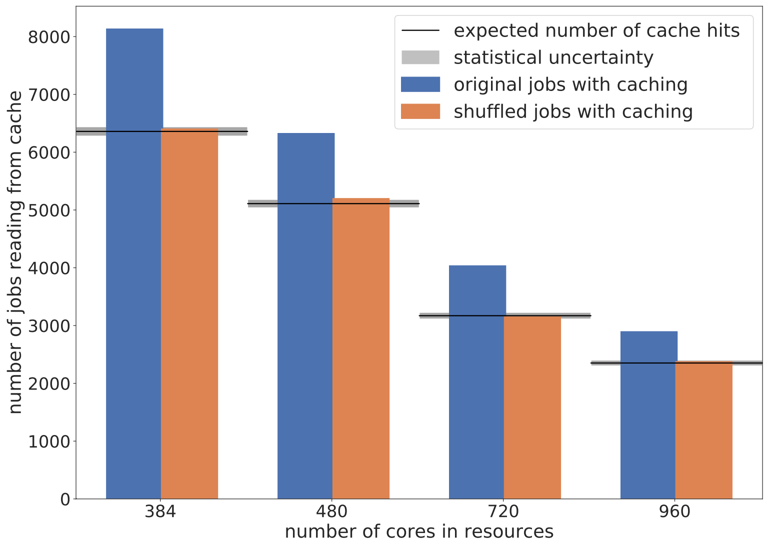


Figure 6.11: Number of cache hits depending on the cluster size. As the jobs are distributed randomly between available resources and the caching cluster does not change with increasing cluster size, the amount of jobs executed on the caching cluster and the number of cache hits decreases with increasing cluster size. For all cluster sizes the number of observed cache hits exceeds the expectations. This is caused by the order of jobs as the number of cache hits is compatible with the expectation when shuffling the jobs in the job input sample (orange bars). Statistical uncertainties of the expected number of cache hits are shown as a gray band.

With the observations regarding the number of jobs executed on the cache cluster and the number of cache hits being understood, the effects of coordination can be examined.

Coordination without `PRE_JOB_RANK`

To focus on the impact of different choices for the `CACHE_RANK` expression documented in equations (6.11) – (6.13), the `PRE_JOB_RANK` is omitted in the following. The performance of the `CACHE_RANK` expressions for one job input and different cluster sizes is compared using the amount of cache hits, shown in figure 6.12.

As observed before, the amount of cache hits decreases with increasing cluster size if there is no coordination because jobs are distributed randomly among available worker nodes. For all scenarios with coordination the number of cache hits is higher than with no coordination. The difference increases with increasing cluster size as the preference implemented in the `CACHE_RANK` has a greater impact if there are more available resources to choose from. The absolute number of cache hits when using coordination is higher for $n_{\mathrm{CPU}}^{\mathrm{total}} = 384$ than for $n_{\mathrm{CPU}}^{\mathrm{total}} = 480$. This can be traced back to the fact that the proportion of the caching cluster to the total system is higher for small cluster sizes. Therefore, more jobs are processed on

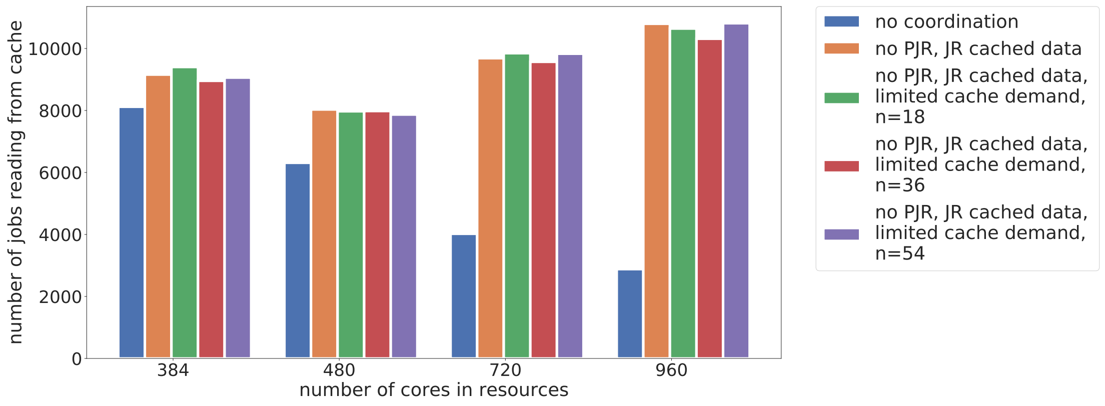


Figure 6.12: Number of cache hits depending on the cluster size for coordination scenarios without `PRE_JOB_RANK` (PJR) for job input sample 1. Different colors denote different `CACHE_RANK` expressions presented in equations (6.11) – (6.13). The limit for the number of jobs that can simultaneously read from a cache is given by n for the corresponding `CACHE_RANK` expressions. Without coordination (blue), the number of cache hits decreases with increasing cluster size as the jobs are distributed randomly. For all cluster sizes the number of cache hits increases when jobs are coordinated. The performance of all coordinating `CACHE_RANK` expressions are similar.

the caching cluster in general and the effects of coordination adds to that. It can also be observed that there is no clear superiority of one cache rank expression. Simulations of the other two job input samples provide similar results and are shown in appendix C.4.

Summarizing, this investigation establishes that if the `CACHE_RANK` is handled with high priority, coordination is achieved and the amount of cache hits increases when jobs are coordinated and that coordination has more impact on less occupied clusters.

Coordination with `PRE_JOB_RANK`

When considering the `PRE_JOB_RANK` into the system the `CACHE_RANK` can only distinguish between worker nodes with the same `PRE_JOB_RANK` score. This limits the `CACHE_RANK`'s impact. Figure 6.13 shows how this affects the number of cache hits and, therefore, coordination.

While the number of cache hits still is smaller without coordination, the number of cache hits increases only slightly when jobs are coordinated. In contrast to coordination without `PRE_JOB_RANK` the difference between the number of cache hits with and without coordination only increases significantly for the largest cluster but the number of cache hits without coordination besides the default `PRE_JOB_RANK` continuously decreases when increasing the cluster size. Again, there is no indication that one specific `CACHE_RANK` expression performs in a superior way. Simulations of other job input samples indicate the same results. The corresponding plots can be found in appendix C.4.

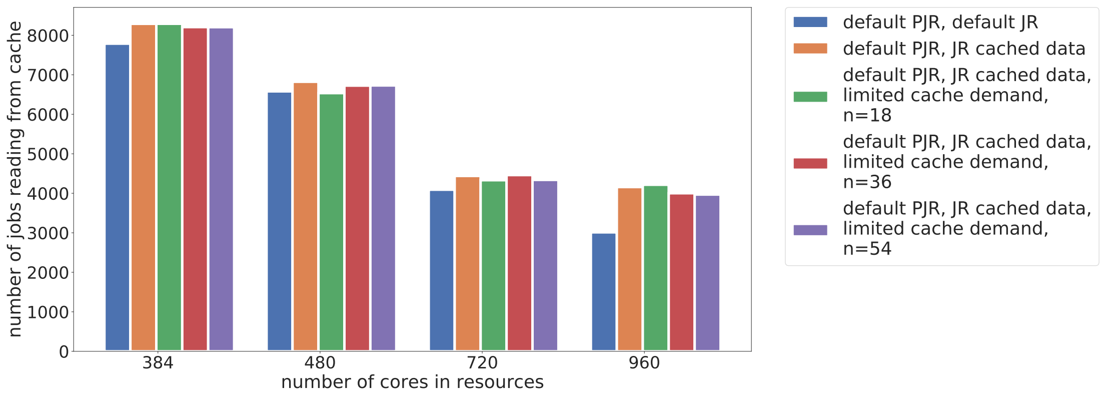


Figure 6.13: Number of cache hits for coordination with `PRE_JOB_RANK` and its dependency on the cluster size as simulated for job input sample 1. Different colors denote different `CACHE_RANK` expressions presented in equations (6.11) – (6.13). The limit for the number of jobs that can simultaneously read from a cache is given by n for the corresponding `CACHE_RANK` expressions. The number of cache hits decreases with increasing cluster size as in the case without `PRE_JOB_RANK` (blue). However, the preselection of worker nodes by the `PRE_JOB_RANK` restrains the effect of coordination resulting in only slightly increased frequency of cache hits. Like without `PRE_JOB_RANK` before, there is no clear indication of the superiority of one coordination expression.

This study shows that the `CACHE_RANK`'s impact decreases significantly when the default behavior of the matchmaking process is re-established by reintroducing the default `PRE_JOB_RANK` expression. Therefor, modifying the `job.RANK` while maintaining the batch system's original behavior is not an optimal realization of coordination. However, there are related approaches to achieve coordination that are interesting topics for future research. For example, a `CACHE_REQUIREMENT` can be introduced to pause the scheduling of jobs with input files for a few scheduling cycles unless worker nodes with access to cached data become available as proposed in section 6.3.2. Alternatively, the `PRE_JOB_RANK` expression itself can be modified to include `CACHE_RANK` expressions as discussed in section 6.3.1.

Summarizing the results of the study of coordination mechanisms presented in this thesis, it was demonstrated that coordination can be achieved if introduced `CACHE_RANK` expressions are able to act. In this case, the throughput of the examined system increases showing that the main goal of distributed coordinated caching was realized. Therefore, it is worthwhile to continue examining how the batch system's original behavior and coordination implementations can be balanced.

Chapter 7

Conclusion and Outlook

In this thesis, the suitability of distributed coordinated caching for improving the performance of data-intensive workflows in Tier 3 centers and thereby the utilization of these resources was examined. Improving the efficiency of the available resources is essential to cope with the challenges posed by the sheer volume of data that has to be handled for many analyses in HEP. On top of this, these challenges are expected to grow in severity with the beginning of the high luminosity period at the LHC.

For several years, KIT has been developing and using prototypes that enable distributed coordinated caching. The focus of the existing considerations was set on feasibility and transparent integration into the existing Tier 3 infrastructure. However, it is not feasible to measure the effects of specific coordination mechanisms and cache behaviors in a real system as side effects and user interaction have to be expected. Thus, the impact of different realizations cannot be guaranteed to be reproducible. Therefore, the systematic optimization of the coordination mechanisms and the cache behavior has not been actively pursued so far. In this thesis, a two-fold optimization approach was followed to enable a systematic investigation of different realizations of distributed coordinated caching.

First, job and data access monitoring was implemented and successfully put into operation for the Tier 3 resources at ETP. Based on the recorded monitoring data and developed characteristics, the workload of this representative Tier 3 computing center was identified as a suitable use-case for distributed coordinated caching. Multiple workflows of this workload were found to be well suited to benefit from caching. Among these workflows, the Analysis stage of the jet energy calibration was identified as particularly well suited. The jet energy calibration workflow was examined in detail with respect to its experimental relevance and caching-related properties and its processing in a real system was observed.

