An Implementation of the ALICE TRD Online Reconstruction

Diplomarbeit

Goethe-Universität Frankfurt am Main
Fachbereich Physik

Institut für Kernphysik

Theodor B. Rascanu

September 2010

Zusammenfassung

In dieser Arbeit wird die Implementierung der Echtzeit-Rekonstruktion, Kalibrierung und Darstellung der Daten des ALICE Transition Radiation Detektors vorgestellt.

Die Rekonstruktion wird auf dem High-Level Trigger, der dritten Stufe des Triggersystems von ALICE, durchgeführt und erlaubt die Online-Kalibrierung und Darstellung der eingehenden Daten. Zusätzlich kann der HLT die Speicherung der aufgenommen Daten steuern, so dass nur physikalisch interessante Events gespeichert werden.

Für diese Rekonstruktion, als auch für die Kalibrierung, werden die bereits vorhandenen Offline-Algorithmen verwendet. Dazu musste die Nahtstelle zwischen dem HLT und diesen Offline-Algorithmen implementiert werden. Um allerdings die notwendige Geschwindigkeit von 2000 Hz in Proton-Proton-Kollisionen, und 200 Hz in Blei-Blei-Kollisionen erreichen zu können, mussten die Algorithmen beschleunigt werden. Um die Engpässe zu finden, wurden die Algorithmen mit speziellen Tools analysiert, um daraufhin betreffenden Kode durch eine Neuimplementierung zu ersetzen oder aber während der Online-Rekonstruktion zu überspringen. Um sicherzustellen, dass dadurch die Qualität nicht zu stark gemindert wird, wurde diese während der Implementierung überwacht.

Abstract

This thesis presents the implementation of the online reconstruction, calibration and monitoring of the data of the Transition Radiation Detector of ALICE.

This reconstruction is performed on the High Level Trigger, the third level of the ALICE trigger system, and enables online calibration and monitoring of the incoming data. Additionally, the HLT can steer the data storage, such that only physical interesting events are saved.

The online reconstruction, as well as the calibration, makes use of the existing offline algorithms. Therefore, interfaces between the HLT and these offline algorithms were implemented. For being able to reach the speed of 2000 Hz in proton-proton collisions, and 200 Hz in lead-lead collisions, the algorithms had to be accelerated. Bottlenecks were tracked down using dedicated tools, and respective code was either re-implemented or it is being skipped during the online reconstruction. The quality of the output data was monitored throughout the implementation, to assure that it is not being cut too much.

Contents

1 Preface

Ever since people asked why nature is as it is, and tried to find laws describing their observation, the understanding of how nature behaves evolved. One such law, for example, describes how apples fall onto Newton's head. Some other describe how planets en-circulate the sun. Very soon it was realised that these laws have a lot in common, and actually describe the same fundamental force from different perspectives, we call this force "gravity". There are some other such fundamental forces, each having its own way of behaving. We call those other forces: electromagnetic, weak and strong force. Many questions have been answered up to now, but as it is very often each answer comes with a price tag labelled in the currency "new more fundamental question". All physics fields like Astrophysics, Solid State Physics, or Nuclear Physics have different perspectives on our universe, and thus they can observe different aspects of the fundamental forces.

The achievements in one field can lead to unveilings in others. Amusingly this mutual impact is very strong between two seemingly very distant fields: astrophysics and nuclear physics. In fact both are not that distant at all when, for example, looking at the goals of current nuclear physics experiments:

- What were the properties of the very early universe?
- Can our models explain the origin of mass?
- What is dark energy?
- Can we explain the evolution of our universe?

These questions are obviously also very fundamental astronomical questions; answering all of those questions is the goal of the experiments built near Geneva at CERN.[1]

Each field has its own very special methods of teasing out more knowledge from nature's seemingly bottomless reservoir. The way nuclear physics is today trying to endeavour its sub microscopic view of the world was first travelled by Ernest Rutherford with his scattering experiment. From his experiment we know that the naming of the so called atom was premature: Since then nuclear physicists are engaged in finding the right description of the substructure of something we call "indivisible".

[1] The first question is the search of the characteristics of the Quark-Gluon-Plasma, which we shall encounter again later; this will be addressed by ALICE. The second is the search after the Higgs Boson; ATLAS and CMS will try to find this last missing particle of the Standard Model. The question about the nature of dark energy is assigned to CMS only, and last but not least LHCb will try to find out more about the CP-violation of the weak nuclear force, which is important for the asymmetry of matter and antimatter.

2 Introduction

2.1 The Pedigree of Particle Physics

The idea that it is impossible to divide matter endlessly was a concept by early Greek philosophers. Thus it was thought that all matter is built out of smallest components, which they called $\alpha\tau o\mu o\varsigma$. Later, chemists of the 17th and 18th century showed that certain substances cannot be further divided by chemical methods. Those substances where called chemical elements and it was assumed that these elements where built out of atoms.

The idea of indivisible atoms had soon to be revised. First signs were the great number of elements and the periodicity of their characteristics. This became clearer as the first steps of the emerging nuclear physics were taken:

William Conrad Röntgen's discovery in 1895, the X-rays, led to an extensive search of other radiation sources. Just a year later Henri Becquerel discovered the radiation of uranium salts by their peculiarity of exposing photographic plates in the dark. In 1898 Marie Curie was able to show that this radiation must be an attribute of the uranium atoms themselves and cannot be explained by chemical reactions. She called this spontaneous radiation of the material "radioactivity". Due to the high radiation, Marie and her husband Pierre found two more elements later that year, Polonium and Radium.

Trying to bring some systematics into the different types of radiation, they were named alphabetically in Greek in the order of decreasing interaction with matter. In 1909 Rutherford's experiment of scattered alpha particles on a gold foil, led to the conclusion that atoms consist of a small positive charged nucleus surrounded by negative electrons. Just two years later Niels Bohr joined the knowledge of quantum mechanics and the outcome of Rutherford's experiment into his model of atomic structure.

The restriction to the distinct energies alpha particles have, being emitted by the sources, let soon arise ideas about how to accelerate those charged particles for further experiments. The particle accelerator was invented.

As in 1932 the neutron was found, the nucleus was complete and together with the prediction of the neutrino in 1930 all observations could be well described. This idyllic atmosphere soon vanished in 1937 as a new particle was detected coming with cosmic rays, the muon. The surprise about this unexpected particle, which was found to be like an electron but heavier, is best summarised in I. I. Rabi's comment: "Who ordered that?"

Until the late 1950's literally hundreds of new particles where found in scattering experiments and could be arranged in two groups: leptons and hadrons. Because of the large number of hadrons the situation was comparable to the situation at the end of the 19th century; where there were simply too many chemical elements for these to be truly called elementary. It was the beginning of the 1960's when it was finally clear that the hadrons, the particles forming the atoms nuclei, could not be elementary either and had to be built up out of even smaller components, which then were called partons. However an unambiguous understanding of their nature was missing until the quark model proposed by Murray Gell-Mann and George Zweig became commonly accepted in 1969 as the predicted Ω^- particle was found.

This brought a high degree of tidiness into the overwhelming big "particle zoo" of the 1960's. Since then this theory evolved into what we today call the Standard Model. [PER, KBK, DEM]

2.2 The Standard Model of Particle Physics

The Standard Model is a theory that describes the known particles and their interactions. In this theory all matter is composed out of quarks and leptons. The possible interactions among particles are given by the fundamental forces, of which this theory leaves out the gravity, and are described as an exchange of force mediating particles.

Quarks and leptons are fermions respecting the Pauli Principle, having half spin and obeying the Fermi-Dirac-statistic, whereas force mediating particles are bosons not respecting Pauli's Principle, having integer spin and obeying the Bose-Einstein-statistic. The Standard Model includes 12 fermions, 12 anti-fermions and 12 known force mediating particles, which are also called gauge bosons. One more boson is predicted and would complete the theory: the Higgs boson.

There are six quarks and six leptons and equally many antiparticles. Quarks posses all known charges, and interact thus via all known forces, which makes them unique in that way. They have a mass, they have electric charge and as all other particles they carry weak charge. In addition to that quarks also carry another type of charge, the colour charge. Unlike electric and weak charge, of which there are two forms (plus and minus), mass has only one form, while colour charge has six: red, green, blue, anti-red, anti-green and anti-blue. Due to the so called colour confinement (or just confinement) quarks are bound together to form colour-neutral composite particles. There are two known quark configurations, although more are imaginable: a quark and an anti-quark forming a meson, while three quarks form a baryon. These colourful names of the colour charge were chosen because of the following similarity: In principle one can think of the classical colour theory (in its additive form) where anti-red would be cyan, anti-green magenta and anti-blue would be yellow; the confinement would demand composite particles to be white.

The remaining six fermions, the leptons, all miss the colour charge, while the three neutrinos also miss the electric charge, which leaves them to interact solely by weak interaction[2], hence making them hardly detectable. Fermions are grouped in three generations, which are sorted by increasing mass. Second and third generation charged particles decay with very short half lives, making all visible matter to consist of only first generation charged particles and the very weakly interacting neutrinos of all generations. Table 1 summarises the properties of the fermions.

When two particles approach each other, the force between those is described in the Standard Model as an exchange of virtual gauge bosons. Each charge produces a correspondent fundamental force, and each force has its gauge bosons to mediate it. The higher the absolute amount of charge, the higher the force and the higher the coupling of the gauge boson to the particle. The bosons have to put back the distance between the interacting particles, by propagating through space. The more

[2]remember, gravitation is not included and is anyhow very weak

	Generation			Charges					
	1	2	3	Mass [eV/c^2]			Electric [e] (z)	Colour	Weak [g] (T_z)
Quarks	u	c	t	2.55 M	1.17 G	171 G	2/3	r,g,b	1/2
	d	s	b	5.04 M	105 M	4.2 G	-1/3	r,g,b	-1/2
Leptons	e^-	μ^-	τ^-	511 k	106 M	1.78 G	-1	0	-1/2
	ν_1	ν_2	ν_3	≈ 0			0	0	1/2

Table 1: Eigenstates of the fermions according to the Standard Model [PDG]. The mass is shown for the mass eigenstates, and the weak charge is shown for the left handed weak eigenstates. The weak eigenstates of the neutrinos ν_e, ν_μ and ν_τ are linear combinations of the mass eigenstates ν_1, ν_2 and ν_3, and the weak eigenstates of the quarks d', s' and b' are linear combinations of the mass eigenstates d, s and b. For the charged leptons the mass and weak eigenstates coincide. Only for the $W^\pm$ bosons does the indicated weak charge directly show the coupling strength. The Z^0 coupling is given by $z - T_z x$, with $x = \sin^2 \Theta_W \approx 0.23$, where Θ_W is the Weinberg angle.

Interaction	Couples to	Gauge Boson	Mass [GeV/c^2]	El. [e]	Col.	Weak [g]	eff. Range	rel. eff. Strength
Strong	colour	8 gluons	0	0	yes	0	10^{-15}	1
Electromg.	electric	photon	0	0	0	0	∞	10^{-2}
Weak	weak	$W^\pm$, Z^0	80, 91	± 1, 0	0	± 1, 0	10^{-18}	10^{-14}
Gravitation	mass	graviton ?	0	0	0	0	∞	10^{-38}

Table 2: The known interactions and their gauge bosons [PER]. Each gluon has one of the following colour combinations: $r\bar{g}$, $r\bar{b}$, $g\bar{r}$, $g\bar{b}$, $b\bar{r}$, $b\bar{g}$, $r\bar{r} - g\bar{g}$, $r\bar{r} + g\bar{g} - b\bar{b}$.

massive the boson the shorter the distance it can propagate. Both, the coupling and the propagation contribute to the effective strength of the force.

The strongest force is the so called strong nuclear force; it's mediated by the gluons between colour charged particles. There are eight gluons, each carrying another colour charge. These are the particles which bind colour charge carrying particles together, even themselves, thus their suggestive name. The next strongest fundamental force is the electromagnetic force, which is mediated by the (electrically neutral) photon between electrical charged particles. Since there is a strong nuclear force, we also know of a weak nuclear force: this is mediated by the massive W^+, W^- and Z^0 bosons between all fermions. This force is the only one to not produce bound compounds, and can thus only be sensed by the transitions it generates, which otherwise would be forbidden; e.g. the beta decay of unstable isotopes. By far the weakest fundamental force is the gravity, which is why it can be neglected in most cases. Though there are some very extreme cases, where it becomes necessary to have gravity included (think of neutron stars); this is the barrier of today's Standard Model. There are ambitions to also include gravity into the theory, thus implying the existence of another gauge boson, the graviton. Table 2 summarises the properties of the bosons. [PER, KBK]

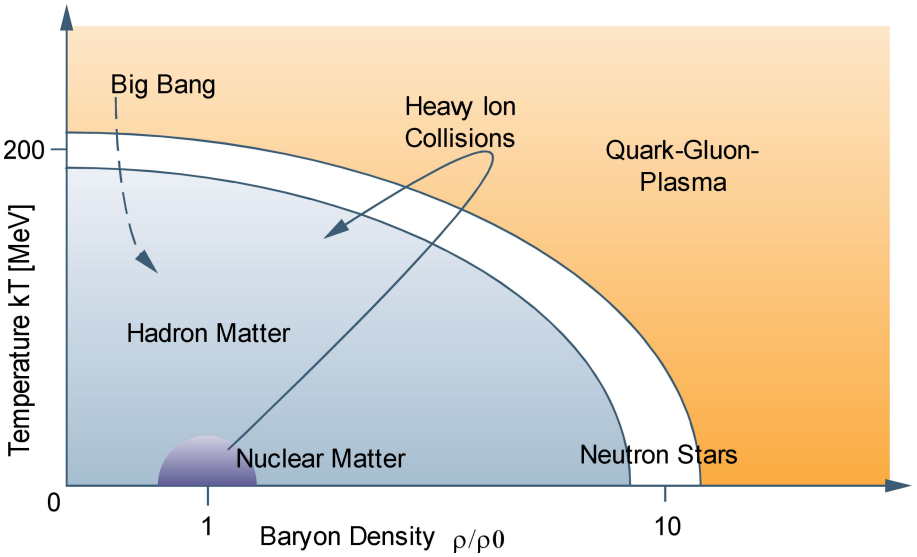


Figure 1: Phase diagram of quarks and gluons. In normal matter quarks and gluons are bound in hadrons, which compose the nuclei, at density ρ_0 and temperature 0. Also shown are the traces of the matter during heavy ion collisions in current accelerators and the early universe. The transition was about 1 μs after the Big Bang. In the core of neutron stars quark gluon plasma might exist at low temperatures already due to the immense density. [TUK]

2.3 The Quark Gluon Plasma

It is expected that moving towards very high densities or temperatures nuclear matter undergoes a phase transition lifting the confinement of the strong interacting quarks and gluons, making them free particles. This phase of matter is called the Quark-Gluon-Plasma (QGP). At small temperatures, ten times the density of nuclei is needed to approach nucleons so much that their wave functions overlap, thus losing their identity and dissolving into a big nuclear lump containing free quarks and gluons. The interior of neutron stars is thought to be such big nuclear lump. On the other hand right after the Big Bang only a low[3] net baryon density was present; but, due to the high temperatures, a high energy density was available. Thus during its first micro second our universe was in a Quark-Gluon-Plasma state. When the universe had expanded enough it cooled down to less than 200 MeV and quarks and gluons combined to hadrons (Figure 1).

In heavy ion collisions similar conditions are generated. Being propelled by the accelerator to great energies, the colliding Lorentz compressed nuclei produce high densities and high temperatures. Due to its high pressure the resulting QGP is expanding into a fireball bringing quarks and gluons back to their confinement eventually. While this fireball expands and cools down, inelastic interactions between hadrons cease out thus fixing the particle composition. This stage is called "chemical freeze out" and is followed by the "thermal freeze out" where also the elastic interactions come to an end, fixing also the momentum distribution [ORS].

[3]The low net baryon density is due to the fact the in the first instances our universe had as much matter as anti-matter. Later this balance lost its equilibrium due to the CP-violation in the weak interaction.

Due to the very short time the QGP state is sustained, it is not straight forward to get a clear evidence of its existence. Thus a series of different observables has to be considered. All observables are monitored depending on the centrality, as the formation of the QGP should be highly dependent on the number of binary collisions. Additionally the observables should be compared between collisions of pp, pA and AA[4] at comparable collision energy per nucleon.

Kinematic Probes

The global experimental observables *average transverse momentum* $\langle p_T \rangle$, *hadron rapidity distribution* dN/dy and *transverse energy distribution* dE_T/dy are directly connected to the thermodynamic characteristics temperature, entropy density and energy density of the fireball, which forms at the collision. At the phase transition towards a QGP a sudden rise of the degrees of freedom should be visible in the change of energy density and entropy as function of temperature.

Another observable is the collective flow, which forms due to the pressure gradient inside the fireball. The formation of a QGP should significantly decrease the flow.

Jet Quenching

When two partons hit head on, they deflect each other back on back out of the fireball. Due to the confinement the partons need to hadronise and form thus two jets. In case a QGP forms, the partons should experience an energy loss due to the strong interaction with the colour charged medium. This should alter the jet structure, and multi particle correlations.

Strangeness Enhancement

Due to strangeness conservation, strange particles can only be produced in pairs. The threshold energy needed is given by the mass of the produced pair. Under normal circumstances the lightest strange particle pair consists of two kaons. However, inside a QGP where the confinement is lifted, strange quarks pairs can be produces directly, lowering the threshold considerably. Thus an enhancement of multi-strange hyperons is expected.

Quarkonium Suppression/Enhancement

There are diverging expectations on how the production of the charmonium and bottomium states changes at the phase transition:

Due to the unbound colour charge of the medium, it is expected that Debye screening should lead to a suppression of the quarkonium states. The suppression of individual states should be depending on the distance of the quark pairs and the temperature of the medium.

Other theories, however, predict an enhancement due to statistical recombination inside the QGP or at the freeze out.

[4]Whereas “pp” mean proton-proton collision and “AA” heavy nuclei collisions, in our case lead

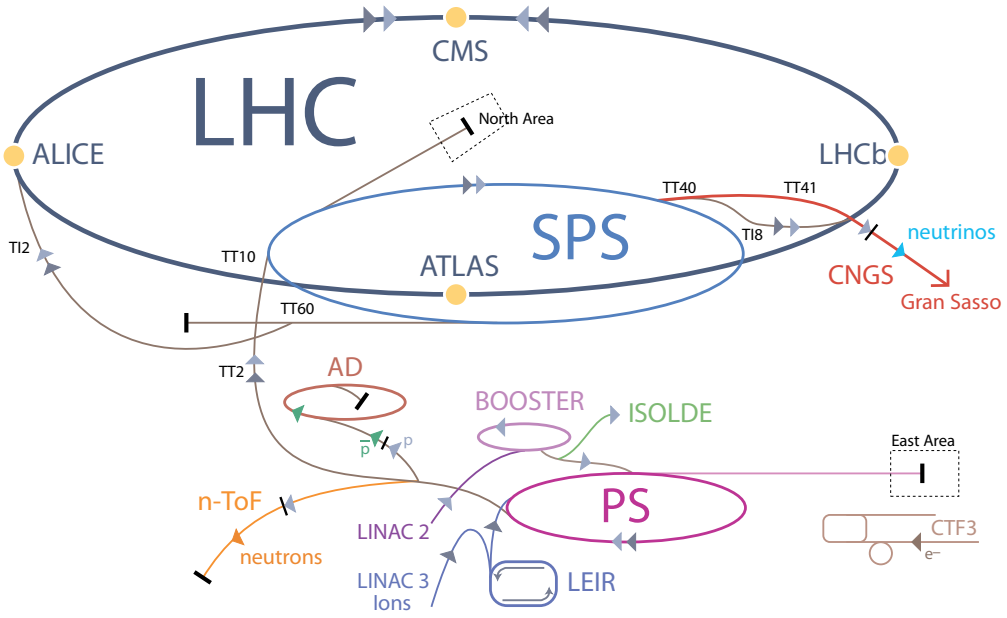


Figure 2: The CERN accelerator complex. Protons are starting their way at LINAC 2 and lead ions at LINAC 3. [CER]

Electromagnetic Probes

Leptons and photons are probes for the earliest moments of the interaction, since they are not influenced by the strong force they can leave the fireball much less obstructed than hadrons. Thus information about the characteristics of the medium before the freeze out can be gained. [QGP1, QGP2, QGP3, JPS, JSN]

2.4 The Large Hadron Collider

The Large Hadron Collider is a ring accelerator at the European Organisation of Nuclear Research (commonly called CERN) situated near Geneva, at the border of Switzerland and France. In 2001 the construction of the LHC started, replacing the predecessor accelerator LEP (Large Electron-Positron Collider) in its 27 km circumference tunnel.

The LHC consists of 1392 superconducting dipole magnets guiding two particle beams of opposing directions on their way through the tunnel. The two beams, which are 20 cm apart, are focused by 392 superconducting quadrupole magnets to a size of only 16 μm in transverse direction. The magnets are cooled to 1.9 K, which is well below the transition temperature of the NbTi alloy windings in the magnets, to maximise the field strength up to the needed 8.3 Tesla required for the highest particle energies and for preserving the Helium in its superfluid phase.

Before being injected into the LHC particles are successively pre-accelerated in an accelerator cascade starting at the linear accelerator LINAC 2 / LINAC 3[5] followed by the accelerator rings PSB / LEIR[5], PS and SPS (Figure 2). Once in the LHC

[5]The first mentioned accelerator is used for protons exclusively, the second mentioned is for lead ions.

Parameter	Value
Total length	26.659 km
Minimal radius	2805 m
Momentum at injection per proton	450 GeV/c^2
Maximal momentum per proton	7 TeV/c^2
Dipole field at 450 GeV/c^2	0.535 Tesla
Dipole field at 7 TeV/c^2	8.33 Tesla
Revolution frequency	11.245 kHz
Nucleons per bunch	$1.15 \cdot 10^{11}$ / 10^8
Bunches per beam	2808 / 592
Transverse beam size at interaction	16.7 μm
Longitudinal bunch size	7.6 cm
Luminosity	$10^{34}\,cm^{-2}s^{-1}$ / $10^{27}\,cm^{-2}s^{-1}$

Table 3: Some parameters of the LHC. [LHCDR, CER]

particles are further accelerated up to the peak energy of 7 TeV in proton-proton collisions and 2.76 TeV per nucleon in lead-lead collision.

