Karlsruher Institut für Technologie

Coordinated Caching for High Performance Calibration using $Z \to \mu\mu$ Events of the CMS Experiment

Max Fischer

Zur Erlangung des akademischen Grades eines
DOKTORS DER NATURWISSENSCHAFTEN
von der Fakultät für Physik des
KIT – Karlsruher Institut für Technologie

genehmigte

DISSERTATION

von

Dipl.-Phys. Max Fischer
aus Karlsruhe

Tag der mündlichen Prüfung: 22.07.2016

Referent: Prof. Dr. Günter Quast
Korreferent: Prof. Dr. Achim Streit

Contents

Chapter 1

Introduction

> It's been a global effort, a global success.
> It has only been possible because of the extraordinary achievements of
> the experiments, infrastructure, and the grid computing.
>
> Rolf Heuer

Over the past decades, particle physics has been a major driving force for scientific progress. Particle physics has been probing nature with steadily increasing energy, precision, and scope. This has uncovered many fundamental rules which govern our universe.

At the same time, the impact of particle physics research reaches well beyond its own scientific fields. Many technological and scientific challenges are posed by particle physics research. This has created a dedicated, interdisciplinary effort driving technological development in general.

Today, the Large Hadron Collider (LHC) and its experiments are popularly seen as synonymous with experimental particle physics research. In fact, the LHC is the biggest, most powerful particle accelerator and collider built to date. It enables the controlled study of elementary particle interactions at unprecedented energies. In addition, its high collision frequency enables the study of extremely rare interactions.

As a result of the collision frequency and level of detail recorded by the LHC detectors, enormous amounts of data are generated. Thus, data analysis by the LHC collaborations require dedicated, powerful data analysis resources. This challenge has only been solved by pioneering developments of both software and infrastructure. Most prominently, collaborations have created the Worldwide LHC Computing Grid (WLCG), the largest noncommercial computing grid in the world.

The LHC collaborations steadily face new challenges in their research. Following the end of the recent shutdown period of the LHC, physics research is resumed under new conditions. Increased collision energy and frequency enable new physics research. In return, performing precision analyses has become even more challenging. At the

same time, the increase in data volume poses a severe challenge for data management and processing.

The research presented in this thesis is firmly rooted in these two areas of particle physics. It is an interdisciplinary work on physics research as well as applied computer science for physics analyses: The determination of the jet energy scale at the LHC creates the foundation for future, high precision physics analyses. This is enhanced by a new approach to physics data analysis, which ensures continued viability of performing physics analyses at high statistics.

End user analyses in modern particle physics process huge amounts of data. Research relies on regular, iterative improvements of analyses based on previous results. This makes analysis speed critical for research progress.

To keep up with increasing data volumes for end user analyses, a new approach to data processing infrastructures has been developed as part of this work. It introduces the data processing paradigm of data locality to particle physics analyses: A distributed caching infrastructure is used to optimise access to data in batch systems. This reflects the distributed, evolving end user data analysis workflows used in particle physics.

A prototype is already in use and demonstrates its applicability for end user data analysis. Compared to existing infrastructure, the prototype improves analysis speed by several factors. Most prominently, it is used to speed up calibration of the jet energy scale of the CMS experiment.

A prominent feature of particle collisions at the LHC are so-called jets. These groups of particles originate from each parton of the actual collision. In many analyses, jets are a prominent signature due to their abundance in collision events. The precise measurement of jet features is critical for high precision analyses.

Since jets are compound objects, calibration of jet measurements is a complex process. As part of this work, the jet energy scale of the CMS detector has been calibrated using Z + Jet events. Being the last step of calibration required for further analyses, both precision and speed are critical.

With the second data taking period of the LHC, jet calibration has to account for new experimental conditions. Several techniques to mitigate biases from Pileup collisions have been studied and calibrated. To achieve adequate precision, two channels are used in parallel, namely $Z \to \mu\mu$ + Jet and $Z \to ee$ + Jet. The achieved precision of CMS jet energy calibration is already approaching the precision of the previous data taking period.

This thesis aims at readers from both particle physics and computer science. Chapter 2 provides a brief overview on the context of this work. The structure and working principle of the CMS detector is outlined. Next, both physical phenomenology

and technical reconstruction of jets are detailed. Finally, infrastructure and workflows used for end user analysis are sketched.

In addition to Chapter 2, a list of acronyms and a glossary is provided after the appendix. These define and briefly explain commonly used technical terms as well as jargon.

Chapter 3 is focused on the new analysis infrastructure. At first, the viability of caching for particle physics end user analyses is outlined; this also includes an overview of existing technologies and related work. Following this, the new approach of coordinated caching for batch systems is introduced and discussed. Next, the prototypical implementation of this approach is detailed. Finally, performance of the prototype is evaluated based on physics analyses deployed on a test infrastructure.

Chapter 4 is dedicated to the calibration using Z + Jet events. A brief description of the calibration procedure used by the CMS collaboration is provided. Afterwards, the analysis of Z + Jet events is described in general. Finally, the calibration for data recorded by the CMS detector in 2015 is detailed.

Individual conclusions and outlooks for parts of the thesis are provided in each main chapter. In addition, Chapter 5 provides an overall conclusion and outlook based on the synergies between both topics: The distributed, coordinated caching as well as the calibration using Z + Jet events.

Chapter 2

Background

The work presented in this thesis is related to many different scientific fields. Obviously, a basic understanding of the detector and methodology of experimental particle physics is required. The processes themselves are subject to considerations of theoretical physics. Finally, the actual realisation is based in the domain of distributed computing and applied computer science.

Within the scope of this chapter, a basic overview on key topics is provided. Covering both computer science and particle physics, descriptions are meant to be adequate for readers from either field. Topics are described in a condensed, fundamental form as required to understand key arguments of later chapters. Thorough descriptions can be found in literature and are out of scope for this chapter.

2.1 The CMS Experiment at the LHC

To study particle interactions at high energies in controlled conditions, particle colliders of different designs are used. The Large Hadron Collider (Large Hadron Collider (LHC)) [1] is the biggest, most powerful of these machines built to date. It was designed to study particle physics in a controlled environment at unprecedented energies. While it has become famous for the discovery of the Higgs boson [2, 3], its scientific scope is much broader.

The LHC is a synchrotron, featuring two adjacent beam pipes. Groups of particles, so called bunches, are accelerated in opposite direction in these pipes. The beams cross at four interaction points, at which high energy collisions of bunches may occur. The main operation mode of the LHC is the acceleration of proton beams.

The advantage of using protons is the reduced loss from synchrotron radiation compared to electron accelerators. Thus, high collision energies can be achieved. A major downside is that protons are not elementary particles, which has significant implications. For example, the fraction of energy in each interaction of two colliding protons' constituents is unknown. Also, elementary particles not part of the main interaction may cause secondary, soft interactions.

After its first data taking period ended in 2013, the LHC underwent upgrades and maintenance. The second data taking period, called *Run II*, started in April 2015 with improved performance. Particle collisions occur at a center-of-mass energy of 13 TeV. The interval between bunches crossing was initially 50 ns but has been reduced to 25 ns in mid 2015.

Particle collision are studied at and around the interaction points. Here, specialised detectors are placed, each designed with a distinct goal in mind. One of the four major LHC detectors is the CMS general purpose detector [4].

2.1.1 The CMS Detector

CMS stands for Compact Muon Solenoid, which signifies the main design features of the detector: It is compact, with little space between subdetectors, placing them as close as possible around the interaction point. It features dedicated, precise muon subdetectors, allowing for clear signatures of processes involving these particles. Finally, it contains a powerful solenoid magnet, enabling high precision momentum measurements. A schematic view of the detector is shown in Figure 2.1.

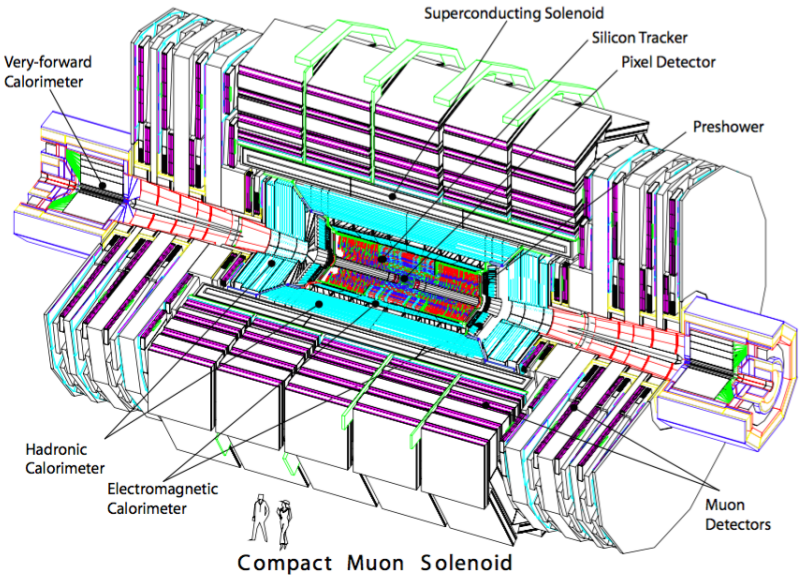


Figure 2.1: Schematic View of the CMS Detector: The CMS detector is designed with a barrel shape and oriented along the beam pipe. Two endcaps close off the edges of the barrel. The forward region is positioned directly around the beam pipe. In each region, several subdetectors are positioned in layers around the interaction point. [5]

Geometrically, the detector follows a barrel shape positioned around the interaction point. The central component, called barrel, is designed as a hollow cylinder aligned along the beam pipe. The barrel is closed off by two circular planes, the endcaps. Protruding from the endcaps is the smaller forward section, which wraps around the beam pipes.

In each of these regions, several subdetectors are present to detect different types of interactions. These subdetectors are positioned in layers stacked on top of each other. The layers are always stacked in the same order in respect to the interaction point.

Not all types of layers are present in every region, and there are technical differences between regions. In general, the following layers are available, in this order starting closest to the interaction point:

- The **tracker** records the trajectory of charged particles.
- The **electromagnetic calorimeter** measures the energy of leptons and photons. It is commonly referred to as ECAL.
- The **hadronic calorimeter** measures the energy of hadronic particles. It is commonly referred to as HCAL.
- The **solenoid** is not a subdetector. However, its magnetic field bends particle trajectories due to charge and momentum.
- The **muon system** records trajectories of muons. It is effectively a second tracker for these particles.

The detector output is an ordered collection of electronic signals. These represent the location and intensity of particles interacting with individual subdetectors. To deduce the type and properties of these particles, they must be reconstructed from the detector output.

2.1.2 The Particle Flow Approach

Each subdetector is sensitive to different properties of particles, as shown in Table 2.1. This allows a direct identification of many particles via these properties. For example, if the tracker records a trajectory pointing at a deposit in the ECAL, this is likely caused by an electron.

The combination of all subdetector signals for reconstruction is called the Particle Flow [6] algoritm. Objects reconstructed with this approach are called Particle Flow candidates. These are the de facto standard for what is used as particles and their features in analyses.

Strictly speaking, the particle flow candidates are only an interpretation of detector signals. While the method is superior to previous methods, its performance varies

Table 2.1: PARTICLE IDENTIFICATION WITH SUBDETECTORS

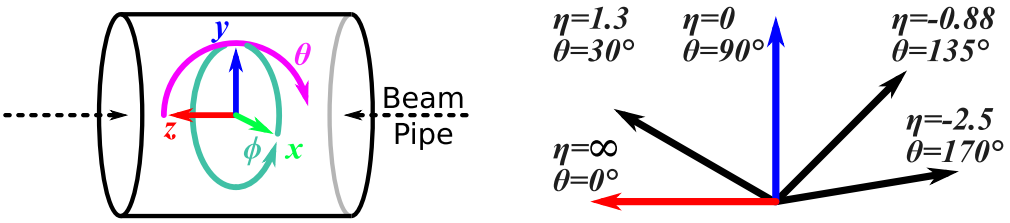
Particle	Identified by			
	Tracker	ECAL	HCAL	Muon System
γ	×	√	×	×
e	√	√	×	×
μ	√	×	×	√
p	√	×	√	×
n	×	×	√	×

with the subdetectors available. This makes it essential to take into account where particles interact with the detector.

2.1.3 Coordinate Systems

Several different coordinate systems are commonly used for describing the CMS detector. Each is designed for a different purpose. All coordinate systems have in common that they are centered to the interaction point. An illustration is shown in Figure 2.2.

The default 3-dimensional cartesian coordinate system is right handed. Its x-axis points to the center of the LHC ring, the y-axis upwards and the z-axis along the beam pipe. Based in this, several derived coordinates are defined. These are more suitable



(a) Cartesian and Spherical Coordinates

(b) Relation of η and θ

Figure 2.2: Coordinate Systems Used for the CMS Detector: The default cartesian coordinate system is right handed and oriented with respect to the beam pipe in z and LHC ring in x. The azimuthal and polar orientation is commonly expressed via the respectives angles ϕ and θ. For describing physical processes, the pseudorapidity η is commonly used instead of the polar angle.

for describing particle physics processes. In addition, the positive and negative z-axis are defined as forwards and backwards direction, respectively.

Reflecting the barrel shape of the detector, the azimuthal angle ϕ resides in the x-y-plane. It starts at the x-axis, meaning that $\phi = 0$ and $\phi = \pi/2$ denote horizontal and vertical orientation, respectively. As both the detector geometry and physics processes are isotropic in ϕ, it is most relevant for expressing relative orientations.

Since the x-y-plane is transversal to the particle beams, the initial momentum in this plane is negligible. This makes any non-negligible, final momentum or energy in this plane important physical quantities. The projection of a quantity onto this plane is denoted by a subscript T. For example, p_T is the momentum in the transversal plane.

Multiple coordinates are available to represent the polar orientation. The obvious one is the polar angle θ, defined with respect to the z-axis, i.e. forward direction. However, the rapidity y and pseudorapidity η are more adequate for describing physics processes.

$$\eta = \frac{1}{2} \ln \left(\frac{|p| + p_z}{|p| - p_z} \right) = - \ln \left(\tan \frac{\theta}{2} \right)$$
$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right)$$

In principle, the rapidity y is more adequate for describing physics: it is invariant under Lorentz transformations along the z-axis. Yet, it implicitly depends on precisely determining an object's mass, which can be difficult or outright impossible at the LHC collisions: The partons colliding may carry an arbitrary fraction of proton energy, making the actual collision energy unknown.

Being a purely geometric quantity, the pseudorapidity η is trivial to deduce and more robust. In addition, pseudorapidity equals rapidity for massless particles.

$$\lim_{m \to 0} y = \eta$$

2.1.4 Detector Regions

The barrel shape of the CMS detector has some drawbacks for measurements. While the detector is roughly isotropic in ϕ, this is not the case in θ. This creates several distinct regions between which detection performance, precision, and coverage varies.

In general, these regions correspond to the detector elements barrel, endcaps, and forward. More in detail, individual sub-detectors partially overlap and transitions regions are realised differently. In the scope of this thesis, several distinct regions are of interest, as shown in Figure 2.3.

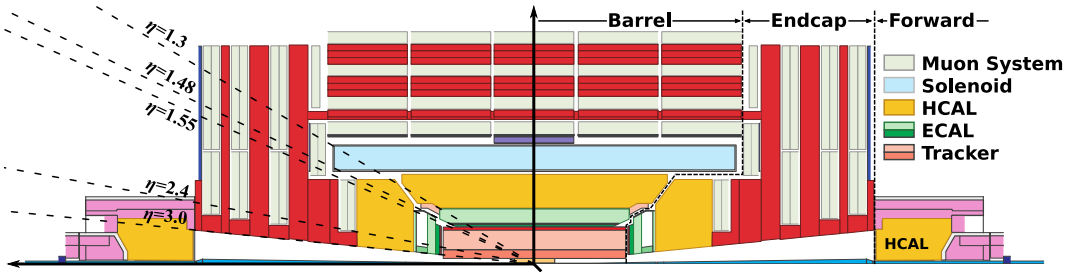


Figure 2.3: Longitudinal Slice of the CMS Detector: The barrel shape along the z-axis leads to distinct regions in the polar orientation. Due to overlap of subdetectors in barrel, endcap, and forward region, the detector precision varies between certain areas. Examples include: The calorimeters of the barrel only cover a region of $\eta < 1.3$. ECAL segments of barrel and endcap join in the transition region of $1.48 < \eta < 1.55$, leading to reduced precision. The total coverage of muon system and ECAL is $\eta < 2.4$ and $\eta < 3.0$, respectively.

The coverage of the barrel with calorimeters is limited to $\eta < 1.3$. Here, energy resolution is best, especially for neutral particles. Since most events of interest lead to high activity in this region, it is critical for many analyses.

Subdetectors targeting leptons are only present in barrel and endcaps, not the forward region. While the muon system provides precise measurements for muons, it only covers the region of $\eta < 2.4$. The ECAL extends further, detecting electrons in the slightly larger area of $\eta < 3.0$. However, the transition between barrel and endcap leads to reduced ECAL precision in the area of approximately $1.48 < \eta < 1.55$.

2.2 Jets in the LHC

The Standard Model of Particle Physics does not allow for isolated partons, i.e. quarks and gluons. Yet, the physical processes taking place in proton-proton collisions practically always include such a state. In practice, the separation of a parton from the interaction causes new partons to form around it. In place of an isolated parton, detectors only register a group of particles instead. Such a group is commonly referred to as a *jet*.

2.2.1 Origin of Jets

The inability to detect individual partons is due to *confinement*: all visible matter is colour neutral, meaning colour charged particles must always be in a bound state. This is due to the strong force, which acts on colour charged particles. In contrast to other forces, the potential of the strong force increases with distance. Thus, a parton

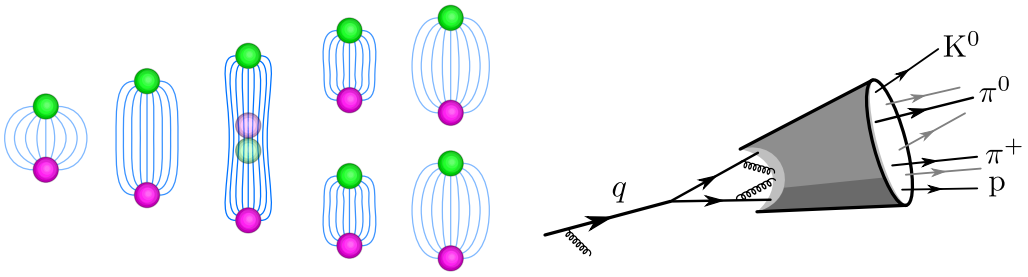
separating from its bound state is met by an increasing potential which can absorb any kinetic energy.

While a parton cannot be freed from its bound state, the confining potential itself may be broken. As a bound state is split, the energy contained in the field binding it increases. At some point, enough energy is available to create a pair of quark and antiquark. Each of these binds to a portion of the initial bound state. The net result are two colour neutral bound states, which may freely separate from each other.

With the energy transferred in collisions at the LHC, each separated parton may cause this process repeatedly. This creates a shower of partons in place of the initial parton. In turn, these partons will bond to hadrons. Further decays of these hadrons may also result in leptons.

The result of this formation, bonding and possible decay is a collimated shower of particles, as shown in Figure 2.4. This shower is roughly comparable in energy and orientation to the parton it originated from. Thus, the shower is used in lieu of the undetectable parton.

Reflecting this, the shower is represented as a single object, a jet. As such, a jet is an abstraction, not a physical object itself. A thorough understanding and modelling is required to use jets in precision measurements.



(a) Splitting of Colour Bound States [7] **(b)** Formation of Particle Jet from Parton [7]

Figure 2.4: Formation of Jets: Jets form due to colour confinement created by the strong force, as visualised in Figure 2.4a. The potential of the strong force increases with distance, potentially until forming new particles is favorable. Partons separating from a bound state thus create new bound states, preventing unbound partons. If a parton is separated with high energy a cluster of particles forms, as visualised in Figure 2.4b. A cascade of bound states splitting iteratively creates a shower of particles. This final shower is abstracted as a *jet*, which is studied in place of the initial parton.

2.2.2 Jet Reconstruction in the CMS Experiment

Abstracting a parton shower to a single object is not unambiguous. Results of *jet clustering*, i.e. combining a group of particles to a jet, depend on the algorithm used. In addition, algorithms vary in the robustness of their underlying model.

In the CMS experiment, the anti-k_t algorithm [8] is commonly used[1]. It uses a distance measure between two particles, as defined in Equation 2.1. The measure expresses how close the particles are to being elements of the same jet. In addition, a dynamic threshold is used, as defined in Equation 2.2.

$$d_{ij} = \min\left(p_{T,i}^a, p_{T,j}^a\right) \frac{\Delta R_{i,j}^2}{R^2} \quad \text{, with } a = -2 \tag{2.1}$$

$$d_k = p_{T,k}^a \quad \text{, with } a = -2 \tag{2.2}$$

$$\Delta R_{i,j}^2 = (\eta_j - \eta_i)^2 + (\phi_j - \phi_i)^2 \tag{2.3}$$

The measure is used for an iterative clustering of jets in the event. In each iteration, the pair of particles (i, j) with the smallest d_{ij} are merged. The merged object is treated as a regular particle for calculating d_{ij} in any following iteration. This effectively merges groups of similarly oriented particles, preferring clusters of high momentum and density.

Any particle k for which d_k is smaller than any d_{ij} is removed from further clustering. Since this requires $\Delta R_{i,k}^2 > R^2$ for any (i, k), it implies that R is a hard limit for the angular jet size.

The anti-k_t algorithms is robust and generally outperforms other common algorithms. Yet, it is still an idealised model. Events recorded by the CMS detector include noise and other background effects. This limits the correspondence between clustered jets and initial parton. Chapter 4 details the process of correcting jets to remedy this, and the correction procedure performed as part of this tesis.

2.3 Computing in High Energy Physics

Scientific research at the LHC is not only a challenge for detector engineering, theoretical physics, and experimental physics. It requires significant effort and resources to handle the vast amounts of data. Being the first to perform research at this scope, the LHC collaborations have developed their own means to manage data.

With research performed by worldwide, publicly funded collaborations, data handling and processing reflects this. On the one hand, data must be available to every collaborator. On the other hand, resources should be contributed equally. To

[1] The name is derived from its initial formulation. There, k_{ti}^{2p} corresponds to what is called $p_{T,i}^a$ in this thesis.

facilitate this, the LHC collaborations have joined their computing resources in a single, worldwide infrastructure.

2.3.1 The WLCG

The Worldwide LHC Computing Grid (WLCG) [9] is primarily composed of hundreds of computing centres. The centres themselves are largely independent, but provide their services via common middlewares and protocols. Each centre provides processing and storage capacities for one or more LHC collaboration. In turn, each collaboration has its own computing model for using these resources. However, the general concepts are the same.

Historically, each computing centre of the WLCG belongs to a specific tier, defining its role. This distinction has become less strict, but it still reflects the principle of data handling. *Tier 0* is closest to the detector, storing raw detector data and performing basic reconstruction. *Tier 1* distributes data globally, creating and storing data ready to be analysed. *Tier 2* provides processing resources, generating simulations and allowing users to analyse data. The tiers are organised hierarchically, as shown in Figure 2.5.

The LHC collaborations use the WLCG for a large variety of computing tasks. Reconstruction workflows convert raw detector data to high level data suitable for physics analysis. This mainly requires processing power, but also considerable data input. Simulation workflows simulate collision events and detector responses. Such workflows practically only depend on processing power. Analysis workflows use data from reconstruction and simulation to analyse physics processes. Their requirements vary, possibly consuming large amounts of data and processing resources.

The size and complexity of the WLCG means that it is mostly used for large, automated workflows. This makes it too inflexible for performing the highly iterative end user analyses. Thus, most analyses are performed in large parts on the local computing resources. Since these resources are loosely coupled to the WLCG, they are commonly called *Tier 3*.

2.3.2 Tier 3 Analysis Facilities

Resources of the *Tier 3* provide access to the WLCG but do not have an active role in its workflows. These resources are managed and used by small groups or individuals. Commonly, they range from processing clusters at universities down to individual workstations. Compared to other Tiers, the smaller scope allows to optimise usability and performance for specific tasks.

Tier 3 analysis facilities provide local batch processing resources to handle the data volume and complexity of LHC analyses. Examples of this are the National Analysis

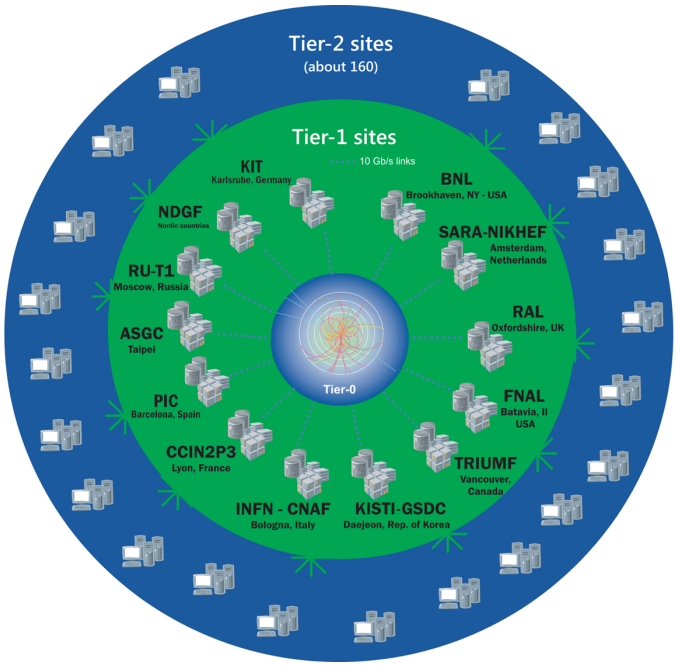


Figure 2.5: The WLCG Tier Structure: The WLCG is composed of hundreds of computing centres, each assigned to a tier. Based on the tiers, computing centres are organised hierarchically: Data from detectors is provided by the Tier 0 to several Tier 1 centres. In turn, each Tier 1 centre serves multiple Tier 2 centres. [10]

Facility (NAF) and the computing resources at the Institut für Experimentelle Kernphysik (IEKP). Such facilities commonly provide three types of resources:

- *Portal*s are interactive login nodes for users. They allow scientists to interactively prepare, test, and manage their analyses.
- *Fileserver*s provide large volumes of storage, sufficient to hold multiple datasets. They are accessible from all other components.
- *Worker Node*s provide processing resources, joined via a *Batch System*. This allows the automatic, parallel execution of analysis applications.

2.3.3 End User Data Analyses

A typical end user analysis consists of multiple stages, each performed in a different environment. In general, each stage reduces data volume and turnaround time. In turn, later stages are repeated more frequently.

Several CMS analyses performed by groups at the KIT use the Artus [11] framework. These analyses can be considered as exemplary for the scope of this work. Figure 2.6 shows the Artus analysis workflow. Most steps of an Artus analysis are trivially parallelised: each collision event is processed individually, without accessing data from other events.

The starting point are datasets provided by the CMS collaboration. These are derived either from the detector or simulations. At this stage, collision events are described via contained logical and physical objects, such as particles. Due to the size and to ensure their availability, such datasets are published in the WLCG.

The process of *skimming* performs a basic selection on the dataset. Events are removed if they are unlikely to contain interesting processes. From each valid event, only the physical objects to be studied are kept. The resulting *skim* is similar in structure to the initial dataset, but highly optimised in terms of size. Usually, every analysis group produces a set of skims for all its members. Due to their size, skims are stored at analysis facilities or at Tier 2 sites.

The *analysis* transforms objects from each event to observables. This includes trivial extractions, such as reading the momentum of a particle object. More complex tasks are handled via immediate objects, e.g. reconstructing the initial object of a decay process. The level of detail available is used for performing complex selection criteria, called *cut*s. The result is an *n-tuple*, a Struct of Arrays where each array corresponds to an observable. Most n-tuples can be easily stored on portals or workstations.

The final step is *plotting*, creating visual or numeric representations of data. This is the only step processing information from multiple events at the same time. It is thus capable of calculating distributions and relations including events. The output is usually a *plot*, though it is not uncommon to also output meta-data, e.g. numerical results of a fit.

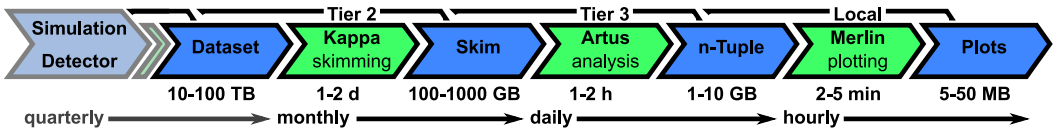


Figure 2.6: Exemplary End User Analysis Workflow Using the Artus Framework: The workflow uses a sequence of processing applications (green) to create increasingly specialised data formats (blue). This process decreases data size and application runtime from TB and days to MB and minutes. Any step can be repeated without repeating previous steps. This allows for high iteration frequencies of later steps.

Chapter 3

Coordinated Caching for End User Data Analyses in High Energy Physics

> If you torture the data enough, nature will always confess.
>
> Ronald Coase

Being a data driven science, High Energy Physics (HEP) strongly depends on efficient data analysis. This is not just a matter of resource utilisation: Analyses undergo constant development and tuning, linking analysis performance to research speed. Improved efficiency allows studying more phenomena, and deriving results of better quality and precision.

Computing in HEP both requires and motivates new developments. The Worldwide LHC Computing Grid (WLCG), and thus experiments at the Large Hadron Collider (LHC), would be impossible without software and infrastructure developments by the HEP community. However, effort has mostly been focused on large scale processing. Infrastructure for end user data analyses usually relies on commonly available technologies. Yet, many new approaches in data analysis cannot be applied to HEP. The mixture of existing workflows and constraints creates a niche that is not well covered by modern, specialised approaches.

This work introduces a generalised approach to iterative data analysis. On the one hand, it uses modern processing principles, most importantly data locality. On the other hand, it frames these principles in a minimalistic way, avoiding constraints and isolating customisations. Being developed primarily for use in HEP, it integrates with and improves existing workflows. The work is applicable to any data science using similar workflows.

The approach builds on two features of data analysis in HEP: First, while sizeable amounts of data are analysed, the total amount of data is considerably larger. Second, data is analysed via batch processing, i.e. distributed processing of multiple chunks of data in parallel. Both features are exploited by providing data from a distributed cache.

Parts of the work presented in this chapter have been previously published [12–15]. Prior reading of these publications is not required for this chapter. All key information and arguments are presented here as well.

An overview on requirements for HEP analyses and existing technologies is given in Section 3.1. In Section 3.2, the concept of Coordinated Caching is introduced and explored in general. The prototypical implementation of the approach and key design considerations are presented in Section 3.3. Experiences and key metrics derived from using the system for analyses of LHC Run II data can be found in Section 3.4. Finally, Section 3.5 contains conclusion and outlook of the project.

3.1 Applicability of Caching for End User Data Analyses

End user workflows, as outlined in Section 2.3.3, are in principle ideal for caching. Each step of a workflow is repeated multiple times. While configurations and applications are tuned constantly, input data is long-living. In addition, input data is often shared amongst several users.

Yet, scope and size of data usage make traditional approaches non-trivial. In addition, existing infrastructure has inarguably been very effective for past research. It is thus critical to more closely define the scope of caching for end user data analyses.

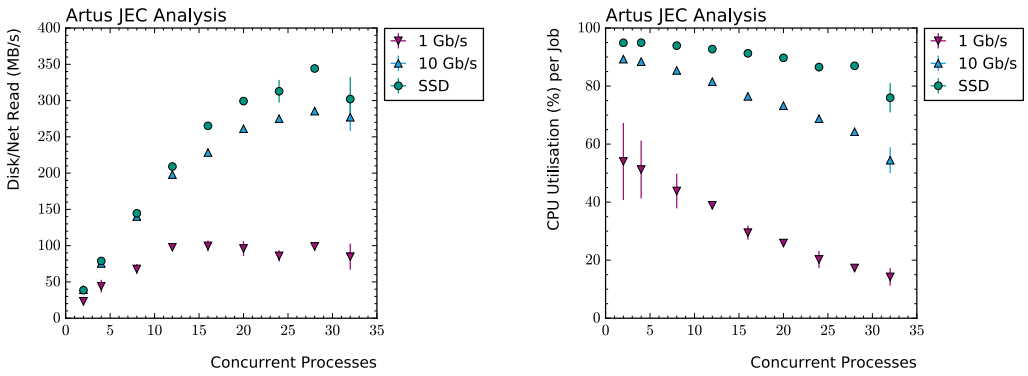
3.1.1 Characteristics of Analysis Workflows

For caching to be beneficial, there must be a potential performance gain. To assess this, benchmarks of different data provisioning technologies were performed. An excerpt is shown in Figure 3.1; a more detailed description and measurement is available in the original publication [14]. Hardware specifications are in Appendix E.1.

The benchmark of 1 Gbit network demonstrates the limitations of current network based infrastructure. Due to network being a shared resource, there is a fixed limit on throughput. Adding more processing resources, even on other hosts, does not increase throughput beyond this point.

Extrapolating the performance makes it apparent that a single host cannot saturate a 10 Gbit network. However, a cluster of three worker nodes is sufficient for partial saturation. Under full load from an efficient analysis, even a small cluster does not perform efficiently. A cache could thus increase efficiency by providing part or all of the required input.

However, any potential cache must provide sufficient performance as well. The demand for parallelisation, throughput, and data volume make this difficult on a single host. The only local data source with adequate performance is a Solid State Drive (SSD). It allows to fully utilise a modern processing node, as used in the benchmark.



(a) Total Input Throughput

(b) Average per Job CPU Efficiency

Figure 3.1: Performance Metrics for Different Input Media: Benchmarks were performed using the same Jet Energy Corrections (JEC) analysis. The type of data access/storage and parallelisation was varied for each benchmark. For the benchmark of 10 Gbit, 48 additional processes reading data were deployed in parallel on other hosts. The performance drop in the last bin is due to saturation of system resources. Here, fewer resources are available as some are consumed by the operating system and other services.