Second, major contributions to the implementation of a simulation tool were designed and developed. This allows to investigate different realizations of distributed coordinated caching in a variety of infrastructures in a fraction of the time compared to testing them in a real system. The extensions made to the simulation tool LAPIS were successfully verified by benchmarks and used for systematic investigations of job coordination. Suitable realizations for this coordination that can be directly transferred to real computing centers were proposed, simulated and evaluated. With the help of the strategy developed in this thesis it was successfully shown that distributed coordinated caching is a promising approach to increase data locality and that further efforts to improve the coordination of jobs are worthwhile.

Summarizing, the work presented in this thesis contributes significantly to a well-founded realization of distributed coordinated caching. Thus, this thesis lays the foundation for a multitude of consecutive studies as the necessary tools, considerations and experience are provided. In this way, new implementations can be verified utilizing the developed simulator.

Outlook

In addition to the studies performed in this thesis, the extensions contributed to LAPIS allow for a wide range of applications to improve the understanding of distributed coordinated caching. While the studies performed in this thesis establish a general concept for coordination mechanisms, further studies on coordination mechanisms that respect both the regular resource usage optimizations of the batch system and the coordination optimizing data locality should be performed. In addition, the performance of coordination mechanisms in different workloads and in different cache and resource setups needs to be studied systematically to understand the detailed behavior of various coordination mechanisms.

Besides examining coordination mechanisms, simulations can be used to investigate the behavior of caches and to identify suitable caching policies and cache eviction policies. Several suitable strategies for the cache eviction policy for HEP workflows are proposed in [51] and should be evaluated with respect to various heterogeneous infrastructures. The caching policy should be studied in detail as well however this has not been performed yet. Beyond investigating the main aspects of distributed coordinated caching separately, further studies are needed to understand their interactions and ensure their compatibility.

Finally, simulations enable studying the impact of distributed coordinated caching and chosen implementations on existing computing centers in terms of performance gains and quality metrics. Especially the integration of opportunistic resources into these infrastructures has not been supported by any simulator so far. This allows the investigation of a dynamic distributed coordinated caching combined with opportunistic resources and offers a unique but technologically feasible and promising use-case for future research.

In summary, the ability to simulate distributed coordinated caching in dynamic infrastructures with opportunistic resources offers enormous potential for highly scalable studies in highly complex environments. This thesis has laid the foundation for these simulations and has shown that distributed coordinated caching qualifies as a suitable strategy with a wide range of applications and will contribute significantly to the efficient use of computing resources.

Appendix A

List of monitored job ClassAd attributes

In scope of this thesis, the workload of the Tier 3 center at ETP was monitored as described in chapter 4. The majority of the information that was gathered about jobs that make up this workload was extracted from the jobs' metadata representation in the HTCondor, the ClassAd. Every job is described by a ClassAd consisting of a list of attributes. Some of these attributes were monitored directly and some attributes were used to calculate additional quantities that were then monitored. A list of both is presented in the following, grouped into four areas: job identification, timing, resources and input file information. An additional reference for ClassAd attributes in HTCondor is given by [61].

- Job identification: attributes that belong to this area are used to identify jobs, their submitter and can indicate to which workflow a job belongs.
  - `ClusterId, ProcessId`: identification of a job in the batch system, individual for each job on the machine the job was submitted on.
  - `Owner`: name of the submitter of a job
  - `AccountingGroup`: group a submitter belongs to that can be used to balance computing resources's usage if there are multiple groups competing for the resources
  - `Machine`: address of the machine a job was submitted on
  - `LastRemoteHost`: address of the worker node a job was processed on
  - `Cmd`: path to a job's executable
  - `JobBatchName`: attribute that can be assigned by the user to identify jobs
- Timing: these attributes are related to a job's time information or calculated based on such information
  - `QDate`: UNIX timestamp indicating a job's submission into the batch system
  - `JobCurrentStartExecutingDate`: UNIX timestamp indicating the beginning of a job's execution
  - `CompletionDate`: UNIX timestamp indicating the end of a job's execution
  - CPUTime = `RemoteSysCpu` + `RemoteUserCpu`: total number of seconds that CPU cores processing this job were active

  - `RemoteWallClockTime`: number of seconds that passed between beginning and end of the job's execution. As a control value, the difference between `CompletionDate` and `JobCurrentStartExecutingDate` was monitored as well
  - Waitingtime = `JobCurrentStartExecutingDate` − `QDate`: number of seconds that passed between a job's submission and its execution
  - CPUEfficiency = $\frac{\texttt{CPUTime}}{\texttt{RemoteWallClockTime}}$: efficiency of a job. If a job allocated multiple CPU cores the CPUEfficiency has to be divided by the number of allocated CPU cores to obtain values between 0 an 1

- Resources: these attributes indicate the amount of resources a job requested for processing and how much it actually used and calculated based on them
  - `RequestCpus, RequestMemory, RequestDisk`: amount of CPU cores, memory and disk space a job requested
  - `RequestWalltime`: amount of walltime requested by a job
  - `MemoryUsage, DiskUsage`: amount of memory and disk space a job actually used
  - `NetworkInputMb, NetworkOutputMb`: amount of data in MB that was transferred to and sent from the job, accessible if the job ran in a docker container
  - ApproxDatarateMb = $\frac{\texttt{NetworkInputMb}}{\texttt{RemoteWallClockTime}}$, average data rate recieved by a job
- Inputfiles: a list containing the path to a job's input files. This ClassAd is only set for the setup at ETP. Information about the input files' sizes had to be gathered separately

Appendix B

Selection criteria of $Z\,(\rightarrow \mu\mu)$ + jet events for absolute residual JEC corrections

$Z\,(\rightarrow \mu\mu) +$ jet events that can be used to determine absolute residual corrections at ETP are selected by applying multiple selection criteria to the two muons, the Z boson reconstructed from the two muons and the jet that is balanced against the Z boson. The selection criteria are chosen to avoid objects being misidentified, to be precisely reconstructed and to be compatible with the desired event topology. In this appendix the selection criteria applied to data recorded in 2016 are presented [45].
For the two muons the selection are the following:

- muons need to pass tight muon identification criteria [62] to avoid misidentified muons
- muons need to pass tight muon isolation criteria [62] to make sure that muons are created in the hard process and do not stem from decaying hadrons of a jet
- muons need to be detected where the tracker is actually instrumented ($|\eta^{\mu}| < 2.3$)
- muons need to have a transverse momentum $p_{\mathrm{T}}^{\mu} > 20\,\mathrm{GeV}$.

The Z boson, reconstructed from the four-momenta of both muons needs to satisfy the following conditions:

- there need to be at least two muons with opposite charges and at most three muons in total
- the reconstructed Z bosons mass needs to be withing 20 GeV of the Z boson mass derived by the particle data group [63]

Selection criteria for the jet balanced against the Z boson are the following:

- the jet needs to satisfy the loose jet identification as defined in [64]
- jets close to one of the muons ($\Delta R < 0.3$) need to be rejected to avoid confusion with the muons
- the events most energetic jet's transverse momentum must be $> 12\,\mathrm{GeV}$ and it needs to be recorded in the detectors central region ($\left|\eta^{\mathrm{jet}}\right| < 1.3$)

To ensure that the events topology is close to the ideal event topology, the following cuts are applied:

- to suppress additional jet activity, the second jet activity α must be < 0.3
- Z boson and the most energetic jet need to be in a back-to-back configuration represented by $\left|\Delta\varphi^{\mathrm{jet1,\,Z}} - \pi\right| < 0.34$

Appendix C

Additions to the performed simulations

This appendix provides additional information concerning chapter 6. This includes a discussion of the assumption that input files are read exactly once as well as additional simulator cross-checks, job input cross-checks and simulation results.

C.1 Examination of the fraction of data read from input files

The purpose of this section is to motivate the assumption that input files are read exactly once. To this end monitoring information of the storage element at GridKa, a Tier 1 computing center of the WLCG located in Karlsruhe [65], was used. This monitoring contains information about the amount of data transferred when a file in the storage element is accessed as well as the file's total size. To determine how much of the input file was read, the ratio of the amount of transferred data and the total file size was calculated. A histogram of this fraction determined for the file accesses of a representative amount of jobs is shown in figure C.1.

For the majority of file accesses, the ratio r of transferred data volume and file size is close to one, reinforcing the assumption that in the majority of cases the whole file is read once. For some file access the ratio exceeds 1 because some parts of the file are read multiple times. Lower ratios can be explained by the structure of a data format that is commonly used in high energy physics. Files of this data format can be used as basis for multiple workflows and hence not all contained information is needed and accessed by all workflows resulting in a lower ratio. The peak at zero is caused by file transfers in which the file was opened but none or only a small amount of data was read.

C.2 Simulator cross-checks and benchmark recreation: additional plots

After extending LAPIS to support caching, the newly included functionality was tested and validated by performing cross-checks and by recreating benchmark measurements.

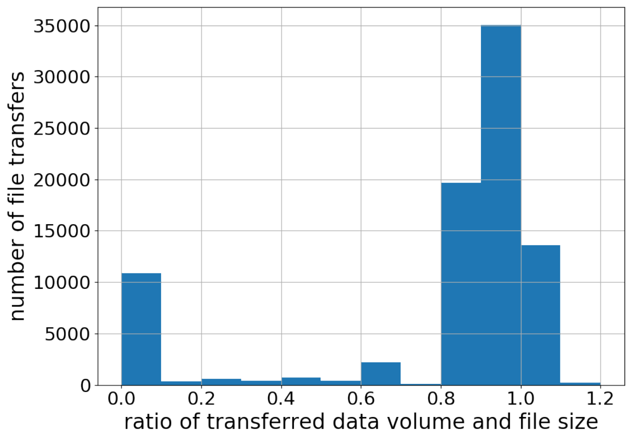


Figure C.1: Histogram of the ratio of the amount of transferred data for a file access and the files total size.

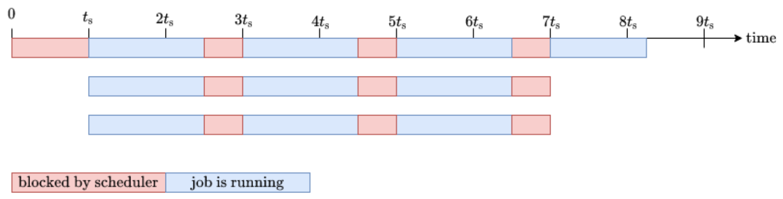


Figure C.2: Schema illustrating the effect of discrete scheduling intervals for the processing of 10 identical jobs on 3 identical slots. t_s denotes the duration of the scheduling cycle. The three parallel lines indicate the three slots. The time that jobs are actually processed on these slots is marked in blue while the time that passes while waiting for the beginning of the next scheduling cycle is marked in red.

C.2.1 Functionality cross-check

To test the simulator functionality, the runtime of jobs and the simulated time that passed until the jobs were completed was examined to determine whether the simulator handles jobs as expected. Therefore, a simple, calculable setup consisting of n identical read jobs and m identical CPU cores, also called slots, was chosen for the following studies.

While the expectation for a single job's walltime is given by equation (6.2), calculating the expectation for the simulated time is more complex as it has to take the impact of discrete scheduling cycles into account. This impact is illustrated in figure C.2 for ten jobs processed in three slots. Thereby, t_s denotes the duration of a scheduling cycle. Jobs start during the next scheduling cycle after their submission at the earliest. As there are three slots, three jobs are processed together in parallel three times and one job is processed alone. After processing the first batch of three jobs, the next scheduling cycle has to be awaited before the processing of the next batch can begin. In this setup, the simulation ended when the last job is finished.