The actual layout of the LHC is not a perfect circle, but includes eight 528 m straights and eight arcs. In the middle of four of the straights the two particle beams cross each other, these are the spots where the collisions happen and where the experiments of the LHC are placed. The other four straights house the beam cleaning, dumping and accelerating facilities. In Table 3 the main parameters of the LHC are shown. [LHCDR]

2.5 A Large Ion Collider Experiment (ALICE)

ALICE is an experiment devoted to the study of strongly interacting matter and the characteristics of the Quark-Gluon-Plasma in heavy ion collisions.

The aspect of ALICE is dominated by the huge magnet of the old LEP experiment L3, which is being reused (Figure 3). Its dimensions are almost 15 m cubed, giving enough room for most of the particle detectors. Only the forward spectrometer of the muon detector resides outside the magnet. The other detectors are placed onion-like around the interaction point.

Scope of this magnet is to create a homogeneous magnetic field, in which charged particles are forced to circular trajectories with radii dependent on their momentum. The particle mass, in fact the quantity of interest, can only be identified with the help of additional measurements.

As a large number of produced particles are anticipated in heavy ion collisions, detectors must be endowed with high spatial resolutions to be able to trace the trajectories and data taking must be capable of routing and saving the enormous amounts of data.

2.5.1 ALICE-Detectors

The ALICE experiment incorporates many detectors. The first four detectors of the upcoming list are the main detectors in the central barrel and all have full azimuthal

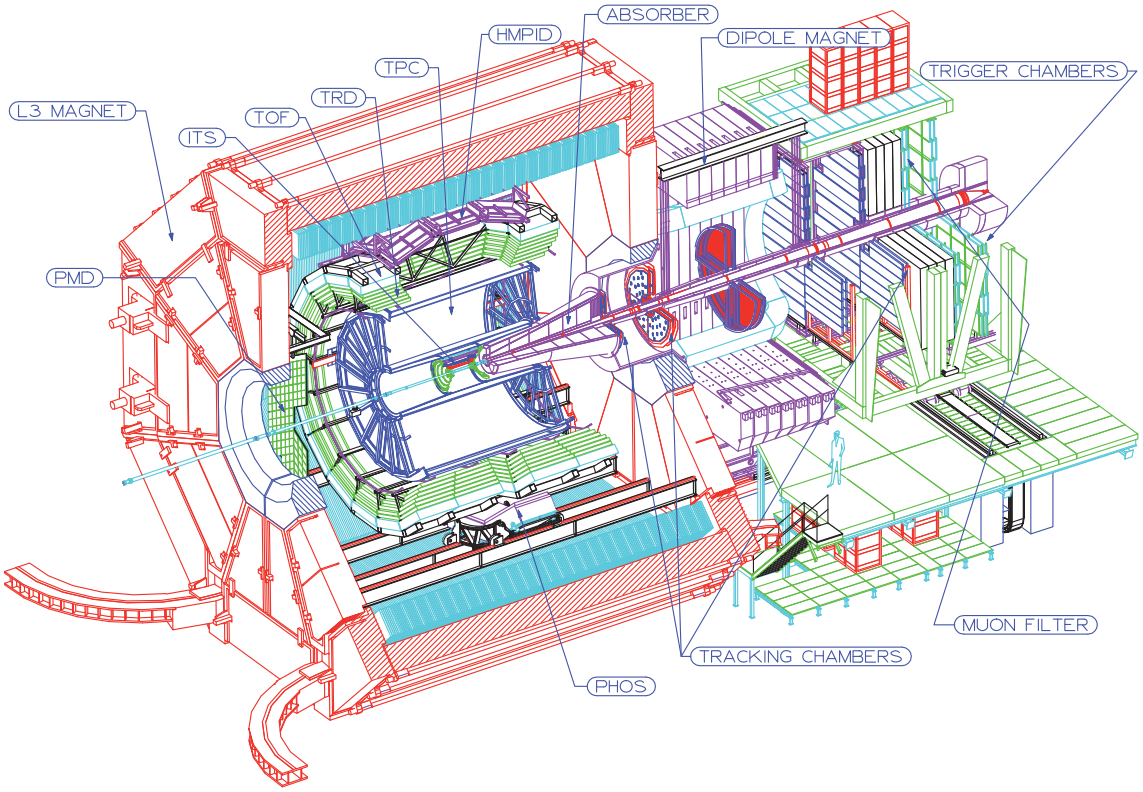


Figure 3: The ALICE Experiment and its detectors.

coverage. Their common pseudorapidity coverage is $|\eta| < 0.9$.

ITS The Inner Tracking System is the innermost detector of ALICE. It consists of three types of two layered silicon detectors, from inside to outside these are: The very high resolution silicon pixel detector (SPD) situated at radii of 4 cm and 7 cm, which is followed by the silicon drift detector at 15 cm and 24 cm, and lastly the layers of the silicon strip detectors at 39 cm and 44 cm. The ITS is optimised for high resolution of the primary vertices, for efficient track finding and for providing particle identification of low energetic particles.

TPC The ALICE Time Projection Chamber is not only the biggest detector in ALICE, it is in fact the biggest TPC ever built. By the gas ionisation in its volume of $95\,\mathrm{m}^3$, it gives the possibility of high resolution track finding and momentum measurement (see also Chapter 3.1). Together with the energy loss measurement the particle identification is realised.

TRD Because of its importance to this thesis the TRD is explained in more detail in the next section.

TOF The Time of Flight completes the particle identification in p_T regions where the other inner detectors do not provide reliable information. It is built out of 160000 multi-gap resistive plate chambers covering $150\,\mathrm{m}^2$ reaching a time resolution of less than 100 ps.

PHOS The Photon Spectrometer is a electromagnetic spectrometer, providing photon and neutral pion identification, and discriminating direct from decay photons. Its acceptance is restricted to the very mid rapidity region, and to 100 degree in azimuth.

EMCal The Electromagnetic Calorimeter completes the measurements of jet properties and enhances the capabilities of measuring high p_T photons, neutral hadrons, and electrons.

HMPID The High Momentum Particle Identification Detector mainly provides $\pi^{\pm}/K^{\pm}$ and $K^{\pm}/\left(p,\bar{p}\right)$ discrimination up to 3 GeV/c and 5 GeV/c respectively.

MUON The MUON detector provides muon identification. Due to the needed absorber it resides exclusively in the forward pseudo rapidity region. Like electrons, muons are part of the electromagnetic decay channels of quarkonia, which are important observables for QGP.

T0 The TZERO detector is a relatively small detector near the beam pipe. At opposite positions and different distances of the interaction point arrays of Cherenkov counters measure the collision time, which is needed by the TOF as reference, the z position of the interaction and provide the wake-up signal for the TRD, which comes prior to the L0 trigger (see also Chapter 4.1).

V0 The VZERO detector consists of plastic scintillator discs around the beam pipe. It provides a minimum bias trigger for the central barrel detectors based on the deposited energy and rejects background events for the MUON detector.

FMD The main purpose of the Forward Multiplicity Detector is to provide information about the charged particle multiplicity at pseudorapidities not covered by ITS.

PMD The Photon Multiplicity Detector measures multiplicity and the spatial distribution of photons, enabling the determination of the interaction plane, and study even-by-event fluctuations in the forward rapidity region.

ZDC The Zero Degree Calorimeter provides a centrality measurement by detecting the spectators and can be used as luminosity monitor.

2.5.2 ECS Control

Detectors must be steered, their state must be monitored and their output data must be saved. In ALICE each of these tasks is deployed by one separate subsystem. In order to direct all subsystems of the experiment there is a control entity called Experiment Control System (ECS). ECS integrates the operator interface and enables the operation of the experiment. It governs all subsystems:

- The Detector Control System (DCS) controls all supporting systems of the individual detectors, configures and monitors their Front End Electronics

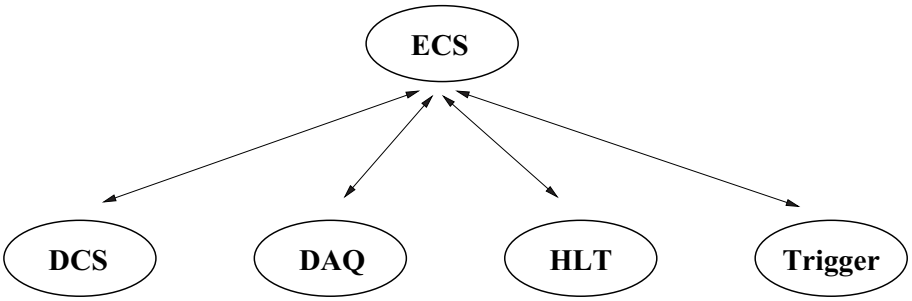


Figure 4: ALICE online systems are steered by ECS.

- Data Acquisition (DAQ) is the entity responsible for the data flow between the detectors and the mass storage
- The Trigger deploys the hardware triggering
- HLT is a software trigger system

Due to their importance to this theses DAQ, and the trigger system will be explained in more detail later.

2.5.3 ALICE-Software

In the ALICE Collaboration each sub-detector group is responsible for building and testing their detectors, mounting them into the experiment, and maintaining them. During operation detectors will generate data, which shall be analysed or reconstructed. The reconstruction algorithms must be in a working, well tested state when the experiment starts. This is only possible when all data output of the experiment can be simulated long before first collisions. This is especially challenging when taking into account the comparable low collision energies of previous experiments: e.g. RHIC has a peak energy of 500 GeV in pp and 200 GeV per nucleon in AA, which means that the models describing the expected characteristics of collisions at LHC are extrapolations with quite big uncertainties.

All reconstruction and simulation algorithms of the detector data are written in C++, and follow the concept of object oriented programming. The individual detector algorithms have a common interface and are concentrated into one package called AliRoot, which uses ROOT as its base. ROOT is a framework, which provides standardisation among different platforms (i386, AMD64, ppc ...), methods for many programming problems (mathematical, input-output, graphical...) and a C++ interpreter. Through this interpreter all methods within the compiled libraries of the AliRoot and ROOT framework can be called directly. Commands to the interpreter can be saved in so called macros. Processing usually takes place on the very machine on which the interpreter runs and uses only one processor.

The AliRoot package is being developed with the help of an online version control system, called Subversion (SVN). All changes are committed into the development repository[6] including a short description. The version control system saves not only

[6]under the SVN nomenclature the development repository is called *trunk*

the newest version, but also all necessary information for extracting previous versions. This is a very important feature as in such a big project as AliRoot it is indeed possible that multiple incompatible changes are committed almost simultaneously by different people. All changes since the beginning of the development can be viewed by help of an online interface[7]. By branching in the current development repository, SVN also provides the possibility of creating stable releases. There the update policy is restricted to bug fixes only.

2.5.4 HLT Software Trigger

The High Level Trigger is the third layer of trigger system incorporated into ALICE. In contrast to the lower two trigger levels, it is implemented in software. It is foreseen to deploy an online reconstruction of the data for the main detectors, and based on this, to perform online monitoring and calibration. This is to happen concurrently in the time it takes the DAQ to built the complete event, out of the fragments coming from individual detectors. Additionally, it decides on event level whether data should be saved by the DAQ on the mass storage, or to be dropped. The online reconstruction is performed on an computing cluster, situated in near proximity to the ALICE cavern.

This thesis describes the development of the software components, needed to perform the online reconstruction for the data coming from the TRD detector. Thus in the next chapter, TRD will be introduced, describing its purpose and its relevant features for this thesis. In Chapter 4 the ALICE trigger system is presented, concentrating on the HLT and its connections to DAQ and ECS. In Chapter 5 the development of the HLT software components is described, and the changes of the TRD offline reconstruction algorithms, which were needed to increase the processing performance of the code. Finally, in Chapter 6 a comparison of the quality of the resulting HLT TRD reconstruction and the TRD stand-alone offline reconstruction is made.

[7] http://alisoft.cern.ch/viewvc/trunk/?root=AliRoot

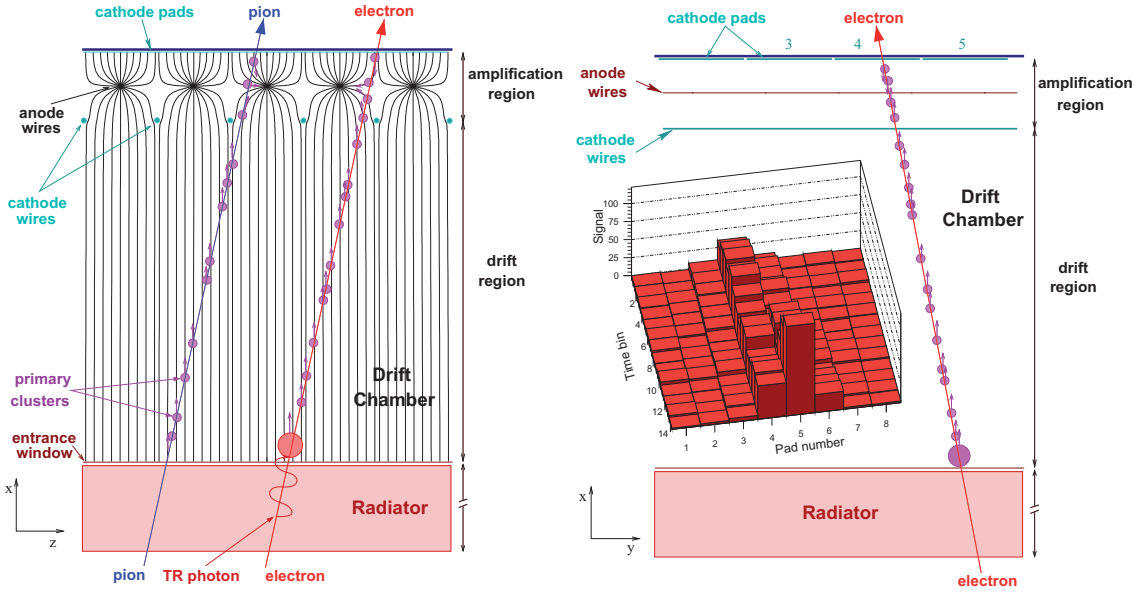


Figure 5: Cuts through a TRD readout chamber. The right image, additionally, shows the generated signals of pads as function of the time bins. [GSI]

3 The Transition Radiation Detector

The TRD is a gas detector, which serves mainly two independent purposes in ALICE: It completes the particle tracking between TPC and TOF and distinguishes between pions and electrons in momentum regimes where TPC and TOF are not efficient. The TRD is sub-sectioned into 18 supermodules, containing 5 stacks of 6 readout chambers each, totalling into 540 readout chambers (Figure 6). As a supermodule is a complete detector and has its own data output a very natural data parallelism is at hand.

As depicted in the Figure 5 readout chambers consist of a radiator of 48 mm thickness and a drift chamber of 37 mm thickness. At the entrance of the drift chamber there is the drift electrode, which is followed by the cathode and anode wires and finally the readout pads. By the applied high voltage between the drift electrode and the anode wires, ionisation electrons drift towards the amplification region. The cathode wires separate the drift from the amplification region and guarantee a homogeneous field in the drift region. Having a very short drift time the TRD is very fast and can be used for triggering purposes.

Each readout chamber has, depending on the size of the chamber, 1728 or 2304 readout pads which are organised in 144 columns and 12 or 16 rows. For decreasing the total number of pads, the pads are slightly tilted by 2 degree in the pad plane. The pads are thus parallelograms rather than rectangles with a typical size of $0.725 \times 8.5\,\mathrm{cm}^2$. As the pads of the chambers in a stack have alternating tilt, the total space resolution along the long side of the pads is improved. The total number of pads included in the whole TRD is 1.16 million covering an area of $736\,\mathrm{m}^2$. Table 4 summarises the main parameters of the TRD.

Parameter	Value
Pseudo-rapidity coverage	-0.9 $< \eta <$ 0.9
Azimuthal coverage	2π
Radial position	2.9 $<$ r $<$ 3.7 m
Length	Up to 7.0 m
Azimuthal segmentation	18-fold
Radial segmentation	Six layers
Longitudinal segmentation	5-fold
Total number of modules	540
Largest module	$117 \times 147\,\mathrm{cm}^2$
Active detector area	$736\,\mathrm{m}^2$
Radiator	Fibres/foam sandwich, 4.8 cm per layer
Radial detector thickness	$X/X_0 = 15\%$
Module segmentation in φ	144
Module segmentation in z	12-16
Typical pad size	$0.725 \times 8.5\,\mathrm{cm}^{-2} = 6.2\,\mathrm{cm}^2$
Number of pads	$1.16 \cdot 10^6$
Pad tilt	2°
Detector gas	Xe/CO_2 (85%/15%)
Gas volume	$27.2\,\mathrm{m}^3$
Depth of drift region	3 cm
Depth of amplification region	0.7 cm
Nominal magnetic field	0.5 Tesla
Drift field	0.7 kV/cm
Drift velocity	1.5 cm/μs
Longitudinal diffusion	$250\,\mu\mathrm{m}\,\mathrm{cm}^{-1/2}$
Transverse diffusion	$180\,\mu\mathrm{m}\,\mathrm{cm}^{-1/2}$
Lorentz angle	8°
Number of readout channels	1181952
Time samples in r (drift)	24-30
Number of used ADC channels	1378944
ADC	10 bit, 10 MHz
Pad occupancy for $dN_{ch}/d\eta = 8000$	34%
Space-point resolution at 1 GeV/c in rφ	600 μm for $dN_{ch}/d\eta = 8000$
Space-point resolution at 1 GeV/c in z	up to 2 mm
Momentum resolution	$^{\delta p}/_{p} = 0.025$ for $dN_{ch}/d\eta = 8000$
Pion suppression at $p \geq 3$ GeV/c	> 100 at 90% electron efficiency
Event size for $dN_{ch}/d\eta = 8000$	11 MB
Event size for pp	6 kB
Trigger rate limits for minimum-bias events	100 kHz
Trigger rate limits for pp	100 kHz

Table 4: Some parameters of the TRD. [PPR1]

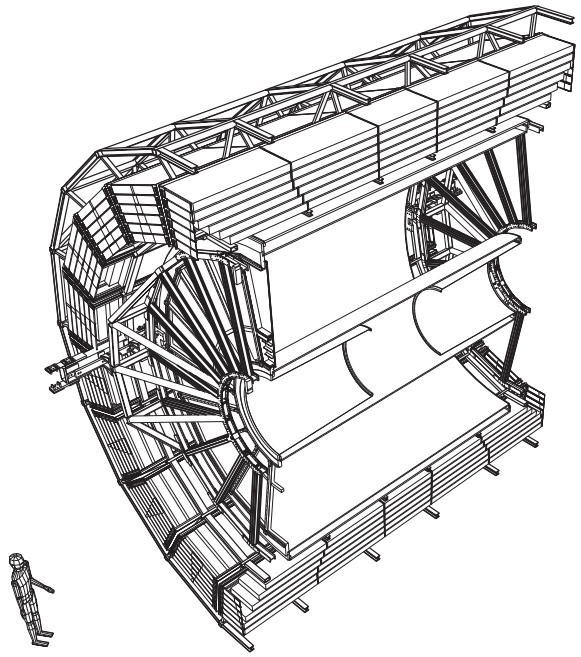


Figure 6: The TRD in its support structure. The ITS is situated in the middle, having the big TPC around. The five-fold segmentation of the TRD stacks, each containing 6 TRD readout chambers, can be clearly seen.

3.1 Working Principles of the TRD

As the TRD has two independent objectives there are two distinct working principles. We shall begin with the principle mainly used for the tracking capability of the TRD, which in the TPC is also extensively used for particle identification.

Gas Ionisation

High energetic charged particles flying through a gas lose energy by colliding with the gas molecules, ionising the gas along their path. The average amount of energy loss per unit length and thus the amount of ionisation electrons is dependent on the properties of the gas and the properties of the particle. It is given by the Bethe-Bloch Formula [SLI]:

$$\frac{dE}{dx} = -\frac{2\pi n e^4}{m_e c^2}\frac{z^2}{\beta^2}\left(\ln\frac{2m_e c^2\beta^2\gamma^2 E_M}{I^2} - 2\beta^2\right),$$

where m_e is the electron mass, n is the electron density of the gas and I is its effective ionization potential. z is the charge of the particle, β is its velocity and γ the Lorentz factor. E_M is the maximum energy transfer allowed in each interaction. In the non-relativistic region this formula is dominated by the term β^{-2} and thus the energy loss falls steeply. With further increasing velocity the energy loss stabilises and rises slightly in the relativistic regime until it saturates in the high relativistic region. Based on the amount of energy loss and combined with the momentum measurement particles can be identified. However, this method has an important limitation: due to the relativistic rise, particles cannot be distinguished in too high energy regions. This is especially problematic in the case of electrons and pions,

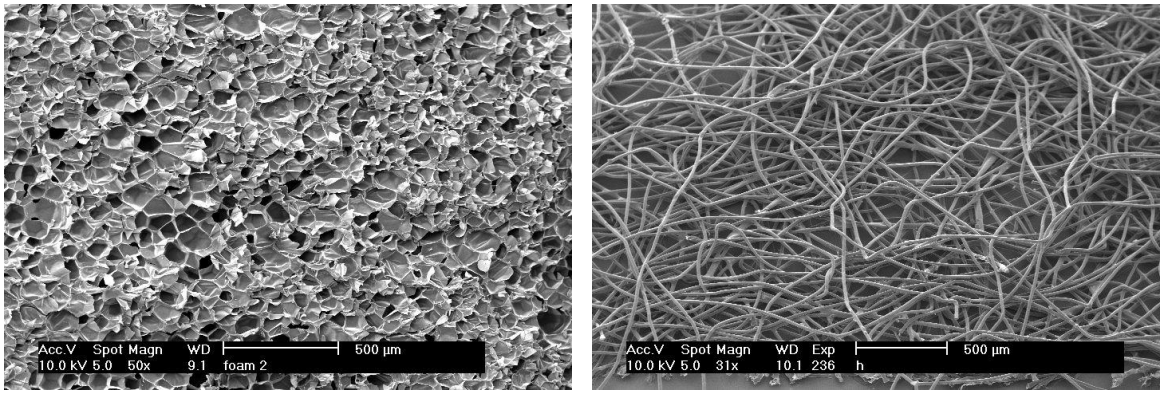


Figure 7: Scanning microscope images of the radiator materials: foam and fibre mats. [ATDR2]

since there are high pion fluxes and electrons are a major observable. Therefore the second working principle of the TRD is essential: the transition radiation.