In addition to raw performance, principles of caching must be applicable to workflows. Most importantly, a notable fraction of data usage must follow a predictable pattern. Thus, analysis input usage patterns were tracked, as visualised in Figure 3.2. All accesses shown are from JEC workflows, belonging to multiple users. Several distinct features relevant for caching can be identified:

- The largest part of accesses is to a few, pivotal datasets. For example, skim 5 is used about ten times more often than skims 1 to 4 combined. It is thus sufficient to cache only a fraction of all data, identified by frequent usage.
- Access to datasets are interleaved, with both pivotal and other datasets overlapping. For example, skims 7 and 8 are detector and simulation data used together. It is thus necessary to assess not the temporary, but the overall benefit of caching specific data.
- Pivotal datasets are only used in a limited time range before being replaced. For example, skims 7 and 8 were replaced by skims 17 and 18. Outdated data may thus be identified by its last access.
- The order of time intervals is days for accesses, and months for relevancy. For example, skim 24 was fully read every 1 to 2 days over 2 months. Thus, a cache must operate at a much larger timescale than usual for e.g. filesystem caches.

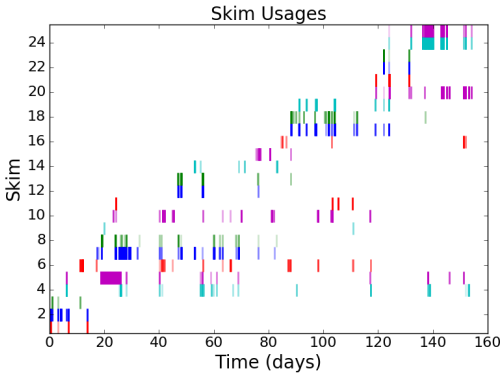


Figure 3.2: Usage of Skims for Analyses: Data was collected from jobs reporting their input files to the batch system. For technical reasons, these are mostly jobs of JEC workflows. Skims are numbered by their occurrence in the batch system. Excluded are skims which are used for benchmarking, were accessed less than five times, or used for less than a week. Skims with similar access behaviour, e.g. skims 7 and 8, are matching detector and simulation datasets.

The characteristics of current workflows make caching a viable strategy. Yet, the use case is not comparable to that targeted by regular caches. To operate efficiently at this scale, an efficient software solution is required.

3.1.2 Existing Approaches and Related Work

A number of existing solutions have been evaluated for their applicability to end user data analyses. This is not limited to cache software. Related technologies which may be used for data locality are included as well. The following is not an exhaustive list, but covers all notable categories.

Local Data Caches

An obvious approach to caching input data are operating system caches. For example, Linux automatically caches recently read data in unclaimed Random Access Memory (RAM). In principle, such a cache could be extended to an SSD. Specialised caches [16, 17] explicitly promote such a use case. Many distributed systems also use comparable, local caches.

Yet, the scale of operation limits the applicability of such caches. The general setup implies several limitations. These can be seen as exemplary for challenges of caching in distributed data processing.

- The underlying algorithms are optimised for small, temporary repeating read operations. In contrast, HEP data easily exceeds cache sizes, and accesses interleave over several days. This makes it likely that cached data is superseded before being accessed again.
- A distributed workflow may repeat the same read operation on different hosts. Yet, caches are only aware of local read operations. Important data may be ignored if the fraction of read operations performed locally is insignificant.
- Similarly, caches are only aware of their own content. As such, data may be cached on multiple hosts. Such duplicates ultimately limit the unique volume of data provided by caches.
- Applications only benefit if data is cached on the same host. However, caches usually do not expose their content, or do so only inappropriately. This makes it impossible to know on which host a job would benefit from already cached data.

Distributed Data Storage

The distribution of workflows on several hosts is a key issue for data locality. Distributed data storage technology likewise allows distributing data onto several hosts.

Parallel file systems extend regular file system behaviour onto distributed environments. Both data storage and access may be performed from several hosts in parallel. The file system itself may span several hosts as well. While not mandatory, it is not uncommon that the same hosts are used for storing and processing files.

However, parallel file systems do not necessarily improve data locality. It is in general not possible to deduce on which host data is stored. Storage allowing to deduce this information can in principle be used for local data processing [18]. However, this implies adhering to the underlying distribution algorithm. These are generally optimised for non-local access and fault tolerance.

To optimise data locality for processing, specialised file system and processing frameworks have emerged in recent years. These usually employ the MapReduce processing paradigm [19]. The paradigm builds on defining tasks in an abstract way: A *map* procedure defines how results are derived from a portion of data. A *reduce* procedure defines how multiple sets of individual results are merged. These procedures allow parallel processing of arbitrary data chunks, and creating a single result from it. Processing frameworks use this to split tasks into jobs matching the data layout on file systems.

The MapReduce paradigm is compatible with HEP workflows. These already process datasets in chunks, merging the results afterwards. However, the formalisation and implementations are not compatible.

The commonly used MapReduce framework Hadoop [20] has several incompatible constraints: Data access is implemented via a unique streaming protocol; in contrast, HEP relies on POSIX for local access, and protocols such as xRootD for remote access. The Hadoop file system [21] splits data into chunks by itself, whereas HEP workflows split data by their own requirements. Similarly, Hadoop expects to split tasks into jobs, which is already performed by HEP job managers. At the implementation level, Hadoop requires active use of its programming model and libraries; a resulting application is not compatible with classical batch systems without extensive modifications.

Caching in Distributed Environments

The general concept of caching in distributed environments is not new. Existing solutions generally perform considerably better than multiple isolated caches. However, the use case of data locality for processing is not adequately addressed yet.

A distributed cache may be created by coordinating individual caches. Issues arising from limited scope, e.g. content duplication, are countered by a centralised coordination component. Simulations show significant improvements in hit ratio and throughput [22]. Yet, implementations usually target resource providers, namely web services. Host locality for data consumers is by design out of scope.

Several caching systems target distributed data processing. The CernVM File System (CVMFS) [23] is used for provisioning of software frameworks and other common applications. Local caches are used to eliminate remote access for often used software. The CacheD service [24] of the HTCondor batch system stores executables used by jobs. This eliminates the need to transfer the same executable each time a job is deployed. Both CVMFS and CacheD show that caching is beneficial for distributed processing. Their use case is on software that is the same for many jobs executing simultaneously. In contrast, input data is the same for individual jobs executing repeatedly.

Attempts have been made by the LHC collaborations to promote data locality for end user data analyses. The ATLAS collaboration implemented a prototype to speedup analyses via a centralised cache [25]. This setup demonstrates the speedup from improved locality as well as the viability of using SSD based caches. Yet, the setup is limited to data access via xRootD and its caching mechanism as well as still accessing the cache via network. Several groups have adapted Hadoop for use with HEP analyses [26, 27]. These approaches demonstrate improved scalability and performance of data locality based processing. However, the modifications required and constraints imposed on existing workflows are severe.

3.2 Coordinated Caching for Batch Systems

As outlined in Section 3.1.1, HEP end user data analyses are suitable for caching. Yet, existing technologies are not applicable without extensive modifications. Still, data locality provided via caching has been shown to provide considerable advantages. Thus, a new caching concept to enable data locality for end user data analyses is proposed as part of this thesis: individual caches on batch system worker nodes are coordinated to form a single cache spanning the entire batch system.

The review of existing technologies as outlined in Section 3.1.2 exposes two major issues: On the one hand, constraints on workflows and jobs are severe. Modifications to analyses required are unfeasible for most end users. Also, modified analyses are incompatible with regular infrastructure. On the other hand, reviewed implementations are optimised for specific use cases. Extensibility for different setups and protocols is limited.

The proposed concept thus targets two main goals, motivating key design features:

- Constraints on end user workflows and software should be minimised. The concept thus targets analysis infrastructure, not applications.
- Extensibility for other setups and protocols must be possible by design. The concept is thus modularised. It offers a generic core and specialised extensions.

An element shared by all HEP analysis infrastructures is the use of batch systems. While implementation details vary, the general design is the same. Thus, the concept suggests a new middleware layer integrating with batch systems[1].

Simply put, the suggested middleware mimics an operating system cache scaled up to cover a batch system. Operating system caches target *applications* using *read requests* to process *blocks* from *files*. The middleware targets *workflows* using *jobs* to process *files* from *datasets*. Key challenges arise from the processing environment being distributed over several hosts or even sites.

3.2.1 Distributed Cache Layers

In general, a cache can be viewed as being composed of three key elements:

- Selection: identifying data most beneficial to cache.
- Provisioning: locally storing and maintaining copies of data.
- Redirection: intercepting accesses to serve local data and track access patterns.

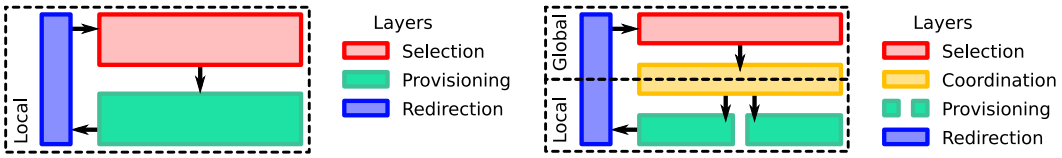
[1] In principle, neither concept nor middleware are strictly limited to batch systems and worker nodes. For example, no modifications are required to target *groups* of worker nodes, e.g. sites in grid or cloud environments. Descriptions presented here assume batch systems as targeted environment for clarity and simplicity.

In a distributed environment of batch processing, these elements must be distributed as well. To promote this, the concept distinguishes local from global scope: each local scope is confined to an individual worker node, while the global scope is the entire system. Based on this distinction, several elements or *layers* can be defined for coordinated caching:

- Selection Layer: operates at global scope, aggregating and analysing all data accesses in the system.
- Provisioning Layer: operates at local scope, maintaining data stored on worker nodes.
- Coordination Layer: links global and local scope, spreading decisions from the Selection Layer to the Provisioning Layer.
- Redirection Layer: intercepts workflows at job and application granularity.

The key difference to regular caches is the addition of a coordination layer. This is required by operating in a distributed environment. The scope of each element is shown in Figure 3.3

The different scopes of each layer imply a tight definition of responsibilities. The provisioning layer is similar to regular caches, though it spans several hosts. It is the coordination layer that handles the distributed nature of the system. Caching logic can be freely chosen and adjusted as the selection layer only ever interacts with other layers. Integration with infrastructure and applications is confined solely to the redirection layer.



(a) Layers of a Classic Cache **(b)** Layers of Coordinated Caches

Figure 3.3: Layers of the Coordinated Caching Concept: The coordinated caching concept is composed of four layers, each handling a different responsibility. The selection layer works at global scope, where all metadata of accesses and data can be aggregated. The provisioning layer consists of multiple elements, each operating at local scope. Connecting the two is the coordination layer, spanning local and global scope. The redirection layer reside in both local and global scope, depending on the targeted use case.

Provisioning Layer

The provisioning layer handles the actual data on worker nodes. As such, it is formed by a number of provisioning nodes, one per worker node. As provisioning nodes are collocated with data processing facilities, the provisioning layer is the only layer with guaranteed access to remote data.

The main purpose of data provisioning is to guarantee that data is locally available. After remote data is copied to local cache devices, periodic checks of metadata ensure the consistency of local copies with remote data. If data is outdated, it is updated or removed.

The provisioning layer does not decide which files to cache. It relies on external input to select files, rate their importance, and assign them to provisioning nodes. Each provisioning node only provides the files that have been assigned to it, preferring important files if space is scarce. Globally managing the precise allocation of files to specific devices is not scalable, however.

Thus, each provisioning node has some limited autonomy. Each provisioning node decides by itself how to deal with limited space. This is required when new files are added or existing files change size. Provisioning nodes take the importance of files into account, but are not strictly bound by it. For example, a provisioning node may decide to evict only the *second* most unimportant file if that frees exactly the required space.

Provisioning nodes are not just the only element with guaranteed access to data. They also have guaranteed access to its metadata, such as size and modification time of files. As such, other layers use the provisioning layer to access file metadata.

Selection Layer

The selection layer is responsible for selecting data that is beneficial to cache. It exclusively works with *metadata* of both jobs and data. Thus, it is sufficient to interact only with other layers of the system.

By design, the selection layer never requires action from elements outside of the system.. All metadata on job and data usage are provided by the provisioning and redirection layers. This frees the selection layer from any constraints from batch system and user applications. It transparently extends the functionality of the middleware by exploiting the interfaces of the redirection and provisioning layer. In turn, this layer can be easily replaced, adjusted, and optimised for different use cases.

In the concept of coordinated caching, selecting data for caching also means rating data. The rating expresses the impact of caching specific data. For example, this may estimate how often data is accessed in the future. The specific rating must be chosen to reflect the expected features of data accesses in user workflows.

The selection layer forwards only the rating, not any details on how it is derived. The only requirement is that other layers properly interpret the rating. That is, they must be able to deduce which rating indicates more important data. This ensures a clear distinction of responsibilities between the layers. In turn, this allows the use of established caching algorithms.

Redirection Layer

The redirection layer is the interface to external elements. Targeting jobs in a batch system requires splitting the responsibility of the redirection layer into two tasks: redirection of queued jobs to worker nodes and redirection of data accesses for executing jobs.

The redirection layer is largely defined by the infrastructure and user applications. In turn, it encapsulates all specifics of these external elements. It is the only layer that handles implementation details of the targeted use case. This makes all other layer implementations independent of the use case.

The redirection of data access for executing jobs is comparable to the redirection by regular caches. Direct data access is rerouted to use local copies of data, if available. This requires overwriting the data access method used by applications. Ideally, this is integrated into the protocol used for data access. Alternate approaches, e.g. modifying job configurations, are possible as well.

The redirection of queued jobs stems is required due to the integration with batch systems. Cached data is only beneficial if jobs execute on nodes hosting their data. Thus, job scheduling must be influenced to prefer nodes with cached input data. Ideally, this is performed by hooking into the batch system's scheduling mechanism. Alternatively, it can be integrated into user submission tools.

The redirection layer puts the only hard constraint on user jobs: in order to redirect queued jobs, they must publish their input data in a standardised way.

Coordination Layer

The coordination layer manages the distributed resources of the system. It connects the selection layer at global scope and the provisioning layer at local scope. This allows using established techniques for selection and provisioning.

The coordination layer is similar to a job scheduler. Data must be scheduled to provisioning nodes, similar to how jobs are scheduled to worker nodes. On the one hand, the coordination layer has metadata for data to be analysed. For example, this may be size, processing speed, or which data is processed in parallel. On the other hand, it has metadata for provisioning nodes. For example, this includes size of caching devices, or the number of attached processing slots. Both metadata of data and provisioning nodes must be matched to find an optimal allocation of data.

Like job scheduling, data scheduling can be optimised for various goals. For example, specific use cases may benefit from high peak locality over average locality. Similarly, one may optimise for speedup of specific workloads versus overall throughput. Most of these decisions depend on the use case. However, some features directly follow from targeting batch systems.

3.2.2 Data Scheduling for Batch Processing

The primary challenge for local caches is the estimation of data popularity. In order to successfully provide data, a cache must predict which data will be used next. In a distributed environment this is further complicated by locality considerations. In addition to estimating which data is required, it is also relevant where it is required.

Batch processing implies several caching scopes for distributed caching. The global scope includes all data cached, regardless of location. The local scope includes all data cached on a specific node. At the job scope, only data used by a single job is relevant. The relation between these scopes is illustrated in Figure 3.4.

Data of any local scope is strictly a subset of the data at the global scope. Selecting data for the global scope has been extensively studied for regular caching. The limiting factor for data locality is the overlap between local and job scope. This determines the efficiency of any coordinated caching system for batch processing.

Optimising the efficiency of coordination is important regardless of use case. To illustrate the core challenges, a simple model of *un*coordinated caching is demonstrated. It mainly avoids edge cases, e.g. partial allocation of data. Conclusions are applicable in general.

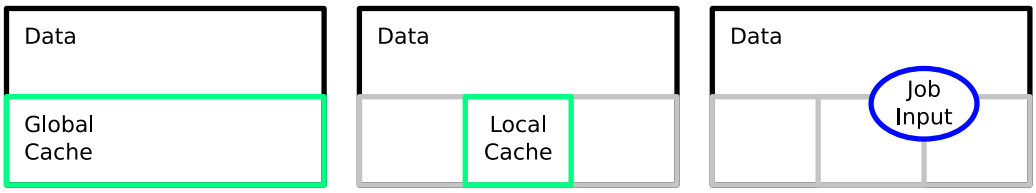


Figure 3.4: Scopes of Coordinated Caching: A single distributed, coordinated cache can be viewed at different scopes. In an idealised view, the global cache is a single entity. Like a non-distributed cache, it simply holds a fraction of the overall data. This fraction is roughly the ratio of cache size to overall data volume. With the goal of optimising data locality, each individual local cache is distinct. The locally cached data is merely a fraction of the globally cached data. This fraction is inversely proportional to the number of caches. To optimise data locality of jobs, the group of files accessed together is important. Even when fully available in the global cache, only a fraction may reside in each local cache.

Assume that the total data has a volume V_{total}, which consists of chunks of data, or *items*. The global cache volume is sufficient to hold a volume V_{cache}. For practically all use cases, V_{cache} is considerably smaller than V_{total}. Further, chunks are small compared to any cache and total storage volumes. Finally, the worst case of a statistical caching strategy is assumed: every item has the same probability of being cached.

Regardless of caching strategy, only a fraction of data can be provided by the cache. This can be thought of as the chance that specific data is cached. As in regular caches, this is the *cache hit rate* P_{global}.

$$P_{\text{global}} \propto \frac{V_{\text{cache}}}{V_{\text{total}}} \tag{3.1}$$

An optimal caching strategy may optimise the use of cache space, and thus P_{global}. Still, the chance of an item being cached is roughly proportional to the fraction of volumes.

Item Locality and Replication

To optimise data locality, the deciding quantity is the amount of data provided locally. Thus the coordinated caching concept adds another metric: the chance of an item being cached on the host it is accessed from. This defines the *local hit rate* P_{local}. The ratio of local and cache hit rate defines the *locality efficiency* η_{local}.

The locally cached data can only ever be a subset of the globally cached data. Assuming processing nodes are interchangeable, the global scope may be equally split across nodes. In this case, each processing nodes has an equally sized local cache volume V_{local}.

$$P_{\text{local}} = \eta_{\text{local}} \cdot P_{\text{global}} = \frac{V_{\text{local}}}{V_{\text{cache}}} \cdot P_{\text{global}} \tag{3.2}$$

At this point, it is also appropriate to address replication. In data storage and processing, it is common to replicate items to multiple nodes. This improves fault tolerance and increases the chance for data locality. For caching, it is important that the global cache volume cannot hold all data and consists solely of the local cache volumes. Using replicas proportionally reduces the volume of *unique* items in the cache. In addition, the number of hosts N_{hosts} constraints the number of replicas N_{replicas}.

$$V_{\text{cache}} = \frac{1}{N_{\text{replicas}}} \sum_{hosts} V_{\text{local}} = \frac{N_{\text{hosts}}}{N_{\text{replicas}}} \cdot V_{\text{local}} \tag{3.3}$$

$$P_{\text{local}} = \underbrace{\frac{N_{\text{replicas}}}{N_{\text{hosts}}}}_{=\eta_{\text{local}} \leq 1} \cdot P_{\text{global}} = \frac{V_{\text{local}}}{V_{\text{total}}} \tag{3.4}$$

Put simply, replication improves the chance to hit cached data. Yet, it reduces the chance of data being cached in the first place. While replication is important for distributed storage, there is no advantage for distributed caching.

Furthermore, the compensation of both effects only holds for statistical data placement. If either cache hit rate or locality efficiency are fixed via scheduling, replication still constraints the other.

Regardless of replication, the local hit rate is tied to the cache volume per node. Compared to a monolithic cache, the hit rate for distributed, uncoordinated caching is inversely proportional to the number of nodes. This is caused by random selection of the node hosting an item and the node executing its jobs.

It is thus important that coordinated caching actively schedules jobs to data. In turn, this means replication should be avoided.

Job Locality and Item Groups

In batch processing and especially HEP, it is common to process a group of items at the same time. In classic batch systems, these groups are assigned externally, before submission.

Each job only benefits from caching if items are locally available. It is sensible to define the *job efficiency* η_{job} as the fraction of items available locally. In turn, this can be expressed via the expected number of local items N_{local} and total items N_{job}.

$$\langle \eta_{\text{job}} \rangle = \frac{\langle N_{\text{local}} \rangle}{N_{\text{job}}} = \frac{1}{N_{\text{job}}} \sum_{N_{\text{job}}} P_{\text{local}} \tag{3.5}$$

$$= P_{\text{local}} \tag{3.6}$$

$$= \eta_{\text{local}} \cdot P_{\text{global}} \tag{3.7}$$

It is important to distinguish between job and locality efficiency at this point. For any single item, locality efficiency η_{local} can be arbitrarily improved via scheduling. This merely requires scheduling jobs to the node caching the item. Disadvantages from distributed caching can thus be easily avoided. There are only two states, both of which are advantageous: An item is either cached and accessible, or it is not cached and thus does not waste any space.

For any job or group of items, job efficiency is bound by the distribution of items. Perfect job efficiency is *impossible* if not all items are collocated on the same node. This adds a third state: An item may be cached but not accessible with the rest of its group, wasting space. This can be modelled as a binomial distribution:

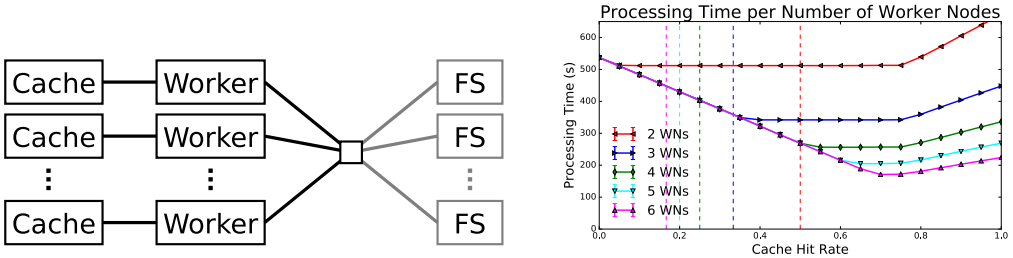
$$Pr\left(\eta_{\text{job}} = \frac{N_{\text{local}}}{N_{\text{job}}}\right) = \binom{N_{\text{job}}}{N_{\text{local}}} \eta_{\text{local}}^{N_{\text{local}}} \left(1 - \eta_{\text{local}}\right)^{N_{\text{job}} - N_{\text{local}}} \tag{3.8}$$

This effect is notable even for small scale processing. Assuming jobs of four items on a cluster of four nodes, 75 % of jobs are limited to job efficiencies of 25 % or less.

It is thus important that coordinated caching schedules data to match usage by jobs. This must be sensitive to the item grouping chosen before submission. Otherwise, it is impossible to optimally schedule jobs to data for execution.

3.2.3 Simulation and Estimates

Performance benchmarks as shown in Section 3.1.1 show that caching can benefit HEP data analysis. However, building a middleware or cluster based on this requires estimating the scale of improvements. Thus, simulations have been performed to estimate effects of coordinated caching. Most importantly, the setup at the Institut für Experimentelle Kernphysik (IEKP) has been studied, as shown in Figure 3.5.



(a) Setup of Cache Throughput Simulation **(b)** Results of Cache Throughput Simulation

Figure 3.5: Simulation of Throughput Using Coordinated Caches on Worker Nodes: The simulation setup is comparable to the analysis cluster at the IEKP, as shown in Figure 3.5a. Every worker node has 32 execution slots and is connected to a local cache and shared fileservers. The bandwidth to individual caches is 4 Gbit/s and 10 Gbit/s to all fileservers. Free parameters are the number of worker nodes and average local cache hit rate. A workflow processing 4 GB at 20 MB/s per process is assumed. The resulting processing time is shown in Figure 3.5b. Straight lines indicate processing time for a given number of worker nodes. Dashed lines are the expected local cache hit rate without aligning job and data scheduling.

Several key features are apparent from the simulation results: Coordinated caching enables scalability. Network bandwidth is an important contributor for overall throughput. Following this, perfect local hit rate is not desirable. Finally, hit rates without scheduling are inadequate.

As expected, data locality via caching enhances scalability of existing infrastructure. The addition of local caches automatically scales throughput to processing capabilities. The actual speedup is determined by network bandwidth and the number of processing nodes. The former determines the initial throughput. The later determines the final throughput. In general, speedups of several factors are possible.

Even with the addition of caching, network is important for overall throughput. Several caches are required to provide the same throughput as network does. Ideally, both cache and network throughput is combined. This implies that perfect hit rates are not desirable: on average, hit rates should balance cache and network throughput.

There is some leeway in optimising hit rates. Limited processing speed creates a plateau between the limits of network and cache. Leaning towards low hit rates allows caching data for more workflows. At higher hit rates, few workflows are reliably provided from the cache. The former is advantageous for many interleaved workflows. The later is more stable with few recurring and many temporary workflows.

3.3 The HTDA Middleware Prototype

As part of this thesis, a prototype of the coordinated caching concept has been created: The High Throughput Data Analysis (HTDA) middleware. It demonstrates both the feasibility of the approach as well as the improved scalability and performance.

HTDA provides all core features of coordinated caching: A provisioning layer supporting POSIX data sources. A combined selection and coordination layer, suitable for usage patterns of HEP workflows. Finally, a redirection layer integrating into HTCondor batch systems and POSIX data access.

The software has been built primarily as a prototype for demonstration. However, it is also designed with future applicability in mind. Dependencies on HEP workflows are kept to an absolute minimum; the middleware is useable outside of the current HEP context. All interfaces to external elements are modular, allowing for future support of additional protocols.

Architecturally, the HTDA middleware builds on distinct nodes. These are joined together to form a pool spanning several hosts. Each node runs one or several services. These services implement the layers of coordinated caching.

There are currently three service types:

- `Provider` services form the provisioning layer. Each stages and maintains data on the host it is running on.
- `Coordinator` services implement the selection and coordination layer. They aggregate metadata on files and jobs, selecting data to cache.
- `Locator` services are a key component of the redirection layer. Each creates a map of files provided by nodes, which is used for scheduling.

In addition, the redirection layer has two more constituents: First, hooks in the batch system collect metadata of jobs and influence scheduling. Second, an overlay in the Virtual File System redirects file access on worker nodes. An overview of the HTDA services is shown in Figure 3.6.

3.3.1 Nodes and Shared Infrastructure

All persistent elements of the HTDA middleware are handled by a single application, the `htda_node`. It provides the basic infrastructure to create a distributed pool: instances launched on different hosts may communicate and collaborate. This forms the basic environment in which HTDA services operate.

The application itself is written in Python 2 [28]. This is motivated by the flexibility and ease of use of the language. In addition, it is widely used in HEP.

The `htda_node` implements the basic environment for running HTDA services. It unifies basic application features, most prominently configuration and logging (see Appendix C). In addition, it provides an abstract interface to the pool of nodes.

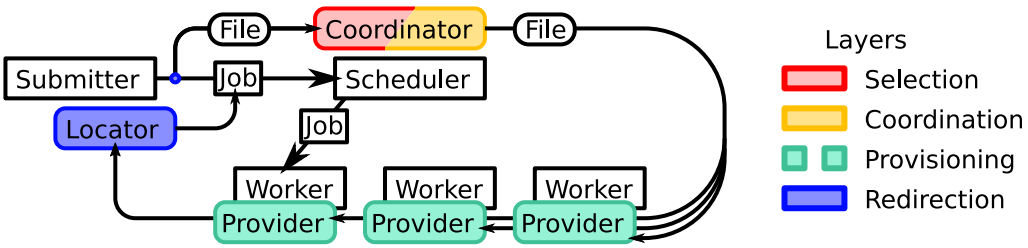


Figure 3.6: Schematic View of the HTDA Services: The HTDA middleware is composed of several services. `Provider` services reside on worker nodes and provide local copies of data. Which files to provide on which node is determined by the `Coordinator` service. To schedule jobs to their cached input data, the `Locator` services provide information about data placement to jobs.

Finally, the `htda_node` monitors deployed HTDA services, restarting them in case of errors. The abstraction is a key feature for the HTDA prototype.

The nodes themselves implement the actual communication between each other. There are two main features to this: Inter-Process Communication (IPC) and *pool mapping*.

Node and Component Communication

Each node runs a server and client to communicate with other nodes. This is currently based on REST requests sent via HTTP/S. The use of HTTP/S is an implementation detail; the design specifically allows for other protocols in parallel. Calls to components on the same node can be resolved locally.

The REST architecture is an integral design decision. Stateless communication makes each `htda_node` only loosely coupled to the pool. Loss of connection between nodes does not corrupt state of the system. Temporary or permanent absence of nodes in a pool is thus not harmful. This allows HTDA to operate on temporary processing resources.

The client/server infrastructure is made available by the node via an abstraction layer. Components can use it to expose some of their methods to other nodes, enabling Remote Procedure Call (RPC). This provides a URI for every functionality, as shown in listing 3.1.

```
1    # Method call via high level Python API
2    HeartbeatAPI(Node<ekpsg01.physik.uni-karlsruhe.de>).
         get_nodes_by_role(node_class="locator")
3
4    # Method call abstract information
5    Node: Node<ekpsg01.physik.uni-karlsruhe.de>
6    Node URI Address: ekpsg01.physik.uni-karlsruhe.de
7    Component: Heartbeat Mapper
8    Component URI Name: heartbeat
9    Method: get_nodes_by_role
10   Method URI Name: GET
11   Method Argument: node_class="locator"
12
13   # Method call via low level HTTP API
14   GET http://ekpsg01.physik.uni-karlsruhe.de/heartbeat/locator
```

Listing 3.1: Method calls between components: Components expose their functionality via high-level APIs, abstracting RPC calls. Interaction between components is performed by calls to such methods on other nodes. The high-level API translates the call to an available low-level API call.

Pool Discovery

`htda_node`s interact by calling exposed methods of other nodes. However, components must first be aware of available nodes. Even in local batch systems, nodes may be only temporarily part of an HTDA pool. This makes dynamic pool discovery vital.

To this end, each `htda_node` runs a generic component, the `pool mapper`. It dynamically discovers other nodes of a pool, as shown in Figure 3.7.

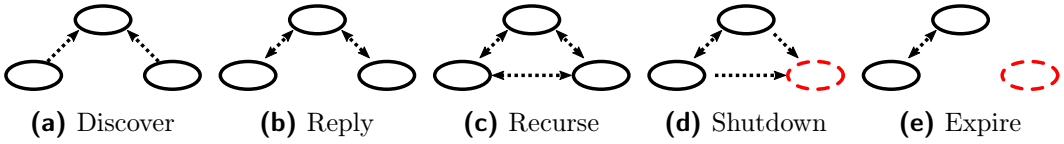


Figure 3.7: Mapping of an HTDA Pool: Each node runs a mapping service, the `pool mapper`. In its discovery stage, the mapper queries known nodes for other nodes (3.7a,3.7b). This is repeated recursively, until all nodes have been discovered (3.7c). In its validation stage, the mapper exchanges heartbeats with known nodes. Any node repeatedly failing a heartbeat exchange is considered defunct (3.7d,3.7e). Both stages are repeated frequently, providing an up-to-date state of the pool. Pool mappers can start discovery either from a configured address, or whenever another node connects.

The pool mapper provides all local HTDA services with the nodes currently available in the pool. This information can be directly used for communication via RPC. The details of availability, e.g. the protocol used for communication, are purposefully not exposed to HTDA services. For example, mapper and underlying protocols may use proxies for communication. This abstraction frees HTDA services from having to deal with the physical layout of the HTDA pool.

3.3.2 Data Provisioning

The data provisioning layer of HTDA is implemented via the `Provider` service. On each worker node, one `Provider` service is run, which maintains local copies of data. In addition, the HTDA middleware uses `Provider` service to persistently store metadata on the location of data copies.

Provisioning for Data Locality

Internally, each `Provider` service is composed of three major elements: The `Catalogue` stores the current state of provisioning. The `Allocator` computes the desired state of provisioning. The `Operator` handles the transition between current and desired state. A schematic overview is schon in Figure 3.8.

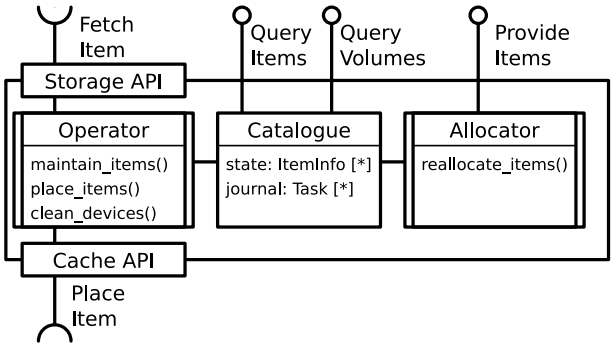


Figure 3.8: Components of the `Provider` Service: Each functionality of the `Provider` is implemented separately. The `Operator` performs the actual provisioning of data, moving it from storage to cache devices. Which data to provide is decided by the `Allocator`. Persistency is ensured by the `Catalogue`, which stores all vital metadata. This complexity is not exposed to other services: Metadata on the `Provider` and its content is exposed, and new items may be suggested for provisioning. Internally, Storage and Cache APIs abstract their resources: storage must only provide a means to fetch data, while a cache must allow placing this data.

The `Catalogue` stores all metadata relevant for data provisioning. On the one hand, this is file information, as shown in listing 3.2. This includes immutable information, e.g. the URI of the file on remote storage, and mutable information, e.g. on which device it is cached at the moment. On the other hand, this is a journal of required actions or *tasks*, e.g. deletions of local copies. This is comparable to the journal of file systems: Instructions for performing actions are stored until all partial operations are successful. This ensures consistency in case of interruptions, e.g. reboots of worker nodes.

```
1    source_uri      : original URI of the file, as addressed by users
2    filename        : dirname + basename on local filesystem
3    storage_id      : storage API providing the source file
4    cache_id        : cache API supervising a local copy
5    size            : size of the source file, in bytes
6    score           : importance of the file
7    cache_time      : when the file was assigned to this node
8    allocation_time : when the file was assigned to a cache API
9    maintained_time : when the file was last validated
```

Listing 3.2: Metadata of files stored by Cache services: Each cache stores immutable and mutable metadata of files assigned to it. Immutable metadata relates to how files are accessed. Mutable metadata includes details on how files are cached.

The `Allocator` implements the autonomous allocation of files to cache devices. At regular intervals, it calculates the currently best allocation. The current algorithm does a sequential allocation based on file priority: All files are sorted by their rating; Cache devices are then successively filled until no space is left. For each file whose allocation changes, a corresponding move or deletion task is created. These tasks are stored in the `Catalogue` journal.