Taking all of this into account, the simulated time that passes when processing n identical jobs transferring a data volume V on m identical jobs is calculated the following way:

If the jobs can be distributed equally among the slots ($n \mod m = 0$) the simulated time $t_{\text{simulated}}$ is given by:

$$t_{\text{simulated}}(h) = t_S \left(1 + (n \div m - 1) \left(1 + t_{\text{job}}^{\text{parallel}}(h) \div t_S\right)\right) + t_{\text{job}}^{\text{parallel}}(h) \tag{C.1}$$

If the jobs can not be distributed equally among the slots ($n \mod m > 0$) the simulated time is calculated using

$$t_{\text{simulated}}(h) = t_S \left(1 + (n \div m) \left(1 + t_{\text{job}}^{\text{parallel}}(h) \div t_S\right)\right) + t_{\text{job}}^{\text{remaining}}(h) \tag{C.2}$$

In both cases $t_{\text{job}}^{\text{parallel}}$ and $t_{\text{job}}^{\text{remaining}}$ are given by

$$t_{\text{job}}^{\text{parallel}}(h) = \max\left(\frac{m \cdot V \cdot h}{s_{\text{cache}}}, \frac{m \cdot V \cdot (1-h)}{s_{\text{remote}}}, t_{\text{calculation}}\right) \tag{C.3}$$

$$t_{\text{job}}^{\text{remaining}}(h) = \max\left(\frac{(n \mod m) \cdot V \cdot h)}{s_{\text{cache}}}, \frac{(n \mod m) \cdot V \cdot (1-h)}{s_{\text{remote}}}, t_{\text{calculation}}\right) \tag{C.4}$$

with s_{cache} and s_{remote} being the data throughput provided by cache and remote storage respectively. The cache hitrate h corresponds to the fraction of input data that is provided by the cache instead of remote storage and $t_{\text{calculation}}$ is the job's calculation time as introduced in section 3.3.1.

To show that the simulated time agrees with the expectations, four scenarios $n = m = 1$, $n > m = 1$ and $n > m > 1$ with $n \mod m = 0$ and $n \mod m > 0$ were simulated. The resulting simulated time and expected simulated time derived in this section agree, as shown in figure C.3, indicating that the simulator behaves as expected. The step structure that can be seen in the dependence between simulated time and cache hitrate h is caused by the discrete scheduling cycle, namely the time that passes after the execution of one batch of jobs until the scheduling cycle is complete and the next batch of jobs can be processed. The steps occur when the value of $t_{\text{jobs}}^{\text{parallel}} \mod t_{\text{S}}$ changes because the job's walltime changes with h resulting in the simulated time being shortened or lengthened by $t_{\text{S}} \cdot (n \div m)$ if $n \mod m > 0$ and $t_{\text{S}} \cdot (n \div m - 1)$ if $n \mod m = 0$. This effect is illustrated in figure C.4.

In each of the three simulated scenarios the same amount of input data is read the input data was allocated to a varying numbers of identical jobs ($n = \{600, 60, 12\}$). These jobs were then processed on a cluster containing $m = 24$ slots. For $n = 600$ there is one step of 24 min, if there are only 60 jobs the step size is reduced to 2 min and for $n = 12 < 24$ there are no steps.

C.2.2 Benchmark recreation

Besides the functionality cross-check, benchmark measurements described in [58] were recreated to show that the simulation tool developed in the scope of this thesis is capable of reproducing existing setups. The recreation of a benchmark using jobs that only read data and do not perform additional calculations on them was presented in section 6.2.2. In addition to this, a similar benchmark based on jobs of the jet energy calibration workflow introduced in 4.2.3 was recreated. It is presented in this section.

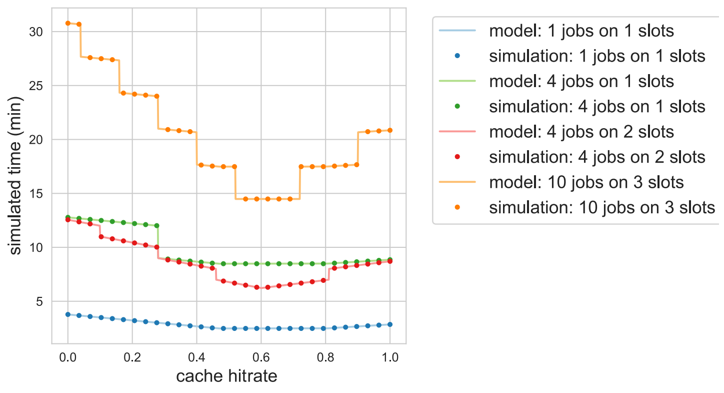


Figure C.3: Cross-check of the extended LAPIS simulator's caching functionality by comparing the simulated time of different simulation scenarios to the expected simulated time. Each simulation scenario consists of n identical jobs on m identical slots and for all scenarios the simulated time agrees with the expectation.

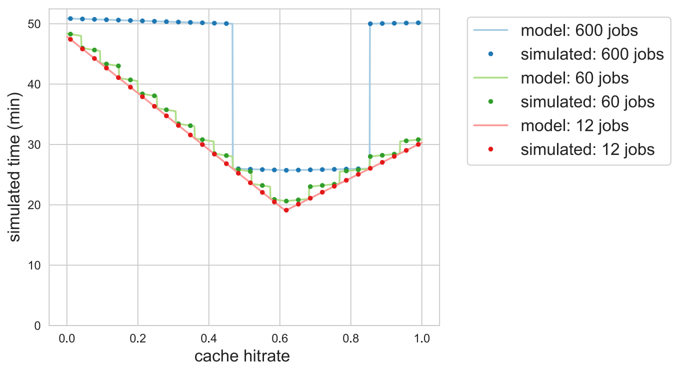


Figure C.4: Effect of a discrete scheduling cycle on simulated time. The same amount of input data is processed by a differing number of identical jobs on 24 job slots to study the step structure in the simulated time. This step structure is an effect of the waiting time between the end of job's execution on a worker node and the allocation of the next job to this worker node due to discrete scheduling cycles. If there are fewer waiting times because there are fewer jobs the step structure becomes less prominent. In case of 12 jobs no steps are present because all jobs can be processed simultaneously and there are no waiting times.

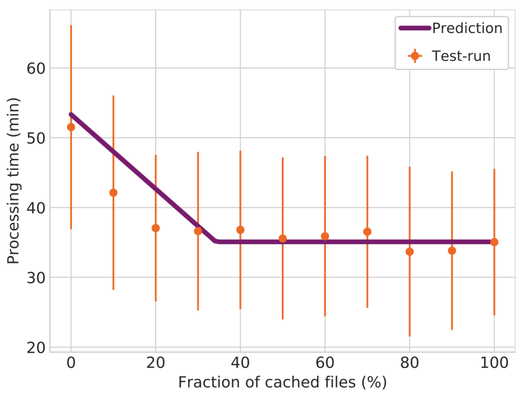
Like in the previous benchmark recreation, 60 jobs requesting 10 input files each are processed on a set of three worker nodes. The worker nodes total to 60 CPU cores with two worker nodes containing 24 CPU cores and one worker node containing 12. All worker nodes share a connection to remote storage with a bandwidth of 3 GBit/s. Each worker node is connected to a cache, each cache connection provides a bandwidth of 1 GBit/s. The total amount of transferred data is $V = 1.2\,\mathrm{TB}$ and the maximal data throughput of the calculations that are performed on this data was measured to be. $s_{\mathrm{workflow}} = 570\,\mathrm{MB/s}$. The jobs were submitted simultaneously to empty worker node and therefore were processed approximately simultaneously. Hence, their walltime is expected to be

$$t_{\mathrm{walltime}}(h) = \max\left(\frac{V \cdot (1-h)}{\frac{s_{\mathrm{remote}}}{n_{\mathrm{jobs}}}}, \frac{V \cdot h}{\frac{s_{\mathrm{cache}}}{n_{\mathrm{jobs}}}}, \frac{V}{s_{\mathrm{workflow}}}\right) \tag{C.5}$$

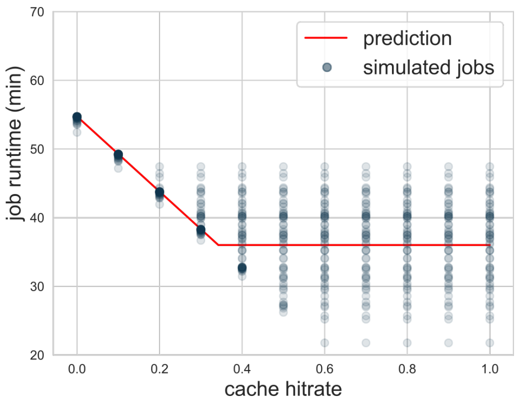
with $n_{\mathrm{jobs}} = 60$ being the number of jobs. The last term represents the calculation time. As the calculation time will differ between jobs as they process different amounts of input data, an approximation based on V and s_{workflow} is used. Like for the cross-checks presented in this section the cache hitrate h is varied between 0 and 1. The average value and variation for the measured walltimes are shown in figure C.5a. The measured walltimes agree with the expectation within their variations. The large variations can be explained by multiple factors. First, the amount of data processed by each job varies resulting in differing duration of data transfer and data processing. Additionally, the bandwidth of remote and cache connections is not constant in reality but can vary in time due to competing transfers and the bandwidth is not necessarily split equally between transfers. Furthermore, the jobs might not have started exactly simultaneously.

To recreate the benchmark using LAPIS, the jobs' metadata was gathered and a worker node setup was configured to match the setup the benchmark was performed on. Hitrate based caching was used as caching mode. Based on s_{workflow} and the data volume transferred by each job, V_{job}, an individual calculation time $t_{\mathrm{calculation}}$ is determined for each job using $t_{\mathrm{calculation}} = \frac{V_{\mathrm{job}}}{s_{\mathrm{workflow}}}$. Figure C.5b shows the simulated jobs' walltime depending on h along with the expected walltime given by equation (C.5). Instead of calculating an average value, the walltime of the jobs is represented by an opaque dot. Up to $h = 0.2$ the walltimes scatter but agree well with the expectation. Above $h = 0.2$ the walltimes start to scatter to higher values and starting from $h = 0.4$ to lower values as well. This is an effect of the jobs' varying calculation time. For low h the walltime is limited by the transfer from remote. Once the jobs' calculation time is reached, the walltime does not decrease further but remains constant as in this setup, the cache connections do not become the limiting factor. Despite the scattering, the average values agree with the expectation as in the original benchmark within their variations.

Concluding, this benchmark recreation shows that the LAPIS simulator can be used to simulate complex systems.



(a) Original benchmark taken from [58].



(b) Benchmark recreation using LAPIS.

Figure C.5: Comparison between a benchmark of jobs of the jet energy calibration workflow performed at ETP (a) and its simulated recreation (b). In both cases the jobs' walltimes were determined depending on the fraction of files that were read from a cache instead of remote storage, also called cache hitrate. For the original benchmark the average walltime and its variance are shown while the spread of the measured walltimes is shown for the simulation. In both cases the measured or simulated walltimes agree with the expectation given by equation (C.5) within their variations.