Transition Radiation

The transition radiation was predicted by Ginzburg and Franz in 1946 [GAF] for non-relativistic particles and was first detected in 1959 by Goldsmith and Jelley [GAJ]. A particle traversing the border of two regions with different dielectric constants may emit transition radiation. The energy is given by the following expression

$$\frac{d^2W}{d\omega d\vartheta} = \frac{2\alpha\hbar\vartheta^3}{\pi}\left[\left(\frac{1}{\gamma^2}+\vartheta^2+\frac{\omega_1^2}{\omega^2}\right)^{-1}-\left(\frac{1}{\gamma^2}+\vartheta^2+\frac{\omega_2^2}{\omega^2}\right)^{-1}\right], \tag{1}$$

where ω is the frequency, ϑ the angle of emitted radiation, $\omega_{1,2}$ are the plasma frequencies of the two media and finally γ is the Lorenz factor of the particle. This equation has a maximum at an angle of $\vartheta = 1/\gamma$ and thus the radiation gets preferably emitted in forward direction. The total energy is given by integrating Equation 1 over all angles and photon frequencies to

$$W = \gamma\frac{\alpha\hbar}{3}\frac{(\omega_1-\omega_2)^2}{\omega_1+\omega_2}, \tag{2}$$

which is linearly dependent on the Lorenz factor of the particle. Hence ultra relativistic particles which have the same charge and the same momentum but different masses can be distinguished by this method.

However as the intensity is very low, the probability of a single emitted photon at one transition is of the order of the fine structure constant α, making many transitions necessary. The total probability for the emission of transition radiation (TR) inside a readout chamber is raised by the radiator, providing many transitions. It consists of a sandwich of two Rohacell HF71 foam sheets and mats of polypropylene fibres (Figure 7). These components were chosen as a compromise between TR yield, radiation thickness and mechanical stability.

Transition radiation from ultra-relativistic electrons is in the energy range of soft to hard X-rays ($< 50\,\mathrm{keV}$ [OBS]), which get absorbed by the gas, thus ionising it, when entering the drift chamber. Providing a high absorption probability is crucial

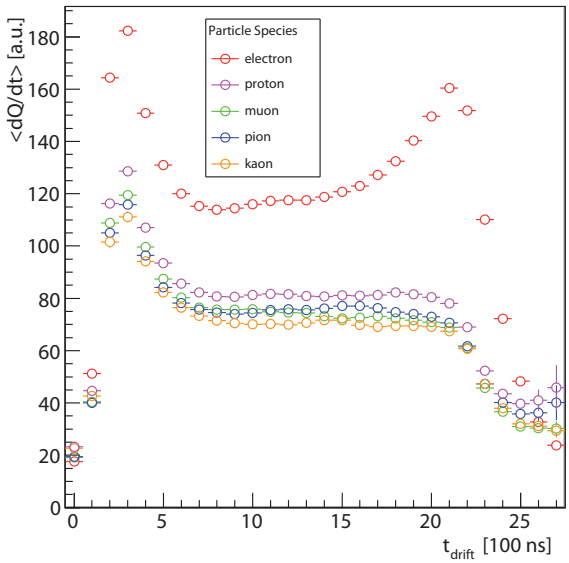


Figure 8: Average pulse height of different particles as reconstructed from simulated data. The electrons have a higher signal then other particles due to the increased energy loss in the gass. At the later time bins, and thus at the entrance of the readout chamber, the energy deposition of the TR photon is visible.

as only then the TR photon can be detected. As the photo effect is proportional to the atomic number of the gas atoms, the gas used in the TRD chambers consists mainly of Xenon which has an atomic number of 54.

Signal Generation

Free electrons from ionised atoms drift along the homogeneous electric field in the drift region towards the cathode wires. There they get accelerated in the highly inhomogeneous field of the anode wires, thus leading to an avalanche of more ionisation. The positive charge of the resulting ion cloud around the anode wire induces a negative charge in the readout pads, which is read out during the drift time of 3 µs with a sampling rate of 10 MHz[8]. Not all atoms ionise, some only get to an excited state which decays by emission of an ultra-violet photon. By the photo effect of these photons hitting metallic surfaces inside the readout chamber, an avalanche of released electrons producing more photons and ionising the gas would be initiated, leading to the breakdown of the chamber. As these UV photons can be efficiently absorbed by organic gases, the gas mixture used in the TRD consists to 85% of Xe and to 15% of CO_2.

The average pulse height of a readout pad due to different particles is shown in Figure 8. As electrons are by far the lightest charged particles, only these reach high enough γ factors for being able to generate considerable amounts of transition radiation.

[8]thus there are up to 30 time bins for each pad at each event

4 Triggers

Modern collider experiments search for improbable and thus rare events. For being able to still have enough statistics, high collision frequencies are needed. Saving continuously all data from all detectors is not an option for the ALICE experiment, since due to the high granularity, the detectors produce huge amounts of data while operating. Thus a trigger system must be implemented, to be able to activate the detectors and the data taking only when physical important events occur. This is especially important for discontinuously working detectors like the TPC, where e.g. the gating grid must be opened to make a measurement possible.

4.1 Hardware triggers (L0, L1, L2)

The Central Trigger Processor (CTP) combines and synchronises the trigger decisions from the triggering detectors, and sends the result to all detectors. Some detectors are continuously operational, some need a very early activation, other can handle longer delays. Thus the triggering system is divided in multiple stages. The first trigger decision, called L0, must be delivered 1.2 µs after the collision took place. For that the triggering detectors contributing to this decision must hand in their decision already 800 ns after the collision. Only very fast detectors can achieve that. Slower trigger detectors, like the TRD, may have a delay up to 6.1 µs and contribute to the L1 trigger signal, which is then delivered after 6.5 µs. After 88 µs the last trigger signal (L2) is delivered. It is used as a past-future protection, to veto against previous L1 triggers. As this is the drift time of the TPC, it cannot handle too many consecutive events within this time in pp collisions. Due to the sheer amount of tracks, pile-ups, i.e. multiple events within the TPC, cannot be reconstructed at all for PbPb collisions. Thus in PbPb collisions any additional L1 trigger within the L2 time would lead to a L2 reject. The data taking of the central detectors is enabled only by the L2 accept. [ATDR1]

4.2 High Level Trigger

Besides the described hardware triggers, which start detectors and data taking, ALICE has foreseen also a third trigger level. The so called High Level Trigger will be used for online calibration, monitoring of the incoming data quality, and as tagging or rejection trigger. For this purpose a fast real time analysis is being performed for the main detectors. Opposing the low level hardware implementation of the other triggers and since it is not constrained to a fixed time budget, this online analysis is implemented in software written in the high level language C++. The online analysis needs an online reconstruction at its base, which also used for the online monitoring and near-time[9] calibration of the detectors. In this context offline processes are executed after events are written to the data storage and online processes are executed during the data taking and thus before individual events get saved. With the help of this online analysis another problematic issue of ALICE can be resolved; the data rate of the detectors can be as high as 25 GB/s in lead-lead

[9] meaning run by run

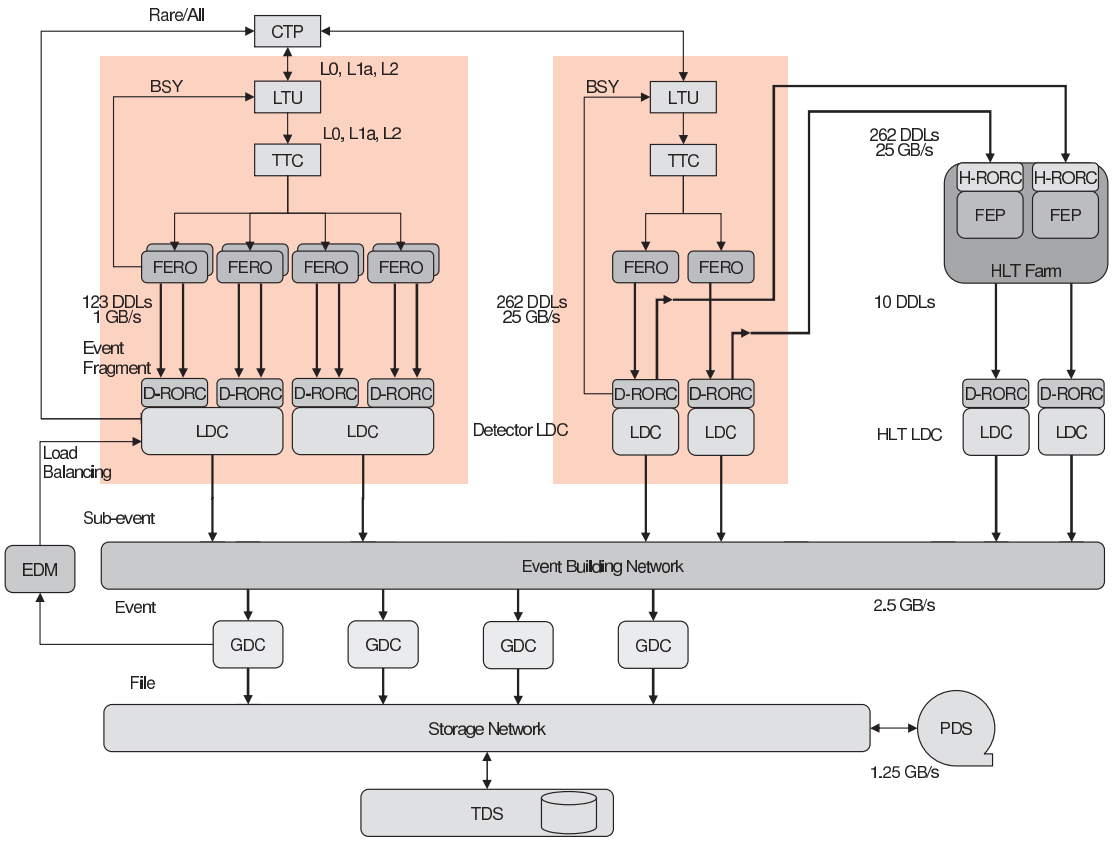


Figure 9: The DAQ principle overview showing the data flow of two detectors, one being connected to the HLT (the right red highlighted area). The Event Building Network connects the Local Data Concentrators, which buffer the incoming sub-events, and the Global Data Concentrators, which build the events alternatively. The Storage Network includes the Transient Data Storage in the proximity of ALICE and the Permanent Data Storage. [ATDR1]

collisions, out of which 2 GB/s are from the TRD. The tapes however, where the data shall eventually be saved, can take only 1.25 GB/s. This is the reason why the rejection trigger is needed. The HLT can use the reconstructed events to reduce the data rate dramatically (up to a factor 10 [ATDR1]) by selecting interesting events or sub-events and rejecting background events.

Although this thesis concentrates on the reconstruction, the possibility of using the HLT as big "compressing machine" shall also be mentioned. In this mode the HLT would just binary-compress the input data. However, this means that neither monitoring nor near-time calibration nor triggering could be performed upon data taking. Hence this mode would address the problem of decreasing the data volume only. Additionally, tests of many different compressions algorithms have shown that compression factors of only about two would be achievable [TPC, LOS].

4.2.1 HLT-DAQ interface

For operation, the HLT of course needs the detector data, but it also needs calibration values and some of the permanently monitored values of the detector survey.

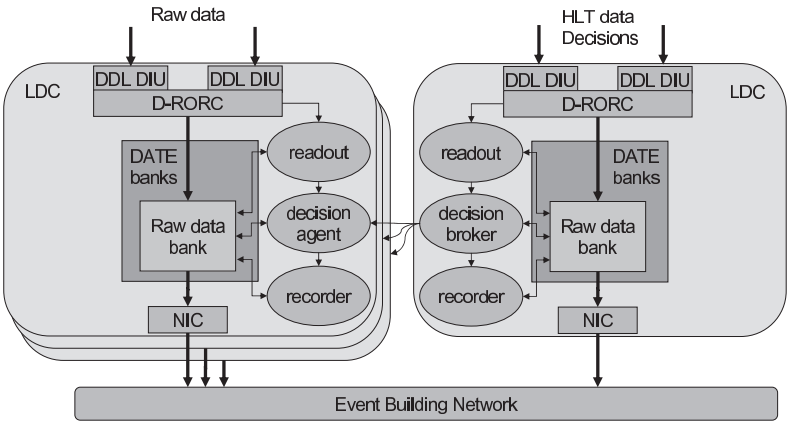


Figure 10: Detector LDCs (left) processing the HLT triggering decisions in mode C. [ATDR1]

Thus HLT is integrated by interfaces to the other ALICE sub-systems.

The interface between DAQ and HLT is required by the data exchange to and from the HLT. As shown in Figure 9 the DAQ is connected to the HLT and to all detectors via optical connections called Detector Data Links (DDL) and integrates additionally a two level network dedicated to the *Event Building*, which is connected to the *Mass Storage*. As the HLT receives and sends data from and to the DAQ, it has separate inbound and outbound connections. The DAQ has three modes of operation which determine the level of activity of the HLT [ATDR1]:

DAQ only - HLT disabled a.k.a. mode A. This is the starting point for the DAQ operation and the DAQ returns to this mode when the HLT is disabled.

DAQ + HLT analysis a.k.a. mode B. The HLT is fully functional, but its trigger decisions are discarded. The purpose of this mode is to give the possibility of checking the HLT algorithms.

DAQ + HLT enabled a.k.a. mode C. The DAQ performs an event and sub-event selection based on the trigger information from the HLT.

In the last two modes DAQ sends a copy of the detector data to HLT. There it is processed and the result is returned back to DAQ. The HLT's result might be compressed data, or trigger decisions with additional data to be added to the event. What happens during that "processing" for the data coming from the TRD is the topic of this thesis and will be discussed in Chapter 5. In case of a negative trigger decision DAQ dismisses the event, whereas in case of a positive trigger decision it further processes the event by also adding the additional HLT data to it. This additional data contains the needed information to have a pre-organisation of the data, which may be used during offline reconstruction or analysis. Figure 10 shows how the triggering decision of the HLT is processed by the LDCs. The event built is then being sent to the *Storage Network*, where it is finally saved. On this saved data offline reconstruction is being performed later on.

4.2.2 Event Processing

A high degree of parallelism is needed, in order to achieve the reconstruction throughput needed to process all detector data. The level at which parallelism is applied in the data processing implies different computing architectures, which lead to different processing circumstances. The computing architecture must be chosen thoroughly in the first step, as it sets the base of the later total computing design and cannot be changed later on.

Sequential Event Processing

Sequential event processing is the most straight forward way of organising the computing design. Whole events are distributed among many machines, where individual events are reconstructed by one process only. Here parallelism is applied at the event level, thus the process has access to all the information of the given event. This however comes at the cost of high data transfer rates onto the individual machines, as the whole events have to be transmitted. Also this means that the whole event must have already been taken in the first place. Considering the event sizes and rates of PbPb events, which sum up to 25 GB/s, this is a strong limitation, which makes online processing impossible with such a design. The ALICE offline event reconstruction makes use of this processing method, which is implemented in the so called computing GRID [ALE].

Parallel Sub-Event Processing

In this approach each machine gets a fraction of the whole event, processes it and hands it further to the next processing step. Thus the bandwidth limitation is circumvented at the cost that each processing step does not have access to all the information of the event. Each process must be independent of the others which are running in parallel. This parallelism however is naturally given by the segmentation of the detectors: as mentioned in Chapter 3 the TRD is composed of 18 independent supermodules. But not only the TRD also the other barrel detectors have high degrees of segmentation. Consequentially the HLT is designed based on this parallelisation method.

4.2.3 Physical Layout

The HLT is a high performance computing cluster with 171 nodes and 1368[10] processors on which the reconstruction is processed in parallel. The nodes have mainstream server hardware and use a mainstream Linux distribution as operating system. The nodes are organised as Front End Processing nodes (FEP) and Computing Nodes (CN). The FEPs are connected to the DAQ DDLs and have custom PCI adapter cards (H-RORC), which receive the data from the DDL and transfer it directly to a reserved area of the machines memory. All nodes are connected through Gigabit LAN. The data is first being processed on the FEPs and then it is further processed on the CNs. As the FEPs are physically connected to specific DDLs they process data from those links only. Whereas the CNs are interchangeable, depending on

[10] these are the number for the pp runs in 2010 [TAL]

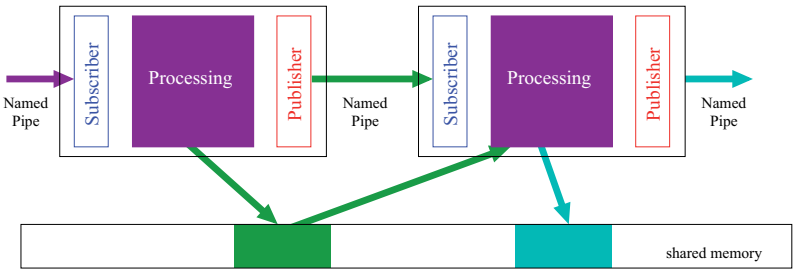


Figure 11: Data exchange between components. After being informed by the publishing component through the named pipe, the subscribing component accesses the data of the publishing component directly on whose shared memory block. [MRD]

the needs. In case of the TRD, 5 FEPs and currently as many CNs can be used. However, the number of CN nodes assigned to the TRD is not a constant and can be changed on short notice. The nodes have 8 CPU cores and FEPs are connected to 2 or 4 DDLs, each coming from one supermodule. The trigger result of the HLT is routed via dedicated nodes, which are connected to the outgoing DDLs.

All following statements concerning speed will be based on a 2 GHz AMD Opteron 2212 processor, if not indicated differently. This was the typical processor in the HLT cluster in the late 2009 runs.

4.2.4 Data Transportation Framework

At each event the response of the HLT to the DAQ is based on the merged outcome of the reconstructed data from all nodes. The nodes must be controlled to behave as one unit, to know what to do with the data coming and where to send the processed data to. This is done by a Publisher-Subscriber Framework, which was specially developed for the ALICE HLT. It takes control of the nodes, starts the necessary jobs for data processing and provides the communication between the nodes. To avoid copying between nodes, the framework also provides a shared memory mechanism (see also Chapter 8.1).

Special interfaces to effective code are implemented, giving the HLT framework the possibility to directly call procedures without the ROOT interpreter. The interfaces are named components, and are in fact very much like executables, which, however, can be used by the framework only.

The framework's name indicates the way components are communicating. The principle is shown in Figure 11. A component which needs data from other components, subscribes to those, which in turn, when data is available, publish the data to all their subscribers. The subscribers do not get the data right away, they get a descriptor of the data, enabling them to decide whether this particular data is of interest or not. If a subscriber decides to use the data, it will read it from the shared memory block, where the publisher has written it to. When finished, the subscriber tells the publisher it may reuse the shared memory block. By this way of communication, only really needed data has to be transferred from one node to the other. [TSD]

Each node buffers its incoming data for being able to be kept busy. If the node is not fast enough and the buffer of a node is filled, the node produces back-pressure

to its preceding nodes. When all buffers of nodes in a chain are filled a BUSY signal from the H-RORC is transmitted to the DAQ, which halts the detector. [ATDR3]

4.2.5 Modular Data Processing

In the HLT processing tasks are carried out by individual processes, which are called HLT components. There are three types of components: *source*, *processing* and *sink*. Components are organized in a chain, which defines the layout of the total data analysis procedure. Source components are used to publish the input data into the processing chain, processing components do calculations on that data, and finally sink components are the endpoint of the analysis, where the output data is saved locally or gets sent to HLT external receivers. Possible receivers are the DAQ via the output DDLs or the monitoring machines via a network connection.

Chains are set up by means of XML configuration files, which are read out by the HLT framework during the initialisation of the run. The components reside in compiled libraries, but are to the framework like command line executables, as during their start up arguments can be handed in, to control some predefined parameters. This modularity is source of great flexibility enabling to change the whole analysis procedure without recompilation, by just altering the configuration files.