The `Operator` performs the actual handling of data. This is implemented via several worker threads each processing work items in parallel. This currently includes three types of work items:

- Handling the transition between allocations: This means processing the tasks created by the `Allocator`. New files are copied from storage to a cache device, copied between cache devices, or released from a cache device.
- Maintaining the current allocation: The `Catalogue` is scanned for allocated files. For each local file, its metadata is compared to the metadata of the source file. If a mismatch is detected, the file is copied from storage again, or released if this fails.
- Cleaning cache devices: Each cache device is scanned for orphaned files. If a file is found that has not been allocated to that cache device, it is released.

Backend Abstraction

The HTDA middleware has been written with the goal of caching files from *remote storage*. This is designed to be performed by copying files to *local cache devices*. For extensibility and portability, elements of `Provider` services do not work directly on files. Instead, an abstraction of protocols via backends is used.

Backends wrap both protocols as well as specific end points. For example, POSIX cache devices can be accessed by the `FSCache` cache backend[2]. A specific end point, e.g. a mounted SSD, is wrapped by instantiating an `FSCache` with the mount point path.

On the one hand, this abstraction simplifies adding additional protocols. Creating a new backend is sufficient to integrate it into all elements of HTDA. This mostly means wrapping metadata retrieval and file access, as shown in Listing 3.3. Backends are easy to implement, requiring about 150 to 250 lines of code.

```
1    class StorageAPIBase(*args, **kwargs):
2        """Backend for Remote Storage"""
3        def supports_uri(source_uri: str):
4            """Check if a given URI is supported by this backend"""
```

[2]The `FSCache` and `FSStorage` backends actually use Python's abstraction of file systems, as implemented by `os.path` [29]. This supports POSIX, Windows and MacOS file systems.

```
5         def get_file_info(source_uri: str):
6             """Provides the FileInfo (metadata) for the given URI"""
7         def refresh_copy(fileview: FileView):
8             """Update a cached copy to match its original source"""
9         def verify_copy(fileview: FileView):
10            """Check the validity of a copy against its source"""
```

Listing 3.3: Abstract API definition for storage backends: Backends are used to abstract accesses from `Provider` services to cache and storage resources. Each backend wraps internal calls to the protocol used by a resource. A FileView implements transactional modifications of the metadata shown in Listing 3.2.

On the other hand, the abstraction loosens the definition of remote storage and cache devices[3]: Remote storage must provide source data and its metadata. Cache devices must allow storing copies of data.

For example, the HTDA prototype implements the `MultiCache` backend. It combines other backends into a single interface. This is currently used to merge multiple SSD devices for allocation by `Provider` components. Access outside the HTDA middleware still targets the bare devices for optimal performance.

Metadata Volatility

Storing file metadata on `Provider` nodes is not only used for internal state. It also serves to provide the ground truth of data locality in an HTDA pool.

Each `Provider` service stores locality metadata in its `Catalogue`. In turn, the `Catalogue` stores its content persistently on its host. This automatically ties locality metadata to the availability of the host. At the same time, locality of data itself is also tied to the host.

As such, all other components rely on `Provider` services for locality information. This information may be cached or proxied. Yet, this replicated information is never binding nor vital. Even after a total system failure, locality information can be reconstructed from `Provider` services.

3.3.3 Data Selection and Coordination

The HTDA middleware combines the selection and coordination layer in the `Coordinator` service. Internally, each layer is still a separate element. The collocation simplifies the management of metadata at smaller scale, however.

Combining the two layers makes it advantageous to synchronise them, as shown in Figure 3.9. First, jobs and files are ranked to identify data worth caching. Afterwards, files are assigned to caches based on this rating. Technical details of the implementation can be found in the master thesis by C. Metzlaff [15].

[3] Technically, the definition of what constitutes a file is also loosened. For simplicity, this is not discussed here.

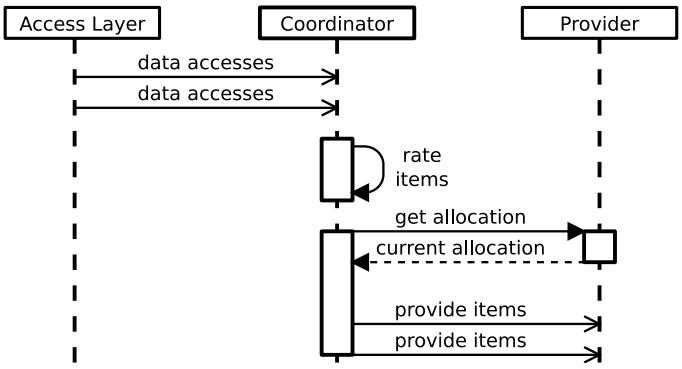


Figure 3.9: Operation Sequence of the `Coordinator` Service: The `Coordinator` relies on the access layer to inform it about data accesses. This information is used to rate the importance of individual data items. The current allocation of items is pulled in from `Providers`. Based on this, a new allocation is calculated and pushed to `Providers` again.

Rating of Data

The requirements on a rating system for HTDA are twofold: On the one hand, the rating must reflect the most relevant features of data usage. On the other hand, the result must allow unambiguous interpretation in a distributed environment.

The second requirement is most easily satisfied by a scalar result. Each file is assigned a score, with higher numbers signifying higher benefit or importance. This allows sorting files by priority, and in turn makes partitioning straightforward. However, it disqualifies multi-dimensional scoring algorithms, such as the Adaptive Replacement Cache [30].

The choice of a scoring algorithm is based on HEP usage characteristics, as shown in Section 3.1.1. On the one hand, important datasets are identified by frequent usage over a period of time. This characteristic is identified by Least Frequently Used (LFU) algorithms. On the other hand, outdated datasets are identified by being unused for a period of time. Least Recently Used (LRU) algorithms are sensitive to this characteristic.

As such, HTDA uses a combination of the two: The score is based on a decaying frequency, as shown in equation 3.9. Here, t_{now} is the time at which the score is calculated. For every job j that accessed an item, t_j is the time of submission to the batch system. The parameter Δt is a configurable tuning constant; Θ is the Heaviside step function.

$$S_{\text{Item}}^{\text{HTDA}}(t_{\text{now}}) = \sum_{j}^{\text{jobs}} \left(1 - \frac{t_{\text{now}} - t_j}{\Delta t}\right) \Theta(\Delta t + t_j - t_{\text{now}}) \tag{3.9}$$

This approach lies in the spectrum of Least Recently/Frequently Used (LRFU) algorithms. Such algorithms perform well for block level caches [31], which HTDA mimics at a larger scale. Owing to the targeted scope, the approach used has some specific features.

First, the parameter Δt limits the observed time period. This sets a limit on the age of metadata that needs storing. Compared to other definitions of LRFU, the linear decay of $1/\Delta t$ sacrifices smoothness. Since the frequency of evaluations is high compared to Δt, continuity is sufficient.

Second, the moment in time t_j of an access to an item by a job is subject to interpretation. A job covers a range of time: it may be queued for hours and run for hours. During the later, it may repeatedly access arbitrary portions of an item. Thus, the middleware uses the time a job is first queued. While not precise, this has several advantages: Actual accesses may be anticipated, allowing data to be cached before a job is run. Jobs rerun by the batch system because of failures are not counted repeatedly. Recorded dates are independent from speedups due to caching; this avoids feedback loops and allows comparing different algorithms.

Finally, the definition is for generic *items*, not files. This reflects that processing is based on groups of files, which are part of datasets. The model used for grouping of files in turn defines how items and files are related.

File Grouping

As outlined in Section 3.2.2, caching individual files only is not adequate. Scheduling data must respect that jobs access *groups* of files. Yet, there are no constraints on this grouping: users are free to choose and change their own partitioning methods.

Analysis of usage data shows some structure to file grouping in most HEP user analyses: Groups of files partition datasets in arbitrary but stable ordering. The size of groups varies, but certain sizes dominate. As a result, specific groupings are more common than others.

Motivated by this, the partitioning model defines scoring in two ways: First, the items rated are the groups of files as used by jobs. This respects that no constraints can be assumed for grouping. Second, each file is assigned the score of the highest ranking group it belongs to. All lower ranking groups are ignored. This approach implicitly prefers workflows with stable grouping.

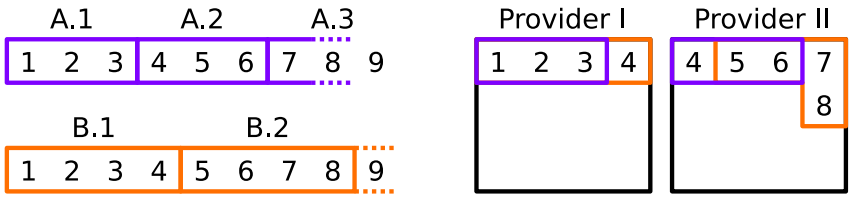
The distinction between file and item score decouples component implementations. All components other than the `Coordinator` operate at the granularity of individual

files. No information on how files relate to each other is provided. Instead, the `Coordinator` is expected to encode relationships in the score provided. This allows for arbitrary scoring mechanisms by the `Coordinator`.

Data Placement

In addition to rating files, the `Coordinator` must also assign them: each `Provider` service is regularly assigned a set of files for provisioning. Similar to scoring, this process must take into account the grouping of files. In addition, it must reflect that groups may overlap.

The HTDA middleware uses a heuristic approach to data placement, as shown in Figure 3.10. Each `Provider` node is abstracted to the sum of its cache volumes[4]. File groups are distributed to nodes in a top-down approach: Beginning with the highest scored group, groups are attempted to be assigned in order.



(a) Grouping of Files by Jobs **(b)** Placement of File Groups

Figure 3.10: File Grouping and Placement: User workflows are free to implement their own partitioning of a dataset to jobs. Figure 3.10a shows the same dataset of files 1, 2, ... partitioned differently: Two workflows using file groups A.1, A.2, ... and B.1, B.2, ... for their jobs, respectively. Placement of files by HTDA attempts to maximise overlap for file groups of the same dataset. Figure 3.10a shows the placement of file groups A followed by B: Since A.1 contains most files of B.1, the missing file 4 is added to the provider of A.1.

Each group assignment is performed with respect to placed files. If a group is already available on a `Provider`, no action is triggered. Otherwise, the `Provider` nodes providing the highest fraction of the group are determined. From these nodes, one is selected randomly.

[4] In principle, each `Provider` node also publishes the metadata of the worker nodes it services. This information is currently not used by the placement algorithm.

3.3.4 End User Workflow Integration

The redirection layer of HTDA is split into multiple elements. The only general element are `Locator` components. Hooks are used to handle the specifics of batch systems and end user applications.

Location Information

The most important information for external resources is the location of files. This is required to schedule jobs, and may be used to optimise file grouping. However, this information is distributed over all `Provider` nodes. Directly querying this information is time consuming and inefficient.

Thus, the `Locator` service is designed to efficiently expose information. Internally, it acts as a proxy for metadata of the HTDA middleware. All `Provider` nodes are periodically queried for metadata of all local files. Intermediate changes are pushed to `Locator` nodes by `Provider` nodes.

The `Locator` is optimised to service the most common external requests. For a given job and file group, the worker nodes with best data locality are provided. For a given dataset, the best partitioning to optimise locality is provided.

By design, the `Locator` is expendable to an HTDA cluster: `Locator` nodes only act as a cache for metadata. This contrasts it to similar concepts, e.g. Hadoop's `NameNode`. Multiple `Locator` nodes may be deployed at the same time to improve fault tolerance and availability.

Integration to HTCondor

The HTDA currently provides hooks to integrate into an HTCondor batch system. The choice for HTCondor is due to its availability at IEKP and its extensibility. The batch system has built-in support for hooks at various points.

HTDA provides hooks which integrate into HTCondor's `job_router` service (see Appendix D). Using the `job_router` serves two functions: First, it performs a preselection of queued jobs. Second, it ensures hooks are invoked in a controlled fashion.

Preselection ensures that HTDA only interfaces with relevant jobs. Being part of HTCondor, the `job_router` efficiently watches the queue and inspects new jobs. Jobs are checked to provide appropriate metadata and possibly to satisfy other quality criteria, such as being limited in execution time. Only appropriate jobs get managed by HTDA via its hooks.

This preselection provides an event-driven handling of jobs in the HTDA infrastructure. In turn, an arbitrary number of jobs not managed by HTDA are allowed without negative effects. This allows deploying HTDA at arbitrary capacity alongside other infrastructure.

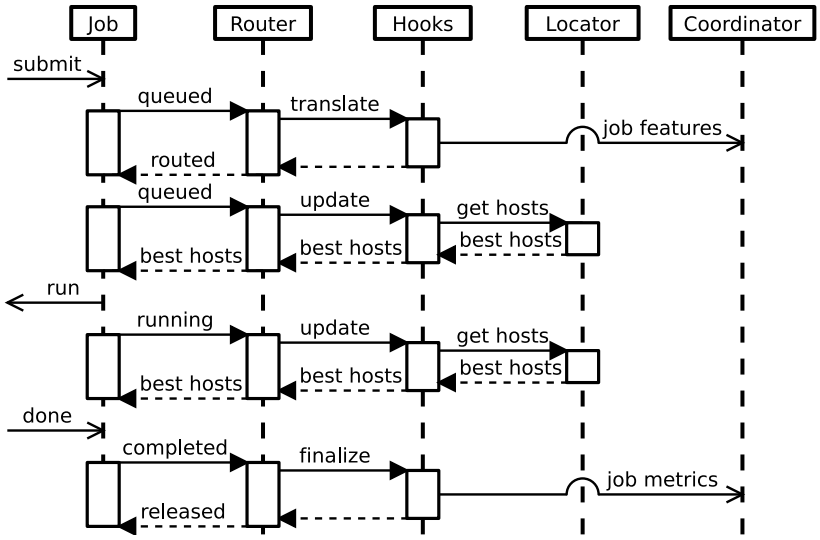


Figure 3.11: Operation Sequence of the HTDA HTCondor Hooks: By using the `job_router` of HTCondor, hooks interact with individual jobs. Hooks publish information to the `Collector` when the job is submitted or done. In turn, hooks add information on preferable hosts while the job is queued or running.

Once a job has been selected as suitable, the HTDA hooks interface with the HTDA infrastructure, as shown in Figure 3.11. There are three distinct stages, each handled by a separate hook: On being selected, jobs are *translate*d. While queued or running, jobs are regularly *update*d. After completion or failure, jobs are *finalize*d.

HTDA use the *translate* and *finalize* stages to collect metadata. Both types of hooks send job metadata to the `Collector`. The *translate* hook collects static information: which files are used, owner, and resources such as memory required. The *finalize* hook collects monitoring information, such as exit code, wall time, hit rates, and worker node.

The *update* stage is used to influence job scheduling. The hook queries a `Locator` for suitable hosts. This information is added to a job's ClassAd Rank statement, as shown in Listing 3.4. In turn, HTCondor uses this statement to prioritise a worker node for execution.

```
1 Rank = ( 0.0 ) + HTDA_RANK
2 HTDA_COLLECTORS = "http://ekpsg03.physik.uni-karlsruhe.de:8082"
3 HTDA_CACHEHIT_RATE = 0.7096774193549999
4 HTDA_LAST_UPDATE = 1465383130.87
5 HTDA_LOCALITY_RATE = 0.548387096774
6 HTDA_LOCATORS = "http://ekpsg03.physik.uni-karlsruhe.de:8081"
7 HTDA_MACHINE_RANK = 0 + ( ( machine == "ekpsg02.physik.uni-karlsruhe.
```

```
     de" ) * 15 ) + ( ( machine == "ekpsg03.physik.uni-karlsruhe.de" )
      * 16)
8 HTDA_RANK = HTDA_MACHINE_RANK
```

Listing 3.4: ClassAd values of a job modified by HTDA hooks. Hooks add metadata required for scheduling and reporting. The RANK statements are used to influence which worker nodes are preferred for execution. Other metadata stores monitoring information and which HTDA nodes to report to.

The HTDA hooks work mostly with the metadata provided by the batch system. The only exception to this are job input files; there is no automated way to extract this from arbitrary jobs. Thus, users are required to publish a list of input files with their job.

Application Redirection

The combination of `Locator` nodes and hooks redirects jobs to worker nodes where input files are cached. However, the actual access by applications must be redirected as well. While not handled by the HTDA middleware, it is required for it to work. Each backend expects a specific scheme for this redirection.

The backend for POSIX accesses uses distinct base paths to distinguish source and cache devices. A prefix based renaming scheme maps every file to a path on a device, as shown in Figure 3.12. This method is derived from the trivial file catalog [32] scheme used in HEP.

In principle, the rules for file access are simple: Read accesses should try any cache paths and fall back to the storage path. Write accesses should directly address the storage path. Yet, implementing this is non-trivial: POSIX accesses are part of the operating system.

User Path: `/storage/a/skims/zjet/Zmm_1866.root`
Source Path: `/htda/storage/a/skims/zjet/Zmm_1866.root`
Cache Path: `/htda/cache1/htda/storage/a/skims/zjet/Zmm_1866.root`

Figure 3.12: Naming Scheme of POSIX Backends: The HTDA POSIX backends use a naming scheme to map files to locations. The basis is the global path of a file, i.e. the path by which users access it. Prefixes are used to signify source and locality. Remote files are identified by the storage path. It adds the mount path of remote storage (purple) as a prefix. Local copies are identified by the cache path. It extends the storage path by prepending the cache device mount path (red).

Three solutions for redirection have been tested:

- Applications aware of the scheme may resolve locations by themselves. However, this requires modifications to the applications, which HTDA is meant to avoid.
- `Provider` nodes may merge local and remote paths using links. The resulting directory structure is effectively read-only.
- File paths may be squashed inside the Virtual File System. This requires specialised file systems.

The suggested method is the use of dedicated file systems for merging paths. Union File Systems allow overlaying local and remote directory hierarchies, as shown in Figure 3.13. The main advantage of this is transparency for users and applications: Read accesses are transparently serviced from cache devices if possible. Advanced union file systems allow write-through, i.e. directing write accesses directly to storage. The downside is that such file systems are not available for all operating systems.

3.3.5 Applicability to other Environments

The environment at IEKP is well suited to develop a prototype for coordinated caching. However, the concept itself is not limited to this specific processing environment. There are other environments where coordinated caching may be beneficial. HTDA has been designed with extensions for this in mind.

The restriction to local cache access is not always adequate. High Performance Computing (HPC) infrastructure usually has small, fast storage shared by worker

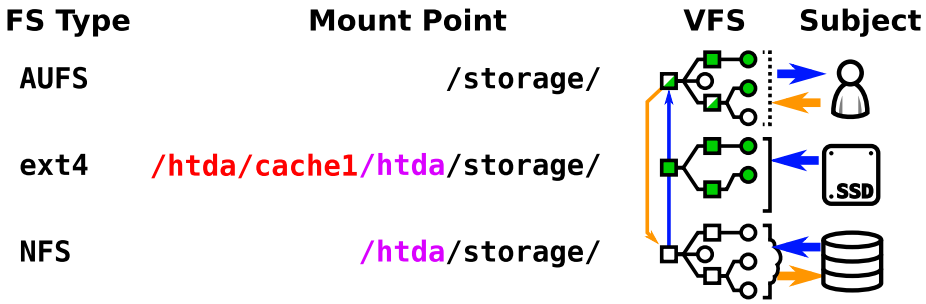


Figure 3.13: Squashing Paths for End Users via Union File System: As users and applications expect a single file hierarchy, paths must be merged. A Union File System squashes individual directory trees present in the Virtual File System. Given distinct local and remote directory trees, a single, merged tree can be presented to users. Complex Union File Systems such as AUFS also allow write-through to the underlying storage.

nodes. An HTDA system could use this as a cache spanning multiple nodes. The middleware already supports POSIX based parallel file system. However, scheduling of data and jobs is built on the assumption that one `Provider` maps to one worker node. Both algorithms and metadata would have to be generalised.

A major motivation of coordinated caching is to speed up remote accesses. This makes protocols other than POSIX suitable. Specifically, common HEP protocols such as xRootD allow access to WLCG storage. As HTDA interacts with resource consumers, not providers, it could transparently overlay WLCG storage. For HTDA itself, this merely requires new `Provider` backends. However, a different local redirection method is required. Related work shows this to be feasible [25] in general.

Finally, a notable fraction of computing resources can be provided opportunistically [33]. This implies that both processing and storage resources are volatile. Technically, the HTDA middleware is built to deal with this. However, it is only handled in terms of fault tolerance. To fully leverage opportunistic resources, scheduling must actively take features of these resources into account. For example, data expected to be important over a long period of time should reside on an equally long lived storage.

3.4 Evaluation of the HTDA Prototype

An instance of the HTDA middleware is already in use at the IEKP. Initially used for development, it is now primarily used to speed up end user data analyses. Being deployed for several months, multiple parts of the system have been evaluated.

3.4.1 Middleware Performance

The HTDA infrastructure is deployed on dedicated hardware, as detailed in Appendix E.1. This hosts worker nodes with SSD cache devices and one `Provider` service each. A single `Collector` and multiple `Locator` nodes are also part of the infrastructure. These are located on separate service and submission machines.

A key design feature of HTDA is transparency. This is not only in regards to user workflows, but also to the processing infrastructure.

Integration with existing Infrastructure

The batch system itself, including submission nodes and scheduler, is shared with other IEKP infrastructure. Modifications are confined to submission nodes, where the HTDA hooks have been installed. Other services, e.g. for dynamic cloud resource provisioning [33, 34], can be deployed in parallel.

Supported sources for data are the file servers available at IEKP. These are overlayed with AUFS [35] union file systems on worker nodes. This redirects accesses to data transparently: Read accesses are serviced from local devices if possible. Write accesses are directly directed to the file services. There are no changes required on the file servers; the default export via NFS is sufficient.

It is worth noting that this mechanism is tied to the availability of an appropriate union file system. The Linux mainline union file system OverlayFS [36] is not suitable for use with HTDA. For the default HEP operating system, Scientific Linux 6 [37], older versions of AUFS are available. However, these implementations are not stable for production use. Operating systems using current Linux kernels 3 or 4 and current versions of AUFS are required. The prototype system deploys CentOS 7 [38] with a 4.5 kernel and AUFS 4.

Overhead and Scalability

The overhead from HTDA is negligible at the scope of a processing cluster. Additional resource consumption, as shown in Table 3.1, is well below resource usage for analyses. Network overhead due to communication is unnoticeable. It is worth emphasising that the system is a prototype running in CPython2.7; neither source code, actual application, nor configurations are optimised for performance of the HTDA middleware itself.

On each worker node, the `Provider` node consumes $(15 \pm 5)\,\%$ of a single core. This is below $1\,\%$ of each worker node's total CPU capacity. Each `Provider` hosts roughly 7000 files, storing $(3.0 \pm 0.5)\,\mathrm{MB}$ of persistent metadata. The memory footprint is roughly 120 MB. Compared to the 500 GB cache volume and 64 GB RAM, both are negligible.

The `Locator` nodes are practically unnoticeable on the batch system's submission hosts. Both CPU and memory consumption is negligible. The integration via the

Table 3.1: HTDA RESOURCE CONSUMPTION: CPU consumption is relative to a single core, i.e. 3200 % equals full capacity. RSS is the resident set size, i.e. memory exclusively held by the application. Persistent Data is the size of data stored on disk. Values have been collected via the `GNU ps` utility.

Component	CPU	RSS	Persistent Data
Cache	$(15 \pm 5)\,\%$	120 MB	$(3.0 \pm 0.5)\,\mathrm{MB}$
Locator	$< 1\,\%$	60 MB	None
Coordinator	$(10 \pm 5)\,\%$	1 GB	250 MB

`job_router` ensures adequate scalability. The limiting factor in scalability tests on job volume and frequency is the batch system itself.

The major challenge for scalability is the `Coordinator`. It is the only component which cannot be scaled out to multiple instances. As such, the current implementation limits the scalability of the selection and coordination layer. However, this limitation is not enforced by other components; scalability only requires refactoring or recreating the `Coordinator`.

In addition, performance of the `Coordinator` [15] is sufficient. For six months of operation, persistent metadata is only 250 MB in size. Algorithm runtime for rating and placing files scales linearly with respect to either file or file group count [15]. Both file and group count are effectively bound by the time period in which accesses are deemed relevant. At the current scale, rating and placement calculation as well as updates to `Provider` nodes take roughly 10 s to complete; due to the scale of data transfer speeds, it is sufficient to perform these steps only every 30 min.

The only element critical for final end user data analysis speedup is the redirection of data access of analysis applications. No other element is actively involved with end user workflows. Benchmarks reveal no notable overhead from redirection via a union file system, as shown in Figure 3.14. This means end users benefit from the full performance of the available processing resources.

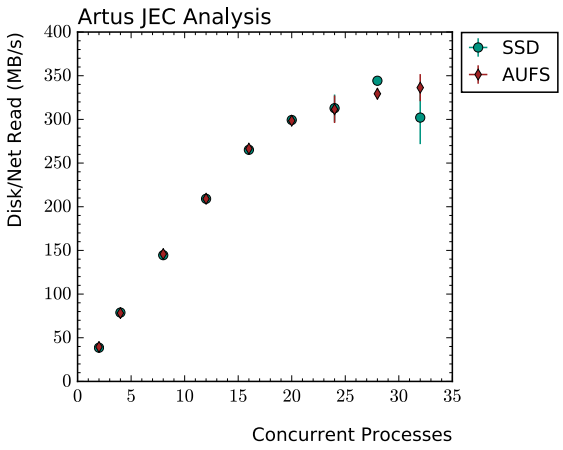


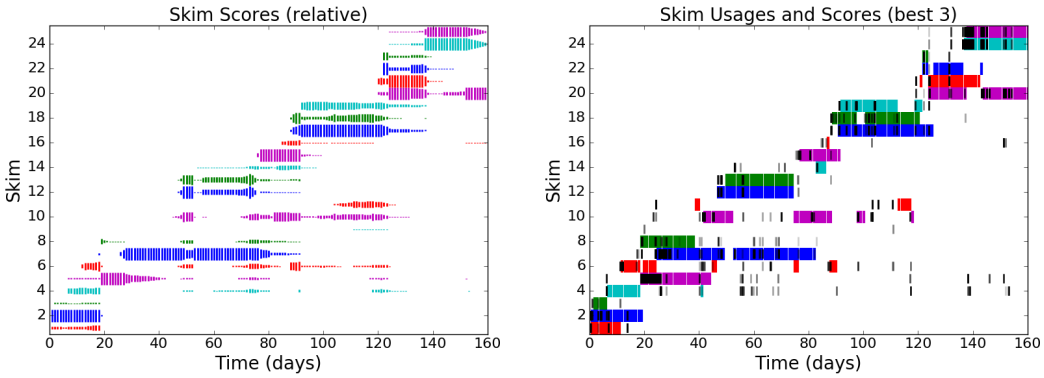
Figure 3.14: Performance of Raw and Overlayed Input Devices: Benchmarks are the same as shown in Figure 3.1a. in particular, the AUFS union file system is benchmarked; it squashes the previously benchmarked SSD and 10 Gbit connected file server. All data was manually copied to the SSD. The benchmark shows no notable overhead from squashing. Differences for high process counts are due to interference with other processes on the host.

3.4.2 Data Scheduling Performance

Being a distributed system, the HTDA middleware has an increased parameter space compared to regular caches. For example, it is not sufficient for data to be cached, it must also be accessible by jobs. This makes it impossible to assess the performance of the middleware with only a single metric like the cache hit rate. The development of system metrics is an ongoing work [39]. Several key features are pivotal enough to be studied in isolation, however.

The global cache hit rate limits the optimal performance. Predicting data accesses is required for all other tasks. To study this, the rating formula defined in Equation 3.9 has been applied to recorded dataset accesses. The resulting rating is shown in Figure 3.15a. Several features can be identified that reflect the behaviour described in Section 3.1.1: A number of pivotal datasets dominate time periods spanning multiple days. At many times, two datasets are ranked equally important, reflecting the use of detector and simulation data in parallel.

To simplify the interpretation of scores, the highest ranking datasets may be selected. This gives an estimate for what would actually be cached. An illustration is shown in Figure 3.15b, while hit rates are listed in Appendix E.2. With two exceptions, all datasets with more than 500 accesses have cache hit rates in the range of 70 % to 90 %. In addition, the selection of datasets is stable: once selected, most datasets keep their rank for several days.



(a) Dataset Ratings Relative per Timeframe **(b)** Dataset Accesses and Highest Scores

Figure 3.15: Selection of Datasets for Caching: Ratings for datasets have been calculated by applying the HTDA scoring formula to recorded accesses. The formula is given in Equation 3.9, while the accesses are shown in Figure 3.2. Figure 3.15a shows ratings as heights. For visualisation, ratings are normalised to the maximum score at every point in time. Figure 3.15b shows accesses in black. Coloured bands indicate that the dataset is amongst the three highest ranking datasets.

The scheduling of data and jobs is only meaningful when studied together. The availability of data automatically limits the effectiveness of job scheduling, as shown in Figure 3.16. In most cases, the partitioning of datasets used by HTDA matches that of jobs. Nearly all jobs have either all or none of their input files available on one worker node. As desired, this maximizes job efficiency, and minimizes unused data. For jobs with cached input data, roughly 75 % are scheduled perfectly.

While effective under most circumstances, scheduling may be further improved. Scheduling of jobs currently uses an all-or-nothing approach to data availability: There is a short delay in which a job is only allowed to start on worker nodes where its input data is cached. Afterwards, jobs may be started on any node. This policy could be relaxed to a soft delay depending on the maximum attainable hit rate: if no or little input data is cached anywhere, there is little benefit in delaying execution. Furthermore, the delay could be adapted if resources are scarce during periods of high utilisation: A longer delay would increase overall throughput, since jobs are more likely to run on cached input data.

Experience shows that file grouping is not handled optimally. On the one hand, the assignment of highest group score as file score can lead to fragmentation. If file grouping changes, previously assigned files may stick to `Provider`s in suboptimal groups: Consider files initially placed in pairs, but later used as sets of four. The pairs would still have the high rating for optimal access, while providing only half the data for this access. Setting the file score on `Provider` nodes to the highest *local* group score would penalise this. On the other hand, using only the groups provided

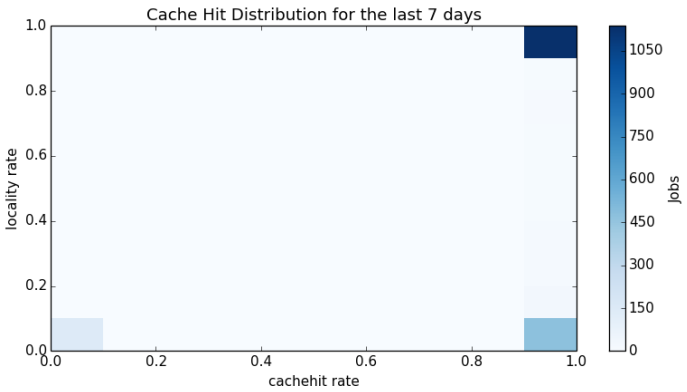


Figure 3.16: Global and Local Cache Hit Rate: Hit rates are recorded by the HTDA middleware. The `cachehit rate` is the maximum hit rate possible for a job on any worker node. The `locality rate` is the actual hit rate on the worker node on which the job is executed. Data shown is an excerpt from monitoring from 15th December 2015 to 22nd December 2015. [39]

by jobs is suboptimal. Smaller groups could often be combined without side effects. For example, jobs using strict subsets of bigger, common groupings ($\{\{1,2\},\{3,4\}\}$ versus $\{\{1,2,3,4\}\}$) could be merged. Identifying file groups via a frequent itemset approach [40, 41] could implicitly resolve this.

3.4.3 End User Data Analyses Performance

A key performance indicator for caches is the benefit on throughput or performance. For a system such as HTDA, many parameters influencing this are impossible to control, however. Instead of monitoring overall throughput, a single analysis has been selected as a benchmark.

The CMS Z + Jet calibration analysis detailed in Chapter 4 is highly data driven. It mostly depends on input rate, as only few calculations are performed compared to other analyses. In addition, processing speed is critical to derive corrections in a timely manner. As such, the Z + Jet calibration analysis is the reference analysis for the HTDA system.

The walltime of jobs is closely related to the performance of workflows. For the reference analysis, this has been tracked both with and without cache, as shown in Figure 3.17. On the IEKP cluster, HTDA improves processing speed roughly by a factor of 4 on average. Notably, HTDA is not at its limit of scalability, while the current infrastructure is; the relative speedup increases with more worker nodes.

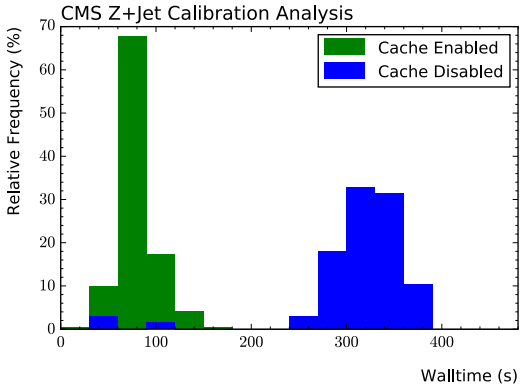


Figure 3.17: Analysis Speed with HTDA: Per-job walltime of the Z + Jet calibration workflow, with enabled or disabled HTDA caching. For both tests, the same workflow was processed on the same cluster, with no other workflows interfering. Data has been collected with the `GNU time` utility. Lower walltime is better. Outliers of the workflow without HTDA cache are executed after the bulk of the workflow. This demonstrates that network bandwidth is saturated when all jobs execute in parallel.

3.5 Conclusion and Outlook

The work presented in this chapter is an important step towards efficient data analysis in HEP. Modern processing paradigms are transparently introduced to existing workflows. Benefits are not limited to short term improvements; the scalability of the approach enables efficient end user data analysis even in the future.

While existing approaches build on similar paradigms, this approach strives for generalisation. It demonstrates how a minimalistic, modularised approach allows integrating modern techniques into established workflows. Furthermore, its focus on extensibility allows introducing additional techniques in the future.

Building on caching and batch processing, the new approach works with well established techniques. Yet, the added scope and scale give rise to new challenges. Solutions chosen in implementing the approach perform adequately for the studied use case. However, extending and optimising the approach is subject to future research.

With the HTDA prototype, the new approach is not just a generic approach. It has already made a notable contribution to the analysis of LHC run 2 data. Inarguably, such a prototype cannot be compared to a production system. Yet, the middleware is in a state adequate for stable operation, providing notable benefit far outweighing its cost.