C.3 Cross-check of the amount of input data in the job input samples

In addition to the discussion of the distribution of the total amount of input data in 30 job input samples presented in section 6.4.3, this section describes the workflow composition of the job input samples' input data. To give an overview over the distribution of input data across workflows, figure C.6 presents a comparison between workflows based on the number of jobs with input files in the job pool from which the job input samples are sampled and the total amount of data transferred by these jobs. While the JEC and Higgs analysis workflow only correspond to a small fraction of the jobs in the job pool, the amount of data processed by these workflows corresponds to more than 75 % of the total amount of processed data. As discussed in section 4.2.2, this indicates that jobs of these workflows have a large data throughput and are therefore likely to benefit from caching as deduced in section 4.2.3.

Figure C.7 shows a histogram of the amount of input data per workflow for 30 job input samples. In this representation only the amount of input data of the JEC and Higgs analysis and Skimming jobs are shown explicitly as they make up the majority of the transferred input data. Again, the dominant role of JEC and Higgs analysis concerning the amount of input data is shown. For all job inputs the amount of transferred data is of the same order of magnitude but scattered across a range of several TB. This can be explained by the fact, that the individual job input samples' workflow composition varies and with varying

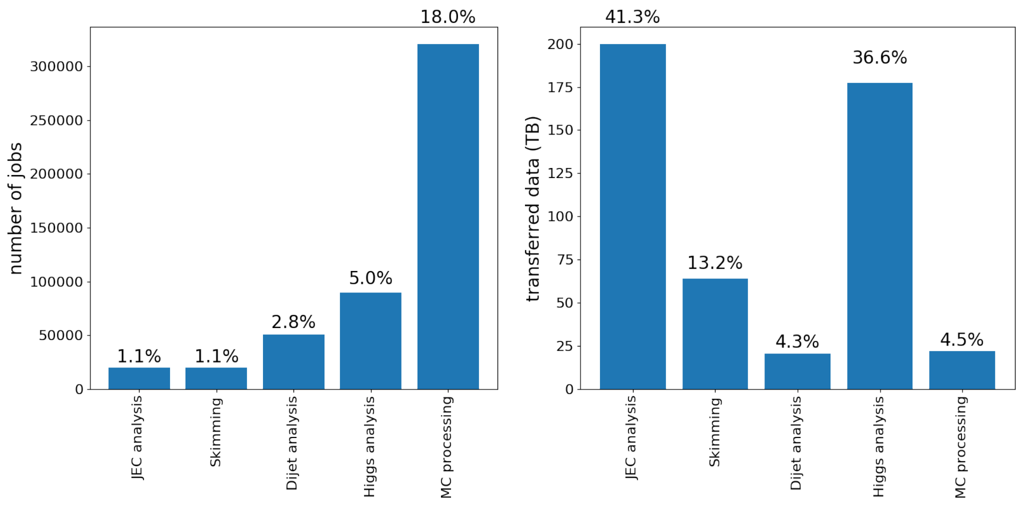


Figure C.6: Comparison by workflow of the number of jobs with input files and known input file size to the total amount of data transferred by these jobs for jobs in the job pool.

workflows the amount of transferred data varies. In addition, workflows differ in the amount of data they transfer.

C.4 Simulation results of additional job input samples and metrics

The plots illustrating simulation results presented in section 6.5.2 are based on the first of three job input samples defined in table 6.2. Simulations using the other two job input samples were performed as well. The resulting plots are presented in this section. Besides using the number of cache hits to determine whether the coordination of jobs was achieved using a specific coordination mechanism, the amount of data provided by the cache can be used as a metric. The resulting plots are discussed in the following.

C.4.1 No coordination: cross-checks & effects of caching

Before examining the influence of coordination, the system's behavior without coordination is studied. The metrics used to do this are the number of jobs that were processed on the caching cluster and the number of cache hits. As described in section 6.5.2, a larger amount of jobs was processed on the caching cluster than expected based on a random distribution alone. It can be concluded that this is an effect of caching. Even without coordination, some jobs whose input files can be provided by the cache are allocated to the caching cluster. The walltime of these jobs is shortened, freeing the resource earlier and thereby making it available to other jobs. Consequently, the job throughput on the caching cluster is increased. To ensure that the effect is caused by caching and not by the intrinsic structure

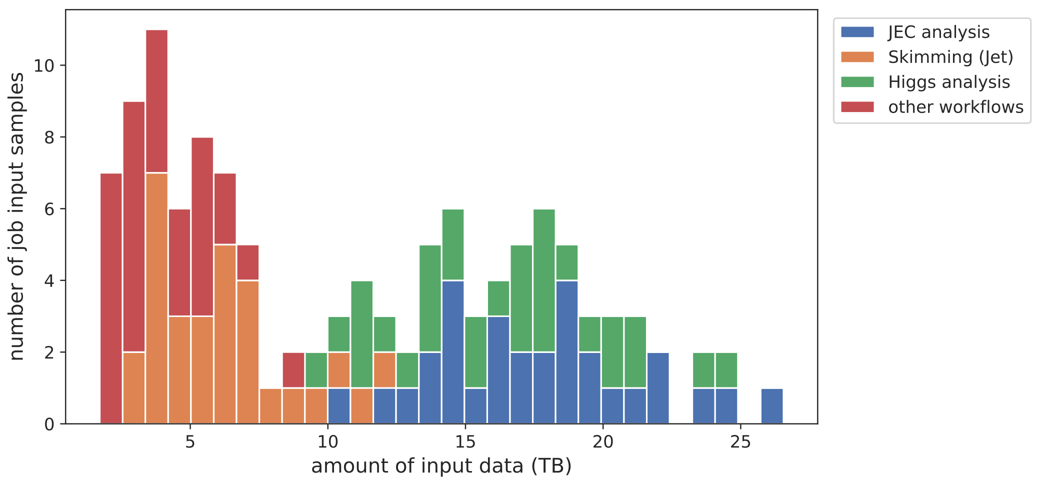


Figure C.7: Amount of input data per workflow for 30 job input samples generated from the job pool.

of the job input, simulations with the same job input, with disabled caching functionality and with modified intrinsic structure of the job input samples were performed. In both cases, the number of jobs processed on the caching cluster is significantly lower and compatible with the expected number of jobs assuming a random distribution of jobs to resources. This is illustrated in figure 6.10 for different cluster sizes and job input sample 1. Figure C.8 illustrates that the same was found using job input sample 2 and 3. The number of jobs processed on the caching cluster decreases in size because the number of CPU cores of the caching cluster is constant while the number of CPU cores provided by resources without caching functionality increases. As jobs are distributed randomly among available resources, the number of jobs processed on the caching cluster decreases with increasing cluster size.

In addition to the number of jobs processed on the caching cluster, the number of cache hits, i.e. the number of jobs that read from the cache, was examined. Based on the observed number of jobs processed on a caching cluster the expected number of cache hits was calculated. Again, the number of cache hits exceeds the expectation. To determine whether this effect is caused by the job input's intrinsic structure, the simulation was repeated with shuffled order of jobs. The excess disappeared indicating that the excess was caused by the order of jobs. In figure 6.11 this is shown for job input sample 1. Figure C.9 shows that the same applies to simulations for job input sample 2 and 3.

C.4.2 Coordination without `PRE_JOB_RANK`

In section 6.5.2, the effects of coordination when omitting the `PRE_JOB_RANK` are illustrated by examining the number of cache hits based on job input sample 1. Figures C.10 and C.11 show that the impact of coordination without `PRE_JOB_RANK` described in section 6.5.2 apply to simulations based on job input sample 2 and 3 as well.

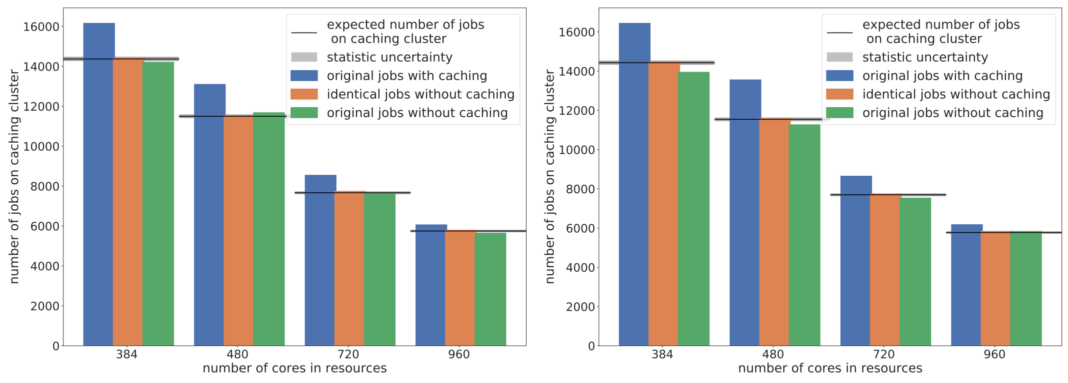


Figure C.8: Number of jobs on caching cluster for varying cluster sizes and job input sample 2 (left) and 3 (right) respectively. The number of jobs processed on the caching cluster (blue) exceeds the expectation based on randomly distributed jobs. This is an effect of caching as it disappears when disabling caching functionality (green). It is not an effect of the job input samples' intrinsic structure as the number of jobs processed on the caching cluster differs only slightly if the intrinsic structure is changed (orange). Statistic uncertainties on the expected number of jobs on the caching clusters are shown as a gray band.

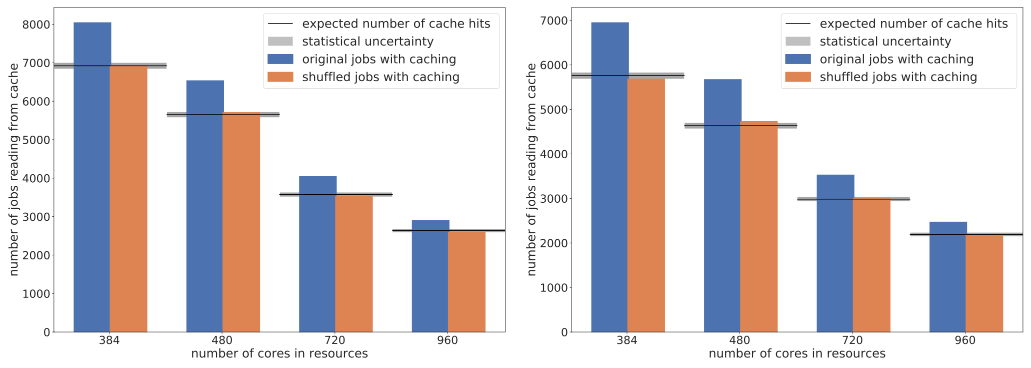


Figure C.9: Number of cache hits for varying cluster sizes and job input sample 2 (left) and 3 (right) respectively. The number of cache hits using the unmodified job input samples (blue) exceeds the expectation. This is an effect of the order of jobs in the job input samples as the excess disappears when shuffling the jobs (orange). Statistic uncertainties on the expected number of cache hits are shown as a gray band.