4.2.6 ECS-HLT Interface

As the whole experiment is steered by ECS, HLT also includes an interface for communication with this system. From ECS' perspective the HLT is a fivefold state machine, whose state transitions are initiated by ECS. The states and the transitions are shown in Figure 12. The procedure of starting a run decomposes in these steps:

- Initializing: First contact of ECS to HLT
- Configuring: XML configuration files are read, containing the information on what node to place which component
- Engaging: ECS run parameters are received, components are started and set up
- Starting: Data taking is started, by engaging data flow components

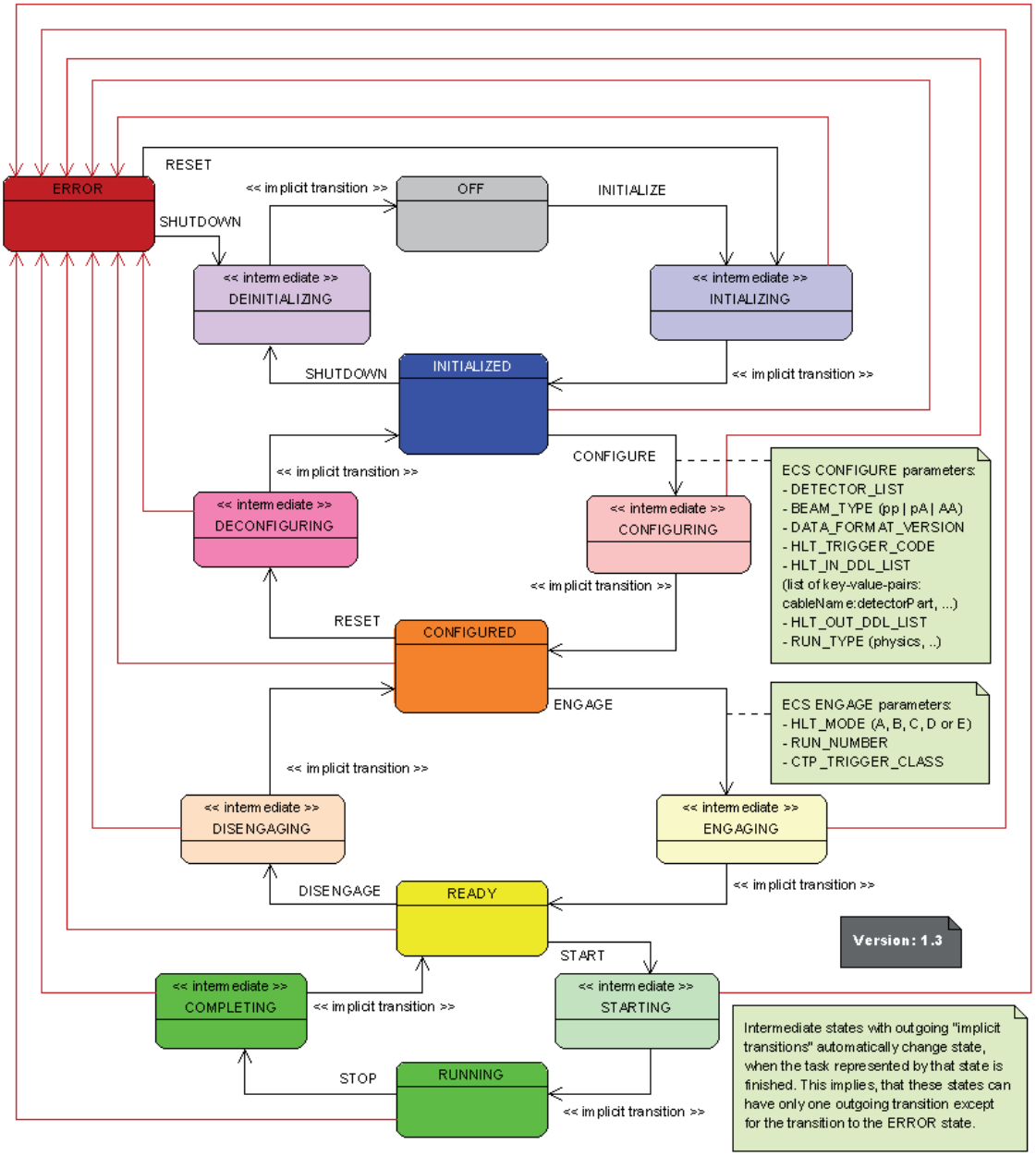


Figure 12: State diagram of the HLT, as seen by the ECS. [AHI]

5 The Development of HLT::TRD

The topic of this thesis is to develop an implementation of the online reconstruction for the TRD data suitable for running on the HLT, including online monitoring and calibration. Under the constraint of sharing with the offline code at least the most important algorithms, the development split in following parts: the code needed to interface the offline algorithms to the HLT Framework was implemented. This was done in multiple steps, the first implementation established a working code base, which was then refined. At the same time a review of the offline algorithms under the aspect of processing performance was undertaken. Based on this, those parts of the code that could be accelerated without quality loss were re-implemented. Where this was not possible, the quality of the reconstruction had to be cut. This was done by skipping or simplifying calculation steps during the online reconstruction. However, the output quality has to meet some quality standards, thus procedures of quality assurance were used.

As mentioned in Chapter 2.5.3 each sub-detector group is responsible for the implementation of the algorithms concerning its detector. This is also true for the HLT online algorithms. In case of the TRD the decision was taken, that the online reconstruction should use as physical relevant algorithms those of the offline reconstruction. This was a unique decision, the other detectors decided to have two completely separate implementations. The upside in having only one code is that this code is already tested and there is no need to maintain two different codes. Using one code under different circumstance also helps discovering malfunctions, which would remain unseen otherwise. However there is a big drawback. The offline reconstruction is used for processing the raw data which has already been saved in the data storage, i.e. data which has passed the DAQ until the end. This is preferably done with the highest available precision. Whereas the online reconstruction is being executed before the data potentially gets saved, as the HLT is choosing whether to actually really save the data or not (see also Chapter 4.2). This is done during the data taking and must thus follow its speed.

In Chapter 5.1 an introduction to profiling is given, as this is the basis for fast software. The software developed must be tested and checked for bugs and malfunctions before being executed on the HLT cluster, this is done on the test server, Chapter 5.2. The results of the code review and the challenges to increase the processing performance of the offline code are presented in Chapter 5.3. The implementation of the code interfacing the offline algorithms to the HLT framework is described in Chapter 5.4. Then, in Chapter 6, the reached output quality of the code in its online mode will be compared to the quality of the offline mode, showing whether and if so, how much the trade-offs degraded the reconstruction quality.

5.1 About Profiling

Developing fast software demands for knowing and understanding what happens inside the computer and especially inside the processor during run time. Compared to directly writing machine code or assembler, using high level object oriented programming languages as C++ eases the development itself as the software can be

	Sampling Profiler	Instrumenting Profiler
Overhead	usually less than 1 %	up to 500%
System wide	Yes (OS, drivers, apps, libs)	No (just applications and libraries)
Collected data	hardware counters	call graph, software counters
Data taking	every N clock cycles	continuously
Environment	effects result	has no effect
Accuracy	statistical and systematic errors	perfectly reproducible

Table 5: Some typical characteristics of sampling and instrumenting profilers. The accuracy of the collected data of sampling profilers includes statistical errors (due to the sampling) and systematic errors (due to the possible influence of the environment on the running programme).

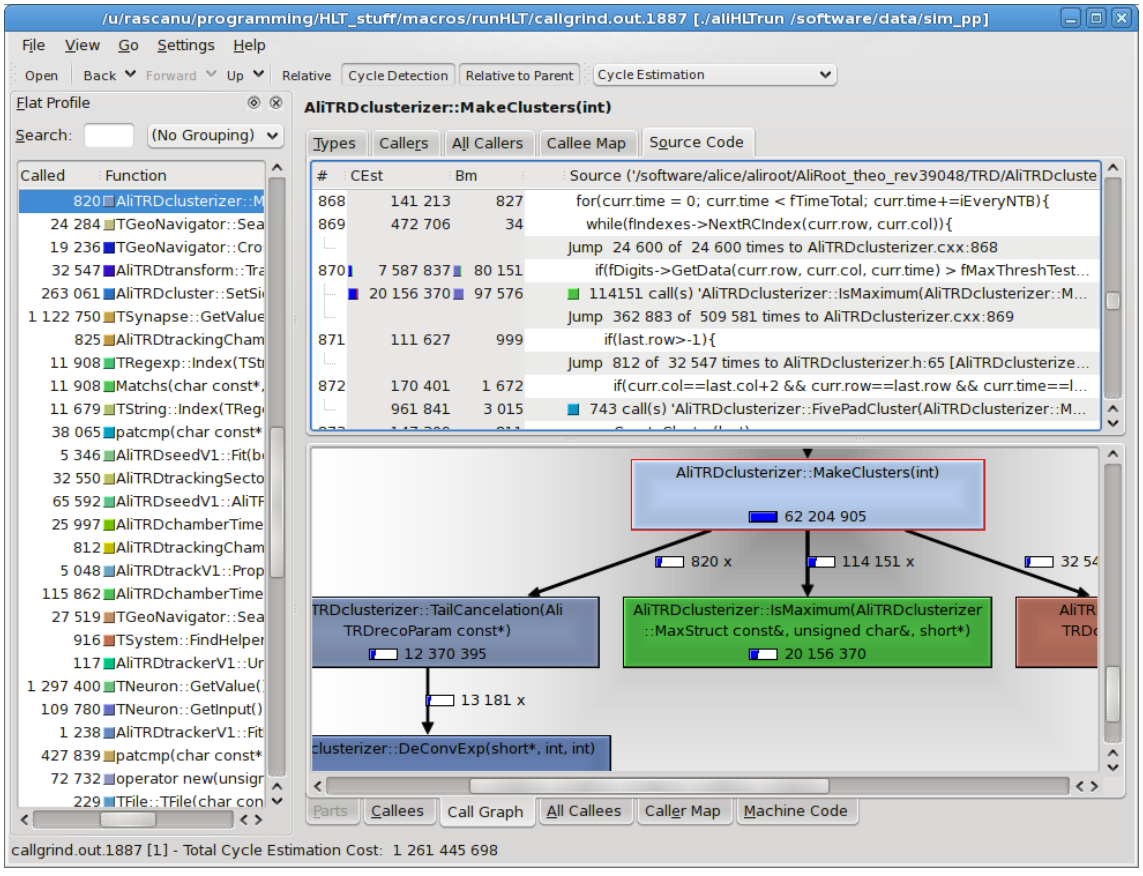


Figure 13: A screenshot of KCachegrind. KCachegrind is a KDE tool used for displaying the profiles taken by the Valgrind profiler. On the left is the function overview with the activated function marked blue (in this case: AliTRDclusterizer::MakeClusters(int)). Top shows the source code of the activated function (here we see the core of the clusterizer) with the estimated number of cycles (CEst) and the count of mispredicted branches (Bm). Bottom is the call-graph, showing the callers of the activated function and the functions called by the activated function.

structured much better. Thus complicated procedures can be hidden behind simple interfaces. This however also hides what happens inside the processor, as it is not obvious any more what instructions are behind a code line[11]. By performance analysis, a.k.a. profiling, this knowledge can be regained. In fact modern profiling tools can do much more than just instruction count.

There are various profiling tools available, which can be classified by their working principle: sampling and/or instrumenting profilers. Some are operating system dependent (like the profiler integrated into the MS Visual Studio), some are processor dependent (like Intel VTune or AMD CodeAnalyst), some are costly and some are free of charge (like gprof, the Valgrind suite or oprofile). Advantages and disadvantages of the working principles are summarised in Table 5. Based on the following requirements the Valgrind suite was chosen as the main profiling tool:

- Easy and fast overview of the collected data
- Detailed line-by-line analysis
- Processor independent
- Running on Linux
- Free of charge

Valgrind is basically a virtual machine in which the programme is run. It simulates the processor and its cache hierarchy and has thus a detailed and very accurate view on what is being processed. Based on this virtual machine there are different highly valuable sub-tools for different objectives apart from the required profiling feature: finding bugs, memory leaks and monitoring memory usage.

The profiler produces a so called call-graph showing functions and function calls (Figure 13). Additionally it counts the number of instructions, L1 and L2 cache misses including the write back of dirty cache lines, branch predictions and its misses and calculates an estimation of the CPU cycle count based on that information. Running programmes in a simulated environment however, of course means that the execution is slowed down significantly, thus making it unusable for real time applications. Additionally Valgrind needs an executable of the programme, which means that it cannot be used to profile already running software, like the operating system itself. Inside the HLT cluster the software is run by the HLT framework. This does not provide a reasonable entry point to the algorithms and, on top of that, the whole processing is distributed among many machines. Thus Valgrind cannot be used to profile the software, while it is being executed on the HLT cluster. For this short-cut interfaces have been implemented to create executables, which were then profiled on a test computer.

Before any real performance tests can be run, simulated data samples must be generated. These data samples should be as realistic as possible, especially when

[11] By no means should this be interpreted as saying that object oriented programmed software is per se slower than assembler programmed. Just using assembler does not necessarily make the end product fast. It is very laborious to develop even small software packages using low level programming and on top it is usually very poorly portable.

it comes to the number of particles per event, their momentum distribution, and noise. When this is settled the first speed test must be run on the HLT cluster, to establish the current total speed of the code. Then a detailed analysis of each called method, preferably for each code line or even for each instruction, should be prepared. Based on this bottle necks can be tracked down. When fast software shall be developed, profiling is part of the development cycle. Each new functionality and each change should be profiled, for making sure that these changes do not interact with the rest in an undesirable way. However before the emphasis is laid on the speed, the algorithm must be working reliably. For being sure not to develop a poorly performing algorithm the data layout and the principle speed should be still estimated, making the development a tightrope walk. Of course not only the speed must be checked but also the quality of the output must be monitored. This is explained in detail in Chapter 6.

5.2 Test Setup

Developing software in a big project such as ALICE demands for reasonable coding standards. Before committing code to the repository, checks concerning compatibility, stability and quality are necessary. For producing high performance code these checks must be extended, including also profiling. The development cycle of HLT components, as shown in Figure 14, has three stages:

- Experimental stage
- Tested stage
- Commissioned stage

After developing a new component or introducing major changes to existing components, the respective components are classified as *experimental*. Only after stand-alone stability tests the components can be assigned *tested*. However, before a component can really be used in the online reconstruction it must pass two additional tests: running inside the whole reconstruction chain while being managed by the HLT Publisher-Subscriber Framework in its stand-alone mode, and, this is the last test, while the HLT framework is itself managed by ECS. Components passing also those tests are *commissioned*.

The first stability tests can be carried out using macros inside the ROOT C++ interpreter with AliRoot libraries loaded. These macros are calling the components as if those were operating inside the HLT cluster. For being able to trace bugs efficiently and for profiling it is however much more useful to have compiled versions of these macros. This was done by the help of the build system *cmake*, which is very well suited for this purpose. Additionally such executables can be very comfortably called on the command line, with command line arguments changing the behaviour. By the use of a debugger (GDB in this case), and the Valgrind suite most bugs can already be found at this stage. Few bugs, however, show up only when the components are executed by the HLT Publisher-Subscriber Framework. This is due to the fact that macros only try to simulate what really happens during an online reconstruction of HLT. More specifically, components interfaced by a compiled macro

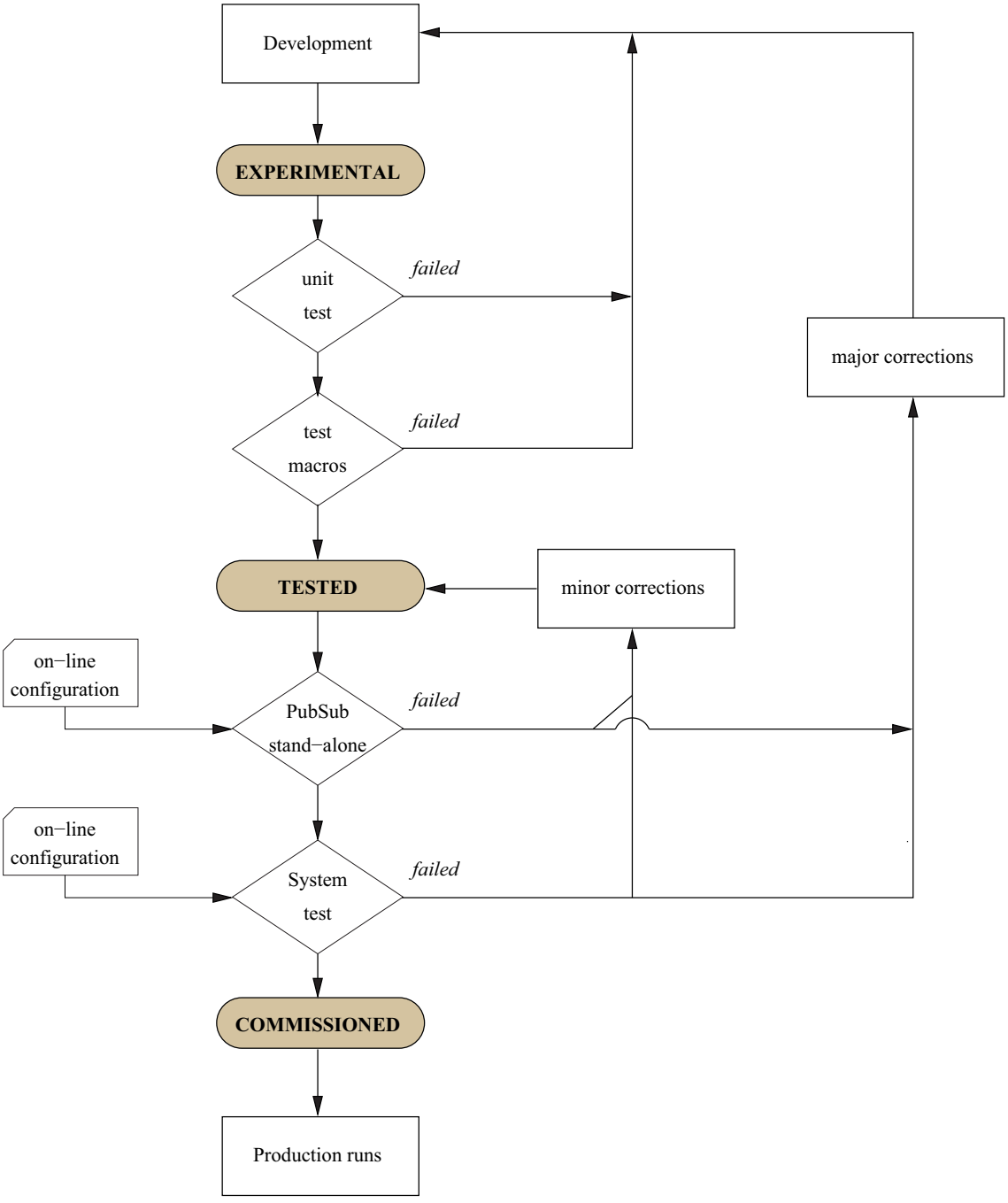


Figure 14: Development cycle of HLT components. [MRD]

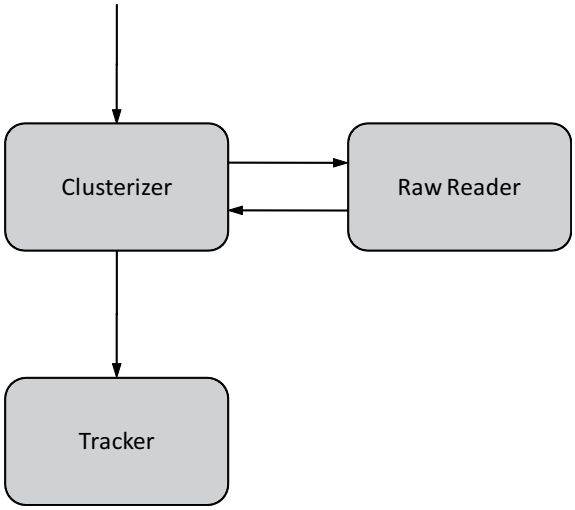


Figure 15: Dataflow between Cluster Finder (a.k.a Clusterizer), Raw Reader and Track Finder (a.k.a Tracker).

are all inside the same *virtual address space*[12], whereas components called by the HLT framework are executed each by its own, and are thus in separate address spaces. On the HLT development computer at Frankfurt a test setup is installed, such that the HLT framework can be deployed just like on the HLT cluster. By that also the first test towards the commissioned status can be accomplished locally. In case of malfunctions this eases the debugging a lot.

For stability reasons, the HLT cluster operates with the latest AliRoot release, not with the experimental SVN trunk. As the HLT TRD components are using offline code, the library containing these components must be linked against the offline libraries containing the offline procedures. This means that development of HLT components must not only be compatible to the trunk but also to the last release. This backward compatibility can only be guaranteed by an own build system of the HLT software. This build system (cmake) checks for the available offline code functionality and compiles the HLT components accordingly. Thus bigger changes of the components usually also ask for updates of the build system.

For testing a new development on different versions of the AliRoot code, all these versions must be available and quickly exchangeable. Therefore a suite of a few command line scripts has been developed for efficiently maintaining several AliRoot code bases with multiple compilations each.

5.3 Offline Reconstruction

The TRD offline algorithms were reviewed under the aspect of processing performance with the goal of meeting the required speed, which is given by the running experiment. For the TRD this means getting data from the 1.16 million readout pads at 30 time bins at an event rate of 2000 Hz[13] in pp. However, first tests showed that only about 40 Hz were reached by the offline code. This resulted not only due

[12] for an explication of the virtual address space see Chapter 8.1

[13] as mentioned in Chapter 4.2.3 these speed indications refer to an AMD Opteron 2212 processor

to the speed of the offline code itself, but also to the data containers used by the offline code. Unifying needs of both objectives, maximal output quality for offline and enough speed for online, into one code without any form of branching means to have the code running at the required speed with the maximum precision. As this was not always possible precision-to-speed trade-offs during the execution of the online reconstruction are necessary. What these trade-offs consist of will be explained in the context of the given processing step.

To get a rough understanding of what is being done for reconstructing data from the TRD detectors and to establish some of the vocabularies used, in the following the necessary processing steps are introduced. Then each processing step is presented in more detail and the implications on the review are shown.

The offline reconstruction in the TRD consists of the following steps:

- Reading and decoding the incoming data
- Searching the data for signal maxima, this step is called cluster finding
- Connection found clusters to tracks, called track finding

These general steps also apply to what is done during the HLT online processing, and also for most of the barrel detectors in ALICE. The first step is in fact the beginning of any real data processing, not just for the ALICE detectors. In case of the TRD it is performed by the *raw reader*.

For accomplishing the second step, the data is searched for maxima of the pads signals. The data is processed in a geometrical way: for each time bin there is in principle a two dimensional array, which represents the readout pads in their plane with their individual signal level as data. In this plane maxima of signals are searched for. These maxima are called clusters, and are comparable to the bubbles in the old bubble chambers. The processing is carried out by the *cluster finder*.

The third step is needed to find clusters that can be connected to continuous lines. Those lines are called tracks and should ideally be where the real traces of the particles were. This processing is performed by the *track finder*. Figure Figure 15 shows the data flow between the three processing steps during the TRD reconstruction.

Offline reconstructions are set up and performed calling methods of the AliRoot framework directly through the C++ interpreter of the ROOT framework. The reconstruction code is steered by the so called *reconstructor*, which also includes the information about the configuration used. Thus under different environments the reconstruction can run with different parameters, like it needs to be done to change from pp to PbPb reconstructions, as an example.

In the following chapters the main reconstruction classes are presented. After an introduction, the changes required to increase the processing performance are described in “Implementation Changes” and the resulting performance is summarised in “Performance Results”.

5.3.1 Raw Reader

The raw reader is used in TRD reconstruction to read the detectors data, decode it and save the result in memory. To decrease the needed bandwidth the data is

zero suppressed meaning that only signals above a certain threshold are transmitted by the detector electronics to the DAQ. Additionally, the information of three time bins is compressed into one binary *double word* (32 bits). Reading this data directly would result in unaligned memory access, which makes the data not easily accessible. Hence it must be converted into aligned data bundled into arrays (see also Chapter 8.1). If decoding of the data was not necessary, it would be possible to only have to save the pointer of the data and directly access it through that pointer.