Today, the HTDA prototype is used to improve the calibration efforts of the CMS collaboration. The Z + Jet calibration analysis described in Chapter 4 is perfectly suited for such a system. It has been adopted for use on the prototype easily. Being a time critical analysis, it greatly benefits from the speedup provided.

Chapter 4

Calibration of Jets with $\mathrm{Z} \to (\mu\mu/\mathrm{ee})$ Events

> It is the mark of an educated man to look for precision in each class of things just so far as the nature of the subject admits.
>
> Aristotle

The size of the Large Hadron Collider (LHC) and its detectors enables unprecedented physics analyses. At the same time, the method by which this is achieved has severe implications on precision and physics performance. Performing high precision analyses requires a precise modeling of how detectors respond to particles from collision events. Dedicated calibration analyses are required to enable other high precision analyses in the first place. The goal is the provisioning of corrections, which relate measured properties to their true scale.

Reflecting the scale of the LHC experiments, calibration is an extensive effort involving multiple analyses. Individual results often influence one another, requiring an iterative approach and ongoing effort. Separate calibrations are performed for different detector components and particles. Jets are amongst the most complicated objects to calibrate, since they consist of multiple particles.

Jets are an integral part of collision events at the LHC. On the one hand, most analyses rely on event signatures featuring jets: jets are used as part of a measurement, or to separate signal and background interactions. On the other hand, jets are present regardless of the interactions studied. Colliding bunches of hadrons are practically guaranteed to result in jets from additional interactions, the so called Pileup.

As such, the calibration of jets is integral for most physics analyses performed at the LHC. The Jet Energy Corrections (JEC) relates the energy of quarks and gluons to the measured energy of their experimental signature, the jets. Due to the complexity of this task, corrections are derived in multiple stages. Since JEC are required by practically all analyses, both precision and speed are crucial.

The basic approach to JEC in Run II is derived from methods used in Run I [42]. However, new conditions due to higher energy and increased interaction frequency

call for new corrections and new methods. In addition, technical problems in data taking have made the speed of deriving corrections even more important.

As part of this thesis, residual corrections for jets have been derived from Z → (μμ/ee) + Jet events. An overview of the JEC effort is given in Section 4.1. Section 4.2 describes the technical and physical background of the Z → (μμ/ee) + Jet analysis in general. The derivation of corrections is detailed in Section 4.3. Finally, Section 4.4 provides a conclusion and outlook of the calibration effort.

4.1 Jet Energy Calibration in the CMS Experiment

The accuracy for measuring properties of jets is inherently limited by jets being compound objects. Reconstructing jets involves combining information on multiple particles from many subdetectors. Reconstruction efficiency is limited by physical processes, algorithms, and the detector itself.

The CMS collaboration has a dedicated calibration effort to counter such limitations. The objective of this calibration is to match the measured jet energy to the parton energy, as shown in Figure 4.1. The calibration is split into several stages, each providing dedicated corrections to specific effects. Both sequence and scope of correction stages are formalised.

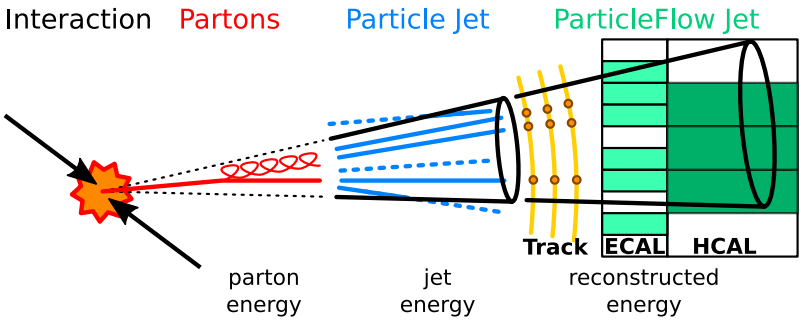


Figure 4.1: Scopes of Jet Energy Measurement: Jets are measured to derive the parton energy. This is the energy of the parton originating from the initial interaction. However, the parton cannot be detected, only the particle jet resulting from it. In the process of clustering a jet, particles may be incorrectly included or excluded. Finally, actually detecting the jet in the detector involves several subdetectors. From the final measurement, features of the particle jet and the parton must be reconstructed. Each intermediate stage is subject to inefficiencies and biases.

4.1.1 Calibration Stages

There are three mandatory stages of corrections for every CMS analysis. Each addresses a different type of inefficiency in determining the initial parton energy. Separate handling of different effects allows the use of dedicated correction methods. In addition, corrections may be derived differently for simulated and recorded events. The three correction stages *Pileup offset corrections*, *simulation based corrections*, and *residual corrections* as well as their sequence are shown in Figure 4.2.

Figure 4.2: Stages of Jet Energy Corrections of the CMS Collaboration: Corrections are handled differently for data from the detector or simulation. Corrections for Pileup background are derived from background studies of simulation and detector data. The expected response is derived from simulation data, but applied to detector data as well. To account for unexpected effects, additional corrections are derived for detector data only.

The first stage is **Pileup offset corrections**. These subtract particle energy erroneously clustered into a jet. A common source for this is Pileup, i.e. particles from additional interactions. This stage also handles phenomenologically similar sources of error, e.g. noise. Corrections are derived separately for data from detector and simulation.

The second stage is **simulation based corrections**, which use complete simulations of interaction and detector. This accounts for all known negative effects of jet reconstruction. Such effects include inefficiencies of detector response and jet clustering. Corrections are derived from simulated events. This gives access to generated partons, simulated particle jets, and simulated detector responses. The same corrections are applied to both detector and simulation data.

The third stage is **residual corrections**. These compensate for differences in simulated and actual response. Corrections are derived separately in two stages from multiple channels. The basic method is the same for all channels: Jets are compared to a reference object that is well understood in simulation and the detector.

4.1.2 Data Driven Residual Corrections

Residual corrections are split into two parts: **Relative corrections** ensure uniform jet energy response across detector regions. **Absolute corrections** ensure uniform jet energy response with regard to jet energy. Combining both parts ensures proper calibration of jet energy in the entire detector.

Both correction parts use balancing methods: Events are studied in which the hard interaction produces a jet and a reference object in opposite direction. Due to the negligible transverse momentum of the LHC's beams, the two objects are balanced in the transverse plane: transverse momentum is the same, but orientation in ϕ is opposite. Measuring the reference object properties provides an estimator for the properties of the parton from which the jet originates. Additionally, by using well known reference objects, jets from simulation and detector can be compared implicitly via the reference object.

To derive the relative corrections, a single jet in the barrel region is used as reference object. This region provides a reliable reference for several reasons, most importantly the uniform detector shape and high range of jet p_{T}. The jet to be corrected may be in any detector region. Corrections are derived from the imbalance of the two jets, parameterised as a function of η.

The absolute corrections use reference objects with good energy resolution. These are matched to jets in the central region. Three analyses, each using a different reference object, derive absolute corrections for Run II. One of these, using a Z boson as reference object, has been implemented and performed as part of this thesis.

4.2 Analysis of Z + Jet Events

Calibration with Z + Jet events has several advantages which offset the low cross section compared to other calibration channels. For example, Z properties have small systematic uncertainties. This allows for precise measurement of jet uncertainties. Different Z decay channels allow crosschecking several subdetectors against each other, as shown in Figure 4.3. Also, using a Z boson as reference object allows corrections at low jet p_{T}. These factors have made previous Z → μμ + Jet calibration studies an important contribution during Run I JEC [43].

Run II introduces new conditions that must be reflected in calibrations. The increased collision energy of 13 TeV and interval of 25 ns makes calibrations even more critical. The entire analysis has been recreated to be viable for techniques used in Run II.

The analysis uses the Excalibur analysis suite [33]. It extends the Artus framework and workflows, adding tools for detailed study of jets at different correction levels. Owing to the impact of CMS JEC, an important focus has been on improving speed and robustness against user errors. This is implemented in the analysis itself via automatisation. It is also further facilitated by an optimised infrastructure, as detailed in Chapter 3.

In addition to the Z → μμ + Jet studies building on experiences from Run I, the calibration has been extended to use Z → ee + Jet events [44]. This increases statistics on the one hand, and allows for internal crosschecks on the other hand.

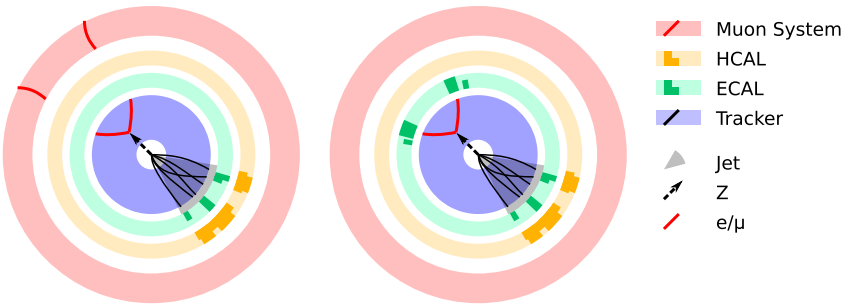


Figure 4.3: Subdetectors Contributing per Channel: The $Z \to \mu\mu$ + Jet and $Z \to ee$ + Jet calibrations use different subdetectors. In both cases, jets are covered by the Tracker and calorimeters. Also, both electrons and muons are visible to the Tracker. For $Z \to \mu\mu$ + Jet, the Z decay products are also covered by the dedicated muon trackers of CMS. In contrast, $Z \to ee$ + Jet relies on the electromagnetic calorimeter.

Finally, new methods such as the PileUp Per Particle Identification (PUPPI) [45] Pileup corrections have been studied and enabled for corrections [15].

The studied number of combinations of channels, corrections, data taking periods, and Pileup mitigation techniques is beyond the scope of this chapter. As such, only a representative selection of plots is shown in this chapter. Unless otherwise noted, plots show $Z \to \mu\mu$ + Jet events with CHS Pileup mitigation. Events are taken from the 2015D data taking period of the CMS collaboration. Configurations of studied variants are listed in Appendix A. Additional plots may be found in Appendix B.

4.2.1 Data Preprocessing

The analysis is implemented using the Excalibur analysis suite. It builds on the Artus framework and thus inherits its general workflow design: Official CMS data is skimmed using the Kappa framework. Observables are extracted using an Artus analysis. Finally, observables are aggregated to create histograms, fits, and other representations.

Before the actual Artus analysis, several preprocessing steps are performed. From a technical side, these reduce data size and complexity. More importantly, they collect and apply metadata relevant for physics results.

Excalibur relies on metadata from external sources to process data in Artus. Fetching, processing, and applying metadata has been automated as a preprocessing step. This ensures that calibration is always performed with up-to-date parameters. Furthermore, it eliminates many sources of user error. Most importantly, it guarantees that settings are used consistently even when manually overwritten.

Skim Preprocessing

Skimming applies basic preprocessing to each event. Leptons with too little transverse momentum are rejected as Z decay candidates. Only events with at least two decay candidates are kept.

In addition, the basic definitions of jets are set: All jets are clustered using the anti-k_t algorithm with multiple cone sizes, as outlined in Section 2.2.2. Results presented in this chapter use the CMS standard of $R = 0.4$. Constituents are particle flow candidates as outlined in Section 2.1.2. Skims of simulation datasets also include so-called generator jets. These are directly derived from simulated particles, without simulated detector interaction and reconstruction. Finally, a transverse momentum threshold is used for all jets to exclude very soft jets from Pileup.

$$p_{\mathrm{T}}^{\mathrm{l,skim}} > 8\,\mathrm{GeV} \tag{4.1}$$

$$p_{\mathrm{T}}^{\mathrm{Jet,skim}} > 5\,\mathrm{GeV} \tag{4.2}$$

Data Certification and Data Taking Periods

Not all data collected by the CMS detector is appropriate for analysis. For example, failure of the solenoid drastically reduces reconstruction reliability. Some data may only be adequate with constraints, e.g. excluding malfunctioning detector regions.

The CMS collaboration coordinates the use of valid data by certifying data taking periods, or *runs*. These are regularly updated by the CMS data certification group. Multiple levels of data quality are available, as shown in Figure 4.4.

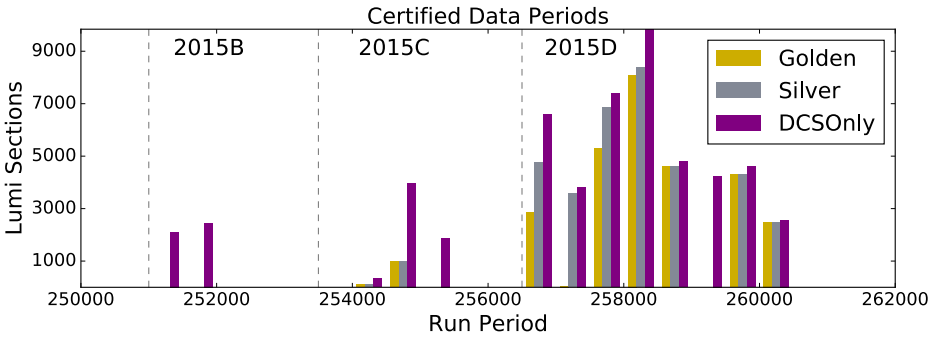


Figure 4.4: Certified Data Taking Periods: The CMS collaboration certifies periods of data taking, so-called *runs*. Certification provides different levels of data quality. High quality data covers less runs, and contains less data in each run. Quality levels used for calibration are Golden (best), Silver, and DCSOnly (worst).

The default for CMS analyses is the *golden JSON*. It offers the best data quality, at the cost of a reduced data volume. Calibration is performed with other data quality levels as well, if required. All results presented in this chapter use the golden JSON standard.

For run 2, data certification lists are also used to specify the bunch crossing interval. Separate lists are available for 25 ns and 50 ns intervals.

Since calibration depends on detector performance, certification information is used at multiple stages. As such, Excalibur does not use raw certification lists. It adds support for cutting and slicing to select appropriate runs. Results in this chapter show data from the 2015D data taking period, unless otherwise noted. This is the most recent period with stable conditions for which corrections have been derived.

The periods defined by certification and slicing are used as a filter on data from the detector. Only events recorded in such a period are processed. In addition, the used data periods implicitly define the Integrated Luminosity. Excalibur automatically queries this value for the specific data periods used.

Pileup Estimation and Weighting

Due to the LHC colliding bunches of non-elementary particles, Pileup adds notable background activity to events. Properly correcting for this effect is critical to achieve adequate precision for calibration. As such, simulations must closely replicate the real Pileup characteristics.

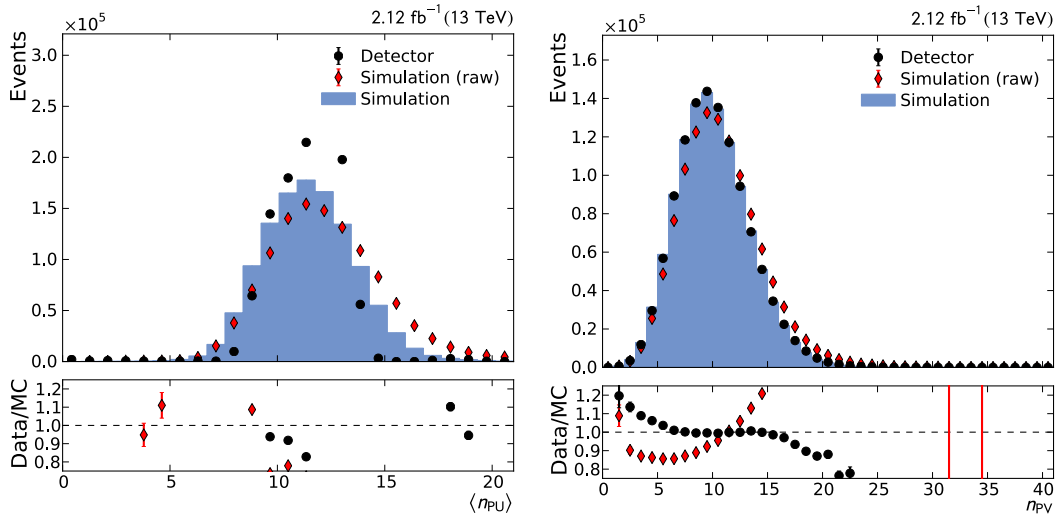
In general, simulations provided by the CMS collaboration do match the Pileup characteristics in the experiment. At the time simulations are performed, Pileup characteristics are often unknown. This is especially true at the beginning of the 25 ns data taking: compared to Run I, Luminosity has been increased and bunch crossing intervals shortened. Thus, simulations were created without knowledge about Pileup. Consequently, simulated events must be weighted to match the distributions found in detector data.

Weighting is performed based on the *expected* number of Pileup interactions μ or $\langle n_{\mathrm{PU}} \rangle$[1]. Recorded data provides only the number of actual Pileup interactions. The expected number must be estimated based on Luminosity and Pileup cross section. Excalibur automatically performs this for a given set of runs. The result is the distribution of the expected number of Pileup interactions for this run.

Simulated events directly expose the expected number of Pileup used in the simulation. Excalibur calculates the distribution of $\langle n_{\mathrm{PU}} \rangle$ for each simulation dataset. The distributions from detector and simulation data are normalised and binned. For each bin, the ratio of events in detector and simulation data is calculated. Each

[1] The canonical name in the CMS experiment is μ. To avoid confusion with muon quantities, the $\mathrm{Z} \to \mu\mu$ analysis uses the longer name $\langle n_{\mathrm{PU}} \rangle$.

simulated event then weighted by the corresponding ratio for its $\langle n_{\mathrm{PU}} \rangle$. This matches the distribution in the simulation to the detector data, as shown in Figure 4.5.



(a) Expected Number of Pileup Interactions

(b) Number of Pileup Interactions

Figure 4.5: Pileup in Detector and Simulation Data: Pileup is a key characteristic of events. Simulations must closely replicate the Pileup observed in the detector. This is ensured by weighting simulated events depending on the expected number of Pileup interactions. Figure 4.5a shows the expected number of Pileup, including the unweighted simulation. Simulation and detector data do not fully overlap, as $\langle n_{\mathrm{PU}} \rangle$ for the later is calculated per event. This excludes smearing from the uncertainty on $\langle n_{\mathrm{PU}} \rangle$. Figure 4.5b shows that the observed, actual number of Pileup agrees between data from detector and weighted simulation.

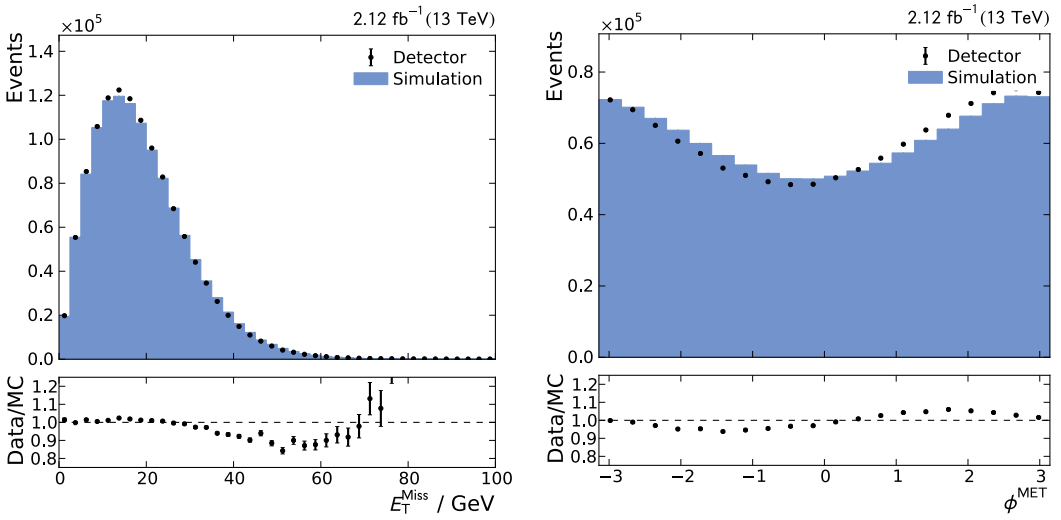
Preliminary Corrections

The purpose of calibration with Z + Jet events is to derive the final residual corrections. This implies that the base level of the calibration uses all previous correction steps. As such, Excalibur applies corrections individually to allow working at specific correction levels. Plots in this chapter include all correction levels before the absolute detector corrections, unless otherwise noted.

Since jet correction levels modify the jet energy, this also influences the Missing Transverse Energy (MET). As such, $\vec{E}_{\mathrm{T}}^{\mathrm{miss}}$ must be recalculated for each correction level, as defined in Equation 4.3. For every jet, its initial energy deposit $\vec{p}_{\mathrm{T,skim}}^{\,i}$ is replaced with the corrected energy $\vec{p}_{\mathrm{T,CL}}^{\,i}$. In addition, the offset for Pileup contribution $\vec{p}_{\mathrm{RC}}^{\,i}$ is added.

$$\vec{E}^{\text{miss}}_{\text{T,CL}} = \vec{E}^{\text{miss}}_{\text{T,skim}} - \sum_{i}^{N_{\text{jets}}} \left(\vec{p}^{\,i}_{\text{T,CL}} + \vec{p}^{\,i}_{\text{RC}} - \vec{p}^{\,i}_{\text{T,skim}}\right) \tag{4.3}$$

The corrections are consistent for data from detector and simulation, as shown in Figure 4.6. This feature is important for one of the absolute calibration methods used. Notably, MET is not isotropic in ϕ, as would be expected from the isotropy of the detector. This is due to a misalignment of the interaction point and the centre of the detector. However, the effect is cancelled out on average.



(a) Missing Transverse Energy

(b) Direction of MET

Figure 4.6: MET after Jet Energy Corrections: The MET is constructed from the energy of all objects in an event. Correcting jet energy for different correction levels thus requires recalculating the MET. As shown in Figure 4.6a, corrections are consistent between data from simulation and detector. The wave like shape in Figure 4.6b is caused by misalignment of the beampipe.

4.2.2 The Z + Jet Event Topology and Selection

For calibration, an ideal Z + Jet interaction produces a Z and a parton, balanced against each other. The parton undergoes showering and hadronisation, creating the jet to be calibrated. The Z decays to two leptons; the calibration is performed for decays to two oppositely charged electrons or muons. An ideal topology is shown in Figure 4.7.

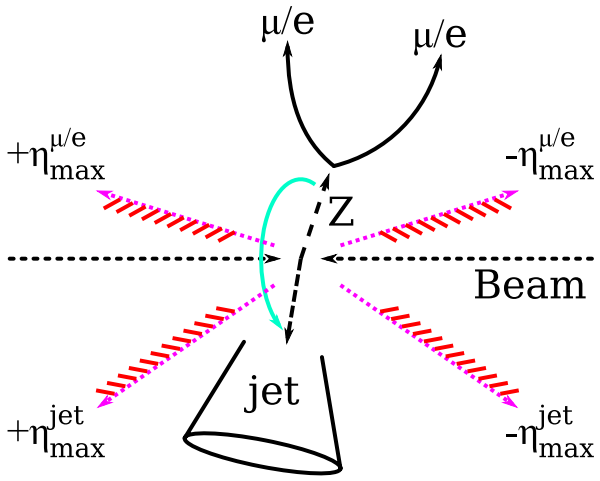


Figure 4.7: Schematic View of Ideal Z + Jet Events Used for Calibration: The actual interaction produces a single Z boson and parton. The initial transverse momentum is negligible. This results in Z and parton being oriented in opposite direction in ϕ. A jet originates from the parton, while the Z boson decays to a pair of electrons or muons. Requiring reliable detector coverage limits the η regions in which jets and leptons are accepted.

Constraints are required to select appropriate events from data. For the Z + Jet channel, background from other channels is low.Instead, additional particles from Pileup are the most common source for misidentifying events.

Constraints from Z Boson Decay

As the Z boson is not visible to the detector, it must be reconstructed from its decay products. To ensure precision, leptons must be fully covered by their respective subdetectors. Thus, a small border region of either ECAL or muon system is excluded as well. A minimum transverse momentum is required to ensure reliable detector response. Any lepton fulfilling these constraints is a potential Z boson decay product.

$$|\eta_\mu| < 2.3 \tag{4.4}$$
$$|\eta_\mathrm{e}| < 2.4 \tag{4.5}$$
$$p_\mathrm{T}^{\mu,\mathrm{e}} > 20\,\mathrm{GeV} \tag{4.6}$$

A valid Z boson is defined as a pair of oppositely charged leptons whose invariant mass is close to the Particle Data Group (PDG) Z boson mass. Since two leptons

are assumed to originate from a Z boson decay, exactly one Z boson must be reconstructable from any pair of leptons. The Z boson must also have a minimum transverse momentum. This avoids probing jet phase space regions with large systematic biases.

$$\left|m_{ll} - m_{\mathrm{Z}}^{\mathrm{PDG}}\right| < 20\,\mathrm{GeV} \quad \left(m_{\mathrm{Z}}^{\mathrm{PDG}} = 91.1876\,\mathrm{GeV}\right) \tag{4.7}$$

$$N_\mu + N_\mathrm{e} \leq 3 \tag{4.8}$$

$$p_{\mathrm{T}}^{\mathrm{Z}} > 30\,\mathrm{GeV} \tag{4.9}$$

Overall, this relaxes the constraints used during Run 1: Z boson reconstruction allowed for only a single pair of oppositely charged leptons. For Run II, an arbitrary number of leptons is allowed.

Jet Definitions and Constraints

The ideal topology includes only one parton and thus one jet. However, this is not a feasible constraint under experimental conditions. For example, additional jets may originate from Pileup or final state radiation from the parton. Jets are thus sorted by their transverse momentum in descending order as first, second, third etc. leading jet. Since additional jet activity is caused by soft processes, it is reasonable to match the first leading jet to the parton. For simplicity, this jet is simply called the leading jet.

Selection criteria for jets apply similar considerations as lepton selection: detector coverage and response, ensured by constraints on η and p_{T}. Only the leading jet is required to fulfil these constraints. Additional jets are not required to satisfy absolute thresholds.

$$|\eta_{\mathrm{Jet1}}| < 1.3 \tag{4.10}$$

$$p_{\mathrm{T}}^{\mathrm{Jet1}} > 12\,\mathrm{GeV} \tag{4.11}$$

A relative threshold is used to veto events with considerable additional jet activity. It is parameterised via α, the estimated relative energy loss of the leading jet. The Z boson momentum is used as an estimate for the true parton momentum. The second leading jet is used as a worst case estimate for leading jet radiation losses.

$$\alpha := \frac{p_{\mathrm{T}}^{\mathrm{Jet2}}}{p_{\mathrm{T}}^{\mathrm{Z}}} \tag{4.12}$$

$$\alpha < 0.3 \tag{4.13}$$

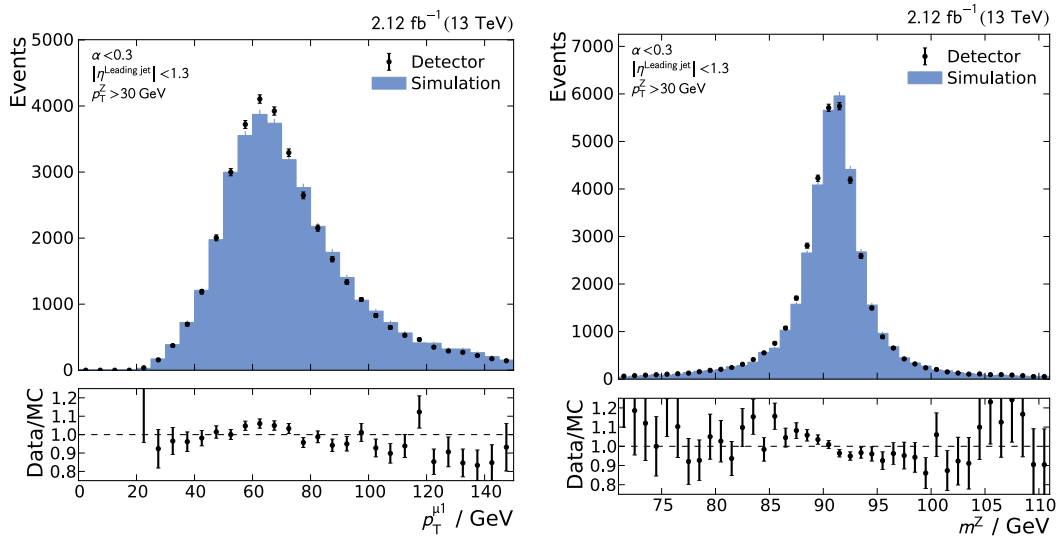
A key assumption for calibration is that the leading jet is balanced to the Z boson. As such, the two are expected to be oriented in opposite direction, or *back to back*. Since only the transverse momentum of the beams is negligible, a constraint is applied only in ϕ. As additional jet activity may bias this balance, a small margin of error is allowed.

$$|\phi_{\mathrm{Z}} - \phi_{\mathrm{Jet1}}| > \pi - 0.34 \tag{4.14}$$

4.2.3 Agreement of Simulation and Detector Data

The choice of Z + Jet events for calibration is motivated by Z boson features being well known. This implies that features of the Z boson must agree in detector and simulation. Thus, calibration requires thorough validation of its features.

In general, both individual leptons and the reconstructed Z boson are adequately described by simulation. Differences are within the expected precision of the detector. However, response of leptons currently seems to be overestimated, as shown in Figure 4.8. This effect is corrected before deriving jet energy corrections.



(a) Transverse Momentum of Leading Muon **(b)** Reconstructed Mass of Z Boson

Figure 4.8: Kinematics of Muons and Z Boson: The calibration with Z + Jet events relies on precise simulation of Z boson features. In general, muons are simulated adequately; detector response is slightly overestimated in simulation. This results in a shift of the Z mass by few permille.

As part of the validation, other features are checked as well. This encompasses kinematics outside the regions used for calibration and the composition of jets. As required by the calibration, data from detector and simulation agree in these features. As such, they are not further discussed here.

4.3 Calibration for CMS Run II

The Z + Jet analysis allows calibrating the jet energy scale. The goals are weighting factors that link the measured jet and initial parton energy, as defined in Equation 4.16. These factors must be separately derived for each detector region and every jet definition.

$$C = R_{\text{jet}}^{-1} = \frac{p_{\text{T}}^{\text{parton}}}{p_{\text{T}}^{\text{jet}}} \tag{4.15}$$

$$= \underbrace{\frac{p_{\text{T}}^{\text{parton}}}{p_{\text{T}}^{\text{jet,sim}}}}_{C_{\text{gen}}} \underbrace{\frac{p_{\text{T}}^{\text{jet,sim}}}{p_{\text{T}}^{\text{jet}}}}_{C_{\text{res}}} \tag{4.16}$$

Calibrations for CMS are parameterised by jet p_{T} and η. Multiple channels are used for best statistics and robustness. The Z + Jet analysis contributes both the $\text{Z} \rightarrow \text{ee}$ + Jet and $\text{Z} \rightarrow \mu\mu$ + Jet channels for Run II. In addition, different jet definitions are supported; at the moment, these are defined by the Pileup mitigation method used before clustering.

4.3.1 Pileup Mitigation

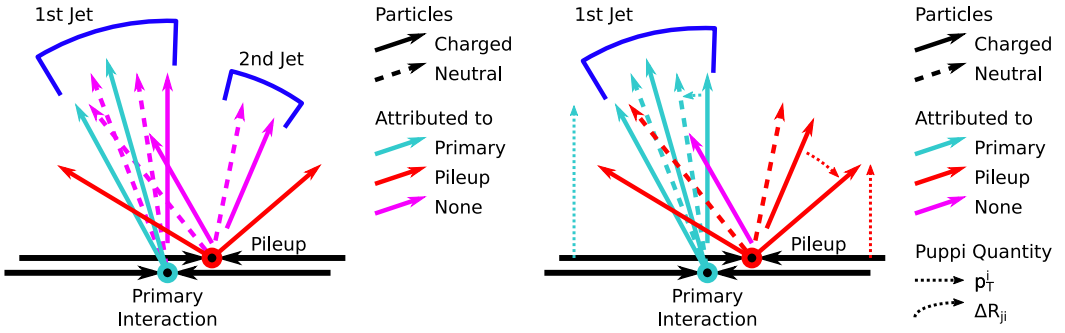
Additional jet activity has a severe impact on calibration performance. The calibration for Pileup effects only removes the average Pileup energy. Yet, jets from Pileup still remain in the event. Applying jet vetoes or studying jet distributions is biased by these artefacts.

As such, identifying and removing Pileup itself is important for calibration. The default method used during Run 1 is the Charged Hadron Subtraction (CHS) method. As charged hadrons are visible to the tracker, their point of origin can be reconstructed. CHS removes any particle that definitely originates from a Pileup interaction.

As CHS only removes charged particles, its effectiveness is limited. To remedy this, the PUPPI method has been introduced for Run II. It generalises the approach of CHS to all particles.

The PUPPI Method

Like CHS, PUPPI uses tracker information to identify charged particles originating from Pileup. However, it uses other features distinct to Pileup to generalise this information, and categorise other particles as well. This includes uncharged particles, but also charged particles whose origin is inconclusive. A schema of this method is shown in Figure 4.9.



(a) Pileup Identification with CHS **(b)** Pileup Identification with PUPPI

Figure 4.9: Pileup Mitigation for Run II: To identify Pileup for removal, the point of origin can be reconstructed from tracks. As this is only possible for charged particles, only these can be clearly categorised. The CHS method removes particles clearly identified as originating from Pileup. The PUPPI method uses this information to categorize other particles as well. It uses the kinetic properties of categorised particles as a classifier for all particles.

The approach uses a generic measure applicable to all particles. It expresses a Pileup likeness α_i for every particle[2] i, as defined in Equation 4.17.

$$\alpha_i = \log \sum_{j}^{\text{particles}} \frac{p_{\mathrm{T}}^{j}}{\Delta R_{ij}} \Theta\left(\Delta R_{ij} - R_{\min}\right) \Theta\left(\Delta R_{ij} - R_{\max}\right) \tag{4.17}$$

The measure is based on the fact that Pileup tends to produce soft jets, in contrast to jets from the hard interaction. A soft jet consists of particles with low energy, spread over a large area. Thus, particles in jets originating from Pileup tend towards low values of α_i. Particles from the hard interaction tend towards high values of α_i. The restriction to a local distance via $R_{\max}$ mimics the maximum size of jets from clustering. Excluding a smaller distance $R_{\min}$ avoids singularities from collinear splitting.

[2]It is important to distinguish between the per-particle measure α_i used for PUPPI and the per-event measure α used for JEC. The two are not equivalent.