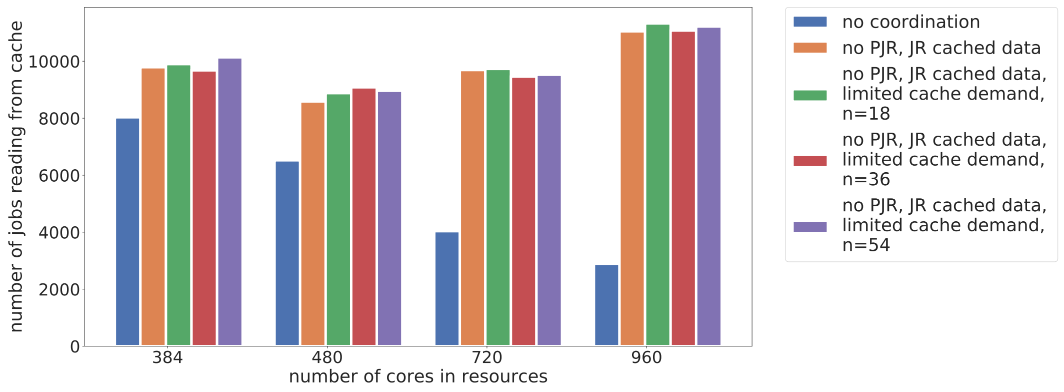


Figure C.10: Number of cache hits when coordinating without `PRE_JOB_RANK` using job input sample 2.

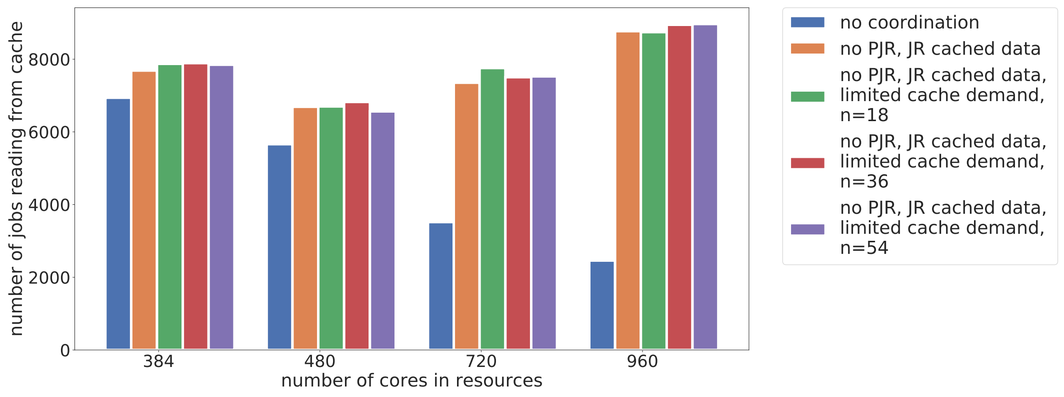


Figure C.11: Number of cache hits when coordinating without `PRE_JOB_RANK` using job input sample 3.

Amount of data read from cache as a metric

In addition to the number of cache hits, the amount of data read from the cache can be used as a metric to assess the impact of coordination mechanisms. If jobs are successfully coordinated to the caching cluster the amount of data read from the cache increases. However, the amount of data read from the cache depends on the amount of data requested by each coordinated job. Consequently, the general interrelations are similar for both metrics. This can be seen in figure C.12 — C.14 that show the amount of cached data for the examined coordination mechanisms and varying cluster sizes for job input samples 1-3 respectively.

As observed for the number of cache hits, the amount of data read from the cache decreases when the cluster size increases if there is no coordination because jobs are distributed randomly among available worker nodes. The amount of data read from the cache is higher for all scenarios with coordination than with no coordination and the difference increases with increasing cluster size since the `CACHE_RANK` has a greater impact.

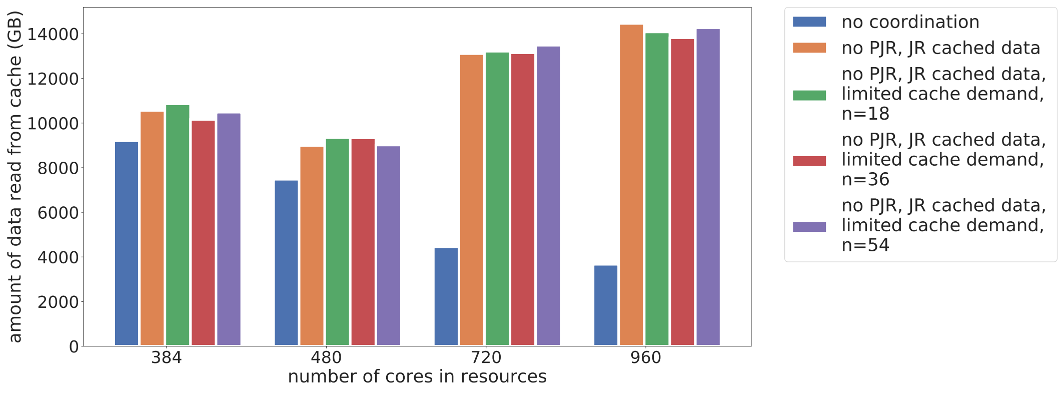


Figure C.12: Amount of data transferred from the cache when coordinating without `PRE_JOB_RANK` using job input sample 1

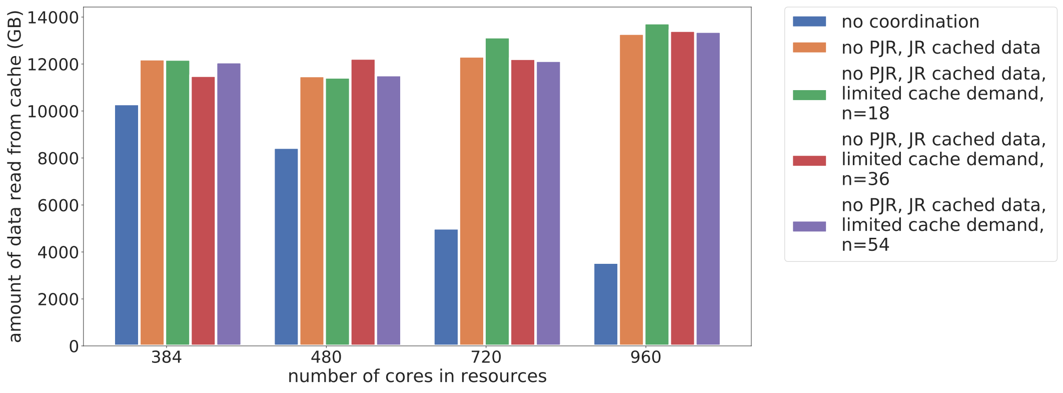


Figure C.13: Amount of data transferred from the cache when coordinating without `PRE_JOB_RANK` using job input sample 2.

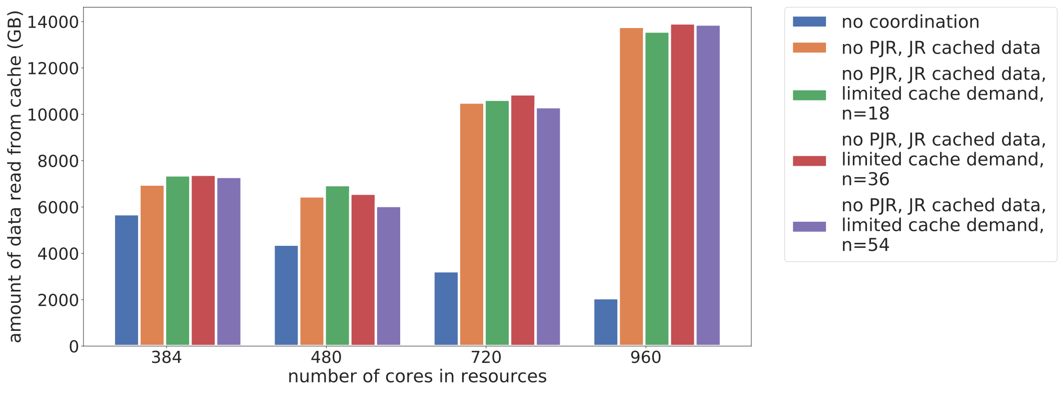


Figure C.14: Amount of data transferred from the cache when coordinating without `PRE_JOB_RANK` using job input sample 3.

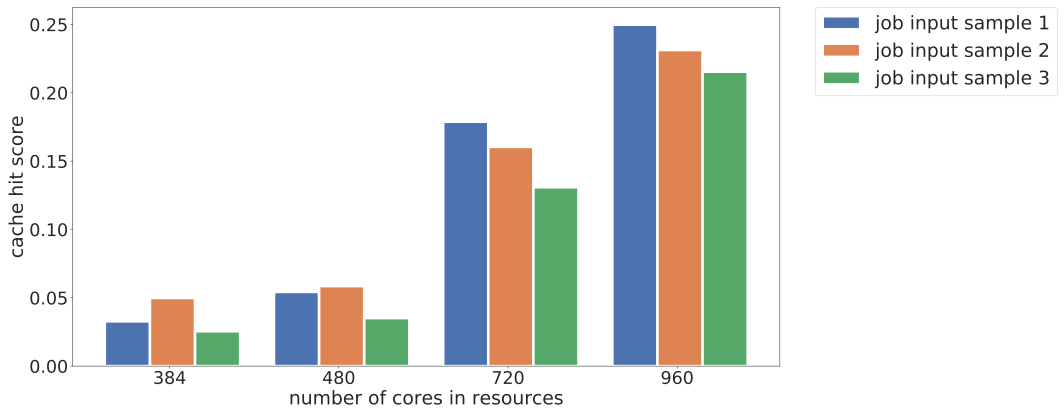


Figure C.15: Cache hit score of job input samples 1-3 for varying cluster sizes when coordinating without `PRE_JOB_RANK`. While the cache hit score compensates for the job input samples' differing number of cacheable jobs, the differences between the job input samples' cache hit scores indicates that there are additional properties of a job input sample influencing the impact of coordination.

Cache hit score as a metric

Besides the number of cache hits and the amount of transferred data, the cache hit score was introduced in equation (6.16) as an alternative metric. Its purpose is to allow comparing the performance of coordination mechanisms between job input samples with varying amounts of jobs that could read data from the cache, in the following referred to as cacheable jobs. Figure C.15 shows the cache hit score of all three job input samples for varying cluster sizes and the coordination scenario in which the amount of cached data was used as `CACHE_RANK`. Other coordination scenarios yield similar results.

While the obtained cache hit score values reflect that the impact of coordination increases with increasing cluster size, the scores differ significantly between job input samples. This indicates that besides the number of cacheable jobs, other properties that the cache hit score does not account for influence the impact of coordination. For example, as stated in table 6.2 the number of CPUh necessary to process all jobs of the input sample within one week, $n_{\mathrm{CPU}}^{\mathrm{total}}$, of job input samples 2 and 3 exceed that of job input sample 1. As the cache hit score is proportional to the number of cache hits that decreases with decreasing cluster size, this motivates that the cache hit score of job input sample 1 exceeds that of job input samples 2 and 3 for large cluster sizes. In addition, the job input samples intrinsic structure influences the number of cache hits as discussed in section 6.5.2 and thereby the cache hit score.