The raw reader gets the starting point in the raw data stream during the initialisation. This is done once for each supermodule by the cluster finder, a.k.a. clusterizer. Then follows a call from the clusterizer for each of the 30 readout chambers of the supermodule. At each such call the raw reader reads the headers of the raw data, makes consistency checks based on this information, decodes the data and saves it in a three dimensional array, which is then passed to the clusterizer to process. The dimensions of this array are the number of rows and columns of readout pads in the readout chamber and the number of time bins. Those are the natural coordinates to address each point of the readout chamber given by the geometry. Later in the reconstruction the time coordinate, also called drift time, is translated to the corresponding drift length. Additionally to the three dimensional array another two dimensional helper index array is filled. This saves the information which pads have some signal at at least one time bin. The purpose of this array is explained in Chapter 5.3.2.

Implementation Changes

First speed tests were made with simulated raw data as real data from the detector was not yet available. Additionally, at that time simulated and real raw data had different formats. The different formats resulted into different raw readers that were used to decode the data, and thus one of the basic principles of profiling had to be ignored; use as realistic data as possible. This made it possible to hide a fundamental problem of that specific real data reader.

It turned out that the raw reader for real data had a conceptual drawback in terms of processing performance. As it was very much devoted to quality assurance (QA) it eased the access to statistics of the reading process, so called error containers. The problem was that the raw reader was thought to decode the raw data of the whole detector, all 540 chambers, and populate a huge (almost 100 MB) substructured array at first call. At later calls it read out of this array chamber-wise. Data was copied twice: once from the raw stream to the raw reader's memory and once from this memory to the clusterizer's memory.

Thus a new reader had to be implemented, which would decode only the needed readout chamber at each call, by preserving the QA possibilities. This was done by the according people assigned to this class.

Another measure for speeding up the reconstruction was the implementation of the index array. This array helps the cluster finder while searching for maxima. In this array the raw reader indexes each pad which has a signal above zero at some time bin. This is explained in detail in the next chapter, but as this helper array is filled by the raw reader is should be mentioned here also.

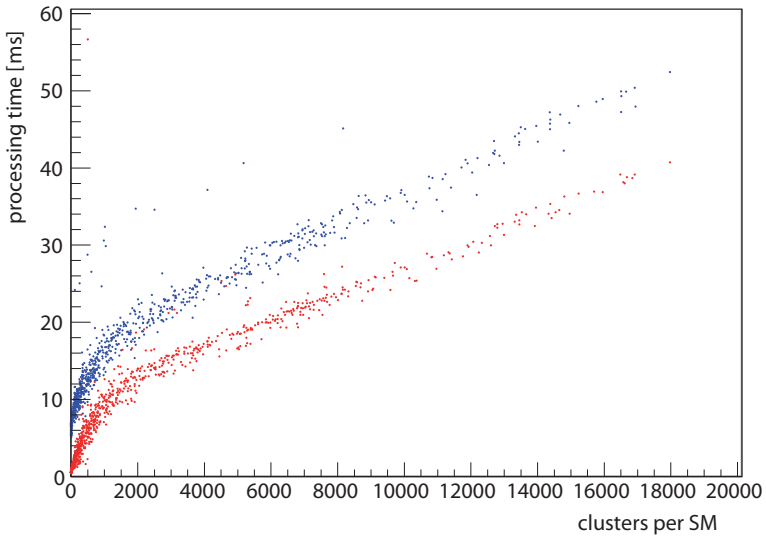


Figure 16: Processing time of the clusterizer for one supermodule. Shown in blue is the initial situation, in red the resulting performance due to changes of the raw reader. The cause of the bending will be explained in the next chapter.

Performance Results

Figure 16 shows a measurement of the processing time of the clusterizer for one supermodule at different event sizes. The inefficiency of the initial implementation of the raw reader is visible at the intercept of the blue points. This is the result of decoding and saving the whole incoming data at first call in a temporary array. The red points, which are the measurement of the more efficient raw reader, are not only shifted by the height of the intercept of the blue points, but also the inclination is slightly reduced. This is due to the fact the that even if the old reader only had to read out of the data array without decoding, the array layout was such that delays occurred. The optimized reader decodes the data just in time and hands the output directly to the temporary array of the clusterizer.

However, the general look of the curve is basically unchanged, as it is dominated by the clusterizer.

5.3.2 Cluster Finder

The main task of the cluster finder, also called clusterizer, is to look for maxima in the decoded data signals. The search is done for each time bin independently. A valid maximum must meet the following criteria:

- the pad at which the maximum is searched for must have a signal above a certain threshold
- its two neighbours of the same row must have smaller signals[14]
- the sum of the signals from those three pads, which is the charge of the cluster, must be above a threshold

[14]Note, neighbouring rows are of no interest, since the pads are far from being quadratic and thus there is barely an influence to neighbours of the same column.

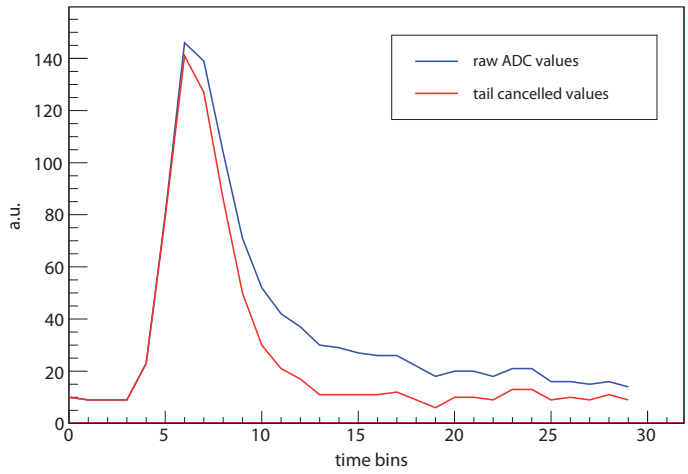


Figure 17: Tail cancellation of the signal of a readout pad.

The embedded subfigure in Figure 5 shows maxima of given pads at different time bins.

In case of the offline reconstruction the clusterizer is called by the *reconstructor*, given the raw data memory block. This memory block is being decoded by setting the raw reader to the beginning of the memory. At the following calls from the clusterizer the reader returns the three dimensional array (compare previous chapter) containing the decoded signals of each pad at each time bin and the index array.

As detectors do not respond ideally, all the data is convoluted with the detectors response function. For the TRD the total effect is decomposed in its row[14] component, called pad response function, and its time component, called time response function. As especially the time effect is very perturbing for the correct operation of the clusterizer, the first thing the clusterizer does after getting the data from the raw reader, is to deconvolute the time response function from the signal. This process is called tail cancellation (Figure 17).

Now the data is ready to be searched for the maxima. For processing the data adequately the clusterizer needs additional information during the maximum finding. The gain of the readout pads is especially important, since if two neighbouring pads have different gain factors (i.e. having the same input level they output signals of different height), their outputs are not comparable without further information and the maximum finding will be disturbed. Furthermore, the total charge of the cluster is also an important value in itself, as it directly correlates to the energy loss of the particle.

At the end of the process each maximum found is converted into a space point. This includes the fitting of the three pads with the known pad response function, thus extracting the y coordinate. The x coordinate is calculated based on the time bin at which the cluster was found, with the help of the drift velocity and the time zero shift of the pad. Due to the geometry of the readout pads, clusters are only distributed in one pad row. Thus the best estimate of the z coordinate is the middle of the pad. The drift velocity and the time zero shift are not absolutely constant in time, like the pad or time response functions. Hence these values are the result of a calibration. Calibration values are saved in the *Offline Conditional Date Base*

(OCDB) in the GRID. For a more detailed explication of the calibration process, see also Chapter 5.4.5.

When the conversion is terminated, all space points found are appended in the output array of the clusterizer, which is then forwarded to the next processing step, the track finder.

Implementation Changes

As already mentioned, the code review was done to increase the processing performance of the offline reconstruction algorithms, by preserving the result quality as much as possible. The first speed profiles (see Chapter 5.1) showed that the clusterizer was the bottleneck, limiting the whole HLT reconstruction to about 40Hz in pp.

Firstly the algorithm of the cluster finding was modularised, increasing source code auto documentation and profiling ease. This also revealed a problem of calculating the charge of overlapping clusters, which was solved as described in the following.

The naive implementation of finding maxima in the three dimensional data array would be to loop over the whole array checking each pad[15]. With the realistic assumptions of track multiplicities, there are less than 100 tracks per chamber, i.e. less than 30% of the pads have a signal, and thus most of this array is empty. To cure this highly inefficient processing the index array was implemented. Each pad at which the raw reader reads a signal at some time bin gets an index in this array. This index tells the clusterizer which pads are worth looking at. Those pads are then checked at all time bins as that index array only has two dimensions[15], column and row. This is due to the fact that particles are mostly crossing the TRD readout chambers perpendicularly, making the time bin information of the index array almost useless. This is also very beneficial for the processing speed, as resetting too big arrays is quite time consuming.

Before being used however, the helper array is sorted by row and column. The data from the detectors is not necessarily saved in this order, although this is important for the following reason: If two particle traces are close, two clusters of one row might overlap, i.e. there is only one pad between the two maxima. To assign a reasonable charge to each of those clusters, the signal of this middle pad must be divided among the two clusters. To be able to process this easily, the two clusters have to be found in a specific order, making the sorting of the helper array mandatory.

Subsequent tests showed that a usual sorting of each of the two dimensions of the helper array is too slow. Thus the two dimensions, row and column, are binary merged to one dimension. Therefore the bits counting the row and those counting the column are simply appended and then interpreted as one number. Thus only a one dimensional sorting has to be done, which can be performed very fast by the C++ sorting function, which is much faster than the higher level sorting algorithms provided by the ROOT package.

[15] the data array has 144*16*30=69120 members, the helper array has 144*16=2304 members

Once the array is searched efficiently, the algorithm of maximum finding itself can be addressed. The method checking whether a signal maximum is at a given pad, is inside the innermost loop of the clusterizer. Thus any small change here has an immediate and significant impact on the whole clusterfinding process. The checks inside this method are ordered very carefully to be able to have as much throughput as possible. Each check which fails, makes the processor return to the loop. Thus the more probable a check fails the earlier the check must occur. However, the more complicated the check is, the later should the check occur. Thus this ordering obviously was done using profiling. A wrong ordering of these checks may easily result in speed penalties of this important method of a factor three or more!

As an exact comparison with the threshold values needs the gain information of the current pad, already the first check would include a call to the data base. Additionally, each function call takes time. To avoid both issues a first rough pre-check against a conservative value is done before entering the method.

To be able to check the quality of the reconstruction algorithms, the code must also cope with simulated data in addition to raw data. Here simulated data does not mean simulated raw data, but a data type which contains additional Monte Carlo (MC) information, which is needed for quality analysis (see also Chapter 6). This is why the code has to provide the necessary facilities for processing both data types, while the processing ways should be as joint as possible. The outcome of this is that for propagating the data array, the helper array and the additional MC information arrays from the reader to the clusterizer, an additional layer of abstraction was implemented. This interface class is used to synchronously manage the arrays. For non-ambiguity and because it could thus be used also during the simulation phase, that class utilises for each of the 540 read out chamber another set of arrays. While this is needed during simulation and reconstruction of the simulated data, it is not necessary during reconstruction from raw data.

However, the overhead is tolerable when reconstructing raw data from all supermodules by one single clusterizer. But in the case of the HLT each supermodule is processed by a separate clusterizer (see also Chapter 5.4), making most of the created sets of arrays useless to each individual clusterizer as it gets data only from the corresponding 30 readout chambers.

Thus this interface class was redesigned to use only one set of arrays for all readout chambers during raw data reconstruction. After the clusterizer finishes its job with one readout chamber, it tells the interface class to reset the used input arrays, and calls the raw reader, which fills them again with the data from the next readout chamber.

Resetting the data array is mandatory because the raw data is usually zero suppressed, i.e. signals below a threshold are not sent by the TRD detector. The resetting of the helper array is fast as the array is only two dimensional[15]. The data array however has three dimensions[15], which makes the resetting already quite time consuming. This is especially unfortunate when the data array is empty or close to empty. Thus the resetting was implemented in two different ways dependent on how much data was added to the array. If the number of pads having a signal is above a

certain level, the array will be reset completely by one function provided by the C language. If below that threshold, the array is reset using the helper array to find out the few pads which had signals and resets those pads only.

Found maxima have to be transformed into space points: A coordinate transformation is necessary, before saving the cluster in the output array. For this, the position of the maximum is estimated by the pad response function and is corrected for the effect of the interaction of the electric and magnetic fields. The x coordinate is calculated based on a constant drift velocity. Additionally, also an estimate of the uncertainties is needed. These are mainly given by pad size and pad response. For all these quantities there are also higher order effects included. For this look-up tables are used. Preventing the copying of these constant arrays from the *static memory* segment onto the *stack*, each time the correction function is called, is an example of easy to achieve speed-ups. (For a detailed explanation, see Chapter 8.1.)

As performance boost of last resort, it is now possible to instruct the clusterizer to produce clusters only on every second time bin. This increases the speed of the clusterizer and the tracker by a factor of two. However, the tracking efficiency will drop steeply for tracks below 2 GeV/c.

Additionally to the described measures the number of dynamically allocated arrays and/or their sizes were reduced whenever possible, speeding up the resetting after each event. In case the array is used only inside a method, size reduction can by very easy to pursue, as there only the highest and lowest expected number must be known. When however the array is needed by multiple methods or even classes, a longer investigation is needed. The least that was tried to be done to any array on the *heap* (see Chapter 8.1), was to fix it in its memory location during a complete run reducing memory fragmentation. This was mainly done by moving objects containing bigger arrays as high up in the loop hierarchy as possible, e.g. from the class methods to the class itself.

When talking about arrays the paradigm of locality should also be mentioned. The reconstruction itself has per definition a bad temporal locality, meaning that once processed, data will actually never ever be processed again. Thus the processor cannot take full advantage of its caches. What remains is to at least ensure good spatial locality, meaning that data which is needed next should be right beside the data processed now. In case of the three dimensional data array of the clusterizer there are two uses, which need a contradicting array topology. The tail cancellation needs all time bins for each column-row combination and the maximum finding needs the columns for each combination of time bin and row. Thus only for one of the two algorithms the spatial locality could be guaranteed. The choice was to opt for the tail cancellation's needs, as here the whole data array undergoes a transformation by a closed algorithm, which accesses the data in a very tight unbranched loop. Thus, the array topology was chosen to have the time as fastest index, then the column and the row as slowest index. This index order is not too bad for the maximum finding, as the number of time bins is very low (usually between 24 and 30) and the column is the second index. Furthermore the maximum finding procedure needs

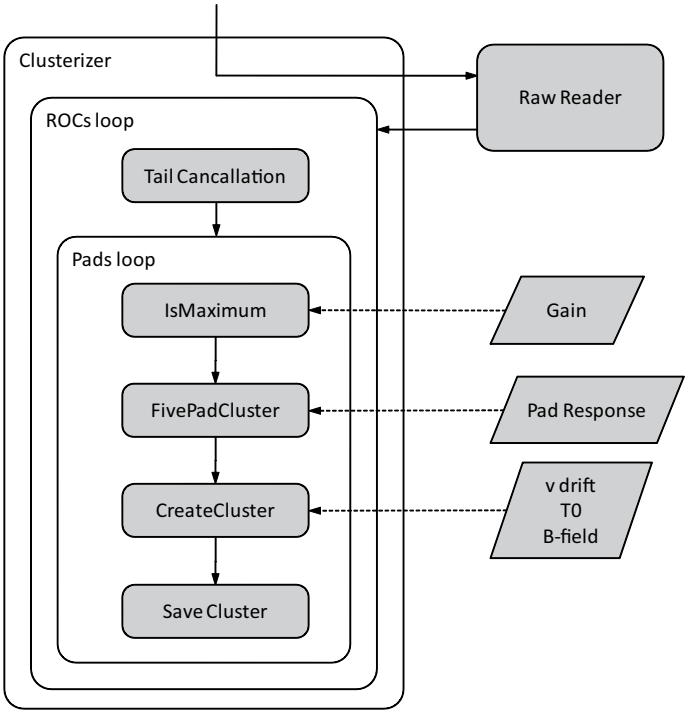


Figure 18: Flowchart of the data inside the clusterizer.

external data access and contains quite some processing inside the inner loop. Thus it would not have made full advantage of the more favourable topology anyhow.

The speed of the streamlined clusterizer is now about 1500 Hz for real data of triggered pp collisions at 7 GeV. This was reached with only very insignificant quality-to-speed trade-offs. This is a big step from the previous 50 Hz. This achievement is put into perspective by comparison to the time it takes of just looping once over the data array, which is 400 μs on the same processor, for a single readout chamber. During the HLT reconstruction however each supermodule must be processed, thus the speed of just doing nothing else but looping once per event over the data arrays of one supermodule would be 83 Hz. This shows the importance of the helper index array. A tiny quality cut, however, was needed to be accepted. It comes from skipping the calculation of an additional value: the count of active pads around each cluster in its pad row. It is used by the tracker to artificially decrease the weight of clusters with more than five active pads. This is done, since such high counts are very unlikely, and thus those clusters could be the result of a not properly working detector, or a noise burst, and might thus be outliers. Processing this value means looping over the data array along the pad row counting all consecutive active pads. The return of this calculation is a minuscule quality improvement and was thus skipped. The fact that such a trivially processable value already has a macroscopic effect shows the processing efficiency of the clusterizer reached very well.

Work Flow

The work flow diagram shown in Figure 18 is the result of the reimplementation of the clusterizer. The incoming helper index array and data array are filled by the raw

```
for(curr.time = 0; curr.time < fTimeTotal; curr.time+=iEveryNTB){
  while(fIndexes->GetNextRC(curr.row, curr.col)){
    if(GetData(curr.row, curr.col, curr.time) > fMaxThreshPre && IsMaximum(curr))
    {
      if(last){
        if(curr.time==last.time && curr.row==last.row && curr.col==last.col+2)
          FivePadCluster(last, curr);
        CreateCluster(last);
      }
      last=curr;
    }
  }
} if(last) CreateCluster(last);
```

Listing 1: The core of the clusterizer as it is implemented in the class AliTRDclusterizer.

reader. The data array is deconvoluted by an exponential tail cancellation, which is executed in the time bins dimension for each pad which has an entry in the index array. The index array sorts itself at the first get-data call. At the last index in the helper array the tail cancellation has finished and the index array resets itself, for being used once again.

This time the data is looped over for the maximum search (Listing 1). Here for each time bin all indexed pads are checked. The function checking for three maximum conditions (*IsMaximum*) is in the innermost loop of the clusterizer, with the rough pre-check ahead.

A maximum passing all conditions is then analysed for overlapping with the previous maximum (*FivePadCluster*). This is the case when the previous maximum was at the same time bin, in the same row, and two columns before the actual maximum. If so, the signal of the middle pad is distributed among both maxima. This is done by an iterative process with the initial condition of an equal distribution of the signal. The process is aborted when the last change of the distribution ratio was less than a given value.

Found maxima are converted into space points, a.k.a. clusters, with tracking coordinates and assigned errors (*CreateCluster*). This coordinate transformation from local chamber coordinates to tracking coordinates needs the information of the pad size, the drift velocity, the magnetic field, the diffusion parameters and the time zero information. These parameters are taken from the OCDB. Created clusters are finally saved in the output array of the clusterizer.

Performance Results

Figure 19 shows the processing performance of the clusterizer before and after the optimizations. The inclination of the red curve is very high, as long as not all readout chambers contain data. In case that a readout chamber does not contain any data, it is almost completely skipped from any processing, and thus here baseline inefficiencies can be hidden. These baseline inefficiencies where mainly array resettings during the maximum finding. When all readout chambers contain data, the curve enters the linear region. In this region the improvements of the maximum finding

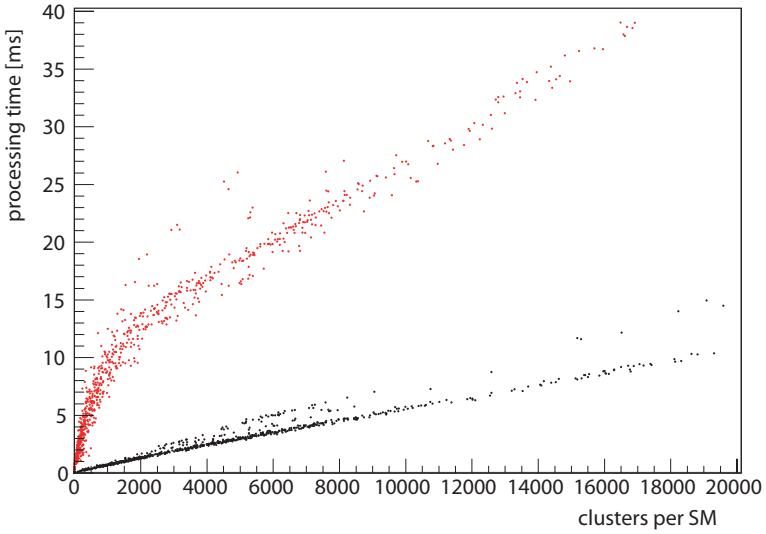


Figure 19: Processing time of the clusterizer for one supermodule. Shown in red is the situation after optimizing the raw reader, it is the same curve as in Figure 16. In black is the resulting performance due to changes of the clusterizer.

procedure itself make the difference.

5.3.3 Track Finder

The track finder or tracker is used for connecting the found space point (a.k.a. clusters) into continuous lines. The TRD tracker has two modes of operation: *barrel* and *stand-alone* tracker. The barrel tracker relies on the fact that during the offline reconstruction trackers from all detectors can exchange data. Thus the trackers can base their calculations on each other. This mode is particularly useful for tracks which can be prolonged into most of the detectors. However, for tracks which do not have prolongations into inward detectors, like particles originating from conversions inside the TRD, the second mode is used. This stand-alone tracker is executed after the barrel tracker and operates only on unused clusters. Since this tracker does not rely on information outside of the TRD detector, this is the mode used by the HLT online reconstruction for all kind of tracks. Here, all detectors are processed in parallel on different machines, disallowing any intercommunication. Thus only this mode was taken into account for the code review and is presented here.