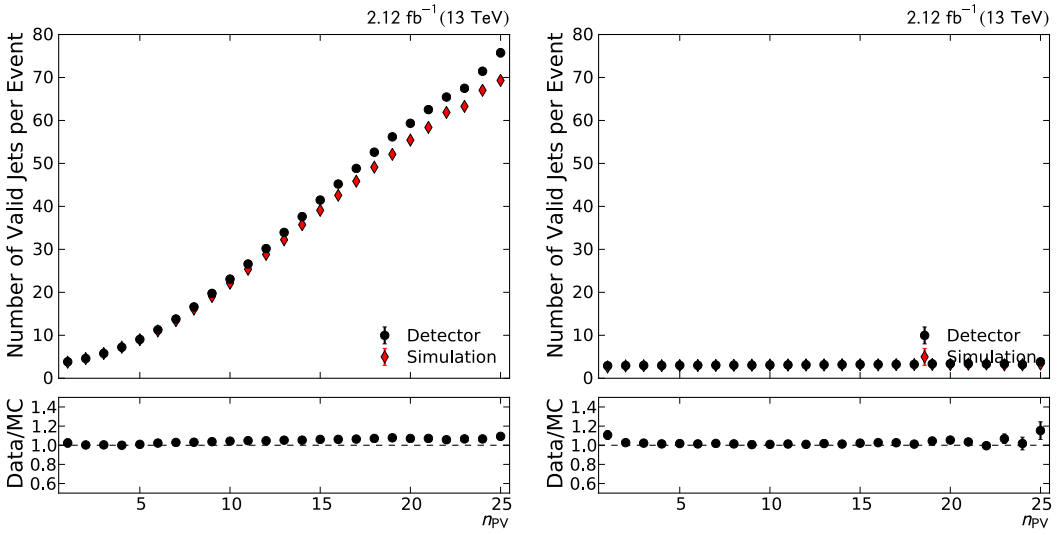
For each event, the PUPPI method uses charged particles to identify the α_i distribution of particles from Pileup. While derived from charged particles, the distribution is comparable for uncharged particles. This allows identifying and removing uncharged particles from Pileup as well.

Calibration using PUPPI

As part of this thesis, the PUPPI method has been integrated into the Z + Jet analysis. Technical validation shows the method to be ready for use. However, side effects on the Z + Jet event topology must be studied as well [15].

Unlike CHS, the PUPPI method inspects all particles. Thus, it can remove entire jets from Pileup. This applies especially to soft jets, after which α_i is modeled. Most jets from additional Pileup interactions are removed, as shown in Figure 4.10.

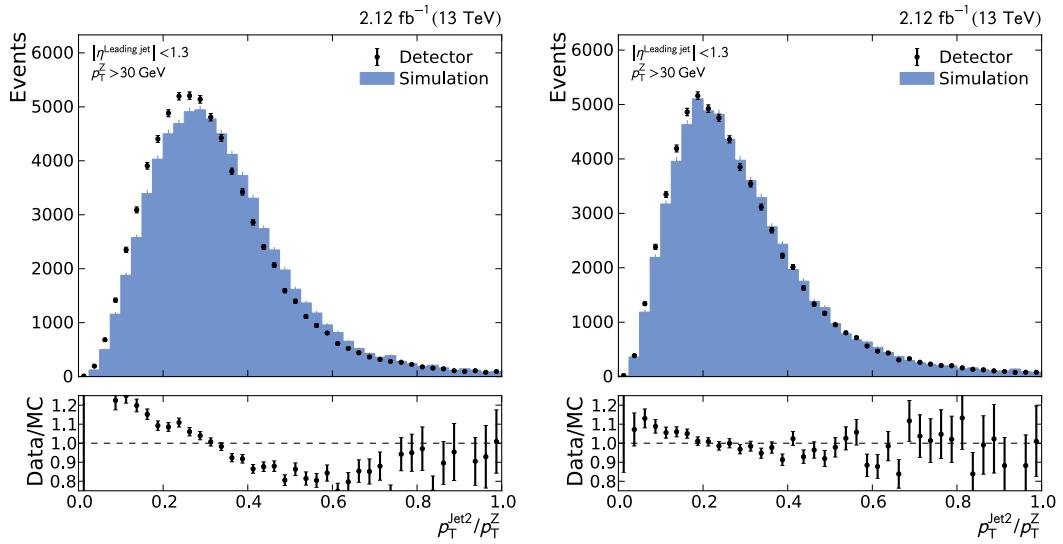
The largest impact on calibration is the purity of events. Second leading jets originating from Pileup are suppressed. In turn, any remaining second leading jet is likely caused by final state radiation. For most events, it is the actual fraction of energy loss. As a result, bias from Pileup on the α constraint is reduced, as shown in Figure 4.11.



(a) Number of Jets with CHS

(b) Number of Jets with PUPPI

Figure 4.10: Number of Jets Depending on Pileup Mitigation: Pileup mitigation removes particles associated with Pileup. The CHS method only removes a portion of charged particles. Pileup jets still remain in the event. In contrast, PUPPI can remove any particle originating from Pileup, possibly removing entire jets.



(a) Second Jet Activity with CHS **(b)** Second Jet Activity with PUPPI

Figure 4.11: Second Jet Activity Depending on Pileup Mitigation: The second leading jet is an estimator for final state radiation. However, jets from Pileup interactions bias this. The PUPPI method largely removes second jets not originating from final state radiation. In turn, the constraint $\alpha < 0.3$ is less biased.

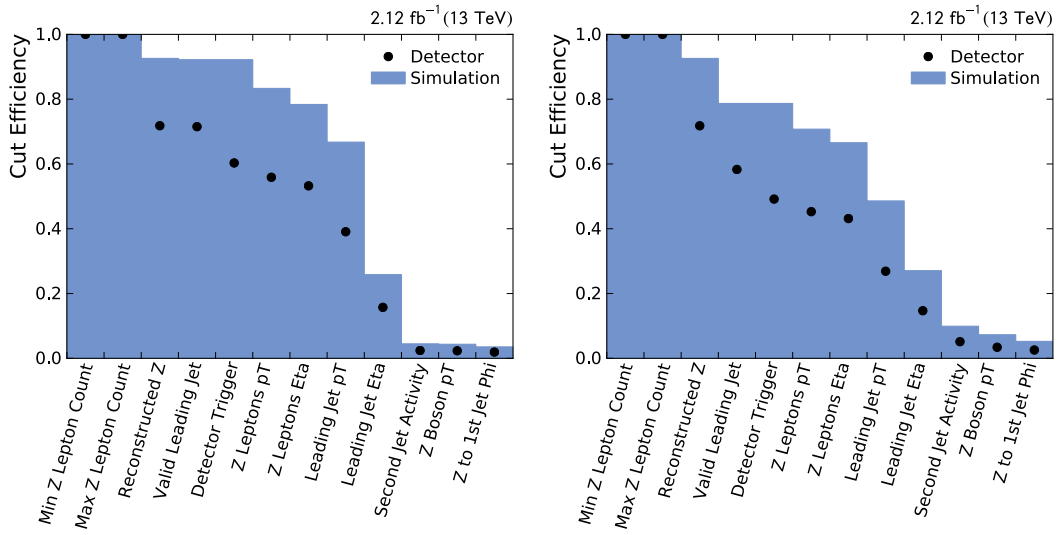
The use of PUPPI is promising for calibration studies. Since it reduces the impact from Pileup in general, calibration is less biased. Furthermore, it provides events with much cleaner topology. This allows selection constraints to be used more precisely. This can be used to improve purity or statistics, as shown in Figure 4.12.

The Z + Jet analysis is an early adopter of PUPPI in the calibration effort. As such, not all calibration stages are consistently available for jets with PUPPI Pileup mitigation. Thus, plots in other sections use the default method for CMS analysis, namely the CHS method. Plots for data using the PUPPI method may be found in Appendix B. However, PUPPI is now officially supported by the calibration effort for data collected in 2016.

4.3.2 Unification of Z → ee and Z → μμ for Calibration

The Z → μμ + Jet and Z → ee + Jet channels are complementary for calibration. Both channels use the same initial interaction. Thus, methods and workflows are largely equivalent. This allows using both channels in parallel.

The Z → ee + Jet channel is a new addition to the Z + Jet calibration effort at Institut für Experimentelle Kernphysik (IEKP). Since the channel shares many



(a) Accepted Events with CHS

(b) Accepted Events with PUPPI

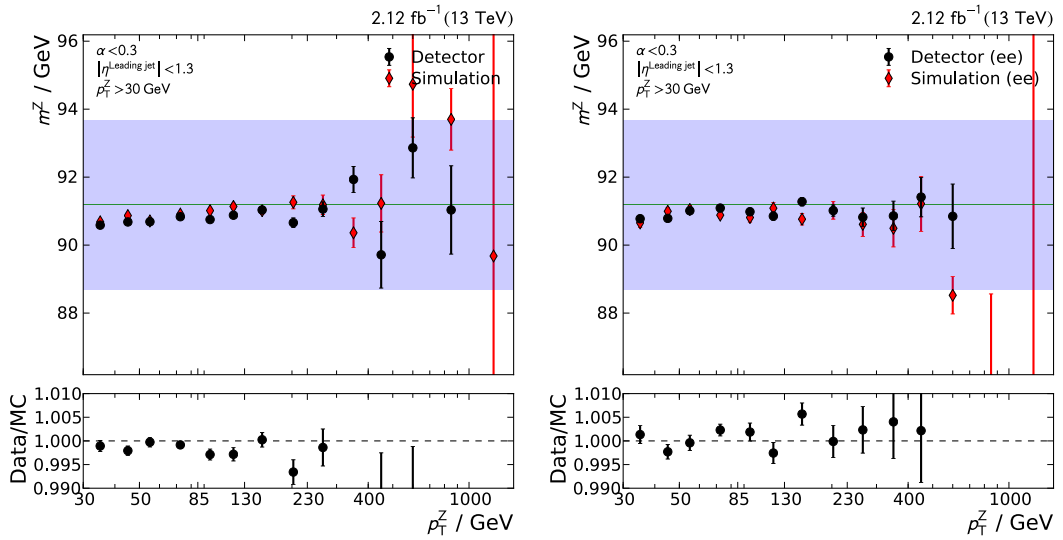
Figure 4.12: Accepted Events Depending on Pileup Mitigation: Multiple constraints used for Z + Jet analyses work on jets in the events. Since jets may originate from Pileup, this introduces biases to constraints. The PUPPI method is superior in removing such jets. This increases the number of accepted events by a factor of roughly 2.5.

elements with the $Z \to \mu\mu$ + Jet channel, many kinematic features are comparable. Still, electrons and muons are not fully equivalent in the detector. This leads to distinct differences that must be reflected in calibration [44].

Most prominently, precision varies depending on transverse momentum of the Z boson, as shown in Figure 4.13. Namely, muons are more precise at low p_T. In turn, electrons cover a wider spectrum of p_T. This is due to the coverage by different subdetectors.

Muons are detected only by the tracker and muon system; both use the trajectory of a particle to measure its momentum. At high momentum, muon trajectory approaches a straight line, making momentum reconstruction difficult. In contrast, electrons are detected by the ECAL. Due to the magnetic field of CMS, electrons emit substantial energy via photons. At low momentum, photon and electron may deviate notably from each other. This makes it difficult to reconstruct them as one entity.

Both channels are analysed as one workflow. Muons and electrons are generically treated as leptons from Z decay; only lepton corrections are applied separately. As a result, kinematics of the two decay channels are mostly equivalent, as shown in Figure 4.14.



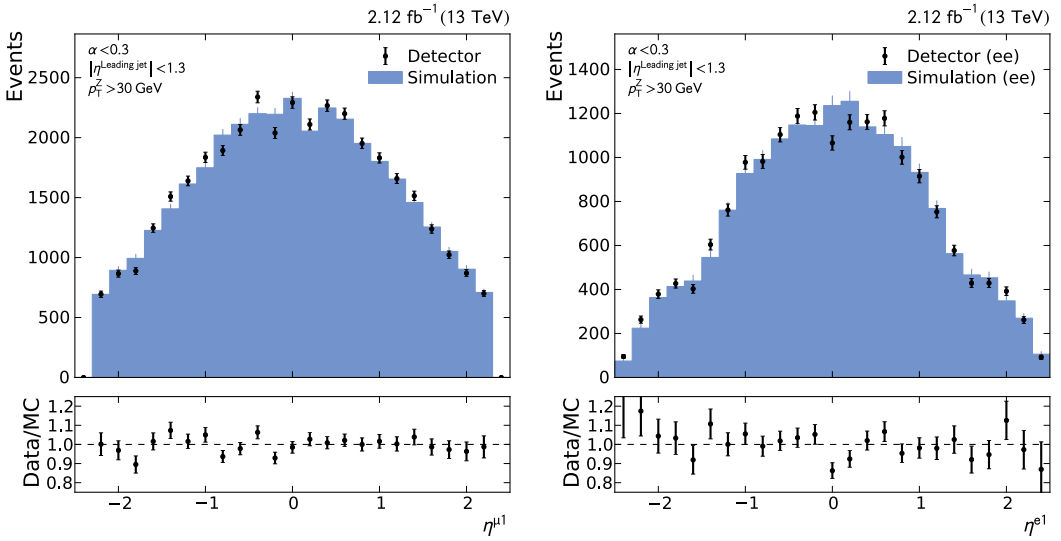
(a) Invariant Mass of $\mu\mu$ **(b)** Invariant Mass of ee

Figure 4.13: Mass of Z Boson Depending on Channel: The precision of the reconstructed Z boson mass depends on the decay products. A low response of muons or electrons also lowers the measured Z mass. In general, the Z → $\mu\mu$ channel becomes unreliable above 400 GeV. In contrast, the Z → ee channel is more robust at higher transverse momenta. The line and band show the Z boson mass and width as published by the PDG.

While processed together, the channels are used separately for calibration. On the one hand, a separate scaling of the Z mass is performed to match data from detector and simulation. It compensates for the different coverage and separate corrections of electrons and muons. On the other hand, treating channels separately allows crosschecking responses from ECAL and muon system. Plots for the Z → ee + Jet channel may be found in Appendix B.

4.3.3 Jet Balancing and Corrections

CMS jet corrections are based on comparing measured and true energy of jets. The ratio of measured and true energy is the *jet response* R_{jet}. Its inverse can be directly used as a correction factor. The challenge is the derivation of the true energy or transverse momentum.



(a) Leading Muon Orientation

(b) Leading Electron Orientation

Figure 4.14: Forward Orientation of Leading Lepton: For both Z $\rightarrow$ ee and Z $\rightarrow$ $\mu\mu$, kinematics of Z decay products are mostly equivalent. Differences arise from the different coverage of subdetectors. The ECAL exhibits a gap between barrel and end caps of the detectors. In contrast, the muon system does not extend as far in forward and backward direction.

Jet Response

Data from the detector obviously does not reveal the true energy of jets. However, the energy can be estimated by exploiting the negligible transverse momentum of particle beams. The Z boson and parton originating from the hard interaction are balanced in the transverse plane. Furthermore, the Z boson features a much better resolution than jets.

Two balancing methods are used for Z + Jet calibration, as shown in Figure 4.15. Both assume that the Z momentum is equal in magnitude to true energy of the *parton*, as defined in Equation 4.18. The difference is in what is considered as the detected equivalent of the parton.

$$\vec{p}_{\mathrm{T}}^{\,\mathrm{Z}} = -\vec{p}_{\mathrm{T}}^{\,\mathrm{parton}} \tag{4.18}$$

p_{T} Balance assumes an ideal topology of one jet balanced against the Z. This implies that the true parton momentum is also the true jet momentum. The p_{T} Balance,

as defined in Equation 4.20, follows directly from this.

$$p_{\mathrm{T}}^{\mathrm{jet}} \approx p_{\mathrm{T}}^{\mathrm{parton}} \tag{4.19}$$

$$R_{\mathrm{jet}}^{\mathrm{bal}} = \frac{p_{\mathrm{T}}^{\mathrm{meas}}}{p_{\mathrm{T}}^{\mathrm{Z}}} \tag{4.20}$$

This method is subject to biases from assuming an ideal topology. Specifically, final state radiation leading to multiple jets from the initial parton breaks this assumption. In this case, response may be significantly underestimated. Constraints on α can reduce this bias, but also reduce statistics when the second jet originates from Pileup.

In turn, the method is robust against soft Pileup. Since only the Z and leading jet are considered, biases from other objects are avoided. This makes the method especially robust against miscalibration of other detector elements.

Missing E_{T} Projection Fraction (MPF) takes any jets into account. The assumption is that the topology is not ideal, but pure: all energy in the detector results either from the Z or the balanced parton. In turn, any MET must originate from imprecisions in jet energy measurement.

$$-\vec{E}_{\mathrm{T}}^{\mathrm{miss}} = \vec{p}_{\mathrm{T}}^{\mathrm{Z}} + R^{\mathrm{parton}} p_{\mathrm{T}}^{\mathrm{parton}} \tag{4.21}$$

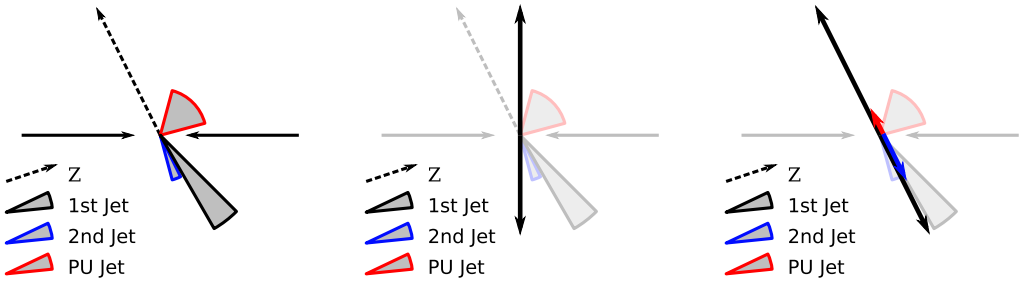
$$R_{\mathrm{jet}}^{\mathrm{MPF}} := R^{\mathrm{parton}} = 1 + \frac{\vec{E}_{\mathrm{T}}^{\mathrm{miss}} \cdot \vec{p}_{\mathrm{T}}^{\mathrm{Z}}}{\left(p_{\mathrm{T}}^{\mathrm{Z}}\right)^2} \tag{4.22}$$

The MPF is subject to biases from the assumption of purity. Especially soft Pileup activity can be significant compared to the p_{T} Balance. Since MET is derived from the entire detector, all elements must be well calibrated.

In practice, MPF is the most robust balancing method. It is implicitly robust against final state radiation and jet clustering errors. Since most biases of the method are isotropic, they cancel out on average.

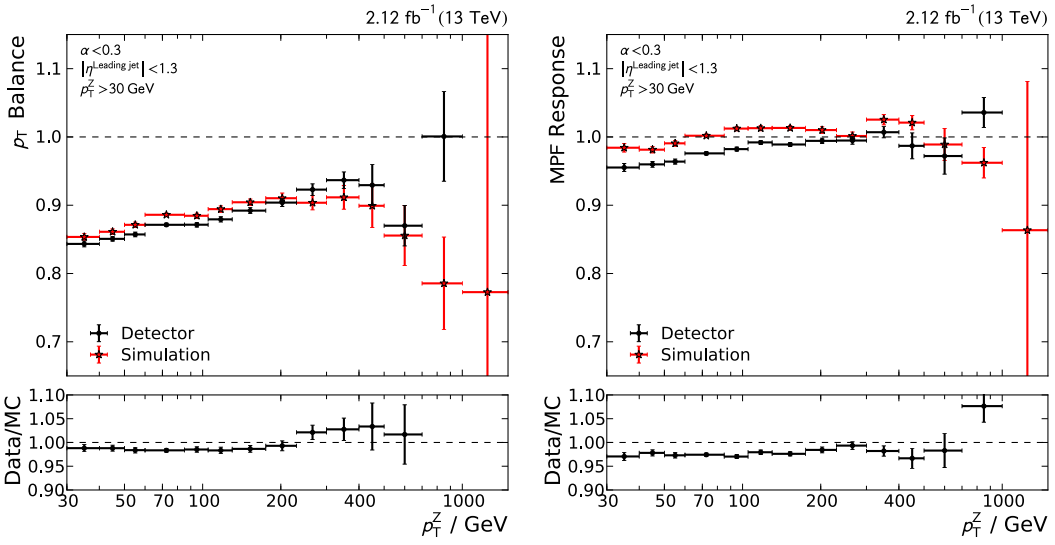
Figure 4.16 shows the performance of the two methods. The scale of $p_{\mathrm{T}}^{\mathrm{Z}}$ serves as an estimate for the true jet energy. As expected, the MPF method is the more robust: its response in simulation is close to 1 on the entire jet energy spectrum. In contrast, p_{T} Balance drops severely for low jet energies. This is mostly an effect of final state radiation having a higher relative effect.

For calibration, the absolute behaviour of responses is of minor significance. In general, both distributions show the same trends in comparison of detector and simulation data: responses of the detector are systematically underestimated. This illustrates the need for distinct corrections of detector responses.



(a) Balancing Topology **(b)** p_T Balance Topology **(c)** MPF Response Topology

Figure 4.15: Jet Balancing Responses: Calibration uses balancing to derive a per-event definition of jet response. The p_T Balance response only uses the p_T of Z boson and leading jet. This avoids Pileup contribution, but also excludes final state radiation splitting the studied jet. The MPF response uses the entire event. This respects arbitrary splitting of the leading jet. Pileup contribution is included, but suppressed by the projection along the Z boson direction.



(a) p_T Balance **(b)** Missing E_T Projection Fraction

Figure 4.16: Jet Responses Before Final Corrections: The two jet response methods p_T Balance and MPF Response are used to derive jet energy corrections. Corrections applied in these figures are based on simulation only. Both methods show a drop in reconstruction efficiency at lower jet energies. The Z + Jet analysis is the only data driven calibration probing the low jet p_T spectrum. Bins above 400 GeV are not yet adequately covered with sufficient statistics.

Topology Extrapolation

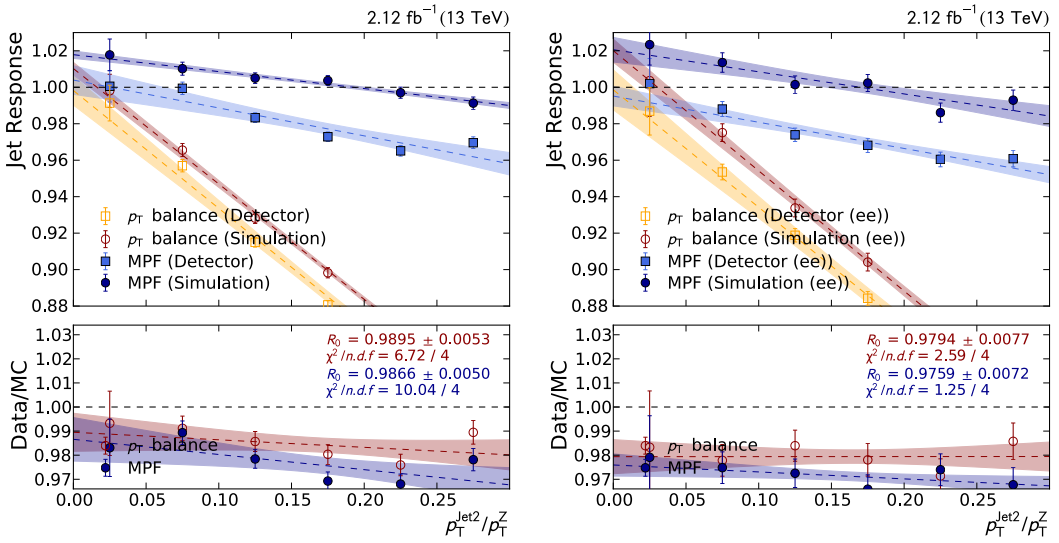
Both the p_T Balance and MPF method are biased if events do not conform to the ideal Z + Jet topology. To derive corrections applicable in general, these biases must be compensated. Ideally, this compensation allows combining both methods.

Biases originate from different effects. Yet, they can be subsumed as additional activity in the event. An effective upper limit on the scale of this is the second leading jet. It constraints jets from both Pileup and final state radiation.

As such $\alpha = p_T^{Jet2}/p_T^Z$, as defined in Equation 4.12, may constrain biases. However, it is not feasible to apply a simple constraint. Especially under the conditions of Run II, many events exhibit notable activity, as shown in Figure 4.11a. Any fixed constraint limiting additional activity would also drastically reduce statistics.

To leverage phase space regions with considerable α, the topology is extrapolated. The phase space is divided into regions of diminishing α, i.e. reduced additional jet activity. Extrapolating responses between these regions allows reconstructing $\alpha \approx 0$. Regions and extrapolation are shown in Figure 4.17.

The extrapolation underlines the characteristics of the p_T Balance and MPF methods. Final state radiation increases with higher values of α. In turn, the leading



(a) Extrapolation for Z → μμ

(b) Extrapolation for Z → ee

Figure 4.17: Jet Response Extrapolation to Ideal Topology: To derive corrections, events are extrapolated to an ideal topology. All constraints mentioned in Section 4.2.2 are applied, with the exception of $\alpha < 0.3$. Responses are calculated for bins in α, and a linear fit applied. Extrapolation provides the responses at $\alpha \approx 0$.

jet only carries a fraction of the initial parton energy. As a result, the p_T Balance drops drastically for high values of α. In contrast, the MPF response is much more robust. Its slope indicates that Pileup offset may be overestimated. High α implies low p_T^Z, reducing the phase space for additional, soft jets.

In the extrapolation to $\alpha \approx 0$, responses from either detector or simulation data converge. The difference between detector and simulation data is still between 1 % to 2 %, however. In principle, the ratio of response from detector and simulation can be used as a correction factor, as defined in Equation 4.16. However, this is applicable only to the single channel responses are derived from. For corrections applicable in general, multiple channels must be combined.

Global Fit and Residual Corrections

Corrections of simulation and detector data are used to enable high precision measurements. In turn, biases from deriving corrections themselves must be kept to a minimum. This is achieved by using multiple channels to derive the final corrections.

For Run II, corrections for CMS are derived based on data from four channels. Two of these channels have been provided as part of this thesis, namely $Z \rightarrow \mu\mu$ + Jet and $Z \rightarrow ee$ + Jet. The γ+ Jet channel can reliably probe high jet momenta. Finally, the Multijet channel is used to calibrate highly energetic jets, using low energy jets calibrated by other channels.

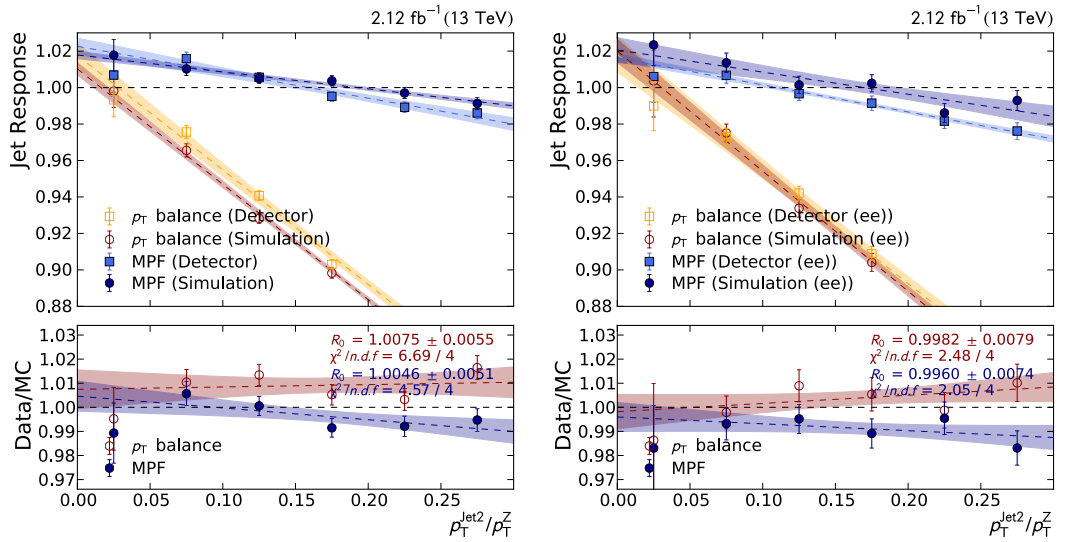
The channels are combined in the so-called global fit. The reference object of each channel, i.e. Z boson, γ or multiple jets, defines the common p_T scale. For each p_T bin, an extrapolation of generalised additional jet activity, as defined in Equation 4.23, is performed. In addition, nuisance parameters are used to account for peculiarities of each channel, e.g. uncertainties on lepton energy.

$$\alpha_{GF} := \frac{p_T^{Jet2}}{p_T^{ref}} \tag{4.23}$$

$$\tag{4.24}$$

The Global Fit produces the final, residual corrections, matching data from detector and simulation. These corrections adjust the jet response in detector data.

As a crosscheck, this effect is validated by the Z + Jet analysis as well. The final corrections further reduce differences between detector and simulation. Corrections derived for the 2015D data taking period consistently reduce differences to below 1 %, as shown in Figure 4.18.



(a) Extrapolation for Z → μμ

(b) Extrapolation for Z → ee

Figure 4.18: Jet Response with Final Corrections: The residual corrections derived from the calibration shift the jet response in detector data. This aligns the response in detector and simulation data. The ratio of the two data sources shows the current precision of corrections. Corrections derived with the Z + Jet analysis reduce differences in jet response to less than 1 %.

4.4 Conclusion and Outlook

The work presented in this chapter is a considerable contribution to the calibration effort of the CMS collaboration. Calibration of jet responses is vital for research at the LHC. It is a key requirement to achieve the high precision required by searches for new phenomena.

The calibration with Z + Jet events is important to align data recorded in the CMS detector with simulations. By using the well studied Z Boson as a reference object, simulation and experiment can be compared. Without the resulting residual corrections, comparing other experimental results to theoretical predictions would not be feasible.

Calibration is an ongoing effort, which extends beyond the timescale of this work. Thus, the Z + Jet analysis has been reimplemented to remain viable for the new conditions of Run II. A stable technical foundation has been created for the ongoing, reliable calibration of the jet energy scale.

The increased collision energy and frequency of Run II directly impacts the so-called Pileup. These additional interactions are a major influence for the measured jet

energy. A new Pileup mitigation technique, which aggressively removes suspected Pileup contribution, has been studied. The corrections derived from these study allow for the adoption of the technique by other analyses. Furthermore, it forms the basis for future in-depth studies of the Z + Jet events in the CMS detector.

To improve the precision of calibrations, two complementary channels have been used: for the first time in the CMS calibration effort, $Z \to \mu\mu$ + Jet and $Z \to ee$ + Jet have been calibrated in parallel. This allows using information from multiple detector components. In addition, it considerably improves statistics and thus precision.

Data taking at 25 ns has only recently begun in summer 2015. Despite this, the jet energy calibration for the CMS collaboration already achieves high precision. The difference in jet response for data from detector and simulation has been lowered to less than 1 %.

Chapter 5

Conclusion and Outlook

The work presented is an interdisciplinary contribution to particle physics. From a technological perspective, a new approach for end user data analysis via distributed, coordinated caching has been proposed. With respect to particle physics itself, the jet energy scale of the CMS experiment has been calibrated to high precision.

The new data analysis approach adds data locality to existing data analysis workflows and infrastructures. By caching data on processing resources, data may be served more quickly, in turn speeding up data analysis as a whole. The approach has been specifically designed to integrate transparently with existing particle physics workflows and infrastructures.

A prototype has been implemented to demonstrate the feasibility of the approach. This prototype is already successfully used to enhance particle physics analyses. In specific, the runtime of calibration analyses for the CMS detector is used as a benchmark. Using the new data analysis approach, such analyses are performed four times faster on average.

Using Z + Jet events recorded at the CMS detector, the jet energy scale has been calibrated for the second Large Hadron Collider (LHC) data taking period. This calibration is crucial for high precision analyses performed by the CMS collaboration. The changed conditions at the LHC necessitate not just a continuation of earlier calibration efforts. An entirely new Pileup mitigation technique has been studied and calibrated. To provide the statistics and robustness required, two channels have been calibrated in parallel. The current precision of calibrations already approaches that of the first LHC data taking period.

Both topics, technology and analysis, benefit from each other. The new middleware allows to perform calibrations with more channels, without sacrificing speed and responsiveness. In turn, the calibration analysis serves as a well defined, stable

benchmark and probe for the middleware. This mutual benefit allows both topics to act as enablers for future research.

Results of the calibration are vital to analyses performed as part of the second LHC data taking period. The contribution of this work impacts the precision of future analyses by the CMS collaboration.

The new techniques introduced to the calibration analysis itself allow for further improvements of precision. The improved Pileup mitigation makes it possible to study the Z + Jet channel with drastically reduced background effects. Combining two decay channels enables an improved study and validation of the detector response in these channels.

The distributed caching middleware prototype is applicable well beyond its current scope. It is only a small step from being suitable for emerging types of processing resources, such as cloud and HPC resources. In addition, both the prototype and its underlying approach are applicable to data analysis in general, even outside of particle physics.

Appendix A

Configuration and Settings for the Z + Jet Analysis

Tools and Settings used for the analysis detailed in Chapter 4.

A.1 Software and Versions

Table A.1: ANALYSIS SOFTWARE USED

Software	Version / Commit
CMSSW (skim)	CMSSW_7_6_3_patch2
CMSSW (analysis)	CMSSW_7_4_0_pre9
Excalibur	a31fe391e8f4f94ea7c4cc4e990b68b290e62073
Artus	a1b8bb10d7f1e4aa3b40e9cee9da6c0309bc79d4
KappaTools	b91c0447a357b35967a2738047ce2837e4355a1a
Kappa	d8472f6b0f3f584767b1355ff02b8aa9c41e0d84

A.2 Configurations

Configurations as provided by the respective repositories and commits. For DCSOnly certification, the config modifier `_DCSOnly` was applied as well. Other run periods have been studied via the modifiers `_2015A`, `_2015B` and `_2015C`.