To illustrate that the cache hit score actually accounts for differing numbers of cacheable jobs, the number of cacheable jobs for job input sample 1 was varied. This was achieved by performing simulation for $h_{\mathrm{caching}} = \{0.33, 0.67\}$ in addition to $h_{\mathrm{caching}} = 1.0$ in all other simulations discussed in this thesis. The resulting cache hit scores are shown in figure C.16. In this case, the resulting cache hit scores agree well. Remaining differences can be explained

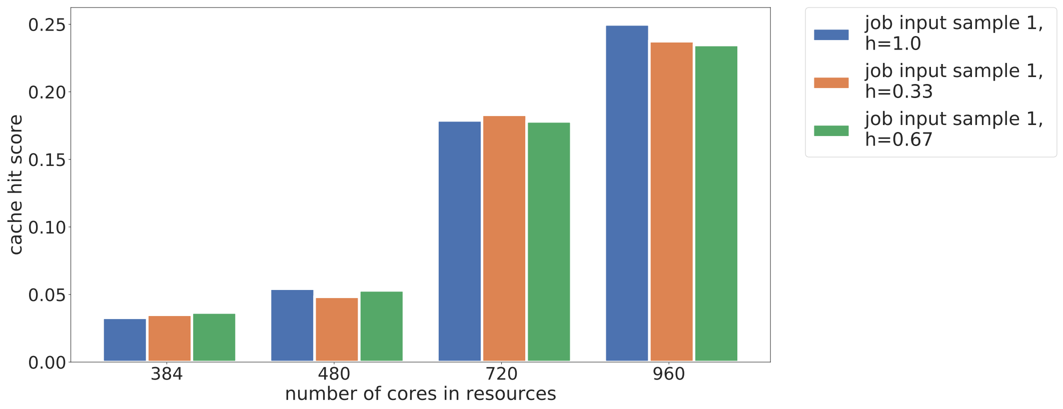


Figure C.16: Cache hit score of job input samples 1 for varying numbers of cacheable jobs and cluster sizes when coordinating without `PRE_JOB_RANK`. As all examined job input samples are closely related, the cache hit scores are compatible.

by statistical effects and the small differences in the modified job input samples' intrinsic structure introduced by the different choices of h_{caching}.

C.4.3 Coordination with `PRE_JOB_RANK`

Similar to the discussion of coordination without `PRE_JOB_RANK`, the results described in section 6.5.2 apply to simulations performed using job input samples 2 and 3 instead of job input sample 1 as well as shown in figures C.17 and C.18.

Amount of data read from cache as a metric

Like when discussing coordination without `PRE_JOB_RANK`, the amount of data read from the cache is studied as an alternative metric to measure the impact of coordination. The results are shown in figures C.19 — C.21. Again, the behavior of the amount of data read from the cache is very similar to the behavior of the number of cache hits.

Cache hit score as metric

For the coordination scenarios with default `PRE_JOB_RANK` expressions, the cache hit score is examined for the coordination scenario in which the amount of cached data was used as `CACHE_RANK` expression as well. Figure C.22 shows the cache hit score for job input samples 1–3 and varying cluster sizes while figure C.23 shows the cache hit score for job input sample 1 with varying number of cacheable jobs. Both figures show that the cache hit scores are significantly lower since coordination only leads to minor changes in the number of cache hits. Therefore, the number of cache hits and thereby the cache hit score become very sensitive to fluctuations resulting in inhomogeneous and even negative cache hit scores for both sets of job input samples.

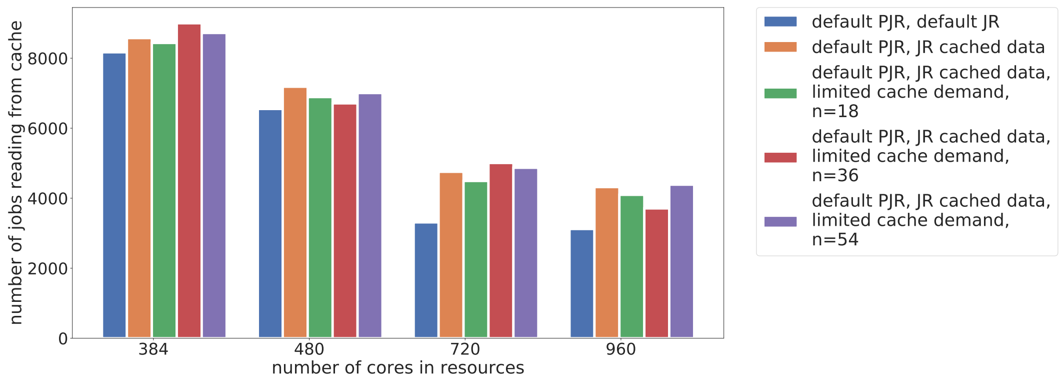


Figure C.17: Number of cache hits when coordinating with default `PRE_JOB_RANK` using job input sample 2.

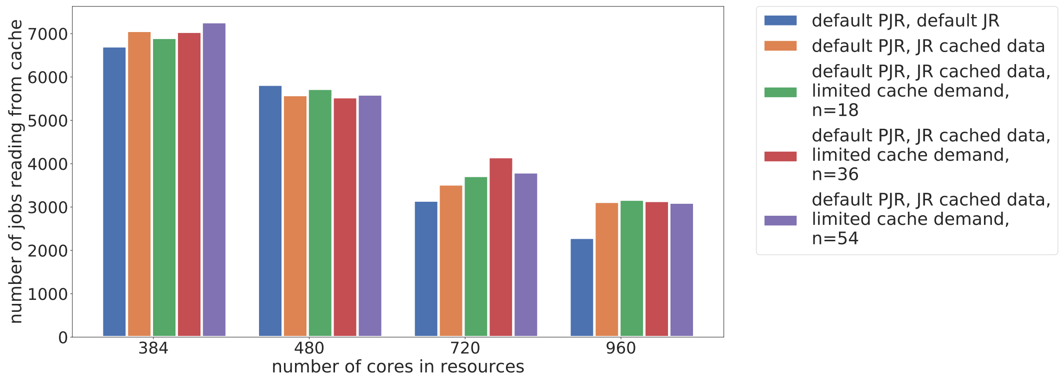


Figure C.18: Number of cache hits when coordinating with default `PRE_JOB_RANK` using job input sample 3.

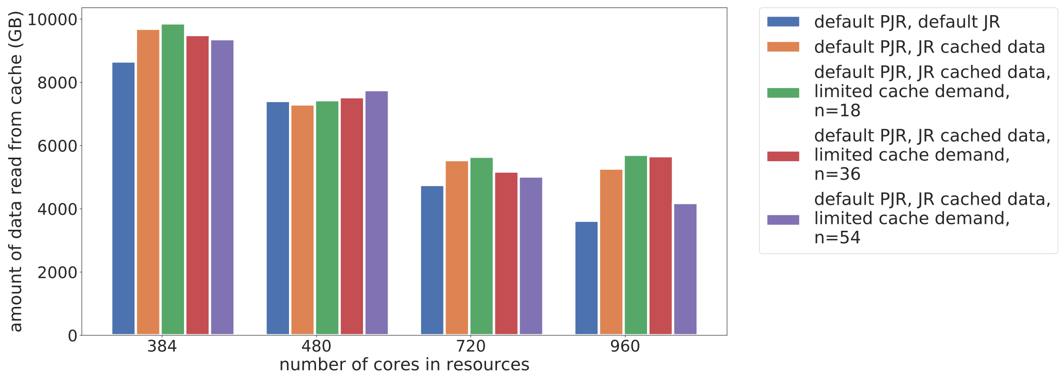


Figure C.19: Amount of data transferred from the cache when coordinating with default `PRE_JOB_RANK` using job input sample 1.

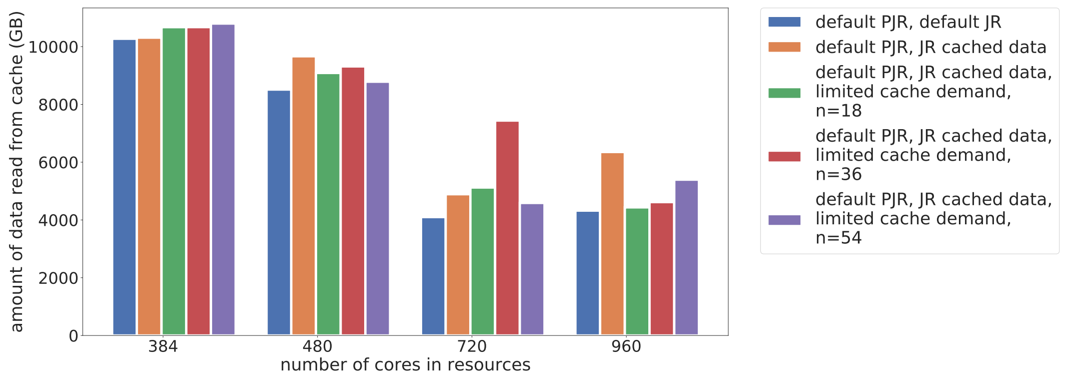


Figure C.20: Amount of data transferred from the cache when coordinating with default `PRE_JOB_RANK` using job input sample 2.

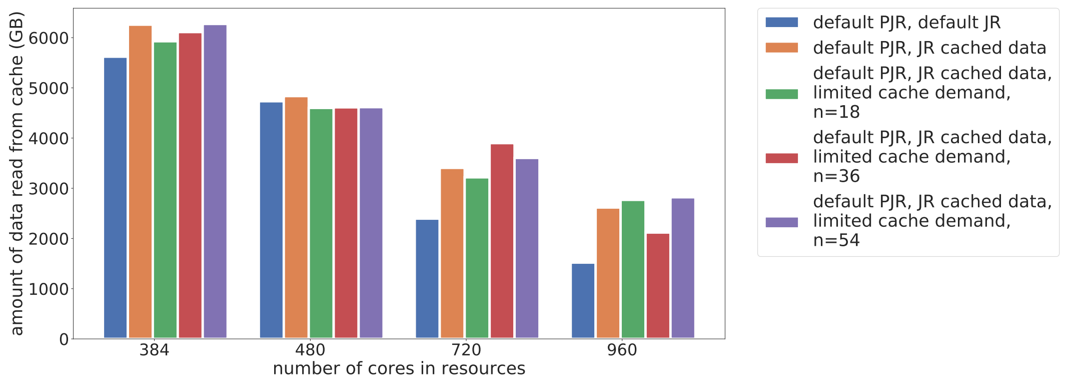


Figure C.21: Amount of data transferred from the cache when coordinating with default `PRE_JOB_RANK` using job input sample 3.