Before explaining the processing of the tracker, a look at the ALICE and TRD geometry is necessary. Particles are originating at the main vertex position and travel to the outer layers of ALICE. Particles[16] are deflected by the magnetic field, which is oriented longitudinally to the beam pipe. As a consequence, particles are moving along helices. The axis is constrained in direction of the field, the radius is determined by the transverse momentum p_T and the pitch by the pseudorapidity η (see also Figure 21). Like all detectors of ALICE, the TRD has a projective geometry, thus particles in the interesting momentum region do not cross stacks. Thus a stack-wise tracking can be applied. A first sketch of the tracking procedure

[16] here only charged particles are of relevance, since neutral particles anyhow cannot be detected by the used gas ionisation detectors

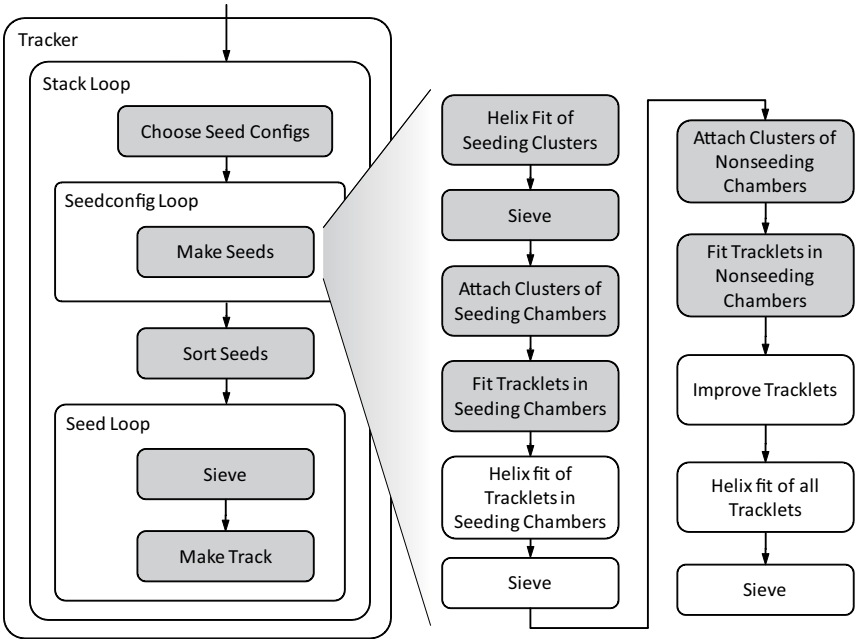


Figure 20: Flowchart of the data inside the tracker. The white steps are skipped during the HLT reconstruction.

provides the best opportunity to explain the vocabulary before entering the details. The processing needed can be summarised by these three steps:

- Choose seeding configuration
- Make seeds
- Make track

The stand-alone tracking is divided in two conceptual disjunctive parts: The track finding, which is represented by the second point, and the track fitting, the third point. Point one is needed as preparation. The final track fitting, or tracking, is done by a *Kalman filter* [KAL]. In principle a Kalman filter could be used for track finding and track fitting in one pass. However, in the stand-alone case this concept was not pursued. In regard of the TRD's peculiarities the two stage processing was followed instead, of which seed finding is the first. Found track seeds are later handed to the Kalman filter producing minimal uncertainty tracks.

In order to produce these seeds, an approximation of the track is used for deciding whether it is a good seed candidate or not. This approximation consists of a helix fit of four seeding clusters. A helix can be generally defined by four points, when however the axis is constrained only three points are needed for defining the helix. The fourth seeding cluster is used to provide a possibility for a quality measurement and based on this a quality cut is applied. In case of passing this quality cut the clusters of the chambers along the seed candidates are attached to the particular candidate. These clusters are then fit by a straight line, which produces the so called tracklets. It is possible to fit the clusters of one readout chamber by a straight line, as the readout chamber is only 3.7 cm thick. Based on these tracklets another helix fit is processed. This delivers the track parameters at the entrance of the TRD,

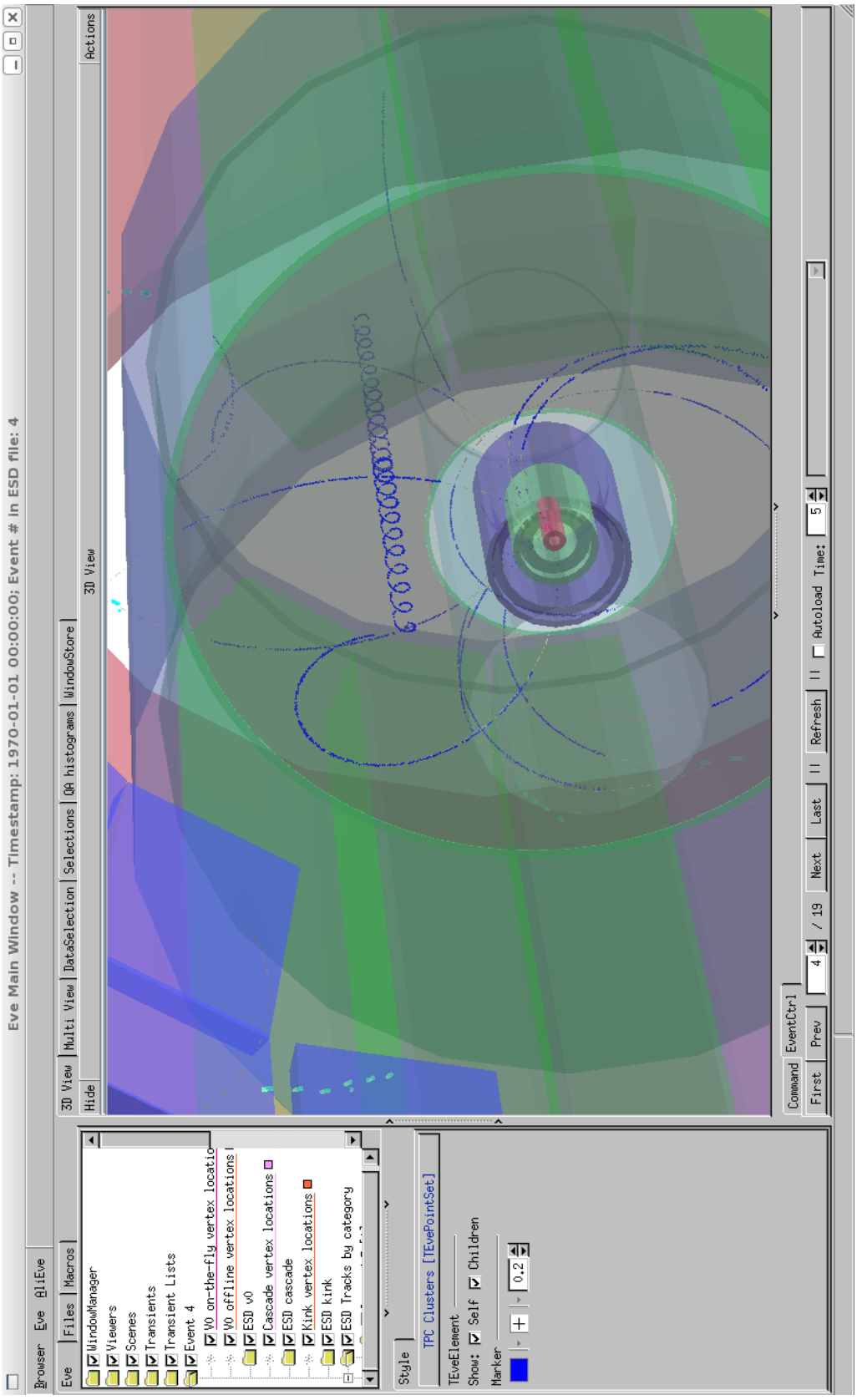


Figure 21: Particles in the TPC. Due to the magnetic field of the L3 magnet, particles move along helices.

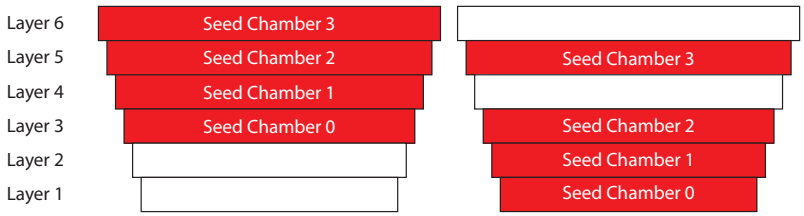


Figure 22: Two of the fifteen predefined seeding chamber configurations.

which are needed as seed for the Kalman filter. The tracklets are then used to update the track information at their specific layers.

Choose Seeding Chamber Configurations

Examining the details of the tracking is crucial for understanding the possibilities of improving its processing performance. The work flow begins with the selection of so called seeding chambers. These are chambers that will produce "good" seeds. There are several predefined seeding configurations each including four of the available six chambers per stack. Three of these seeding configurations are selected for being processed during the seed finding.

The three seeding configurations are selected by a quality measure, which includes two components:

$$q_i = q_{i,a} \prod q_{i,j}$$

Here $q_{i,a}$ is the a priory configuration quality, $q_{i,j}$ is the quality of the j'th chamber included in that seed configuration, and finally q_i is the total quality of the seeding configuration. The three configurations with the highest q_i are chosen. In Figure 22 two of the predefined seeding configurations are shown.

The a priory quality is basically given by the number of consecutive chambers and the number of gaps between the chambers. The quality of each chamber is based on the deviation of the number of clusters in that chamber to the mean number of clusters multiplied by an integer value bigger than zero. The higher the deviation the lower the quality.

This quality measurement provides a dynamical way of finding chambers useful for producing seeds by discarding empty, and thus broken chambers.

Make Seeds

This next step is done for each of the three seeding configurations. In each of the chambers of the given seeding configuration seeding clusters are produced. These are given by the centre of gravity of nearby clusters. The assumption here is that a seeding cluster is a good representative for the clusters of a tracklet. As the TRD chambers have a height of only 3.7 cm and tracks are mainly crossing it perpendicularly, that assumption is justified. However for making sure that also intersecting tracks are found, not only one seeding configuration is used, but multiple.

For producing the first approximations of the tracks, all "possible" combinations of seeding clusters are used. The possible combinations are found by starting at the seeding chamber 3 (see Figure 22). For each seeding cluster in this chamber

all seeding clusters of seeding chamber 0 are taken into account which are inside a window defined by two angles. From here all clusters of seeding chamber 1 are considered which are again in a window. At seeding chamber 2 only that seeding cluster is taken which is nearest to the position of the linear interpolation of the two seeding clusters from chamber 0 and 3. For all such combinations of seeding clusters found a helix is fit. Fits with a too high χ^2 are dropped, as this means that the probability of the fit describing the data is low. For passing fits, clusters around the seeding clusters are attached to tracklets, which in turn are attached to the track. The tracklets are fit based on the clusters by a straight line fit. Then a new helix is fit based on the tracklets. Now the likelihood of the tracklets is calculated and checked. This is a cumulative value based on the χ^2 of the tracklets and the total number of clusters. Along the extrapolation of the helix to the remaining two chambers which were not the seeding chambers clusters are, if available, attached to tracklets, which are then fit. Before performing the last helix fit, an iterative process of improving the cluster sets attached to the tracklets is started. This process replaces individual clusters with other neighbours, recalculates the tracklet based on this new set and compares the quality.

Finally the found seed candidates which survive also the last χ^2 are saved in an array. Each step and each performed check which fails leads into discarding the tracklets of the current set of seeding cluster and finding the next possible set.

Make Track

Each of the seed candidates found describes a track. Thus *Make Track* is inside the loop of all found seed candidates (Figure 20). However, as the seed candidates originate from three seeding configurations of the same stack, each track is represented by up to three seed candidates. In order to not produce multiple tracks of the same particle trace, seed candidates with too many clusters which are used by other tracks are rejected. For being sure that only the best seed candidate of each track is used for the track generation the seed candidates are first sorted by the likelihood produced in Make Seeds.

The actual generation of the track is based on the helix fit and the tracklets of the seed candidate. Using the parameters of the helix fit the Kalman seed is produced describing an approximation of the track at the entrance of the TRD. Then the seed is updated by the information of the tracklets to generate a minimum uncertainty track. Actually exactly one point along the tracklet is used, this point is called reference point. There the y coordinate uncertainty is minimal. This point is also of importance for the Quality Assurance Chapter 6.

Implementation Changes

The code review for increasing the processing performance had to include the tracker as its speed in its offline version was about 100Hz for minimum bias pp events. The tracker however is a much more complex construct than the clusterizer. Thus the code review was restricted to slim it down to the absolute necessary processing steps, without a general reorganization. Still, output quality had to be monitored very closely for choosing the right combination of calculations. At some few critical

spots, however, changes had to be introduced similar to those of the clusterizer: increasing processing performance without losing precision.

First of all, the procedure of choosing the seeding configurations had to be altered. In case of the complete offline reconstruction, the stand-alone tracking is performed after the barrel tracking has finished. Thus only clusters from tracks without prolongations to the inner detectors remain at this stage. In case of running the stand-alone tracking without preceding barrel tracking, the situation however changes radically. In this case it does not make sense to have the chamber-wise quality of *Choose Seeding Chamber Configurations* strictly dependent on the deviation of the number of clusters to an integer multiple of the mean number of clusters. Having big number of clusters statistical fluctuations lead to fake decisions. Thus the importance of this deviation must be reduced with increasing number of clusters.

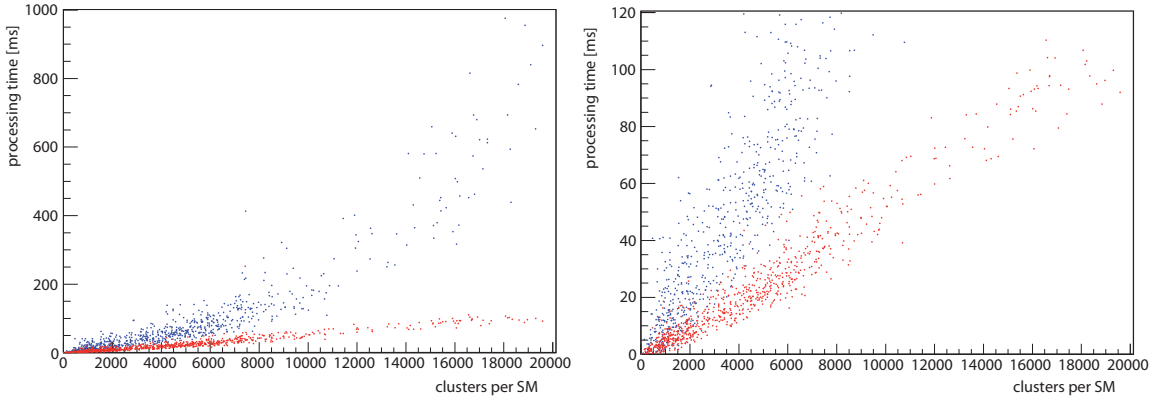
The first obvious possibility to increase the speed is to reduce the number of returned seeding configurations. In the offline reconstruction the three seeding configurations with the highest quality are returned. By decreasing this to the best two configurations only, the processing speed of the tracker is increased by one third, while the tracking efficiency slightly drops by 1% at 1 GeV/c. For reaching even higher processing speeds of course the number of returned seeding configurations can be decreased to one. This however includes a much stronger efficiency cut (5% at 1GeV/c).

Another step towards a streamlined tracker was to cut the procedure of *Make Seeds* just after the fitting of the tracklets to the clusters the first time. Thus especially the iterative procedure of refitting the tracklets was dropped completely. As this includes a big amount of refits, which even increases with the number of clusters per chamber, a big leap in processing performance can be attributed to this change. However, this change also includes a great degradation in terms of quality. Since the second helix fit, which is based on the tracklets, is completely skipped, the spatial quality of the track is based only on the first helix fit. This includes only four seeding clusters in the four seeding chambers. Additionally the curvature is not determined there, thus no momentum information is left. Thus for the HLT reconstruction a helix fit was added at the beginning of *Make Track*. This helix fit is however different to the one performed during the offline reconstruction in *Make Seeds*: it is constrained to the primary vertex[17]. As most tracks originate near to the primary vertex, this restriction only affects tracks without prolongations into the inner detectors. By moving the second helix fit into *Make Track* the helix fit is called only for seeds which survive the cut in front of *Make Track* and thus it is called much less often then it would when being inside of *Make Seeds*. The drawback of this procedure is that there is no sorting of the tracks depending on their χ^2 of the helix fit possible any more, but this is the quality cut which was necessary.

One major change, which increased performance by preserving the quality, was the inclusion of a self-written linear fitter, which is used by the tracklets. This replaces the very high level implementation of the ROOT framework, which was very poorly performing. The self-written implementation uses the fact that a linear fit can be processed by just using sums of the values and their squares.

Additionally at specific spots the error propagation of the clusters toward the

[17] additionally to the constrained axis



(a) (b)

Figure 23: Processing time of the tracker with offline (blue) and online (red) configuration for one supermodule at different event sizes. Figure (b) shows a zoom to the red points.

tracklet was simplified. First of all the uncertainties of the first helix fit at the position of the tracklet is skipped completely, only pre-set values are used. Secondly, the cluster error is not recalculated based on a Gaussian fit of the signals when it is attached to the tracklet.

With the measures described the tracker has been accelerated from 100 Hz to about 1000 Hz in minimum bias pp events. The bulk of this performance increase was done in a selective way. As contrary to the clusterizer quality losses were accepted, the resulting quality had to be monitored very closely. The procedure and the results of this Quality Assurance are described in Chapter 6.

Performance Results

The resulting processing performance of the speed optimized stand-alone tracker as it is run on the HLT cluster compared to the stand-alone tracker as it is run in the offline reconstruction is shown in Figure 23. Especially the higher order dependency of the standard tracking is very problematic for realtime usage. This is mainly due to the big number of refits.

5.4 Online Reconstruction

Data coming from the TRD detectors during the online reconstruction basically takes the same processing steps as in the offline reconstruction. There are however many administrative differences which can be summarized with these words: parallelism, transport, and control.

In the HLT TRD reconstruction the parallelism is applied at the supermodule level. This means that each supermodule is reconstructed separately. The processing is modularised to be flexible and to be able to distribute needed work to as many different computers as possible. After being received and processed by the FEPs data is transferred to other computers (CNs) via ethernet. Thus a huge amount of different jobs is being processed on many different computers, which makes the control of all of these being essential. These tasks are carried out by the

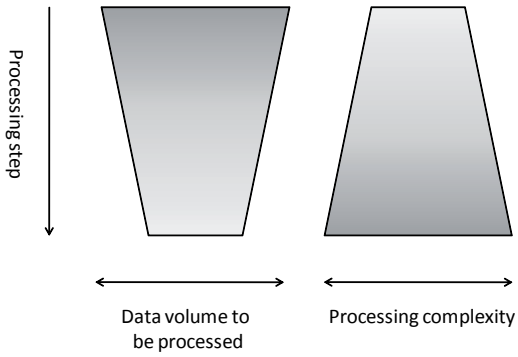


Figure 24: The principle of HLT processing. Each processing step shall decrease the data volume such that the next processing step may have a more complex processing.

Publisher-Subscriber Framework. It takes care of the jobs on the machines and the communication between them. Processes which shall be started by the framework are described in classes, which inherit from one base class and are called components. The development of the components needed for the data reconstruction, calibration and monitoring was part of this thesis. The components are in general closed analysis units. In case of the TRD the reconstruction and calibration components are interfacing offline code, like clusterizer and tracker. This code is thus shared between the HLT online and the offline reconstruction. The needed speed-up of this code was described in the previous chapter. Now the focus will be the components itself and the communication between them. Another important issue addressed in the following is the placement of the components onto the available computing machines.

To be able to cope with the high amount of raw data, each reconstruction step reduces the data volume thus the next more sophisticated processing step has to run only on a concentrated data sample (Figure 24). The first reconstruction step is the clusterizer, which is followed by the tracker. Those two reconstruction components prepare the data for the analysis components, monitoring components and calibration components.

In this chapter the components developed are presented. As the implementation of exchanged data containers is a very important issue, this is the starting point prior to turning to the components.

5.4.1 Data Exchange between Components

Since the processing on the HLT cluster is performed by many computers, network transmission between is needed for the communication between processes. As network can quickly turn into a bottleneck, it is mandatory do have a close look at what has to be transmitted and how this should be done. At the first tests it already became clear, that the data types used in the offline reconstruction are not suitable for the data exchange in the HLT. The reason is the parallelism of the HLT in conjunction with the object oriented, inheritance based dynamic polymorphism used by virtually all classes of ROOT and thus also of the AliRoot framework.

When data passes from one component to another, the executable it is processed by changes. Data might even find itself on another machine, as the participating components are distributed among different nodes, depending on how the compon-

```
1  class AliHLTTRDCluster {    // class implementing the micro cluster
2  public:
     AliHLTTRDCluster();
     AliHLTTRDCluster(const AliTRDcluster* const inCluster);
     void ExportTRDCluster(AliTRDcluster* const outCluster) const;
   private:
7    UInt_t   fSignals;        // Signals in the cluster
     UChar_t  fPadCol;         // Central pad number in column direction
     UChar_t  fPadRow;         // Central pad number in row direction
     UChar_t  fPadTime;        // Uncalibrated time bin number
     UChar_t  fBits;           // Bits of the cluster
12 };

   struct AliHLTTRDClustersArray {  // header for the micro clusters
     AliHLTTRDClustersArray(Int_t det):fDetector(det),fCount(0){}
     Short_t  fDetector;            // detector number
17   UShort_t fCount;               // number of clusters following the header
     AliHLTTRDCluster fCluster[];   // open array of clusters following the header
   };
```

Listing 2: Cluster and cluster-header exchanged between clusterizer and tracker component.

ents happen to be arranged (see also Chapter 5.4.7). This has a crucial implication as each running executable lives in its own address space, which is mapped to real addresses by the operating system (see also Chapter 8.1). When an instance of a class using dynamic polymorphism, or having pointers as data members, is written out and read by another executable, the pointers become useless as in this new address space they point to locations with different meanings. Among those pointers is also the hidden virtual table pointer (*vpointer*) pointing to the virtual table, which is needed for polymorphic objects and is an array of pointers pointing to the virtual functions. The only way to reliably ship such an object from one executable to another is to stream it. That means that not only this object must be copied, but anything pointed by its pointer-data-members must gain a meaning in the new address space. What makes this a problem is that all classes within both the ROOT and the AliRoot framework are based on one abstract base class, making all inherited classes polymorphic. Thus already small events, with only very few tracks have quite a considerable amount of pointers, which makes the streaming very slow.