Table A.2: EXCALIBUR CONFIGURATIONS

Channel	Data	MC	Modifiers
$Z \to \mu\mu$ + Jet (CHS)	`data15_25ns`	`mc15_25ns`	`_2015D`
$Z \to ee$ + Jet (CHS)	`data15_25ns_ee`	`mc15_25ns_ee`	`_2015D`
$Z \to \mu\mu$ + Jet (PUPPI)	`data15_25ns`	`mc15_25ns`	`_2015D _puppi`
$Z \to ee$ + Jet (PUPPI)	`data15_25ns_ee`	`mc15_25ns_ee`	`_2015D _puppi`

A.3 Settings

This is an excerpt from above mentioned configurations. Due to their length, only key portions are listed.

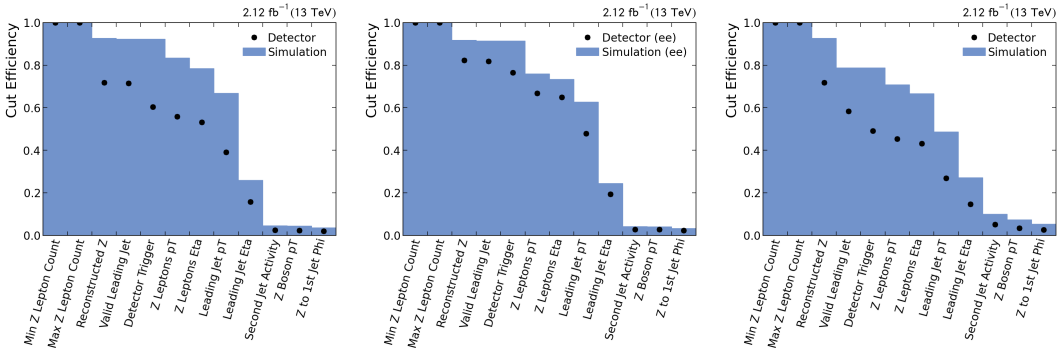
Table A.3: ANALYSIS SETTINGS

Option	Channel	Value
Dataset	$Z \to \mu\mu$ (Data 2015B, 50ns)	`/DoubleMuon/Run2015B-16Dec2015-v1/AOD`
	$Z \to \mu\mu$ (Data 2015C, 50ns)	`/DoubleMuon/Run2015C_50ns-16Dec2015-v1/AOD`
	$Z \to \mu\mu$ (Data 2015D, 25ns)	`/DoubleMuon/Run2015D-16Dec2015-v1/AOD`
	$Z \to \mu\mu$ (MC NLO, 25ns)	`/DYJetsToLL_M-50_TuneCUETP8M1_13TeV-amcatnloFXFX-pythia8/RunIIFall15DR76-PU25nsData2015v1_HCALDebug_76X_mcRun2_asymptotic_v12-v1/AODSIM`
	$Z \to ee$ (Data 2015B, 50ns)	`/DoubleEG/Run2015B-16Dec2015-v1/AOD`
	$Z \to ee$ (Data 2015C, 50ns)	`/DoubleEG/Run2015C_50ns-16Dec2015-v1/AOD`
	$Z \to ee$ (Data 2015C, 25ns)	`/DoubleEG/Run2015C_25ns-16Dec2015-v1/AOD`
	$Z \to ee$ (Data 2015D, 25ns)	`/DoubleEG/Run2015D-16Dec2015-v2/AOD`
	$Z \to ee$ (MC NLO, 25ns)	`/DYJetsToLL_M-50_TuneCUETP8M1_13TeV-amcatnloFXFX-pythia8/RunIIFall15DR76-PU25nsData2015v1_HCALDebug_76X_mcRun2_asymptotic_v12-v1/AODSIM`
JEC	$Z \to (\mu\mu/ee)$ (Data)	`Fall15_25nsV2_DATA`
	$Z \to (\mu\mu/ee)$ (MC)	`Fall15_25nsV2_MC`
JSON	$Z \to (\mu\mu/ee)$ (Data)	`/afs/cern.ch/cms/CAF/CMSCOMM/COMM_DQM/certification/Collisions15/13TeV/Cert_246908-260627_13TeV_PromptReco_Collisions15_25ns_JSON_v2.txt`
Minbxsec	$Z \to (\mu\mu/ee)$	`69.0 mb`
Jet ID	$Z \to (\mu\mu/ee)$ (Data)	`2015 loose`
HltPaths	$Z \to \mu\mu$ (Data)	`HLT_Mu17_TrkIsoVVL_Mu8_TrkIsoVVL_DZ`

Appendix B

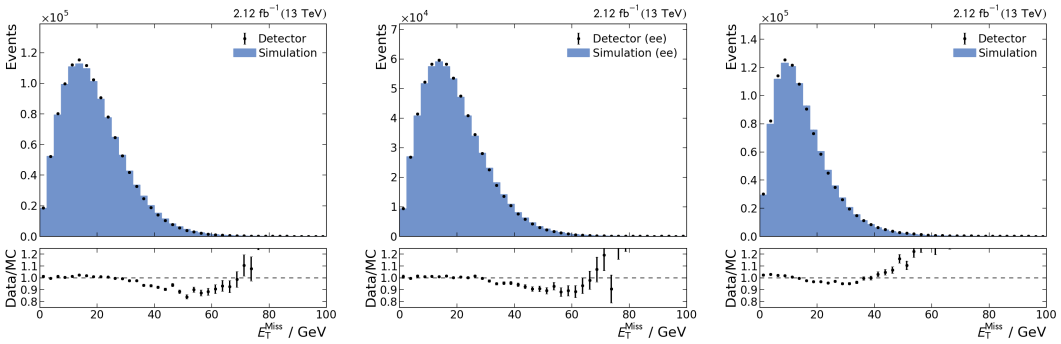
Additional plots of Z + Jet analysis

B.1 General



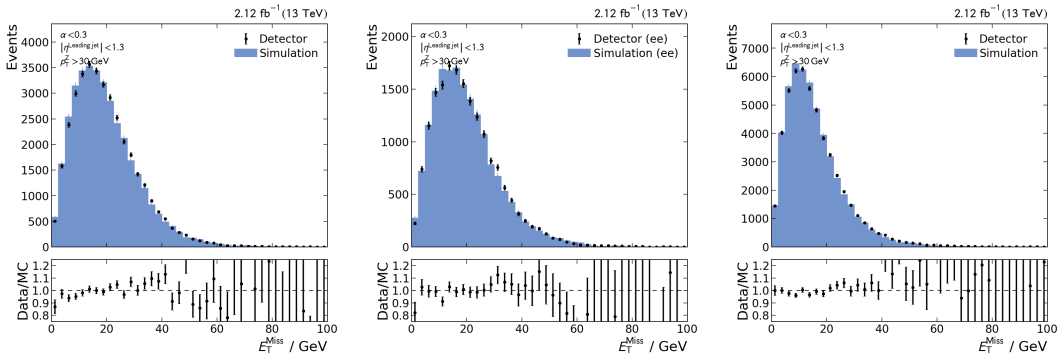
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.1: Constraint Efficiency with all constraints, and Simulation based Corrections



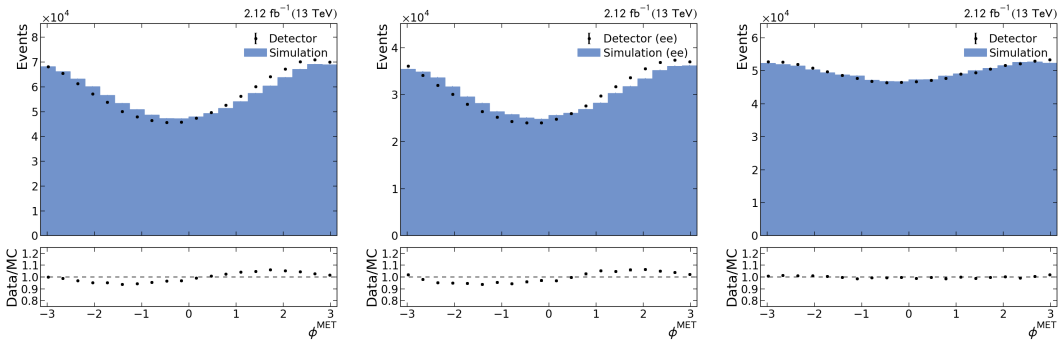
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.2: Missing Transverse Energy without any constraints, and Simulation based Corrections



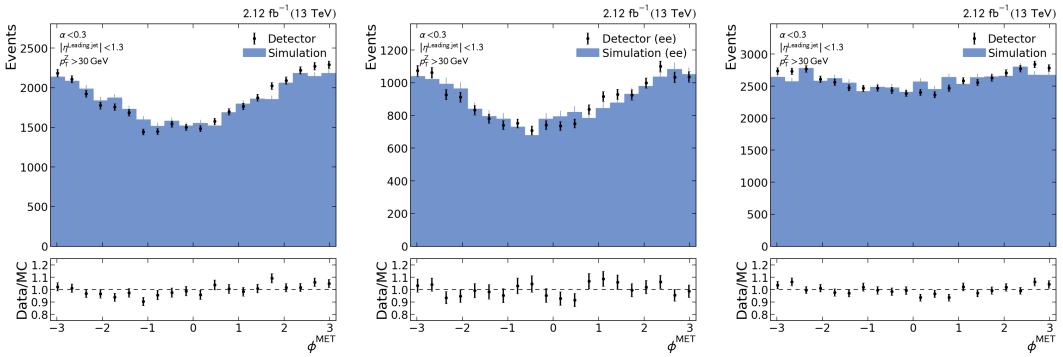
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.3: Missing Transverse Energy with all constraints, and Simulation based Corrections



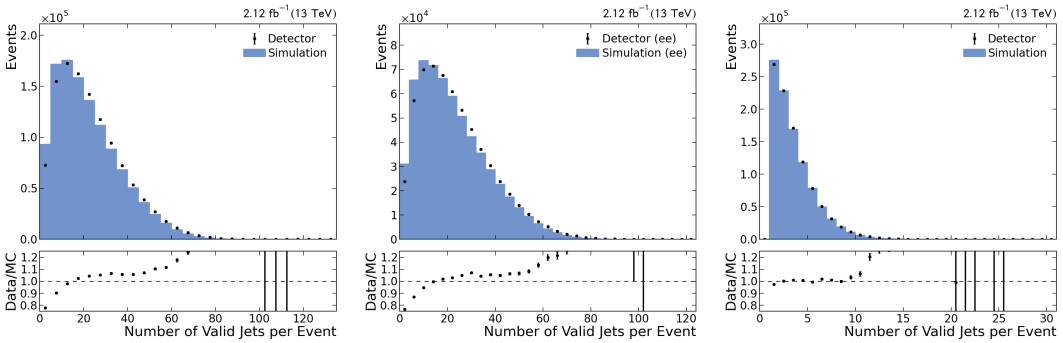
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.4: Azimuthal Orientation of Missing Transverse Energy without any constraints, and Simulation based Corrections



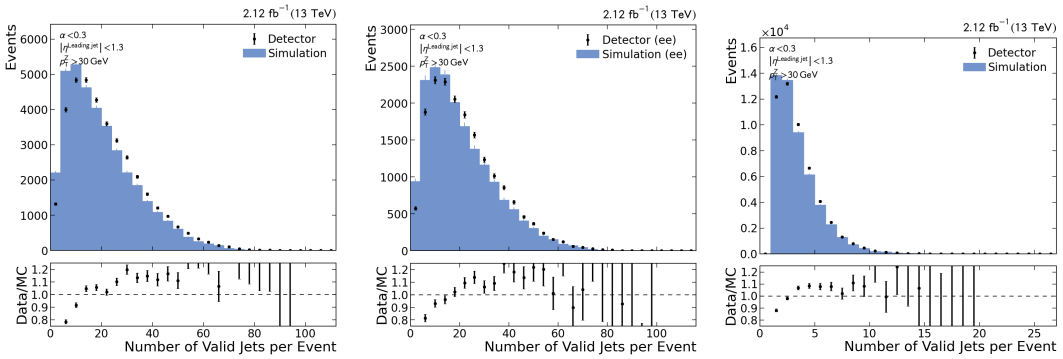
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.5: Azimuthal Orientation of Missing Transverse Energy with all constraints, and Simulation based Corrections



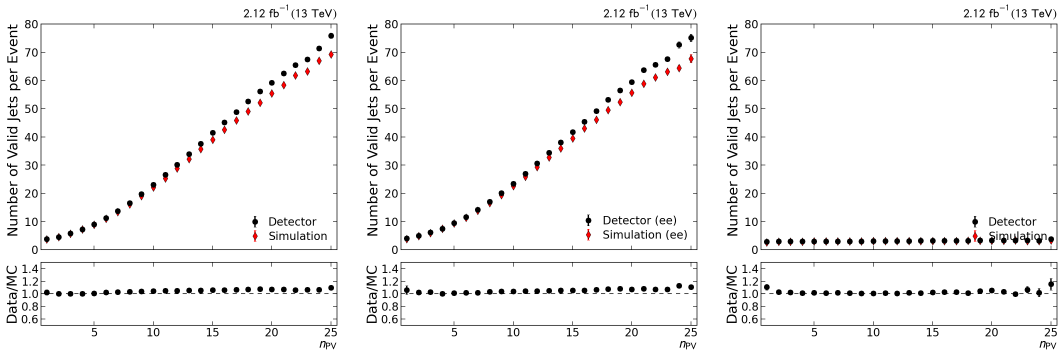
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.6: Number of Reconstructed Jets without any constraints, and Simulation based Corrections



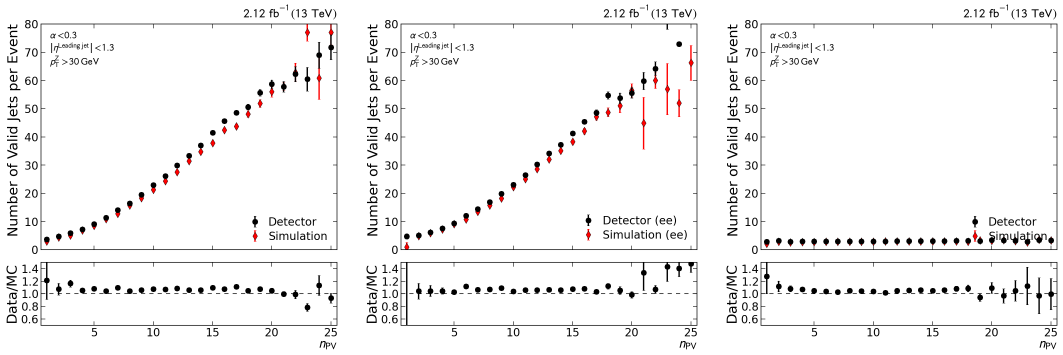
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.7: Number of Reconstructed Jets with all constraints, and Simulation based Corrections



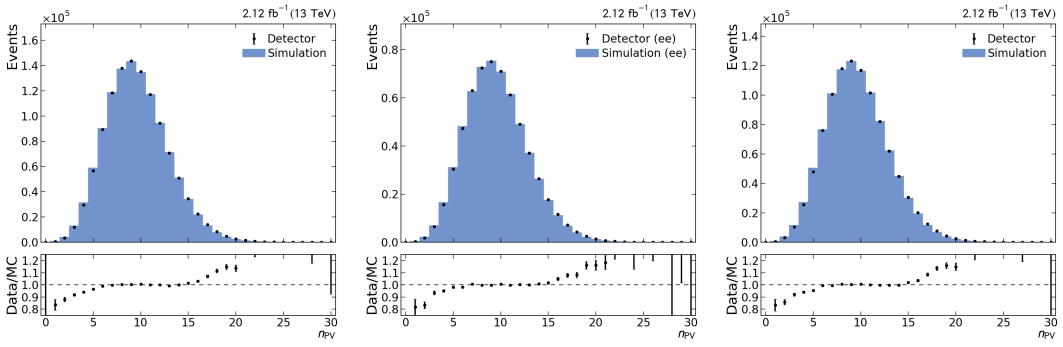
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.8: Number of Reconstructed Jets versus Number of Primary Vertices without any constraints, and Simulation based Corrections



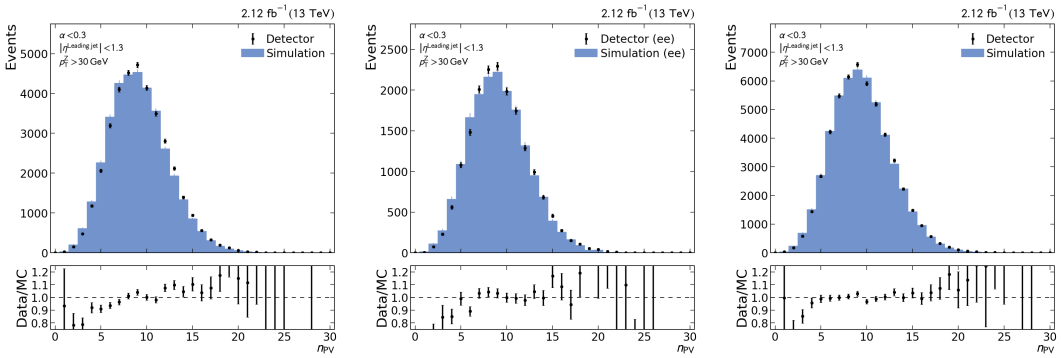
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.9: Number of Reconstructed Jets versus Number of Primary Vertices with all constraints, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

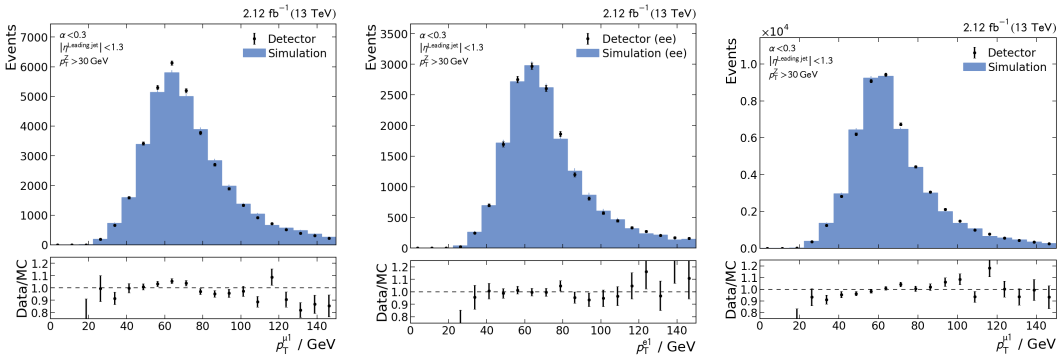
Figure B.10: Number of Primary Vertices without any constraints, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

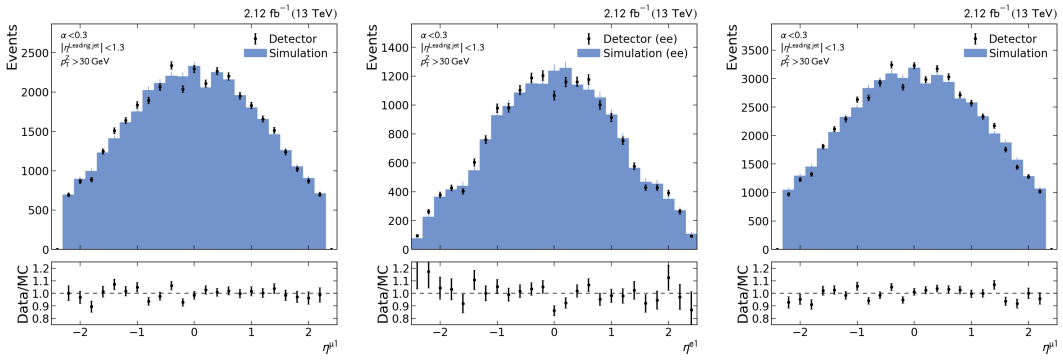
Figure B.11: Number of Primary Vertices with all constraints, and Simulation based Corrections

B.2 Z Boson



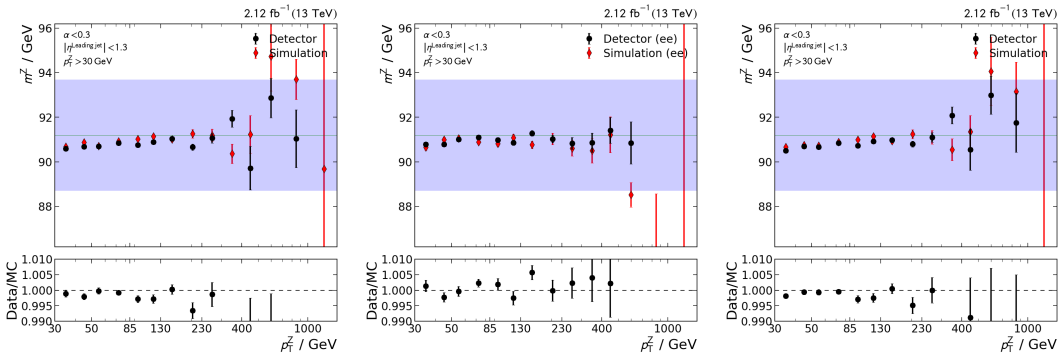
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.12: Transverse Momentum of Leading Z with all constraints, and Simulation based Corrections



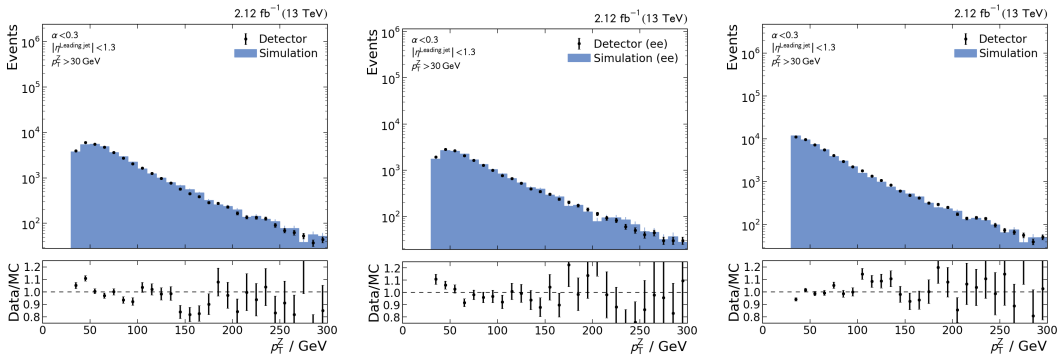
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.13: Orientation of Leading Z Lepton with all constraints, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.14: Mass of Z Boson versus Transverse Momentum of Z Boson with all constraints, and Simulation based Corrections

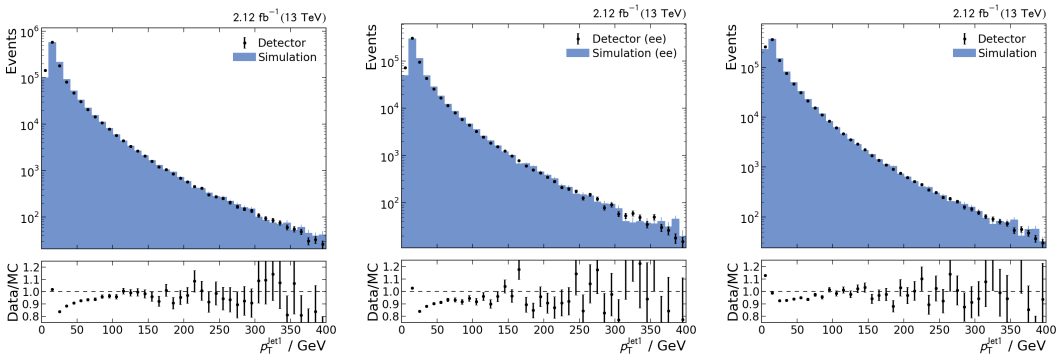


(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.15: Transverse Momentum of Z Boson with all constraints, and Simulation based Corrections

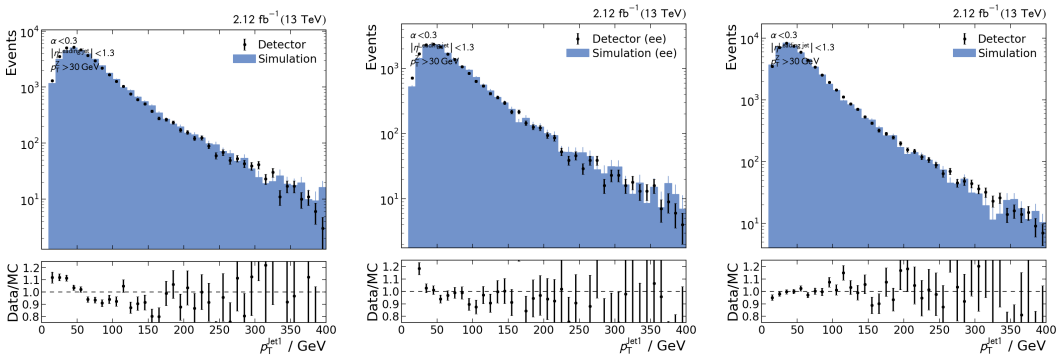
B.3 Jets

B.3.1 Leading Jet



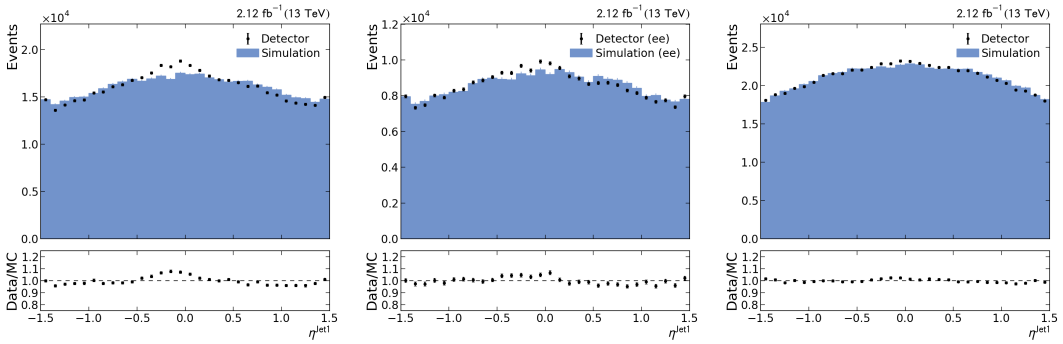
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.16: Transverse Momentum of Leading Jet without any constraints, and Simulation based Corrections



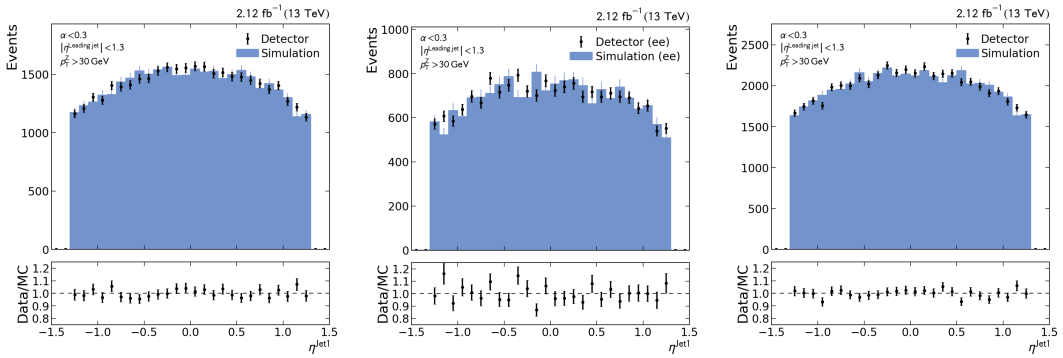
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.17: Transverse Momentum of Leading Jet with all constraints, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.18: Forward Orientation of Leading Jet without any constraints, and Simulation based Corrections

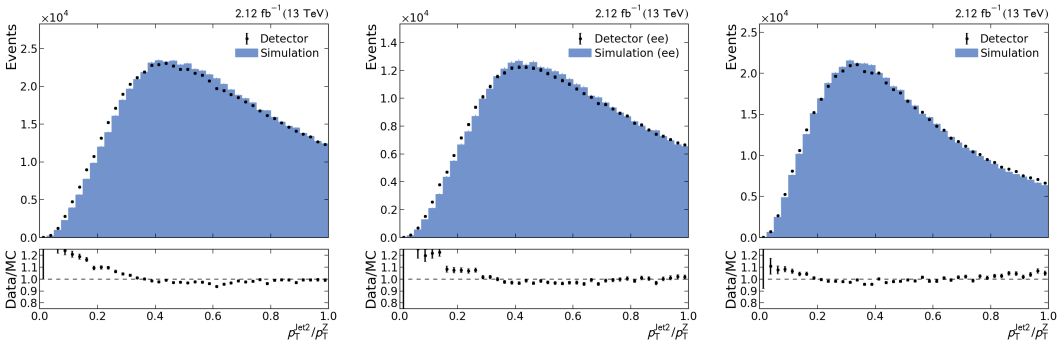


(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.19: Forward Orientation of Leading Jet with all constraints, and Simulation based Corrections

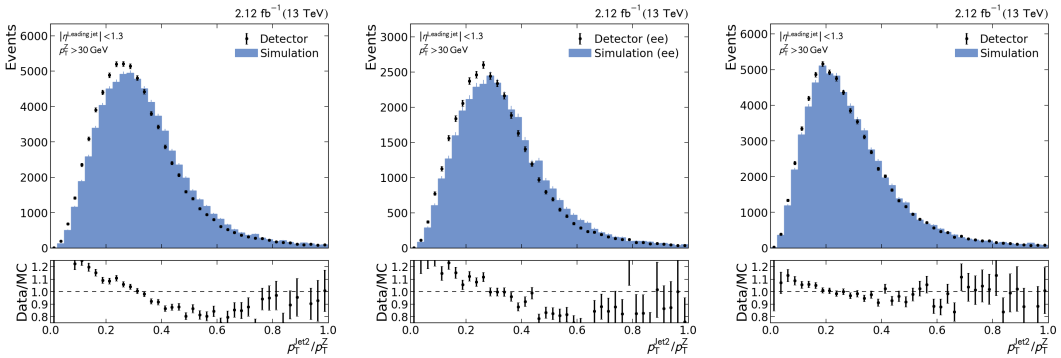
B.3.2 Second Leading Jet

Alpha Constraint



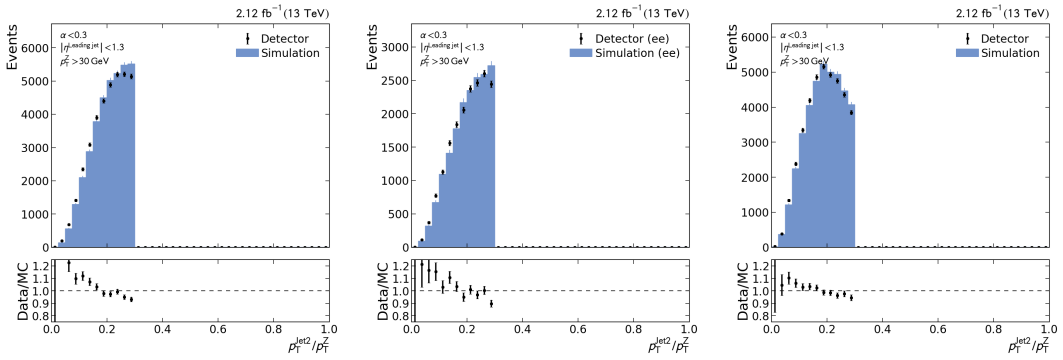
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.20: 2nd Jet Activity without any constraints, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.21: 2nd Jet Activity without constraints on α, and Simulation based Corrections

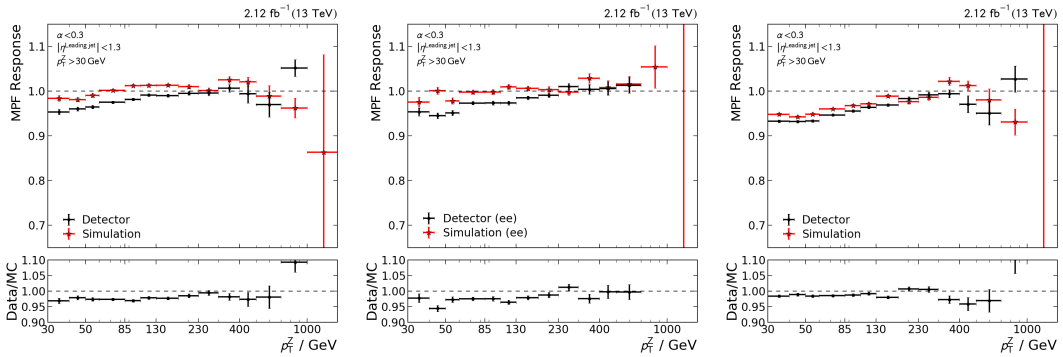


(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.22: 2nd Jet Activity with all constraints, and Simulation based Corrections

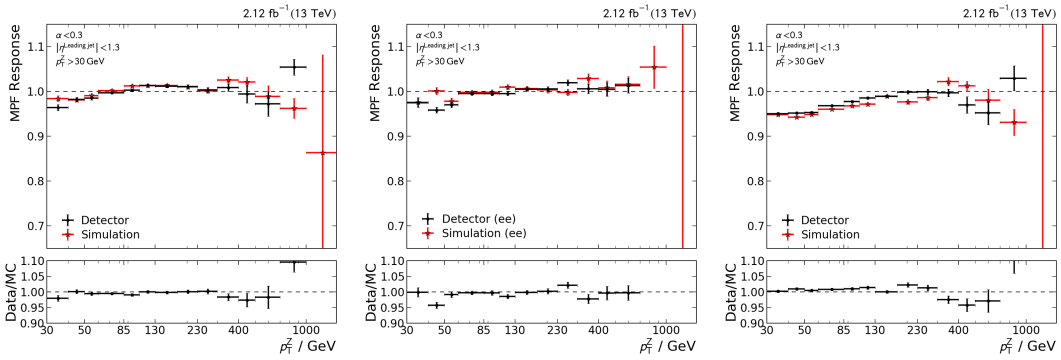
B.4 Responses

B.4.1 Missing E_T Projection Fraction (MPF)