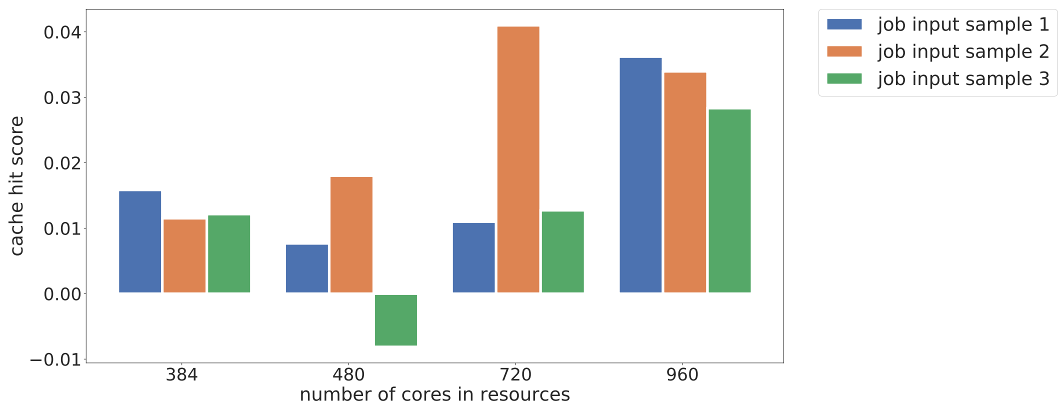


Figure C.22: Cache hit score of job input samples 1–3 for varying cluster sizes when coordinating with default `PRE_JOB_RANK`. As the number of cache hits does not change much when coordinating with `PRE_JOB_RANK`, the resulting cache hit score values are small and subject to fluctuations.

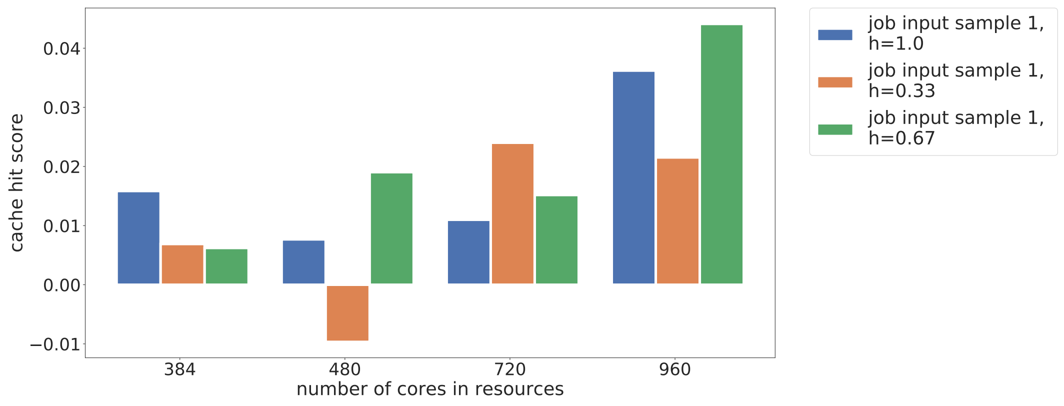


Figure C.23: Cache hit score of job input samples 1 for varying numbers of cacheable jobs and cluster sizes when coordinating with default `PRE_JOB_RANK`. As the number of cache hits does not change much when coordinating with `PRE_JOB_RANK`, the resulting cache hit score values are small and subject to fluctuations.

Appendix D

Basic considerations for implementing a caching policy

While realization options for the coordination of jobs are examined in this thesis and cache eviction policies are investigated in [51], the caching policy has yet to be examined in future studies. This section presents basic considerations on possible choices for the caching policy. They are based on the experience gained in the scope of this thesis and are supposed to provide a basis for considerations in the scope of future examinations of the subject.

As the name suggests, the purpose of the caching policy is to decide what data will be cached. It is employed when a job is processed on a worker node with access to a cache that does not contain the job's input data. Using the XRootD caching plugin described in section 3.5.1, the caching policy defaults to caching all requested files. While this is a simple implementation, often the resulting behavior is not ideal. Usually, caches will have a limited volume and caching every requested file of every job is likely to lead to a continuous flushing of cache content. Even with a suitable cache eviction policy it is unlikely that data remains in the cache until it is requested again if cache space has to be freed permanently to store new files. Consequently, the caching policy has to reduce the amount of cached data by determining what data is worth caching. As transparent caching is applied when using the XRootD caching plugin, this decision has to be made at the beginning of a job's processing.

Consequently, choosing a suitable caching policy becomes a complex task. In the following, some approaches that could be followed to face this challenge are presented in order of increasing complexity.

A first, rather simple approach is to reduce the flushing of cache contents by caching a randomly selected fraction of all requested files. While not all the files selected for caching are accessed again, the reduced amount of cached data could facilitate the work of the cache eviction policy. If the cache eviction policy is designed to keep files that are likely to be accessed again, even this simple caching policy could be beneficial.

However, an active preselection of files that are likely to be accessed again is expected to be more beneficial for the overall performance of the system. This preselection could be realized by identifying characteristics of workflows suitable for caching and then actively caching the input files of jobs belonging to these workflows. However, this approach arises with two challenges. First, the jobs have to be identified based on the metadata that they provide before processing. Second, the workflows and their characteristics are subject to evolving changes as a research group's activity and the workflows' configuration are time-dependent. Therefore, detecting workflows based on metadata as shown in section 4.2.3 is not practical as `JobBatchName` or paths to files change with the software environment or the people submitting the jobs. More technical metadata that are available before a job is processed such as the size of the jobs input data or requested resources are subject to changes as well and do not necessarily allow for identification of workflows on a long term basis either.

While identifying the workflows of jobs easily becomes a very complex task, the characteristics of jobs and workflows suitable to benefit from caching as presented in section 4.2.1 could be used in the caching policy. These criteria will most likely not allow a selection as specific as identifying jobs of workflows that are known to be well suited to benefit from caching. However, the continuous effort of identifying workflows well-suited for caching and extracting their properties is avoided and the overall procedure is less dependent on changes in the research groups activity or workflow configurations.

Not all characteristics of jobs that are suitable to benefit from caching can be used in a caching policy. For example a job's timing information like CPU time and walltime is not available before the job is finished. In contrast, the amount of data requested by the job is available before the processing. However, the amount of requested data and an estimated value for the CPU time could be used to indicate the data throughput required by a job. As shown in figures 4.6 and 4.8, this quantity allows separating jobs of analysis workflows such as the JEC, Dijet and Higgs analysis from jobs of other workflows. An estimate for the CPU time could come from the user or from historic data. If the users provided this estimate base on local execution of test jobs and add it to the job's ClassAd attributes, the average data throughput required by the job could easily be calculated. However, depending on efforts made by the user this is not an ideal strategy. However, the experience at ETP showed that users tend to become surprisingly capable of learning when being rewarded with increased job performance and supply of additional resources.

As an alternative to user estimates, historical data could be used. For example, metadata of jobs processed in the last few weeks could be monitored. Submitted jobs could be checked for similarity to already processed jobs. If similar jobs were found, an estimate of the CPU time could be calculated. However, such a procedure can easily become too elaborate or computationally expensive. Alternatively, a list of already requested files could be introduced and files could be cached if they are requested repeatedly within a certain time frame. Additionally, information about the content of all caches could be provided to avoid unnecessary replication of already cached files.

Concluding, the discussion of this section hints at the complexity of finding a suitable caching policy making the choice of caching policy an interesting and challenging topic for future research.

List of Figures

List of Tables

Bibliography

[1] The ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider." In: *JINST* 3 (2008), S08003. DOI: 10.1088/1748-0221/3/08/s08003.

[2] The CMS Collaboration. "The CMS Experiment at the CERN LHC." In: *JINST* 3 (2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004.

[3] The ALICE Collaboration. "The ALICE experiment at the CERN LHC." In: *JINST* 3 (2008), S08002. DOI: 10.1088/1748-0221/3/08/S08002.

[4] The LHCb Collaboration. "The LHCb Detector at the LHC." In: *JINST* 3 (2008), S08005. DOI: 10.1088/1748-0221/3/08/S08005.

[5] The CMS Collaboration. "Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC." In: *Phys. Lett.* B716 (2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021. arXiv: 1207.7235 [hep-ex].

[6] The ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC." In: *Phys. Lett.* B716 (2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].

[7] *High-Luminosity LHC*. https://home.cern/science/accelerators/high-luminosity-lhc. accessed 24.01.2020.

[8] J. Albrecht et al. "A Roadmap for HEP Software and Computing R&D for the 2020s." In: *Comput. Softw. Big Sci.* 3.1 (2019), p. 7. DOI: 10.1007/s41781-018-0018-8. arXiv: 1712.06982 [physics.comp-ph].

[9] *CMS detector design*. http://cms.web.cern.ch/news/cms-detector-design. accessed 24.01.2020.

[10] G. P. Salam. "Towards Jetography." In: *Eur. Phys. J.* C67 (2010), pp. 637–686. DOI: 10.1140/epjc/s10052-010-1314-6. arXiv: 0906.1833 [hep-ph].

[11] The CMS collaboration. *CMS celebrates the end of LHC Run 2*. https://cms.cern/news/end-of-LHC-Run2. accessed 27.4.2020.

[12] J. Shiers. "The Worldwide LHC Computing Grid (worldwide LCG)." In: *Comput. Phys. Commun.* 177 (2007), pp. 219–223. DOI: 10.1016/j.cpc.2007.02.021.

[13] *WLCG tiers*. accessed 07.03.2020. URL: https://wlcg-public.web.cern.ch/sites/wlcg-public.web.cern.ch/files/inline-images/WLCG-TiersJun14_v9-transp.png.

[14] The CMS Collaboration. *CMS computing: Technical Design Report*. Technical Design Report CMS. Submitted on 31 May 2005, page 64, 75. Geneva: CERN, 2005. URL: https://cds.cern.ch/record/838359.

[15] I Bird et al. *Update of the Computing Models of the WLCG and the LHC Experiments*. Tech. rep. CERN-LHCC-2014-014. LCG-TDR-002. Apr. 2014. URL: https://cds.cern.ch/record/1695401.

[16] K. Meier et al. "Dynamic provisioning of a HEP computing infrastructure on a shared hybrid HPC system." In: *J. Phys. Conf. Ser.* 762.1 (2016), p. 012012. DOI: 10.1088/1742-6596/762/1/012012.

[17] S. Richling, M. Baumann, and V. Heuveline. "ForHLR: a New Tier-2 High-Performance Computing Systemfor Research." In: *Proceedings of the 3rd bwHPC-Symposium: Heidelberg 2016* (2017), pp. 73–75. DOI: 10.11588/heibooks.308.418.

[18] *Helix Nebula Science Cloud*. http://www.hnscicloud.eu/. accessed 16.04.2020.

[19] M. Fischer et al. *MatterMiners/cobald: v0.10.0*. Version v0.10.0. Oct. 2019. DOI: 10.5281/zenodo.3469929.

[20] M. Giffels et al. *MatterMiners/tardis: ErUM Data Cloud Workshop Aachen*. Version 0.3.0. Feb. 2020. DOI: 10.5281/zenodo.3688615.

[21] M. Fischer. "Coordinated Caching for High Performance Calibration using $Z \to \mu\mu$ Events of the CMS Experiment." PhD thesis. Karlsruhe, 2016. DOI: 10.5445/IR/1000067354.