To make an example: Let's have an event with two tracks. Considering the most common, each traverses all six TRD layers, thus there are six tracklets per track, and each tracklet has an average of 26 clusters. Thus there are 26*6*2=312 clusters, 6*2=12 tracklets and the two tracks. All those 326 objects are instances of classes with more or less complicated heritage from the abstract base. Streaming these objects takes about 60% of the CPU time needed to calculate them.

Data Containers

In order to circumvent this problem, data containers holding just the needed data members of the tracks, tracklets, and clusters have been implemented. These contain only the absolutely necessary values, and are tuned for high conversion speeds and

small sizes. Using encapsulation these containers can initialise themselves based on the offline objects, or create an offline object based on themselves. These containers are implemented as classes. To decrease the time needed for conversions these HLT data containers are befriended to the offline classes, gaining unobstructed and fast access to the data. To decrease the size of these containers to a minimum, data members are sorted by size (see also Chapter 8.1).

What is sent from the clusterizer component to the tracker component are not the clusters but only the not transformed maxima. These are saved in one data container which, out of historical reasons, is called cluster data container, or just cluster. To avoid confusion, these clusters will be called micro clusters in the following. These micro clusters contain just the signals of the three neighbouring pads of the maximum, and the number of the row, column and time bin at which the maximum was found. The signals of the three pads are merged into one binary *double word*, just like it is done for the data transmission from the detector to the HLT. The three local coordinates can each be represented by one byte. The remaining byte for completing the alignment is used for a flag indicating whether the cluster was overlapping with another one. This is used later on by the tracker for determining the quality of the cluster. In total that leads to a size of a micro cluster of 8 bytes. This information is sufficient to unambiguously identify a maximum in its read out chamber. However, this is not enough for the tracker as it needs to know global coordinates. Thus also the number identifying the readout chamber itself must be transmitted. There are 540 readout chambers, and to identify each 10 bits are needed. Adding this to the micro cluster data container would result in adding 16 bits[18]. In each readout chamber there are however many clusters, which share this information. To exploit this evidence the following implementation was chosen:

All micro clusters of one chamber are following a header, which contains the detector number and the number of following micro clusters. Additionally this header contains as last data member an open array of the micro clusters. Via that array all micro clusters of the readout chamber can be addressed comfortably. Listing 2 shows the structure of the micro cluster and its header.

The tracker component needs to ship tracks and tracklets to the monitoring and calibration component. Each tracklet is the straight line fit of clusters that are part of the same track in one readout chamber. Beside of the spatial information, the tracklet contains also all clusters that were used for the fit. The tracker reduces the data size, as all clusters originating from noise do not lead to tracks. Thus it is possible to output tracks, which contain clusters, that are less compressed: the clusters saved in each tracklet are an inherited version of the micro clusters, including the global tracking coordinates. Thus these are fully-fledged clusters and are directly usable by the following components, without the need of a coordinate transformation. The tracklet is now being used as the header, thus it also contains the open cluster array providing a transparent access to the clusters.

This leads to the track which, actually, is the object of interest. Each track is calculated out of up to six tracklets. The track is used as header to the other tracklets, just as the tracklet is a header for the clusters. However, due to the varying number of attached clusters, tracklets do not have a defined size, which

[18]This is because either way data addresses would be unaligned

means that the concept of open arrays cannot be applied here. Instead the tracklets are addressed by pointer arithmetic. For this each tracklet needs to know its own size, including all the clusters.

Alternative to Data Containers

For completeness, the other discussed and tested method of solving the streaming problem shall also be mentioned. Considering the pointer data members are not of interest or can be restored. The only remaining problem is the hidden virtual table pointer. What is feasible is to treat the offline objects as the HLT containers, i.e. save them directly into the shared memory block and read them out by the next component. Then, for using the read objects, a new offline object is created and its *vpointer* is copied onto the read objects. Like this, the *vpointer* is restored and the read objects are fully usable. This method only works because the HLT computers are all binary compatible and although it is very easy to implement, it has a drawback: the offline objects are much bigger than the corresponding HLT objects. For comparison let's take the two tracks from before: the total size of the offline objects would be 41056 Bytes, whereas the size of the HLT objects totals into 9760 Bytes. For events with hundreds of tracks this makes quite a difference. This of course is only a problem for data that must be transferred to other computers. However this, let's say C++ hack, is dependent on a specific implementation of the C++ compiler and the target architecture. Even though these issues could be covered during the configuration of the source code, the decision was taken to opt for the more secure implementation with the data containers.

Procedure of Data Exchange

The data shipment from one component to the next is done as follows: Based on the offline objects coming from the offline processing algorithm, HLT data objects are created in the shared memory block of the processing component (see also Figure 11). When the processing of the event has finished a descriptor is set up containing some crucial information: What kind of data it is (cluster, tracks, histograms...), what supermodule this data is from, how big the data block is and where it can be found (i.e. the pointer to the beginning of the data in the shared memory block). This descriptor is sent to the subscriber component which reads it. If interested, it uses the HLT data objects from the publisher components shared memory block right away, or it converts them back to the offline objects. So there is at least one conversion: from the offline object to the HLT object. But, as emphasis was laid on it, this conversion is very fast: less than 0.5% of the total CPU time is invested into that.

5.4.2 Reconstruction Components

The TRD reconstruction components are used to interface the offline processing units, like clusterizer or tracker for being used by the HLT framework. The components set up the needed environment for the offline processing unit and prepare the data exchange to other components.

```
<Component ID="CF">
  <ComponentID>TRDClusterizer</ComponentID>
  <Parent>DDL</Parent>
  <Options>output_percentage 100 -lowflux -experiment
           -tailcancellation -faststreamer -yPosMethod LUT</Options>
  <Shm blocksize="250k" blockcount="1000"/>
  <Multiplicity>1</Multiplicity>
  <Library>libAliHLTTRD.so</Library>
  <ForceFEP/>
</Component>
```

Listing 3: XML configuration of the clusterizer component. The framework is notified that a component named *TRDClusterizer*, which can be found in the library *libAliHLTRD.so* needs the data coming from the *DDL*. The arguments setting up the component are given between the *Option* keywords. The node at which the component shall be executed, as well as the amount of shared memory needed is given by the *ForceFEP* and *Shm* keyword respectively.

During an offline reconstruction where all reconstruction steps are performed by only one process, the different processing steps can easily communicate with each other, as they share the same memory space. Even though the HLT framework provides a shared memory system, one has to remember that data still needs to be transmitted through a network to reach other nodes. As communication must be kept to the absolute minimum, so must the dependencies between processes.

Component Initialisation

As shown in Figure 25, all HLT components have five states which are invoked by the HLT framework, which in turn is steered by ECS. While the framework is being configured by ECS, it is given the ECS configuration parameters and it reads the XML configuration files, which define the available processing nodes, the needed components, and the placement of the component on the nodes. During engagement of the framework, it creates the jobs on the nodes loading the shared libraries where the components are saved in. The components are created by a call to the classes *Constructor*, where the most basic initialisations are carried out, like creating the class members or allocating the needed heap memory space (see also Chapter 8.1). Then the *DoInit* method is called, given the arguments of the XML configuration files. The offline processing units (clusterizer or tracker) get their configuration parameters from the reconstructor, a class used to steer the whole offline reconstruction. Thus before initializing the offline processing unit the *DoInit* method sets up the reconstructor with the parameters taken from the parsed arguments of the XML file. After this is finished the component is ready for data processing. For this, the component enters the event processing loop. When the run is being terminated, the component exits this loop, cleans its memory space and finally is deleted. Listing 3 shows the XML configuration of the clusterizer component.

The XML configuration files are saved locally on the HLT cluster. As the component arguments are usually not changed during a run period and as it is useful to

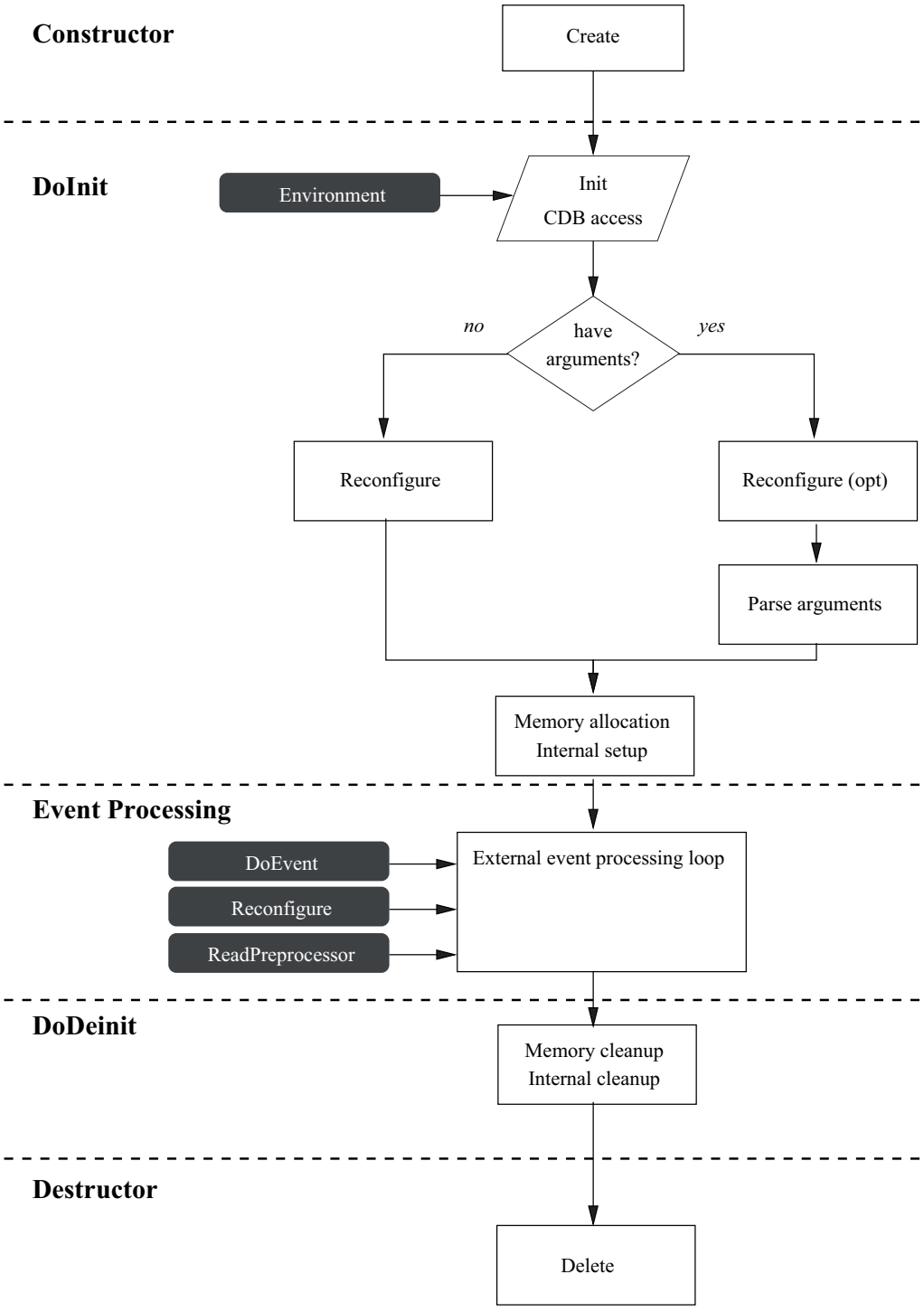


Figure 25: The work flow of HLT components includes five stages. [MRD]

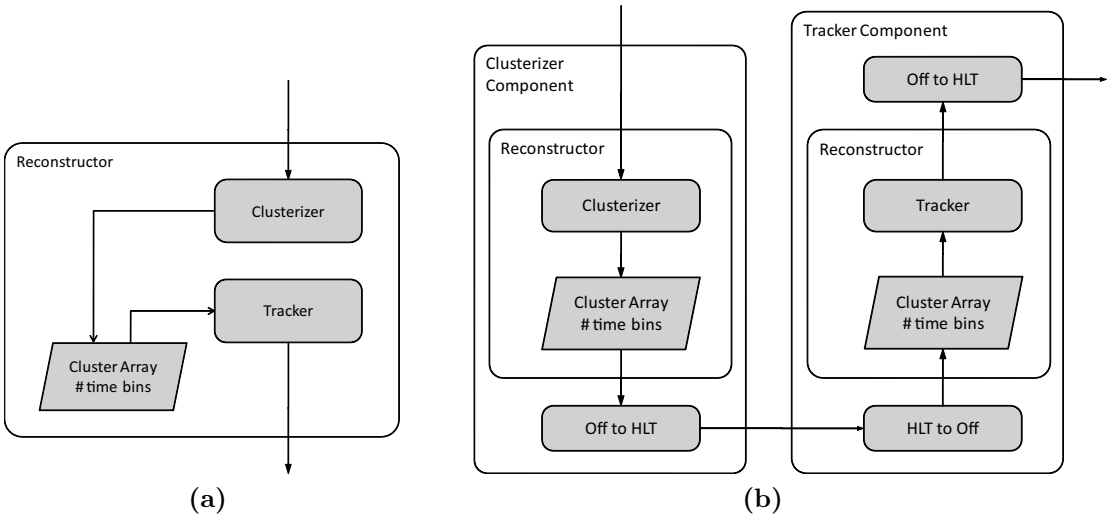


Figure 26: Flowchart of the data during an offline (a) and a HLT reconstruction (b).

have those arguments saved in a place accessible from outside of the HLT cluster, these arguments may also be saved as OCDB entries. These are parsed by the *Reconfigure* method, in case no other arguments are given. By that all necessary information to reproduce the state of the components during a specific run can be retrieved from all over the world. But still in case of malfunctions the components can quickly be given different arguments in the XML files.

Implementation

During the offline reconstruction the offline processing units are steered by an instance of the reconstructor and communicate through it. As this communication is not possible in the HLT, this behaviour must be emulated during the online reconstruction to make the processing units work, even if those are in their own address space, separated from the rest (Figure 26). This is done by running each of the reconstruction components with its own instance of the reconstructor. Depending on how the reconstructor is set up by the component arguments, the components can be used to process the data at all levels of accuracy. Thus the performance optimisations implying quality cuts, described in Chapter 5.3, of the tracker and clusterizer can be enabled, or disabled, leading to HLT online, or stand-alone offline reconstruction result qualities.

Apart from this, the task of the reconstruction components during data taking is to prepare the incoming data before it is forwarded to the offline processing units. After the offline processing unit finishes its processing, data is converted to the HLT data containers and the data descriptor is filled.

For the clusterizer component, which gets its input data from the HRORC adapter card[19], this preparation consists in these steps:

- Setting up the raw reader to the address of the memory where the HRORC has saved the raw data to

[19]see also Figure 9 and Chapter 4.2.3 for an explication of the HRORC

- Reading the DDL number out of the Common Data Header of the raw data and forwarding this information to the raw reader

Based on this information the raw reader checks the integrity of the raw data and returns warnings on failures, indicating potential data corruption or misconfiguration of the detector. As this can be seen by the HLT operator, errors in the detector configuration can be immediately unveiled already at the very first events of the run.

The clusterizer used in the clusterizer component is an inherited version of the offline clusterizer, which further eases the conversion from the offline data types to the HLT data containers. This is done by polymorphically overriding the method which saves the created cluster into the output array. Generally however, dynamic polymorphism should be applied with care as methods have a slight speed decrease compared to normal methods when being called. This overridden method converts the offline clusters to HLT micro clusters and saves them in the output memory block. The beginning of this block is given by the component before starting the cluster finding. When the clusterizer has finished its job the component is returned the size of the written data. The component adds the description information and sends out the written part of the block.

The tracker component gets the micro clusters from the shared memory block of the clusterizer component. As it has been discussed in Chapter 5.4.1 the clusterizer component sends only the information of each cluster which is actually describing a maximum. In order to extract all the other necessary information the tracker component needs to do the same as it is done by the offline clusterizer in the last processing step: transform the maximum into clusters. This means a coordinate transformation from the local readout chamber coordinates into tracking coordinates. These transformed clusters, which are all appended into an array, are given to the offline tracker.

Once the tracker has finished, the tracker component grabs the output array of the tracker, which contains all found tracks with their tracklets and clusters, and converts them into the HLT data containers. These are created directly on the output shared memory block. After adding the description information the block is sent out.

5.4.3 Monitoring Components

In order to facilitate an overview of the status of the online reconstruction during data taking, the monitoring components send out different monitoring plots to the display at the ALICE control room. These plots are shown on the operator display and can also be viewed remotely via the internet.

There are two types of monitoring components: the cluster monitoring component and the track monitoring component. The cluster monitoring component gets its data from the clusterizer component, the track monitoring component from the tracker component. The following distributions are shown:

- charge per cluster
- time bin per cluster

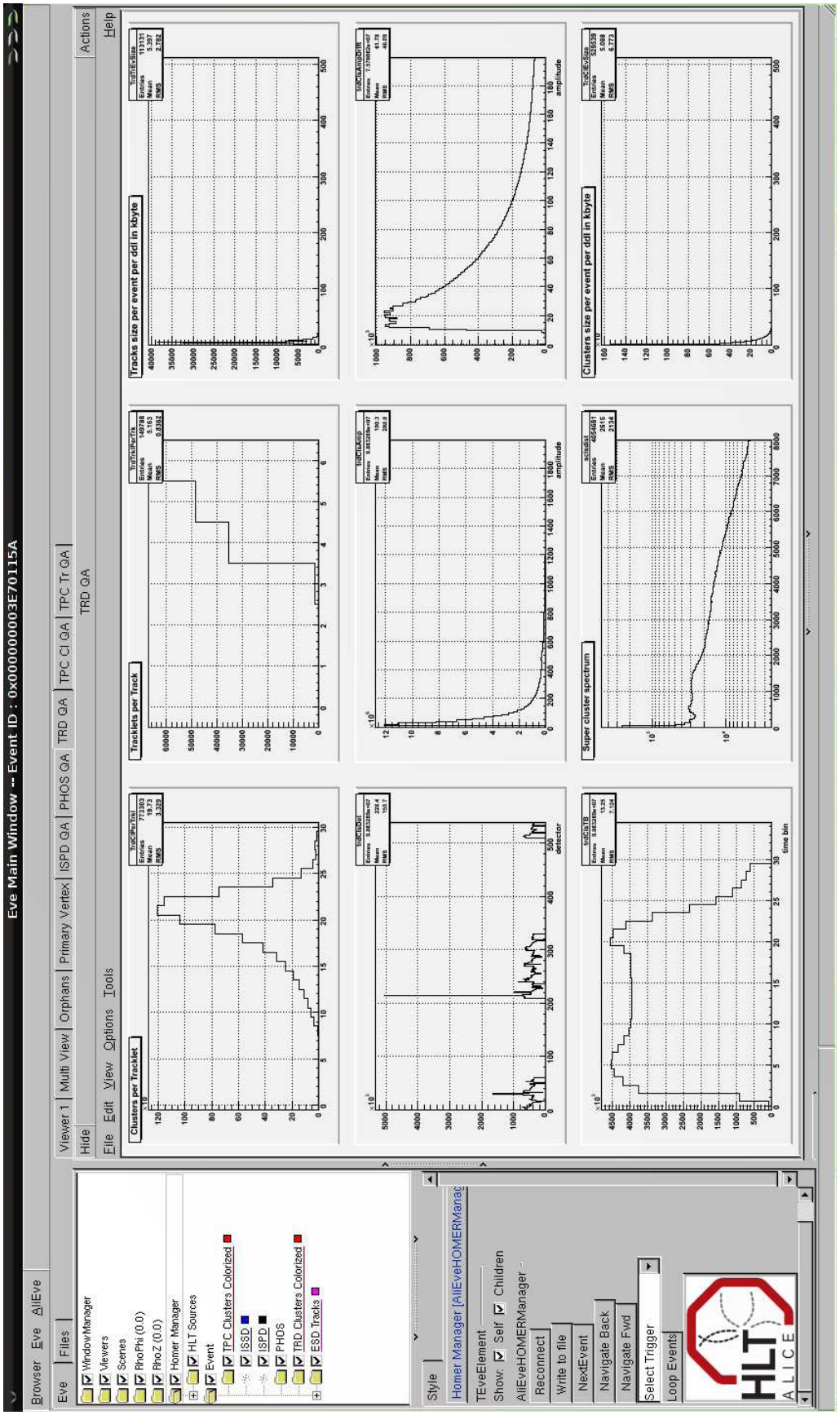


Figure 27: Screenshot of the TRD online histograms during reconstruction.

- number of clusters per detector
- total charge per detector
- incoming size per event
- clusters per tracklet
- tracklets per track
- η, φ and p_T distribution of tracks

As shown in Figure 27, these histograms can be seen by the operators in the ALICE Control Room. As these plots show the output of the reconstruction process which is based on the offline reconstruction, problems shown here will most likely also extend to the adjacent offline reconstruction of the saved data. In case of a misconfiguration of the detector that would have remained unseen, these monitoring plots give a unique possibility to cross check. In case the histograms look peculiar, the operators can call an additional expert, who may view the plots remotely via internet. The decision of aborting the run and reconfigure the detector can then be then taken by the expert.

Implementation

The monitoring components get the input data from their parents' components. To avoid network bottlenecks, there is not one central component getting the data from all nodes, but instead, on each node a separate monitoring component is executed. For each input object (cluster or track) the above mentioned quantities are determined and filled into a histogram. These histograms integrate the data over the run time.