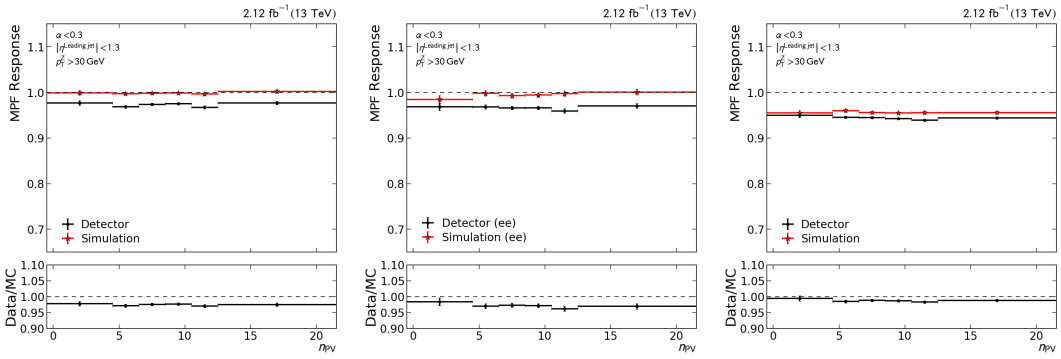
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.23: Missing E_T Projection Fraction versus Transverse Momentum of Z Boson with all constraints, and Simulation based Corrections



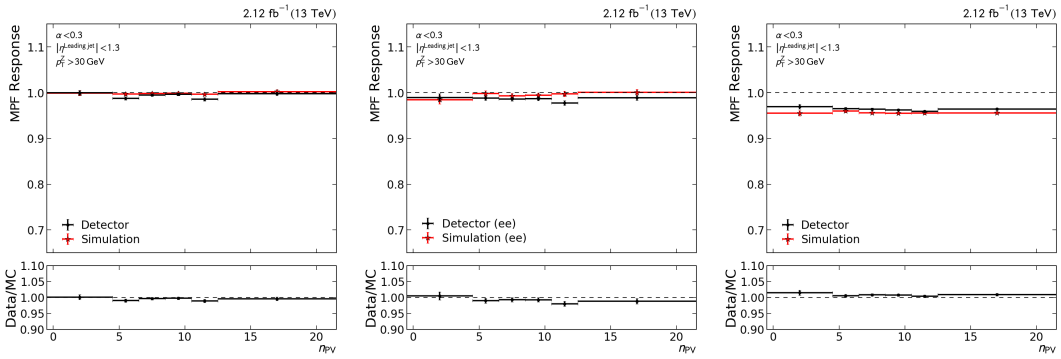
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.24: Missing E_T Projection Fraction versus Transverse Momentum of Z Boson with all constraints, and Simulation and Residual Corrections



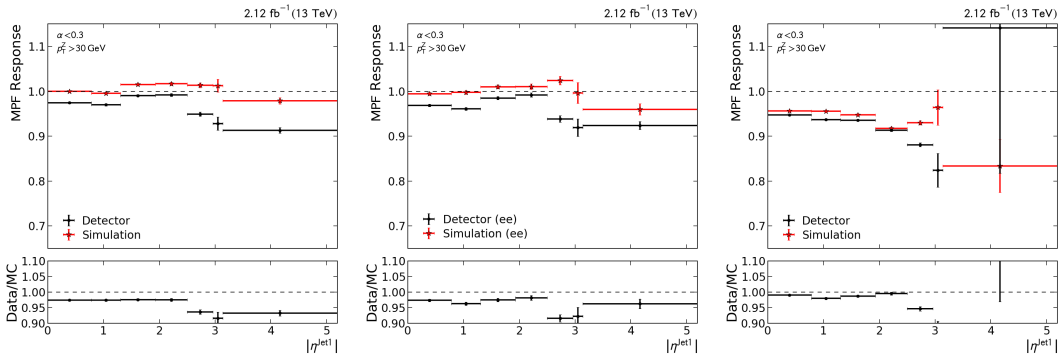
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.25: Missing E_T Projection Fraction versus Number of Primary Vertices with all constraints, and Simulation based Corrections



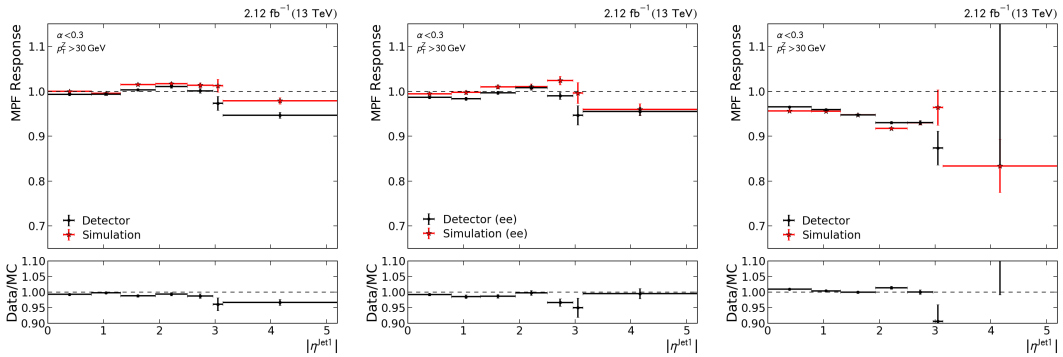
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.26: Missing E_T Projection Fraction versus Number of Primary Vertices with all constraints, and Simulation and Residual Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

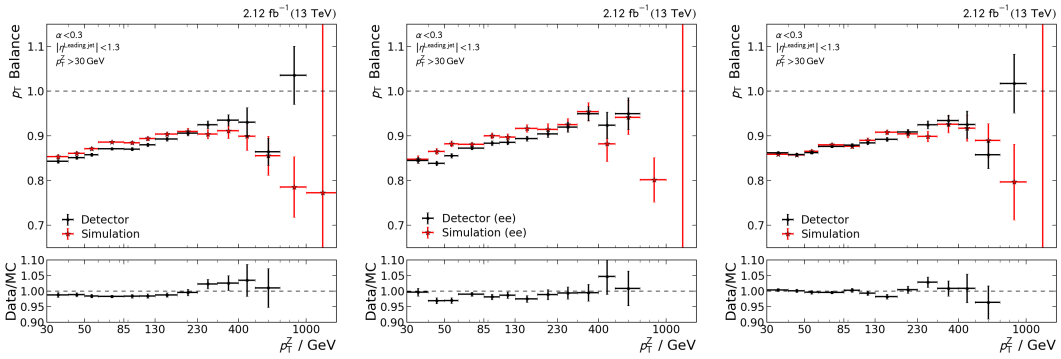
Figure B.27: Missing E_{T} Projection Fraction versus Absolute Forward Orientation of Leading Jet without constraints on η, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

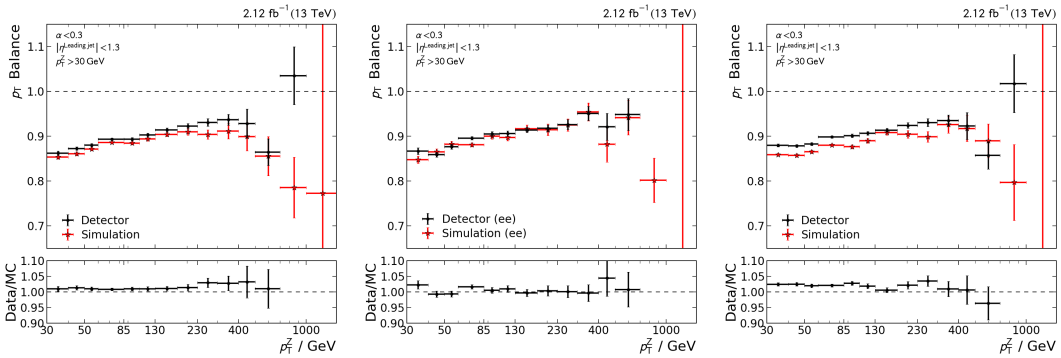
Figure B.28: Missing E_{T} Projection Fraction versus Absolute Forward Orientation of Leading Jet without constraints on η, and Simulation and Residual Corrections

B.4.2 p_T Balance



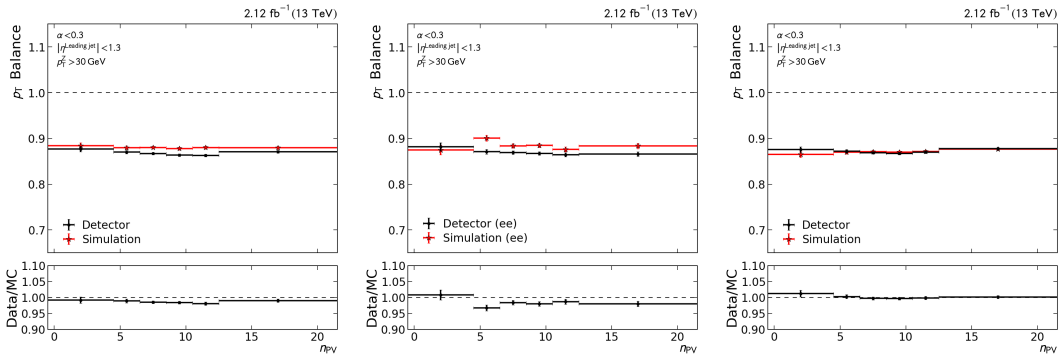
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.29: p_T Balance versus Transverse Momentum of Z Boson with all constraints, and Simulation based Corrections



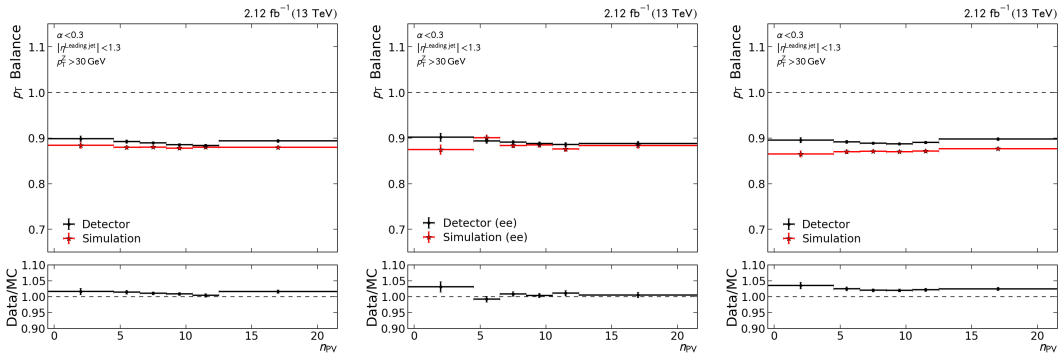
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.30: p_T Balance versus Transverse Momentum of Z Boson with all constraints, and Simulation and Residual Corrections



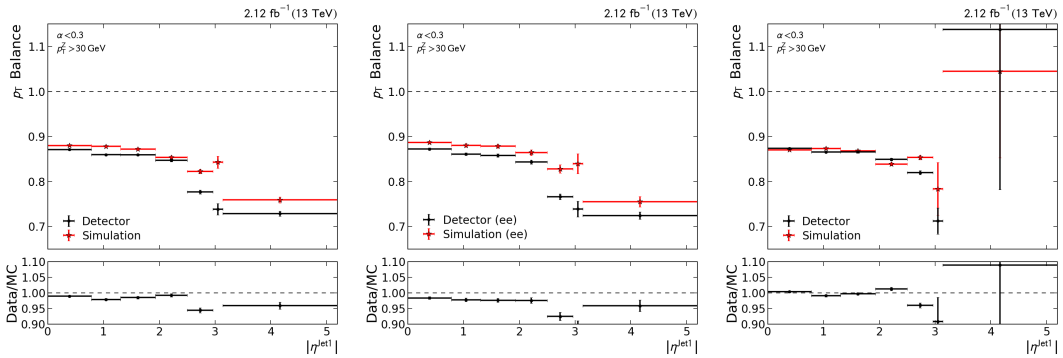
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.31: p_T Balance versus Number of Primary Vertices with all constraints, and Simulation based Corrections



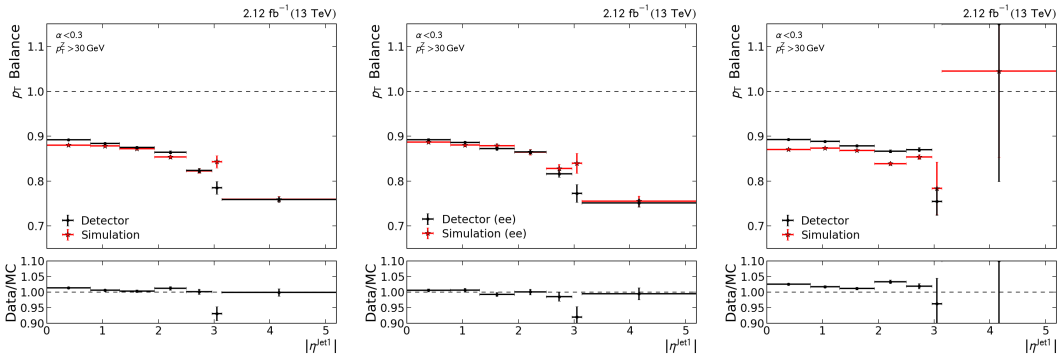
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.32: p_T Balance versus Number of Primary Vertices with all constraints, and Simulation and Residual Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

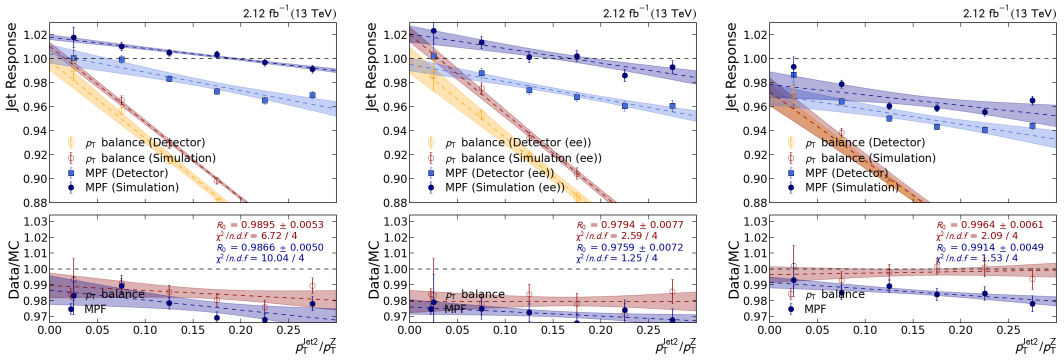
Figure B.33: p_T Balance versus Absolute Forward Orientation of Leading Jet without constraints on η, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

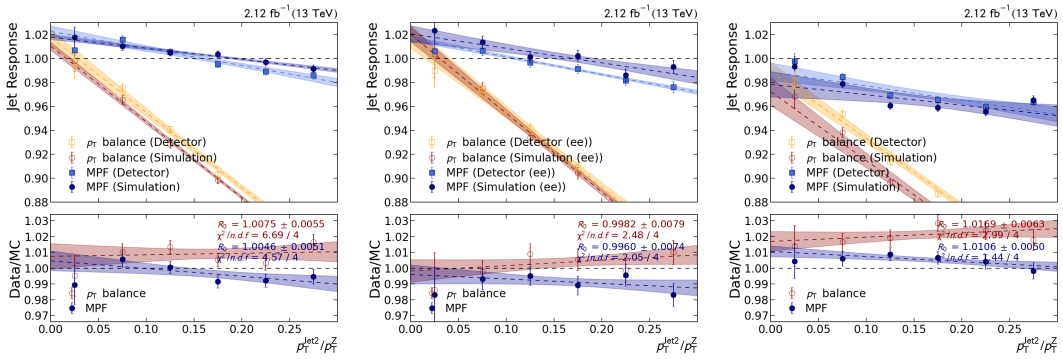
Figure B.34: p_{T} Balance versus Absolute Forward Orientation of Leading Jet without constraints on η, and Simulation and Residual Corrections

B.4.3 Extrapolation



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.35: Extrapolation in α without constraints on α, and Simulation based Corrections

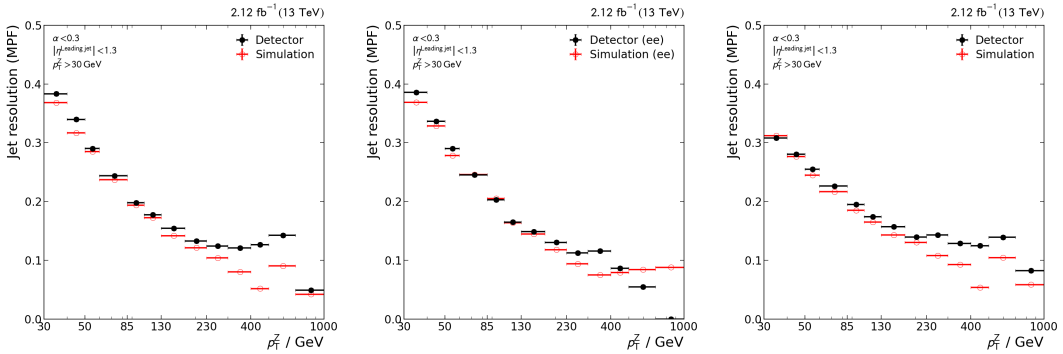


(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.36: Extrapolation in α without constraints on α, and Simulation and Residual Corrections

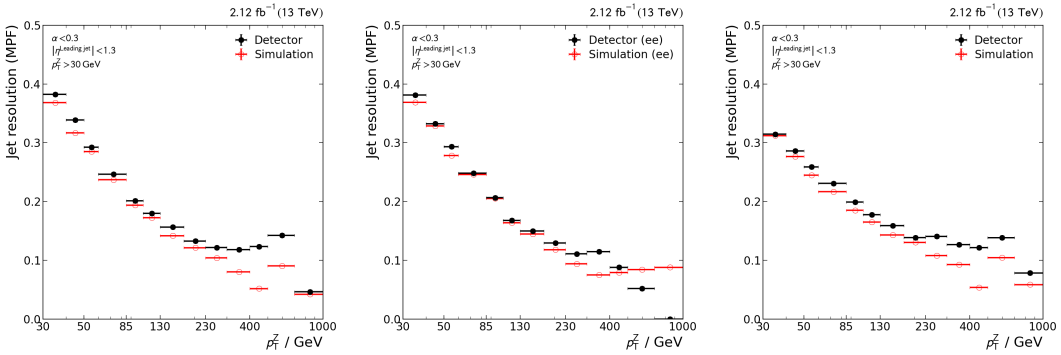
B.5 Resolution

B.5.1 Missing E_T Projection Fraction (MPF) Resolution



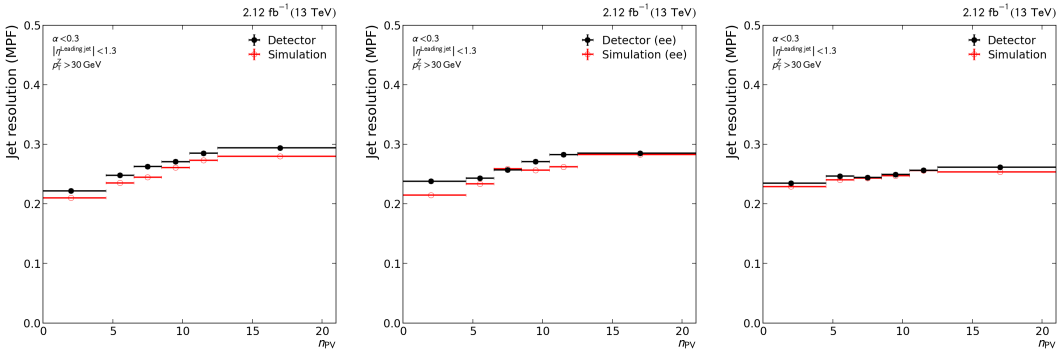
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.37: Resolution of Missing E_T Projection Fraction versus Transverse Momentum of Z Boson with all constraints, and Simulation based Corrections



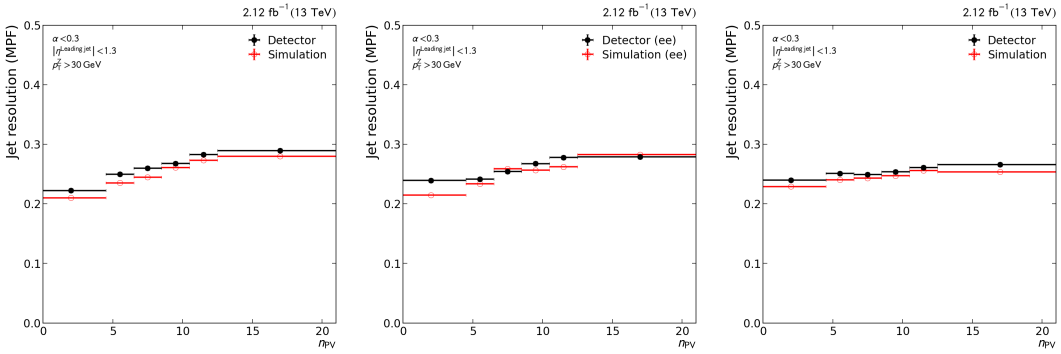
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.38: Resolution of Missing E_{T} Projection Fraction versus Transverse Momentum of Z Boson with all constraints, and Simulation and Residual Corrections



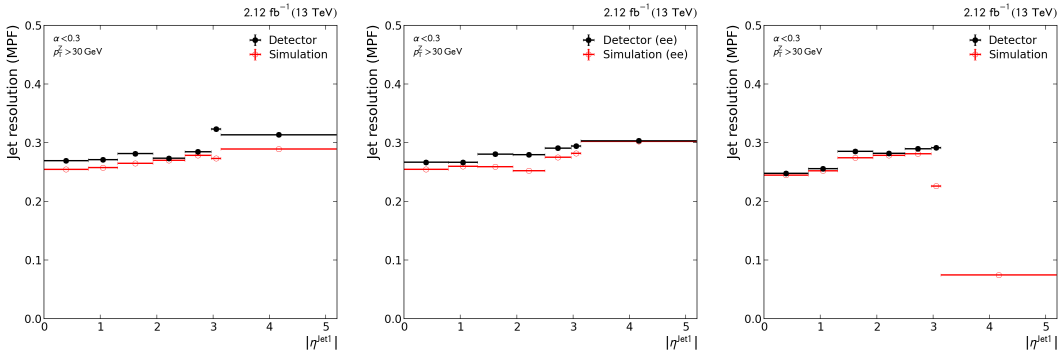
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.39: Resolution of Missing E_{T} Projection Fraction versus Number of Primary Vertices with all constraints, and Simulation based Corrections



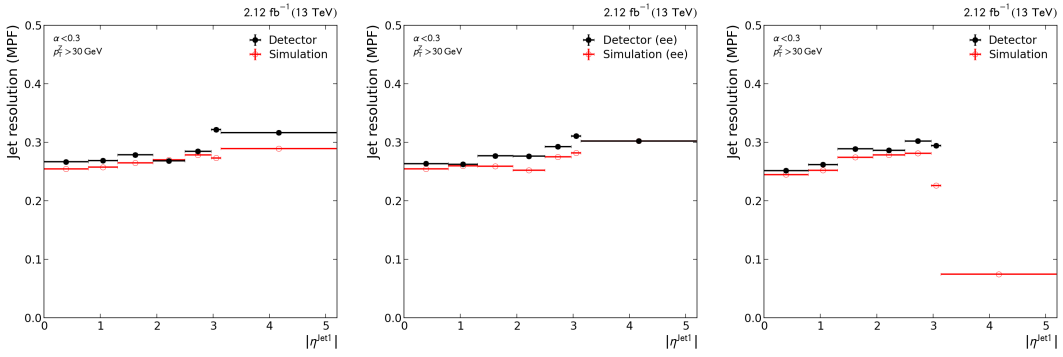
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.40: Resolution of Missing E_{T} Projection Fraction versus Number of Primary Vertices with all constraints, and Simulation and Residual Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

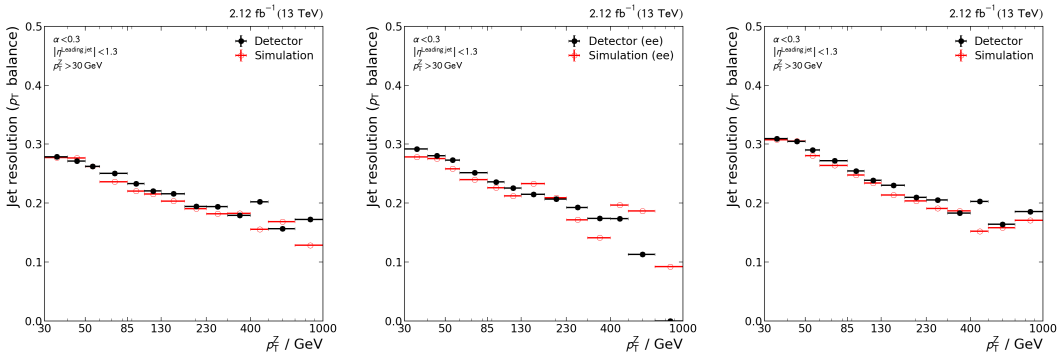
Figure B.41: Resolution of Missing E_{T} Projection Fraction versus Absolute Forward Orientation of Leading Jet without constraints on η, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

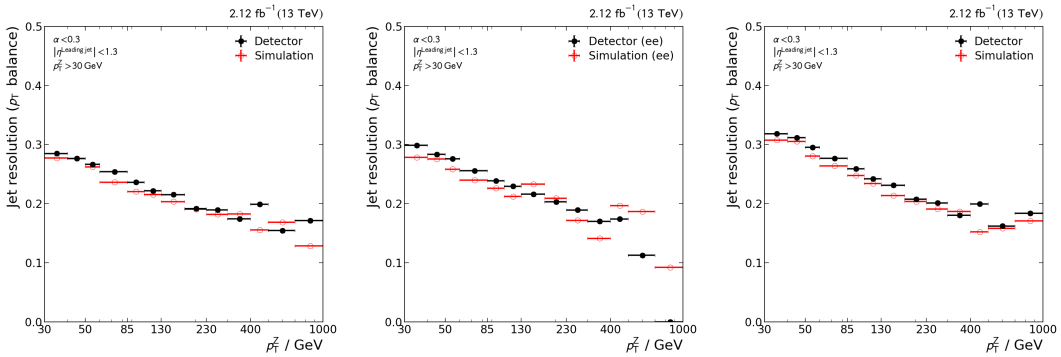
Figure B.42: Resolution of Missing E_{T} Projection Fraction versus Absolute Forward Orientation of Leading Jet without constraints on η, and Simulation and Residual Corrections

B.5.2 p_T Balance Resolution



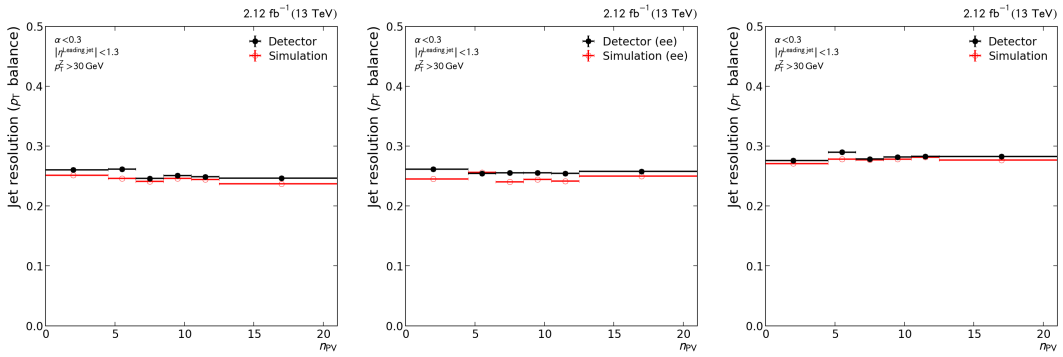
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.43: Resolution of p_T Balance versus Transverse Momentum of Z Boson with all constraints, and Simulation based Corrections



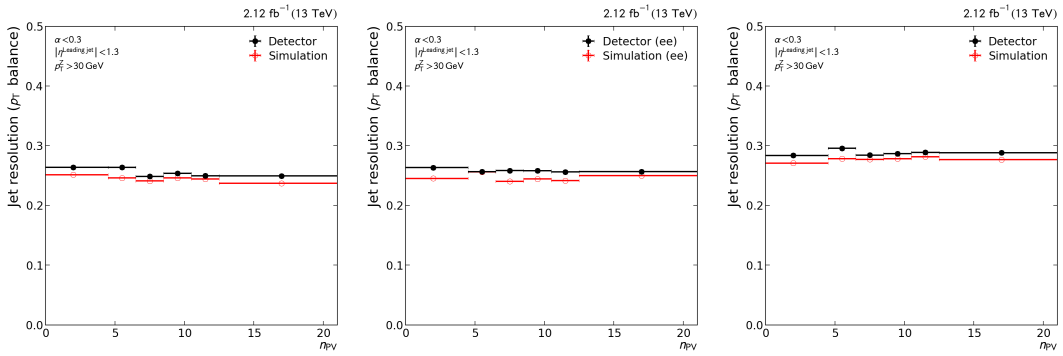
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.44: Resolution of p_T Balance versus Transverse Momentum of Z Boson with all constraints, and Simulation and Residual Corrections



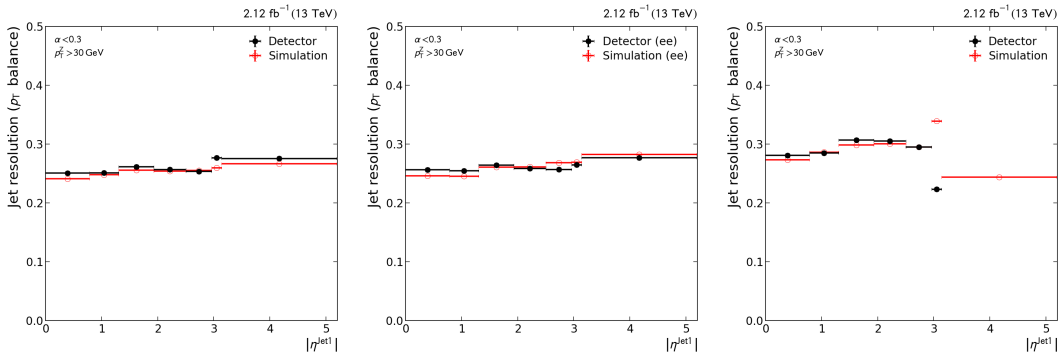
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.45: Resolution of p_T Balance versus Number of Primary Vertices with all constraints, and Simulation based Corrections



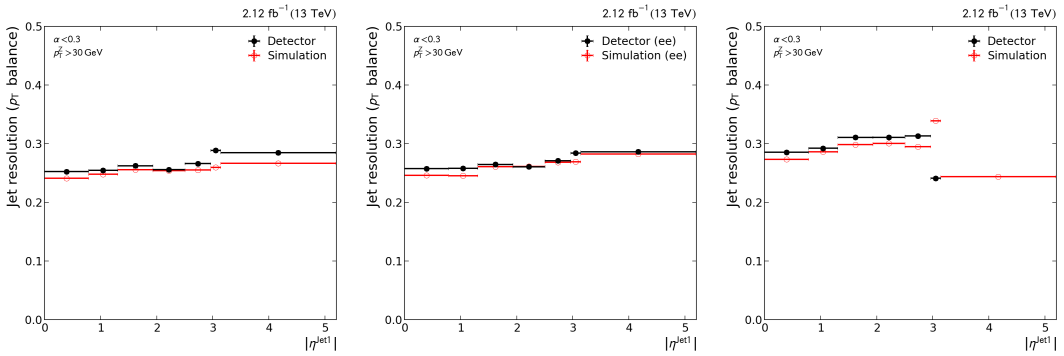
(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.46: Resolution of p_T Balance versus Number of Primary Vertices with all constraints, and Simulation and Residual Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.47: Resolution of p_T Balance versus Absolute Forward Orientation of Leading Jet without constraints on η, and Simulation based Corrections



(a) $Z(\mu\mu)$ with CHS **(b)** $Z(ee)$ with CHS **(c)** $Z(\mu\mu)$ with PUPPI

Figure B.48: Resolution of p_T Balance versus Absolute Forward Orientation of Leading Jet without constraints on η, and Simulation and Residual Corrections

Appendix C

HTDA Configuration

This is the HTDA Sphinx documentation on configuring HTDA nodes and hooks. Configuration setting blocks are autogenerated.

HPDA uses configuration files with an extended `ini` style. In addition, a number of command line arguments may be used for convencience.

The HPDA system consists of two generic component types: **Nodes** that host the core services, and **Hooks** that link to an existig batch system. In order to setup an HPDA cluster on top of a batch system, several components must be configured:

Cache On every worker node with cache devices, an HPDA Node with a **CacheMaster** must manage the caches. It maintains the files on the caches and provides file and device meta-data to other components.

Locator A Frontend to the Cache meta-data, used for scheduling jobs to worker nodes. At least one **Locator** is required to be accessible from every submit host. Ideally, there is one **Locator** running on every host used for user job submission.

Coordinator The scheduler of the HPDA cluster, deciding which files to stage and where to put them. Exactly one **Coordinator** is required, and it must support bi-directional communication to all **Caches** and **Locators**.

Collector Trackers for job, user and file usage statistics. Currently a subcomponent of the **Coordinator**, so it cannot be explicitly setup.

Job Hooks The link between job meta-data and scheduling on the one hand and the HPDA components on the other. When using an **HTCondor** batch system, these must be configured on every submit host (*schedd*).

C.1 Syntax

The configuration file format uses extended INI syntax. It supports key-value pairs organized in sections. Values may be lists and use references to each other. Comments

may be used, which are simply ignored by the parser. Formally, a config file consists of the following elements:

- *Sections* enclosed in square brackets (`[]`) at the start of a line.
- *Keys* at the start of the line, followed by an equal sign (`=`) to set or a plus-equal sign (`+=`) to append.
- *Comments* starting with a semicolon (`;`).
- *Values* anywhere else.

Sections appearing multiple times are concatenated. Attributes appearing multiple times are concatenated if defined with plus-equal sign (`+=`) or reset if defined with equal sign (`=`). It is not an error to concatenate to an undefined attribute - this is equivalent to a new definition. Keys are case insensitive and have included whitespace converted to underscores. In general, whitespace is stripped from all elements.

There is a reserved section called `global`. Any attributes appearing before a section are assigned to it. In addition, an explicit section global may be used for assigning attributes.

Values may appear anywhere after a key. Lists of values are created simply by having each individual value on a new line. Values may also include references to other values. The syntax for this is `&<section:attribute>s`. If section is ommited (`&<:attribute>s`), it defaults to the current section. If only an attribute is named (`&<attribute>s`), the `global` section is used.

> **note** References are substituted as plain strings, without any type conversions.

Multiple configuration files may be used. Sections and attributes are created in parsing order. Note that references are resolved after parsing all files - they may thus refer to attributes set in other files.