[22] R. Caspart et al. "Modeling and Simulation of Load Balancing Strategies for Computing in High Energy Physics." In: *EPJ Web Conf.* 214 (2019), p. 03027. DOI: 10.1051/epjconf/201921403027.

[23] R. Caspart et al. "Advancing throughput of HEP analysis work-flows using caching concepts." In: *EPJ Web Conf.* 214 (Jan. 2019), p. 04007. DOI: 10.1051/epjconf/201921404007.

[24] W. L. Chang, N. Grady, and NBD-PWG NIST Big Data Public Working Group. *NIST Big Data Interoperability Framework: Volume 1, Big Data Definitions*. Special Publication (NIST SP) - 1500-. 2015. DOI: 10.6028/NIST.SP.1500-1.

[25] *HEP-SPEC06 (HS06)*. https://w3.hepix.org/benchmarking.html. accessed 03.02.2020.

[26] *Computing and Software - Public Results.* `https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults`. accessed 26.03.2020.

[27] A. Dorigo et al. "XROOTD/TXNetFile: a highly scalable architecture for data access in the ROOT environment." In: (Jan. 2005), p. 46. URL: `https://www.researchgate.net/publication/234817900_XROOTDTXNetFile_a_highly_scalable_architecture_for_data_access_in_the_ROOT_environment`.

[28] S. Lange. *Analyse und Vorhersage von Nutzungsverhalten und Ressourcenverbrauch in einer wissenschaftlichen Cloud-Umgebung.* Master thesis. 2018.

[29] M. Barisits et al. "Rucio: Scientific Data Management." In: *Comput. Softw. Big Sci.* 3 (Aug. 2019), p. 11. ISSN: 2510-2044. DOI: `10.1007/s41781-019-0026-3`. arXiv: `1902.09857 [cs.DC]`.

[30] R. Egeland, T. Wildish, and S. Metson. "Data transfer infrastructure for CMS data taking." In: *PoS* ACAT08 (2008), p. 033. DOI: `10.22323/1.070.0033`.

[31] The ATLAS collaboration. *The PanDA Production and Distributed Analysis System.* `https://twiki.cern.ch/twiki/bin/view/PanDA/PanDA`. accessed 12.03.2020. 2018.

[32] T. Maier et al. "Performance and impact of dynamic data placement in ATLAS." In: *EPJ Web Conf.* 214 (2019), p. 04025. DOI: `10.1051/epjconf/201921404025`.

[33] Yutaro Iiyama et al. "Dynamo – Handling Scientific Data Across Sites and Storage Media." In: (Mar. 2020). arXiv: `2003.11409 [cs.DC]`.

[34] *XRootD Proxy File Cache - Readme.* `https://github.com/xrootd/xrootd/tree/master/src/XrdPfc`. accessed 08.03.2020.

[35] N. Hartmann. *Xcache status.* 2020. URL: `https://indico.physik.uni-muenchen.de/event/30/contributions/267/attachments/117/177/nikolai_lmu-atlas-belle-meeting-24.02.2020_xcache.pdf`.

[36] E. Fajardo et al. "A federated Xrootd cache." In: *J. Phys. Conf. Ser.* 1085 (Sept. 2018), p. 032025. DOI: `10.1088/1742-6596/1085/3/032025`.

[37] T. Tannenbaum et al. "Condor – A Distributed Job Scheduler." In: *Beowulf Cluster Computing with Linux.* Ed. by Thomas Sterling. MIT Press, Oct. 2001.

[38] M. Soysal, M. Berghoff, and A. Streit. "Analysis of Job Metadata for Enhanced Wall Time Prediction." In: *Job Scheduling Strategies for Parallel Processing.* Ed. by D. Klusáček, W. Cirne, and N. Desai. Cham: Springer International Publishing, 2019, pp. 1–14. ISBN: 978-3-030-10632-4.

[39] F. Stober et al. "The swiss army knife of job submission tools: grid-control." In: *CoRR* abs/1707.03198 (2017). arXiv: `1707.03198 [cs.DC]`.

[40] H. Kirschenmann. "Determination of the Jet Energy Scale in CMS." In: *J. Phys. Conf. Ser.* 404 (Dec. 2012), p. 012013. DOI: `10.1088/1742-6596/404/1/012013`.

[41] A.G. Agocs, G.G. Barnafoldi, and P. Levai. "Underlying Event in pp Collisions at LHC Energies." In: *6th International Workshop on High-pT physics at LHC*. Aug. 2011. arXiv: `1108.1705 [hep-ph]`.

[42] A. Perloff. "Pileup measurement and mitigation techniques in CMS." In: *J. Phys. Conf. Ser.* 404 (2012). Ed. by N. Akchurin, p. 012045. DOI: `10.1088/1742-6596/404/1/012045`.

[43] V. Khachatryan et al. "Jet energy scale and resolution in the CMS experiment in pp collisions at 8 TeV." In: *JINST* 12.02 (2017), P02014. DOI: `10.1088/1748-0221/12/02/P02014`. arXiv: `1607.03663 [hep-ex]`.

[44] The CMS collaboration. "Determination of jet energy calibration and transverse momentum resolution in CMS." In: *JINST* 6.11 (Nov. 2011), P11002. DOI: `10.1088/1748-0221/6/11/p11002`. arXiv: `1107.4277 [physics.ins-det]`.

[45] T. Berger. "Jet energy calibration and triple differential inclusive cross section measurements with Z ($\rightarrow \mu\mu$) + jet events at 13 TeV recorded by the CMS detector." PhD thesis. Karlsruher Institut für Technologie (KIT), 2019. 139 pp. DOI: `10.5445/IR/1000104286`.

[46] The CMS Collaboration. *Jet energy scale and resolution performance with 13 TeV data collected by CMS in 2016-2018*. Tech. rep. CMS-DP-2020-019. Apr. 2020. URL: `https://cds.cern.ch/record/2715872`.

[47] *Kappa Analysis - Karlsruhe Package for Physics Analysis*. `https://github.com/KappaAnalysis`.

[48] *Excalibur - Analysis repository for Z+Jet studies*. `https://github.com/KIT-CMS/Excalibur`.

[49] J. Berger et al. "ARTUS - A Framework for Event-based Data Analysis in High Energy Physics." In: (2015). arXiv: `1511.00852 [hep-ex]`.

[50] *Karma*. `https://github.com/dsavoiu/Karma`.

[51] P. Skopnik. Master thesis, to be published. 2020.

[52] R. Raman, M. Livny, and M. Solomon. "Matchmaking: Distributed Resource Management for High Throughput Computing." In: *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*. 1998.

[53] *Configuration Macros - HTCondor Manual documentation*. `https://htcondor.readthedocs.io/en/latest/admin-manual/configuration-macros.html`. accessed 27.01.2020.

[54] *User Priorities and Negotiation - HTCondor Manual documentation.* `https://htcondor.readthedocs.io/en/v8_9_5/admin-manual/user-priorities-negotiation.html`. accessed 27.01.2020.

[55] E. Kühn et al. *tfesenbecker/lapis: Initial release.* Version v0.1. May 2020. DOI: `10.5281/zenodo.3819500`.

[56] M. Fischer and E. Kühn. *MaineKuehn/usim: μSim - Simply Simulate.* Version v0.4.3. May 2020. DOI: `10.5281/zenodo.3813588`.

[57] E. Kühn et al. *MatterMiners/lapis: v0.4.0.* Version v0.4.0. May 2020. DOI: `10.5281/zenodo.3822379`.

[58] M. Sauter. *Effizienzsteigerung von Datenanalyse in der Teilchenphysik durch Datenlokalität.* Master thesis. 2019.

[59] *classad - Python package to parse and interpret HTCondor classads.* URL: `https://github.com/MaineKuehn/classad`.

[60] M. J. Schnepf. "Triple Differenetial Cross Section Measurement with Z($\rightarrow \mu\mu$) + Jet Events at 13 TeV Recorded by the CMS Detector Using Oportunistic Computing Resources." preliminary title, to be published. PhD thesis. 2020.

[61] *Job ClassAd Attributes - HTCondor Manual reference.* `https://htcondor.readthedocs.io/en/v8_9_5/classad-attributes/job-classad-attributes.html`. accessed 30.04.2020.

[62] The CMS Collaboration. "Performance of the CMS muon detector and muon reconstruction with proton-proton collisions at $\sqrt{s} = 13\,\text{TeV}$." In: *JINST* 13.06 (June 2018), P06015. ISSN: 1748-0221. DOI: `10.1088/1748-0221/13/06/p06015`. arXiv: `1804.04528 [physics.ins-det]`.

[63] M. Tanabashi et al. "Review of Particle Physics." In: *Phys. Rev. D* 98.3 (2018), p. 030001. DOI: `10.1103/PhysRevD.98.030001`.

[64] The CMS Collaboration. *Jet algorithms performance in 13 TeV data.* Tech. rep. CMS-PAS-JME-16-003. Geneva: CERN, 2017. URL: `https://cds.cern.ch/record/2256875`.

[65] *Grid Computing Centre Karlsruhe (GridKa).* `http://www.gridka.de/cgi-bin/frame.pl?seite=/welcome.html`. accessed 02.05.2020.

Danksagung

Mein ganz besonderer Dank gehört Prof. Dr. Günter Quast. Ihm möchte ich sowohl für die engagierte und motivierende Betreuung dieser Arbeit als auch für seinen unermüdlichen Einsatz für die Modernisierung der Lehre im Physikstudium am KIT danken. Sein Einfluss hat meinen Studienverlauf maßgeblich geprägt und mein Interesse an Themen über die Physik hinaus geweckt.

Besonders danken möchte ich auch Priv.-Doz. Dr. Klaus Rabbertz für die Übernahme des Korreferats und das Verständnis für den hohen technischen Anteil dieser Masterarbeit.

Darüber hinaus möchte ich der gesamten Arbeitsgruppe für die angenehme Arbeitsatmosphäre, zahlreiche Hilfestellungen, produktive und weniger produktive Unterhaltungen und nicht zuletzt das Korrekturlesen beim Schreiben dieser Arbeit bedanken. Besonders möchte ich dabei natürlich dem Computing Team am Campus Süd, Christoph Heidecker, Matthias Schnepf, Florian von Cube und Maximilian Horzela danken für viele Diskussionen, das geduldige Beantworten vieler Fragen sowie generell die gute und mitunter auch sehr unterhaltsame Zusammenarbeit.

Mein besonderer Dank gilt außerdem René Caspart, Eileen Kühn, Max Fischer und Manuel Giffels in vielen Aspekten. Zuerst natürlich für die freundliche Aufnahme am Campus Nord sowie für die Inspiration für und Unterstützung bei der inhaltlichen Umorientierung dieser Arbeit. Darüber hinaus für die vielen, hilfreichen Diskussionen mit und ohne Kaffee, eure Motivation und allgemein die angenehme Zusammenarbeit. Besonders dankbar bin ich für eure Betreuung in den letzten Monaten. Ihr habt mich fachlich, menschlich, in vielen verschiedenen Situationen und zu jeder Uhrzeit auf die beste Art und Weise unterstützt, die ich mir hätte wünschen können.

Nicht zuletzt möchte ich meiner Familie und meinen Freunden, besonders aber meinen Eltern für die Unterstützung während dieses Studiums danken.