C.1.1 Example

```
times = 2 ; global value

[Section1]
foo_value = bar
times_seconds = &<times>s seconds ; refers to global value times

[Section2]
some_minutes = &<times>s minutes  ; refers to global value times
times = ; a value may already be defined here as well
```

```
  &<Section1:times_seconds>s
  &<:some_minutes>s ; refers to local value

[Section1]
foo_value = foo ; overwrites previous definition
times_seconds += 16 seconds ; append to previous definition
```

Indentation and line breaks before values are optional. The value of `[Section2] times` will actually be the list `[''2 seconds'', ''16 seconds'', ''2 minutes'']`

C.2 General

C.2.1 Logging

All components provide logging facilities based on the Python `logging` package. This allows sending debugging and runtime information to streams and log-files. (Other `logging` targets, e.g. email, may be supported in the future.)

It is generally sufficient to configure Handlers. Configuring Loggers provides detailed control when a single Handler receives messages from multiple Loggers.

[handler.nick1], [handler.nick2], ...

The actual output targets for logging. They must refer to loggers used in the application in order to receive any logging messages.

type is `buffer`

Buffer for other handlers, delaying and clustering output of records.

capacity (`int` = `32`) Number of records to buffer at most

flush_close (`bool` = `True`) Flush buffer when closing

flush_full (`bool` = `True`) Flush buffer when full

flush_level (`str` = `CRITICAL`) Minimum required level of messages to flush

logger (list[`str`] = [`<empty>`]) logger name(s) to get messages from

targets (list[`str`]) Name of handler(s) to buffer

unlink_targets (`bool` = `True`) Remove original sources from targets

type is `file`

Redirection of logging output to a file.

destination (str) name of file to direct output to

logger (list[str] = [<empty>]) logger name(s) to get messages from

max_age (seconds = -1) Maximum file age; use -1 for no limit

max_count (int = 3) Maximum number of files to use

max_size (bytesize = -1) Maximum file size as byte size; use -1 for no limit

minimum_level (str = NOTSET) -> {DISABLED, CRITICAL, ERROR, WARNING, STATUS, I
Minimum required level of messages

permissions (permission = rw-r--r--) Permissions of log file

type is stream

Redirection of logging output to a stream, i.e. stdout or stderr.

destination (str = stderr) sys stream to direct output to

logger (list[str] = [<empty>]) logger name(s) to get messages from

minimum_level (str = NOTSET) -> {DISABLED, CRITICAL, ERROR, WARNING, STATUS, I
Minimum required level of messages

[logger.nick1], [logger.nick2], ...

Internal aggregation pipelines for logging messages. Each section's nickname is used as the name the logger configured. The root logger is always defined.

minimum_level (str = NOTSET) -> {DISABLED, CRITICAL, ERROR, WARNING, STATUS, INFO,
Minimum required level of messages

propagate (bool = True) Propagate to all lower loggers

C.3 Nodes

C.3.1 Node

Every node, regardless of active components, provides a core of essential functionality. This infrastructure is shared by all components running on the node.

The NodeServer should be configured with respect to firewall settings and other local services (the default port 8080 is the http alternative default). The NodeMapper

requires to be configured with respect to the components running on the node (see below).

[NodeMaster]

Controller for any component's main threads. The NodeMaster regularly checks component lifesigns, restarting them in case of failures.

components (**list**[`str`] = []) components to start on this node

restart_delay_add (`seconds` = `10.0`) delay to add to thread resurrection per death

restart_delay_max (`seconds` = `600.0`) minimum delay for thread resurrection after death

restart_delay_min (`seconds` = `0.0`) minimum delay for thread resurrection after death

restart_interval (`seconds` = `0.0`) interval for checking thread lifesigns and restarts

[NodeServer]

HTTP server used by all components for inter-node communication.

host (`str` = `scc-wkitx-cl-cn-199-116.scc.kit.edu`) The address/hostname the server is listening on.

port (`int` = `8080`) The port the server is listening on.

threads (`int` = `5`) Maximum number of worker threads to service concurrent requests

timeout (`seconds` = `30.0`) Timeout for incomming requests.

[NodeMapper]

Discovery of other nodes via hearbeats. For either two nodes to be able to interact, at least one must notify the other as a `listener`

interval (`seconds` = `60.0`) Delay between sending/checking heartbeats

listeners (**list**[`str`] = []) URIs of nodes to actively notify about this nodes.

max_misses (`int` = `5`) Allowed missed heartbeats before ignoring node

variance (`seconds` = `5.0`) Variance of delay

C.3.2 Cache

At least one `Cache` and `Storage` is required for the node to fullfil its functionality. The catalogue should reside on a media that is as persistent as the caches.

Nodes hosting a **Cache** component should address all other **Locator** and **Coordinator** nodes.

[CacheCatalogue]

Persistent store for the state of the meta-data of the cache.

state_base_path (`str`) Directory to store current state.

state_cache_size (`int = 4`) Number of chunks to cache in memory.

state_chunk_count (`int = 64`) Number of chunks to store state in.

tasks_base_path (`str`) Directory to store outstanding tasks.

[CacheAllocator]

Manager of the cache resources. Assigns files to backends or removes them.

interval (`seconds = 300.0`) Interval between starting work cycle.

variance (`seconds = 10.0`) Variance on work cycle interval.

[CacheOperator]

Maintainer and janitor of the cache. Fetches, validates and unlinks files and performs other maintenance duties.

cleaner_interval (`seconds = 43200.0`) Interval between attempting to spawn a cleaner for a single cache.

cleaner_max_threads (`int = 1`) Maximum number of worker threads tending to cache cleaning duties.

cooldown (`seconds = 0.01`) Delay for each worker between a work action.

duration (`seconds = 60.0`) Minimum duration of a work cycle.

interval (`seconds = 300.0`) Interval between starting work cycle.

pool_size (`int = 3`) Maximum number of worker threads to use for work items of any kind. Must be `>2` and should by `>1+cleaner_max_threads`.

variance (`seconds = 10.0`) Variance on work cycle interval.

[cache.nick1], [cache.nick2], ...

Resources available for caching files. The nickname is arbitrary but should not change between configuration modifications.

type is `multi`, `multiapi` or `master`

Pseudo cache combining several APIs.

selector (`str` = `random`) Placement strategy to use for assigning files to slaves. `['random', 'hash', 'sequence']`

sequence (`float` = `0`) Order in which the backend is tried. Higher sequences are tried first, negative ones are ignored.

slaves (list[`str`] = []) Nickname of slaves to be handled by this API.

type is `fs`, `fsapi`, `posix` or `filesystem`

Cache device accessible via POSIX commands. Any kind of r/w mount point or subfolder is adequate.

sequence (`float` = `0`) Order in which the backend is tried. Higher sequences are tried first, negative ones are ignored.

size_limit Limit of the space to use for caching files.

(`percent` = `0.75`) Fraction of the total device volume to use.

(`bytesize`) Absolute size to use on the device.

uri_prefix (`str`) Start of all URIs of files managed with this backend. This is commonly the path to a device mount point or subfolder.

[storage.nick1], [storage.nick2], ...

Resources providing data files. The nickname is arbitrary but should not change between configuration modifications.

type is `fs`, `posix`, `fsapi` or `filesystem`

Data source accessible via POSIX commands. Any kind of readable mount point or subfolder is adequate.

raw_mount (`str`) Mount point of the source, without any caches overlayed.

sequence (`float` = `0`) Order in which the backend is tried.

uri_prefix (`str`) Start of all URIs of files available from this backend. This should be the mount point as seen without cache infrastructure.

[WorkerInfo]

Source of information to publish about an attached worker node. Used for scheduling.

type is `htc` or `htcondor`

dynamic (`bool` = `False`) Whether slots may be dynamically resized.

pool (`str` = `default`) <undocumented>

type is `static`

cpus (list[`int`] = [`1, 1, 1, 1`]) Cores available per slot.

dynamic (`bool` = `False`) Whether slots may be dynamically resized.

memory (list[`float`] = []) Memory available per slot.

name (`str` = `scc-wkitx-cl-cn-199-116.scc.kit.edu`) Name of the worker node in the batch system.

pool (`str` = `default`) Nickname of the batch system pool. Only relevant for multiple batch systems.

C.4 Hooks

[HTCRouter]

Hooks interfacing jobs to service and cache nodes.

collectors (list[`str`] = [`http://localhost:8081`]) URLs of collectors/coordinators to report to about jobs.

enforce_local Policy for waiting for locally available files.

(`seconds` = `0`) Minimum time to wait before running on host without local files; `0` means do not wait, `-1` means wait indefinitely.

(`str` ...) ClassAd expression that must evaluate to True once the job may start. Use '%s(machine_rank)s' for the number of files on a Startd.

hook_timeout (`seconds` = `10`) Maximum time after which the hook aborts.

hook_variance (`seconds` = `1`) Maxmimum range of random variance on query times.

input_key (`str` = `INPUT_FILES`) Key to the job ClassAd attribute listing input files.

locators (list[`str`] = [`http://localhost:8081`]) URLs of locators to query for file locations.

prefix (`str` = `HPDA`) Prefix to expect/apply for ClassAd insertions' keys, e.g. `'<prefix>_RANK'`.

query_all (`bool` = `False`) Whether to query all locators or just the first available.

update_delay (`seconds` = `120`) Minimum delay between updating jobs. Even if the HTC Job Router launches more often, hooks will not perform action.

Appendix D

Job Router Integration

This is the HTDA job hook configuration for use with HTCondor's ***job_router****.*

```
### HPDA Schedd Setup
### --------------------
### To be used on nodes that allow job submission

# launch job router to hook into submitted jobs
DAEMON_LIST = $(DAEMON_LIST), JOB_ROUTER

# intrinsic job router:
#  - test for HPDA required attribute(s)
#  - disable HTCondor grid routing
#  - inject routing keywords to activate hooks
# NOTE: This requires new style ClassAd syntax
JOB_ROUTER_ENTRIES = \
        [ \
                name = "HPDA"; \
                requirements = (target.INPUT_FILES isnt undefined); \
                MaxJobs = 1000; \
                MaxIdleJobs = 500; \
                TargetUniverse = 5;\
                set_HPDA_Route = True; \
                set_HookKeyword = "HPDA"; \
                GridResource = "NONE"; \
                OverrideRoutingEntry = True; \
        ]


# router can poll frequently as hooks may skip frequent updates
JOB_ROUTER_POLLING_PERIOD = 10

### external HPDA hooks
# docs specify HPDA_HOOK_JOB_FINALIZE, 8.2 expects HPDA_HOOK_JOB_EXIT
HPDA_HOOK_TRANSLATE_JOB   = /opt/hpda/repo/bin/htc_translate.py
HPDA_HOOK_UPDATE_JOB_INFO = /opt/hpda/repo/bin/htc_update.py
HPDA_HOOK_JOB_EXIT        = /opt/hpda/repo/bin/htc_finalize.py
HPDA_HOOK_JOB_FINALIZE    = /opt/hpda/repo/bin/htc_finalize.py
```

```
36
37
38 # Disable PROCD as job_router runs as submitting user, not condor/root
       like PROCD does
39 JOB_ROUTER.USE_PROCD = False
40
41
42 # Increase ClassAd parser buffer character size (defaults to 10240) to
       handle large job ClassAds
43 PIPE_BUFFER_MAX = 102400
44
45
46 # Require jobs to wait for router hooks, IF they are applicable
47 # NOTE: Keep this up to date with JOB_ROUTER_ENTRIES above
48 APPEND_REQUIREMENTS = ( (INPUT_FILES =?= UNDEFINED) || (HPDA_Route =?=
       True) || (( CurrentTime - QDate ) > ( $(
      JOB_ROUTER_POLLING_PERIOD) * 3 )) )
```

Appendix E

HTDA Test Cluster

E.1 Specifications

The High Throughput Data Analysis (HTDA) test cluster is part of the Institut für Experimentelle Kernphysik (IEKP) HTCondor cluster. Five worker nodes are used as listed in Table E.1. Two of these worker nodes use only half of the maximum Random Access Memory (RAM) and CPU.

Table E.1: TEST CLUSTER WORKER NODE

OS		Scientific Linux 6 (Kernel 2.6.32)
CPU	2x	Intel Xeon E5-2650v2 @ 2.66 GHz (à 8 cores, 16 threads)
Memory	8x	8GB RAM
SSD	1x	Samsung SSD 840 PRO 512 GB or
	2x	Samsung SSD 840 EVO 256 GB
HDD	4x	WDC WD4000 4 TB
Network	1x	Intel X540-T1 (10GigE/RJ45)

A total of 7 file servers is covered by the HTDA test cluster. In total, they provide 305 TB of storage.

E.2 Performance

Table E.2: HTDA hit rate for Datasets: The hit rate is defined as number of accesses with data cached over total number of accesses. See Section 3.4.2 for details on how scores are derived.

Index	Dataset (Basename)	Accesses	Cache Hit Rate
25	Zmm_DoubleMu_Run2015C_PRV1_13TeV	1448	0.72
24	Zmm_DoubleMu_Run2015D_PRV4_13TeV	1921	0.93
23	Zee_DoubleEG_Run2015C_PRV1_13TeV	52	0.05
22	Zee_DoubleEG_Run2015D_PRV4_13TeV	139	0.07
21	Zee_DYJetsToLL_M_50_aMCatNLO_Asympt25ns_13TeV	240	0.00
20	Zmm_DYJetsToLL_M_50_aMCatNLO_Asympt25ns_13TeV	1110	0.55
19	DoubleMu_Run2015C_Jul2015_13TeV	324	0.52
18	DoubleMu_Run2015D_Sep2015_13TeV	551	0.93
17	DYJetsToLL_M_50_aMCatNLO_Asympt25ns_13TeV	212	0.75
16	DYJetsToLL_M_50_aMCatNLO_Asympt50ns_13TeV	64	0.00
15	DoubleMu_Run2015D_Sep2015_13TeV	110	0.41
14	DYJetsToLL_M_50_aMCatNLO_Asympt50ns_13TeV	84	0.00
13	DYJetsToLL_M_50_aMCatNLO_Asympt25ns_13TeV	512	0.84
12	DoubleMu_Run2015C_Aug2015_13TeV	540	0.89
11	2015-07-28_ee-backgrounds_Run2012	201	0.00
10	2015-07-28_ee-data_Run2012	364	0.32
9	2015-08-06_ee-mc-gen_Run2012	14	0.00
8	DoubleMu_Run2015B_Jul2015_13TeV	686	0.28
7	DYJetsToLL_M_50_aMCatNLO_Asympt50ns_13TeV	16513	0.92
6	2015-07-28_ee-mc_Run2012	272	0.16
5	2015-05-18_DoubleMu_Run2012_22Jan2013_8TeV	12224	0.91
4	2015-05-16_DYJetsToLL_M_50_madgraph_8TeV	368	0.01
3	DoubleMu_Run2015B_Jul2015_13TeV	29	0.20
2	DYJetsToLL_M_50_aMCatNLO_Asympt50ns_13TeV	361	0.58
1	DoubleMu_Run2015B_Jul2015_13TeV	91	0.47

Acronyms

CHS Charged Hadron Subtraction 61, 69–73, 123, 128, 137, *Glossary:* Charged Hadron Subtraction

CVMFS CernVM File System 26, 123, *Glossary:* CernVM File System

HDD Hard Disk Drive 123, 128, *Glossary:* Hard Disk Drive

HEP High Energy Physics 21, 22, 25–27, 33–36, 42, 43, 47, 49, 50, 55

HPC High Performance Computing 48

HTDA High Throughput Data Analysis 35–38, 40–50, 52–55, 121–123, 133–135, 143, *Glossary:* High Throughput Data Analysis

IEKP Institut für Experimentelle Kernphysik 18, 34, 45, 48–50, 54, 72, 121, 133

IPC Inter-Process Communication 37, 123, *Glossary:* Inter-Process Communication

JEC Jet Energy Corrections 23, 24, 57, 58, 60, 70, 123, 132, *Glossary:* Jet Energy Corrections

LFU Least Frequently Used 42, 123, *Glossary:* Least Frequently Used

LHC Large Hadron Collider 5, 6, 9, 10, 12, 13, 16, 17, 21, 22, 26, 55, 57, 60, 63, 80, 83, 84, 127, 129

LRFU Least Recently/Frequently Used 43, 123, *Glossary:* Least Recently/Frequently Used

LRU Least Recently Used 42, 123, *Glossary:* Least Recently Used

MC Monte Carlo Simulation 123, *Glossary:* Monte Carlo Simulation

MET Missing Transverse Energy 64, 65, 76, 87–89, 123, 137, 139, *Glossary:* Missing Transverse Energy

MPF Missing E_{T} Projection Fraction 2, 76–79, 96–98, 102–104, 138, 140, 141

NAF National Analysis Facility 17, 124, *Glossary:* National Analysis Facility

PDG Particle Data Group 66, 74, 124, 138, *Glossary:* Particle Data Group

PUPPI PileUp Per Particle Identification 61, 69–73, 124, 137, *Glossary:* PileUp Per Particle Identification

RAM Random Access Memory 24, 50, 121, 124, *Glossary:* Random Access Memory

RPC Remote Procedure Call 37, 38, 124, *Glossary:* Remote Procedure Call

SSD Solid State Drive 22, 24, 26, 40, 41, 49, 51, 124, 135, *Glossary:* Solid State Drive

WLCG Worldwide LHC Computing Grid 5, 17–19, 21, 49, 124, 132, *Glossary:* Worldwide LHC Computing Grid

Glossary

Array of Structs Data structure for storing data of similar objects. Uses a sequence where each position corresponds to a single object holding its attributes. Efficient for working with all attributes of an individual object. Contrast with Struct of Arrays. 128

Artus Analysis framework for the Kappa data format. 19, 60, 61, 125, 132

CernVM File System File system providing access to remote software repositories. Repositories are compiled, maintained, and provided globally. The actual file system allows to easily make the content available on many processing hosts. 26, 123

Charged Hadron Subtraction Pileup mitigation technique, the default method of the CMS collaboration. As charged hadrons are visible to the CMS tracker, their point of origin can be reconstructed. Charged hadrons originating from pileup interactions can thus be classified as such. Charged Hadron Subtraction excludes these hadrons from jet clustering. 69, 123, 128

ClassAd Data structure and format used by HTCondor to describe any resource. This includes pool resources, e.g. worker nodes, and user resources, e.g. jobs. ClassAds are used to publish metadata required for scheduling in an HTCondor pool. Structurally, ClassAds are mappings containing key-value or key-expression pairs. Expressions may be evaluated in respect to other ClassAds, e.g. a job requiring memory on a worker node. 46

ECAL The Electromagnetic Calorimeter, or ECAL for short, is a subdetector type of the CMS detector. It detects the energy of leptons and photons, with the exception of muons. Energy is detected by kinetic absorption of particles; the heavy muons can pass through the ECAL absorber. 66, 73–75, 138

Excalibur Analysis suite for $Z \to (\mu\mu/ee)$ + Jet events, built on the Artus framework. 60, 61, 63, 64

Hadoop Data processing framework focusing on data locality. Usually used as a processing cluster, similar to a batch system. Special parallel file systems, such as HadoopFS, may publish locality information for a Hadoop cluster. The cluster may split and distribute analysis processes to optimise the locality of data access. 26, 45

Hard Disk Drive Persistent data storage device, using magnetic disks. Relying on simple and established technology, high capacities are available at low cost. 123, 128

High Throughput Data Analysis Prototype of a coordinated caching middleware. 35, 121, 123

HTCondor Batch system for high throughput computing. It excels at combining spatially and administratively distributed resources. As such, it is commonly used for managing grid, cloud, and other opportunistic resources. In addition, it offers a wide range of settings and hooks, allowing for a high level of customisability. 26, 35, 45, 46, 121, 125, 134

Integrated Luminosity A measure for the volume of interactions that happened in a time period. It is derived by integrating the Luminosity over a given period of time. While the Luminosity allows calculating the *rate* of a process, the integrated Luminosity provides the *count*. 63

Inter-Process Communication Mechanism for distinct processes to share data. IPC is typically used to perform requests from one process to another. 37, 123

Jet Energy Corrections Formalised corrections to match reconstructed jet energy to actual jet energy. 23, 57, 123, 132

Kappa Framework for creating and reading optimised datasets. 61, 125

Least Frequently Used Cache selection algorithm rating items by their frequency of accesses. Most effective if items are used repeatedly over a long period of time. 42, 123

Least Recently Used Cache selection algorithm rating items by their last access time. Most effective if items are used repeatedly over a short period of time. 42, 123

Least Recently/Frequently Used Cache selection algorithm rating items by both frequency and age of accesses. Most effective if items are used repeatedly over a limited period of time. 43, 123

Linux A family of operating systems building on the same central component, the Linux kernel. In general, one can distinguish between the kernel version and distribution. The kernel version defines the low-level functionality available, such as drivers or file systems. The distribution roughly defines the auxiliary tools and infrastructure around the kernel. 24, 50, 128

Luminosity A measure for the potential rate of interactions in a particle collider. It can be defined as the flow of particles through an area per time. For any process, its expected rate can be calculated from luminosity and interaction cross section of the process. 63, 126

MapReduce Processing paradigm used for local data processing. Workflows are specified in separate *map* and *reduce* steps. The former maps functions onto chunks of data, extracting information. The later reduces the chunks of information into one result. 25, 26

Missing Transverse Energy Transverse component of missing energy in a particle collision. Missing energy is an indication of particles not covered by the detector, e.g. neutrinos. In collision events at the LHC, the initial transverse energy is negligible. A non-negligible transverse energy in the final state implies a missing balancing energy deposit. 64, 87–89, 123, 137, 139

Monte Carlo Simulation Simulation technique for physical processes. Uses random sampling to model complex processes, by chaining simple processes with known random distributions. 123

National Analysis Facility Tier 3 analysis facility for german scientists, provided by DESY. 18, 124

NFS The *Network File System* is a protocol for accessing remote directories and files. It allows mounting remote directory trees similar to a local file system. NFS is commonly used to access distributed storage in local networks. 50

Particle Data Group International collaboration publishing particle properties. 66, 124, 138

Pileup Additional, soft collisions in particle collision events. To achieve high event rates, bunches of particles are collided instead of isolated ones. In addition to possible hard interactions, this practically always results in multiple soft collisions. The *pileup* collisions add a background of mostly low-energetic particle to events. 6, 57, 59, 61–64, 66, 67, 69–73, 76–81, 83, 84, 136–138

PileUp Per Particle Identification Pileup mitigation technique, generalising the Charged Hadron Subtraction (CHS) method. As with CHS, charged particles are used to identify particles from pileup interactions. From this, the general characteristics of pileup particles are derived, regardless of charge. These characteristics are then used to filter uncharged particles as well. 61, 124, 137

POSIX The *Portable Operating System Interface*, a definition of APIs provided by operating systems. Commonly defines the system calls of Unix operating systems, in turn implying some architectural features. Linux does not strictly follow the standard. However, it can be considered POSIX compliant for most purposes. 26, 35, 40, 47, 49, 134

Random Access Memory Fast data storage, typically volatile. Mostly used to hold data actively used by a computer, i.e. its main memory. The term RAM is commonly used to refer to both software and hardware implementing the main memory. 24, 121, 124

Remote Procedure Call Mechanism for executing calls in remote processes. Typically used to unify calls to local and remote resources through a common interface. 37, 124

REST The *REpresentational State Transfer*, a design model for handling requests. It is an abstract definition, focusing on architectural considerations. REST builds on a uniform communication interface, separating clients sending requests and servers handling them. A key feature is statelessness, i.e. each request is distinct. Actions depend only on the request and state of the server. 37

Solid State Drive Persistent data storage device, using integrated circuits. The lack of any moving mechanical parts allows for fast, concurrent reading, and writing. In general several times as expensive per capacity as an Hard Disk Drive (HDD) 22, 124, 135

Struct of Arrays Data structure for storing data of similar objects. Uses multiple sequences, where each sequence corresponds to an attribute and each object has its own index. Efficient for working with individual attributes of all objects. Contrast with Array of Structs. 19, 125

Virtual File System An abstraction layer unifying the interfaces of actual file systems. A VFS allows providing multiple file systems of different types in a single interface. In modern operating systems, users only interact with the VFS. Linux represents the VFS as the root directory, into which other file systems are mounted. 36, 48, 135

Worldwide LHC Computing Grid The data storage and processing infrastructure of the LHC collaborations. 5, 17, 21, 124, 132

xRootD Remote data access protocol and service. The xRootD protocol allows reading of data from local and remote data providers. 26, 49

List of Figures

List of Tables

Bibliography

[1] Lyndon Evans and Philip Bryant, "LHC Machine", *Journal of Instrumentation* **3** (2008) S08001, URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08001.

[2] S. Chatrchyan et al., "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC", *Physics Letters B* **716** (2012) 30–61, DOI: http://dx.doi.org/10.1016/j.physletb.2012.08.021.

[3] G. Aad et al., "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC", *Physics Letters B* **716** (2012) 1–29, DOI: http://dx.doi.org/10.1016/j.physletb.2012.08.020.

[4] The CMS Collaboration et al., "The CMS experiment at the CERN LHC", *Journal of Instrumentation* **3** (2008) S08004, URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08004.

[5] CMS Collaboration, "Detector Drawings", CMS Collection., 2012, URL: http://cds.cern.ch/record/1433717.

[6] Florian Beaudette, "The CMS Particle Flow Algorithm", *Proceedings, International Conference on Calorimetry for the High Energy Frontier (CHEF 2013)*, 2013, 295–304, [arXiv:1401.8155].

[7] Max Fischer, "Workflow Management auf verteilten Computingsystemen für Analysen in der Teilchenphysik", Dipl. Thesis, Karlsruhe Institute of Technology (KIT), 2013, URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48317.

[8] Matteo Cacciari, Gavin P. Salam and Gregory Soyez, "The Anti-k(t) jet clustering algorithm", *JHEP* **04** (2008) p. 063, DOI: 10.1088/1126-6708/2008/04/063, [arXiv:0802.1189].

[9] Ian Bird, "Computing for the Large Hadron Collider", *Annual Review of Nuclear and Particle Science* **61** (2011) 99–118, DOI: 10.1146/annurev-nucl-102010-130059, eprint: http://dx.doi.org/10.1146/annurev-nucl-102010-130059.

[10] WLCG Office, *WLCG Public Homepage*, 2016, URL: http://wlcg-public.web.cern.ch/.

[11] J. Berger et al., "ARTUS - A Framework for Event-based Data Analysis in High Energy Physics", *ArXiv e-prints* (2015), [arXiv:1511.00852].

[12] M Fischer and E Kuehn, "Data Locality via Coordinated Caching for Distributed Processing", *CLOUD COMPUTING 2016* (2016).

[13] Max Fischer et al., "Tier 3 batch system data locality via managed caches", *J. Phys. Conf. Ser.* **608** (2015) p. 012018, DOI: 10.1088/1742-6596/608/1/012018.

[14] M Fischer et al., "High Performance Data Analysis via Coordinated Caches", *J. Phys.: Conf. Ser.* **664** (2015) 092008. 8 p, URL: https://cds.cern.ch/record/2134645.

[15] Christian Metzlaff, "Optimierung eines Cachingsystems für einen hohen Datendurchsatz bei der Analyse von *pp*-Kollisionen am LHC", M.S. Thesis, Karlsruhe Institute of Technology (KIT), 2016, URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48755.

[16] David Howells and Red Hat, "Fs-cache: A network filesystem caching facility", *In Proceedings of the Linux Symposium*, p. 2006.

[17] Kent Overstreet, *bcache*, 2016, URL: https://bcache.evilpiepirate.org.

[18] M J Schnepf et al., "Benchmark of a Cubieboard cluster", *Journal of Physics: Conference Series* **664** (2015) p. 052032, URL: http://stacks.iop.org/1742-6596/664/i=5/a=052032.

[19] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Commun. ACM* **51** (2008) 107–113, DOI: 10.1145/1327452.1327492.

[20] The Apache Software Foundation, *Apache Hadoop*, 2016, URL: https://hadoop.apache.org.

[21] Konstantin Shvachko et al., *The Hadoop Distributed File System*.

[22] S Paul and Z Fei, "Distributed caching with centralized control", *Computer Communications* **24** (2001) 256–268, DOI: http://dx.doi.org/10.1016/S0140-3664(00)00322-4.

[23] J. Blomer and T. Fuhrmann, "A Fully Decentralized File System Cache for the CernVM-FS", *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, 2010, 1–6, DOI: 10.1109/ICCCN.2010.5560054.

[24] Derek Weitzel, Brian Bockelman and David Swanson, "Distributed Caching Using the HTCondor CacheD", *Proceedings for Conference on Parallel and Distributed Processing Techniques and Applications*, 2015, URL: http://stacks.iop.org/1742-6596/513/i=3/a=032054.

[25] W Yang et al., "Using Solid State Disk Array as a Cache for LHC ATLAS Data Analysis", *Journal of Physics: Conference Series* **513** (2014) p. 042035, URL: http://stacks.iop.org/1742-6596/513/i=4/a=042035.

[26] S A Russo, M Pinamonti and M Cobal, "Running a typical ROOT HEP analysis on Hadoop MapReduce", *Journal of Physics: Conference Series* **513** (2014) p. 032080, URL: http://stacks.iop.org/1742-6596/513/i=3/a=032080.

[27] S Lehrack, G Duckeck and J Ebke, "Evaluation of Apache Hadoop for parallel data analysis with ROOT", *Journal of Physics: Conference Series* **513** (2014) p. 032054, URL: http://stacks.iop.org/1742-6596/513/i=3/a=032054.

[28] Python Software Foundation, *Python*, 2016, URL: https://www.python.org.

[29] Python Software Foundation, *Python os.path documentation*, 2016, URL: https://docs.python.org/2.7/library/os.path.html.

[30] Nimrod Megiddo and Dharmendra Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache", *In Proceedings of the 2003 Conference on File and Storage Technologies (FAST)*, 2003, 115–130.

[31] Donghee Lee et al., "LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies", *Computers, IEEE Transactions on* **50** (2001) 1352–1361, DOI: 10.1109/TC.2001.970573.

[32] N Ratnikova et al., "CMS Space Monitoring", *Journal of Physics: Conference Series* **513** (2014) p. 042036, URL: http://stacks.iop.org/1742-6596/513/i=4/a=042036.

[33] Georg Fleig, "Dynamic Integration of Cloud Resources into Local Computing Clusters and Calibration of the Jet Energy Scale with the CMS Detector", M.S. Thesis, Karlsruhe Institute of Technology (KIT), 2016.

[34] M Giffels et al., "Dynamic provisioning of local and remote compute resources with OpenStack", *Journal of Physics: Conference Series* **664** (2015) p. 022022, URL: http://stacks.iop.org/1742-6596/664/i=2/a=022022.

[35] Junjiro R. Okajima, *AUFS Project Homepage*, 2015, URL: http://aufs.sourceforge.net.

[36] Neil Brown, *OverlayFS Design Specification*, 2016, URL: https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystems/overlayfs.txt.

[37] Scientific Linux, *Scientific Linux*, 2016, URL: https://www.scientificlinux.org.

[38] CentOS Project, *CentOS Project*, 2016, URL: https://www.centos.org.

[39] Sebastian Brommer, "Meta-Monitoring for performance optimization of a computing cluster for data intensive analysis", B.S. Thesis, Karlsruhe Institute of Technology (KIT), 2016.

[40] Rakesh Agrawal and Ramakrishnan Srikant, "Fast algorithms for mining association rules", *Proc. of 20th Intl. Conf. on VLDB*, 1994, 487–499.

[41] Jiawei Han, Jian Pei and Yiwen Yin, "Mining Frequent Patterns Without Candidate Generation", *SIGMOD Rec.* **29** (2000) 1–12, DOI: 10.1145/335191.335372.

[42] The CMS collaboration, "Determination of jet energy calibration and transverse momentum resolution in CMS", *Journal of Instrumentation* **6** (2011) P11002, URL: http://stacks.iop.org/1748-0221/6/i=11/a=P11002.

[43] Dominik Haitz, "Precision Studies of Proton Structure and Jet Energy Scale with the CMS Detector at the LHC", PhD Thesis, Karlsruhe Institute of Technology (KIT), 2016, URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48801.

[44] Julia Handl, "Jet Energy Calibration of the CMS Detector with Z ($\rightarrow$ee) + Jet Events at $\sqrt{s}$ = 13TeV", M.S. Thesis, Karlsruhe Institute of Technology (KIT), 2016.

[45] Daniele Bertolini et al., "Pileup Per Particle Identification", *JHEP* **10** (2014) p. 059, DOI: 10.1007/JHEP10(2014)059, [arXiv:1407.6013].

Danksagung

Meine langjährige Forschung und schlussendliche Entstehung dieser Arbeit wäre ohne die Unterstützung vieler Leute nicht möglich gewesen. Die Menschen, die mir diesen Weg eröffnet und mich darauf begleitet haben, sind zu viele, um sie zu benennen. Jedem einzelnen gebührt mein Dank.

Ein spezieller Dank gilt er Karlsruher Schule für Elementarteilchen- und Astroteilchenphysik (KSETA). Dies betrifft nicht nur die Förderung dieser Arbeit, sondern auch die Ermöglichung eines fachübergreifenden Austausches.

Einen ganz besonderen Dank verdient Prof. Dr. Günter Quast. Ohne ihn hätte ich diesen Weg weder eingeschlagen, noch wäre ich ihm bis zu seinem Ende gefolgt. Seine Motivation und Begeisterungsfähigkeit waren Inspiration weit über fachliche Belange hinaus.

Besonderer Dank gebührt auch Prof. Dr. Achim Streit. Seine Aufgeschlossenheit und Geduld bei dieser interdisziplinären Arbeit haben sie überhaupt erst möglich gemacht.

Einen herzlichen Dank verdienen die Kollegen meiner Arbeitsgruppen. Jeder einzelne war Motivation und Inspiration auf meinem Weg. Ohne euch hätte es keinen Spaß gemacht.

Ich bedanke mich über alle Maßen bei meiner Familie und Freunden, für ihre Unterstützung und Verständnis.

Erklärung der selbständigen Anfertigung der Dissertationsschrift

Hiermit erkläre ich, dass ich die Dissertationsschrift mit dem Titel

»*Coordinated Caching for High Performance Calibration using* $Z \to \mu\mu$ *Events of the CMS Experiment*«

selbständig und unter ausschließlicher Verwendung der angegebenen Hilfsmittel angefertigt habe.

Max Fischer
Karlsruhe, den 05. Juli 2016