Before the histograms can be sent towards the display, the histograms of all histogramming components must be merged. This is done in a subsequent component, explained in the next chapter. There however the problem of network bottlenecks might arise again, and exactly this was the reason for having one monitoring component per node. Additionally, as the histograms are objects of the ROOT framework, they must be streamed, which is inherently slow. However, as the histograms are integrating over run time, these problems are simply solved by sending the data more rarely than once per event. For achieving this, the time at which the last sending occurred must be saved. Until a set time interval did not elapse, any other output of the component is prohibited. The time scale for this interval can be adjusted freely; a useful setting is several seconds up to one minute. Thus the monitoring components process all the data, the observer however is shown a snapshot only every now and then, which is completely sufficiently.

5.4.4 Histogram Merging Component

The histogram merging component is needed since the monitoring components take their data per node. In order to get a fast overview of the detectors state, it is however more practical to not display each histograms form each node separately.

Additionally some of the histograms of the tracker monitoring component are only useful when the whole detector is included into one histogram.

As the task of adding histograms is independent of the data included in the histograms, there exists only one implementation of the merging components, which is executed in two instances. Each instance exclusively computes the data of the cluster or track monitoring component.

Implementation

The histogram merging component gets its input data from the cluster or track monitoring component of all nodes. Each histogram of one monitoring component is added with the corresponding histograms of the other monitoring components.

The design of a component doing only this would be almost trivial. The monitoring components, however, send their data only after a time interval elapsed. The interval is fixed during a run, but this does not mean that the sending of all components occurs at the same time. This would not even then be the case when all computers sustained their system times perfectly. As components can only send out data when input data is available. Not all supermodules must necessarily have tracks in exactly that event when the time interval elapses. Thus the arrival times have a spread. In order to deal with this fact, the merging component waits to proceed with its own outbound sending until it gets data from the same supermodule twice. Consequently the merging component waits roughly one time interval to aggregate the histograms from all supermodules. Finally it sends the histograms to the displaying computer.

5.4.5 Calibration Components

The detectors are built very carefully, however, physical attributes like drift time or gain are not constant[20] and must be monitored. It is important to extract these quantities, as they are the base for the calculation of all observables: spatial coordinates, momentum and energy loss. This is of course also done by the Offline group, but to get a first idea of the state of the detector and to be able to directly analyse the taken data, an online calibration is useful. Indeed, it provides the possibility for the HLT to use this calibration data at the next run already, improving the results of the reconstruction. For this purpose an online version of the TRD calibration is performed during the HLT reconstruction. Just like the reconstruction components, this component interfaces offline TRD code to the HLT framework.

The calculations are based on the clusters and tracklets, which are attached to the tracks coming from the tracker component. The following quantities are extracted by the calibration component detector wise:

- Gain: This defines how much the primary ionisation is being amplified by the anode wires
- Drift velocity: This is the speed by which the ionisation electrons travel through the drift region

[20] This is because of changing temperature, pressure, gas mixture and so on

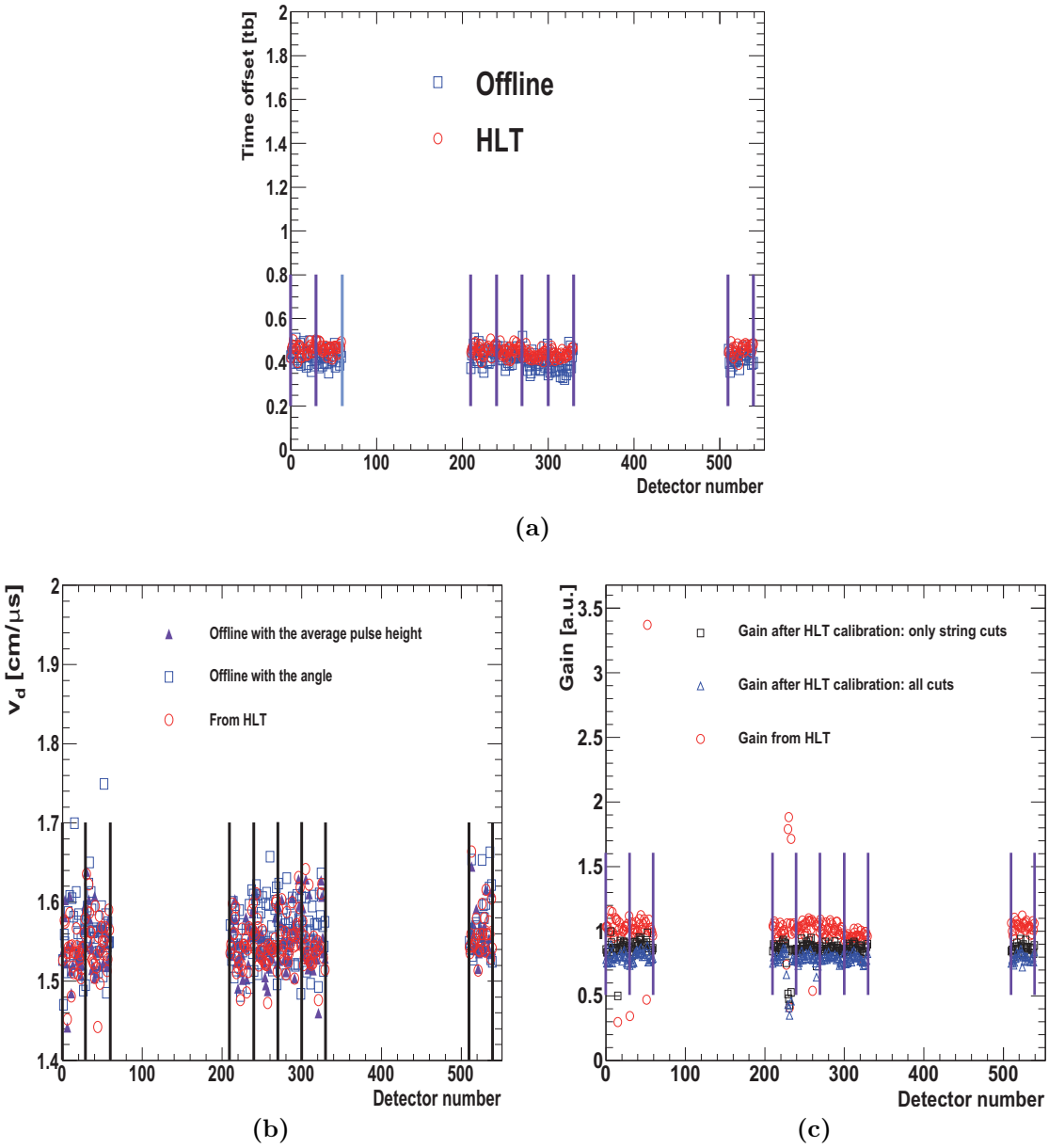


Figure 28: Comparison of the calibration results for the HLT and offline.
Figure (a) shows the results for the time zero offest of individual detectors.
Figure (b) shows the extracted value for the drift velocity,
and figure (c) shows the gain. [RB]

- Time zero: Together with the drift velocity this gives the distance travelled by the drifting electrons [RB]

Once the run is terminated, the reference histograms are saved on the so called *File Exchange Server* (FXS), from where they are picked up by the so called *Shuttle*. The histograms are fit there and the final resulting values are saved in the GRID OCDB. From there these values can be picked up and transferred back into the HLT's copy of the OCDB, to be usable for the next runs. Additionally, the first pass of the offline reconstruction may use these values. More accurate calibration values are generated by an iterative process utilizing subsequent reconstruction passes. The online calibration saves one such iteration, which is by itself a very important reason to run the online calibration, as processing power is an always scarce commodity.

Implementation

The actual implementation of the calibration and its link into the HLT TRD processing chain is very similar to the monitoring components. Based on the tracks coming from the tracker the calibration histogramming component generates the reference histograms (dE/dx-spectra and average pulse height) for the detectors through which the tracks were going. After the elapse of a time interval, a copy of these histograms is sent to the calibration merger component. There the histograms coming from all calibration histogramming components are merged. On the *End Of Run* event (EOR) just before the component is taken down the histograms are saved on the File Exchange Server. Due to the importance of the calibration, and even though the processing is very similar to the monitoring, it is processed separately giving the possibility to be used in a minimal fail save configuration.

Unlike the HLT calibration, which is performed per detector, the offline calibration is segmented much finer; for groups of pads, or even single pads. However pressure, temperature, gas mixture and high voltage have only small spatial fluctuations inside one chamber. The results of the HLT calibration as compared to the offline calibration is shown in Figure 28. These plots are a lookahead to the Quality Assurance in 6, since the values match quite well, the reconstruction of the TRD inside the HLT obviously has a reasonable quality.

5.4.6 Trigger Components

The previously discussed components are all necessary for the reconstruction process itself or for calibration or monitoring. These are important goals of the HLT, however the triggering facility is as important. For this, dedicated components are needed, taking the output tracks from several detectors and check for specific signatures. Only events with such signatures would be triggered and thus saved by the DAQ.

The first available and running triggering component was the TRD cluster multiplicity component. It was used during the last session of cosmic runs in autumn 2009, such that the TRD could take data without other dedicated triggering detectors. It was the first test of the triggering capabilities of the HLT including a detector. The implementation of the component itself is in fact trivial, as the component takes the data from the clusterizer and checks for events with more than a

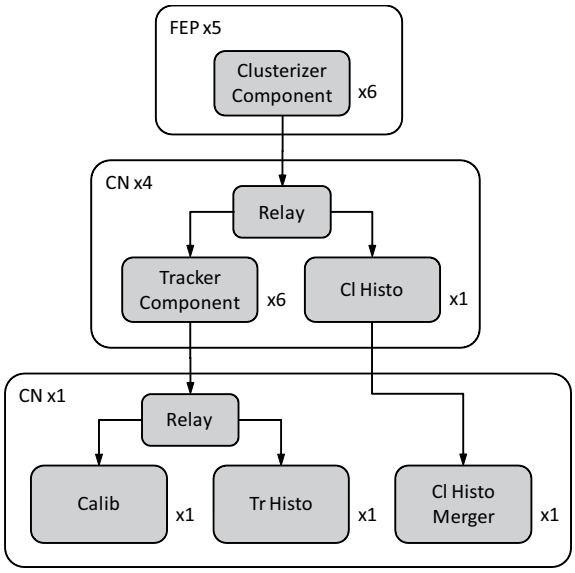


Figure 29: Data flow of the TRD components during the pp run 2010.

defined number of clusters. In this case it sends out the positive triggering answer to the DAQ, which in turn saves the data coming from the detector.

Up to know there are no implementations of physics trigger components, such that the first PbPb run in autumn 2010 might not make use of the triggering facility.

5.4.7 Component Placement

In order to get the highest possible performance, the components of the analysis chain must be diligently arranged onto the available computing nodes. Therefore the processing power of each node and the needed bandwidth for data exchange must be taken into account. Especially as some data has more than one recipient it must be assured that this data is still not being sent out twice. The placement of the components on the available nodes is set up via XML configuration files, which are read in by the HLT framework during the initialization of the run. In Figure 29 an overview of the organization of the HLT TRD components during the pp run 2010 is shown.

The data coming through the DDLs from the supermodules is first processed by the clusterizer components on the five FEP nodes. Up to four supermodules are processed by one computing node. However, during the 2010 pp run only seven of the eighteen supermodules were installed, and thus only up to two supermodules had to be processed by each node. The clusters are transferred from the FEPs to four CN nodes where they are processed by the trackers. On each of the CNs where trackers are executed one instance of the monitoring components is executed, as well. It would have been too much inbound traffic for one node to take the clusters from all FEPs, thus the cluster monitoring component cannot run only on one CN. The histograms of the multiple cluster monitoring components must be merged, this happens on the last CN. There, also the track histogramming component and the calibration component process the input tracks from all trackers. The inbound rate of the tracks is less by around a factor of two compared to the total clusters rate,

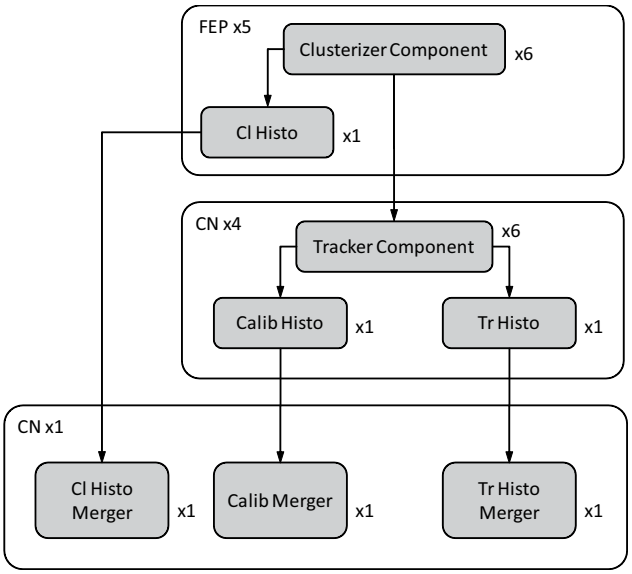


Figure 30: Expected data flow of the TRD components during the PbPb run 2010.

thus it was possible run the processing components on one node only. The additional *Relay* is used for splitting the incoming data on multiple outputs. Without the *Relay* each subscriber would demand for its own data connection to its publisher, thus the FEPs would have to send the same data twice through the network to the CNs: once to the *Tracker Component* and once to the *Cluster Histo Component*. To compete with the required data taking speed of collisions at 2000 Hz three clusterizers and trackers were run in parallel. This includes one component redundancy.

The layout shown in Figure 29 is stable and fast enough, however the usability of this layout is restricted to pp data only. As PbPb collisions generate much higher data flows the layout was further optimised. The ethernet connection of the node running Calibration and Track Histo Component would have been a bottleneck. Figure 30 shows the expected layout for the PbPb run in autumn 2010. To circumvent the network bottleneck at the last CN node, here the calibration and track histogramming is done directly at the trackers' CN nodes. Only the histograms are sent out to the last CN node where those are merged. Additionally the cluster historamming component was moved from the trackers' node to the clusterizers' to equalize the needed processing times. Because of the immense amount of data in PbPb collisions, the number of components needed to process all data would exceed the number of available processor cores vastly. Even including the upgrade to the 24 core nodes the expected processing speed would probably not reach more than about 50 Hz. This means that events must be dropped, and thus the HLT TRD could not perform as trigger. However calibration and monitoring will still be possible.

5.4.8 Offline Compatibility Components

In order to keep the required bandwidth to a minimum, the data which is exchanged between HLT components is saved in data containers holding only the absolutely necessary data members (Chapter 5.4.1). However, it is important to be able to check the quality of the output data during the development. For doing so the monitoring

Component	Speed pp [Hz]	Speed PbPb [Hz]	Memory usage [MB]	Input / Output size
Clusterizer	1500	66	160	2:1
Tracker	1000	16	140	1:1
Cluster Histo	>2000	200	40	-
Track Histo	>2000	>200	45	-
Calibration	>2000	32	50	-

Table 6: The performance of the main HLT TRD components.

components may be used, but a more accurate understanding can be gained by using simulated data. For being able to use quality assurance based on Monte Carlo (MC) information additional data is needed to be exchanged and saved, the so called MC labels. For this, dedicated components have been implemented, which have as input and output format the format used by the offline reconstruction, but process the data just like their online counterparts. The decision to use the data types of the offline reconstruction was taken in order to not unnecessarily overcomplicate the classes of the HLT software, and to really have perfectly compatible output data of the online and offline reconstruction. The offline compatibility reconstruction components inherit from the usual reconstruction components (Chapter 5.4.2) and can thus be used by the same environment without implying other changes than the swapping of the component itself.

Additionally, an *ESD writer component* was implemented. This component gets the tracks from the offline compatibility tracker component and saves those in a file on disk, which is organized exactly like the file used by the offline reconstruction. This enables the use of the Quality Assurance framework to process also the data coming from the components.

5.4.9 Summary of the Components Developed

The components developed interface the offline code to the HLT framework. By the performance increase of the offline code, these components can be efficiently used in the HLT cluster. The speed achieved is enough to be used during the pp run periods with a full reconstruction. During a PbPb run periods online monitoring and calibration is possible. Table 1 summarises the properties of the components. The speed for pp is measured with real data, and the speed for PbPb is measured using simulated data generated by the HIJING [HIJ] peripheral setup.

In regard of the speeds achievable for the PbPb collisions, it is worth noting that a fast clusterizer is much more important than a fast tracker. The trackers can be distributed on many CN nodes, however it is very usefull if the clusterizers are only on the FEP nodes that recieve the data. Thus the amount of network traffic can be reduced.

6 Quality Assurance

In order to reach the required speed of the data taking some processing steps have to be skipped during the HLT online reconstruction. To verify that this does not degrade the output quality too much, quality assurance tests have been performed. A test setup for TRD data existed already for the offline reconstruction. Thus instead of re-implementing an already existing infrastructure, emphasis was laid on making the output of the HLT online reconstruction compatible to the offline reconstruction (Chapter 5.4.8).

The Quality Assurance framework has two modes of operation. The first mode demands for Monte Carlo data attached to all reconstruction entities (clusters, tracklets and tracks). This mode gives a very detailed measurement of the absolute resolution. However as the additional Monte Carlo data is needed, this mode cannot be operational for real data. As especially for real data it is crucial to get at least a rough understanding of the achieved resolution, the second mode of the Quality Assurance framework deals with real data. Of course the detail and accuracy of the analysis is truncated. However, real data is to be reconstructed by the HLT, and thus the difference in quality between the HLT reconstruction and the stand-alone offline reconstruction of this data is decissive.

The QA analysis is performed on data files saved during the reconstruction, which is performed on the HLT test server using the offline compatibility components (Chapter 5.2 and Chapter 5.4.8). The components are interfaced by a compiled macro, which accepts command line arguments. By changing the argument indicating the reconstruction mode (HLT online vs. non HLT offline) all necessary reconstruction parameters describing either mode are set by the macro as arguments to the components (Chapter 5.4.2). The QA analysis is performed separately on the output of both reconstruction modes and the resulting QA plots may then be compared.

The quantities of interest are spatial and momentum resolution and tracking efficiency. However these are only definable if simulated data including Monte Carlo information is reconstructed. By comparison of the reconstructed data with the Monte Carlo data the following quantities are extracted:

- y and z resolution of clusters
- y and z resolution of tracklets
- φ resolution of tracklets
- y and z resolution of tracks
- p_T resolution of tracks
- total tracking efficiency
- particle identification

To get a hint on the achieved quality in real data reconstructions auxiliary quantities must be looked at. In this case these are the residuals of the reconstruction entities:

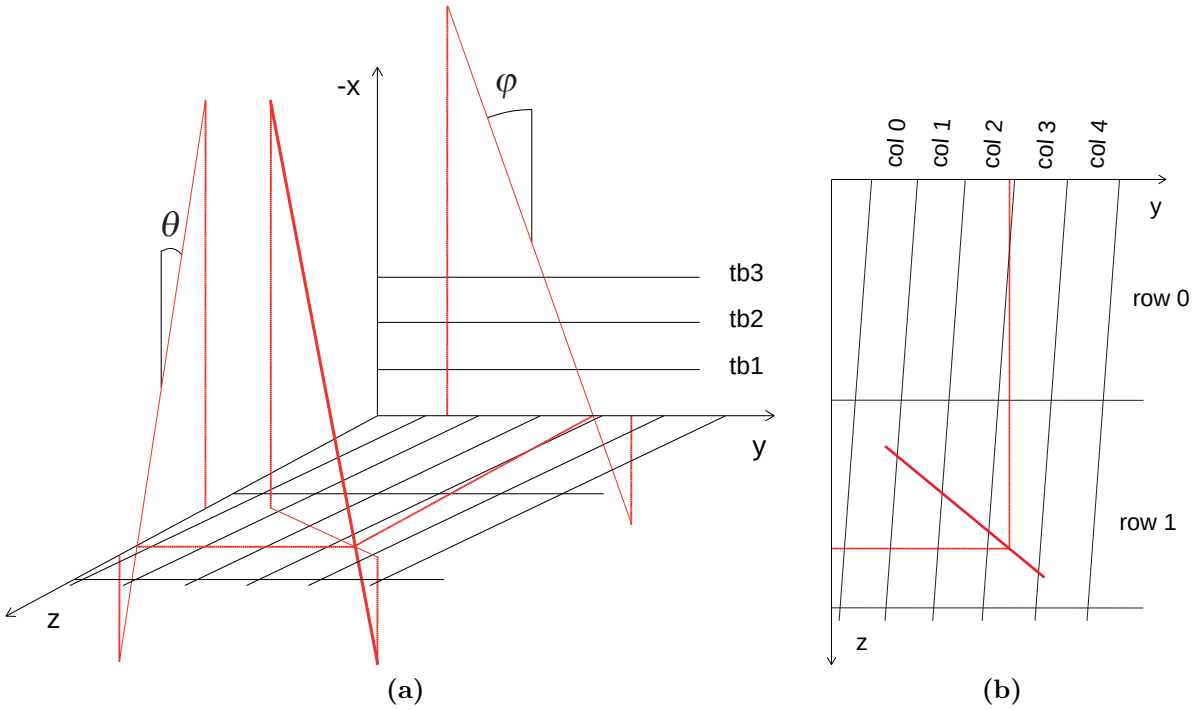


Figure 31: The geometry of a TRD chamber in an axionometric (a) and a top view (b). The z axis is contracted by a factor three. The -x axis points to the vertex of ALICE. The yz-plane contains the pad plane with its tilted pads. The thick red line represents a track going through the chamber. The thin red continuous lines are projections of the track to the axial planes and the thin dotted lines are auxiliary lines helping to visualize the illustration. φ and θ are the two track inclination angles.

- y coordinate residuals of clusters to the tracklet
- y and z coordinate residuals of clusters to the track
- y and z coordinate residuals of tracklets to the track
- φ residuals of tracklets to the track

Before showing the results of simulated and real data in Chapter 6.2 and 6.3 respectively, some TRD specific concepts must be explained beforehand, for understanding what the exact meaning of these observables is and how they are extracted.

6.1 Pre-requirements

The particular geometry of the TRD chambers has consequences for the Quality Assurance. The definition of coordinates and angles is shown in Figure 31. As the pads are very long[21] typically only one pad row senses a point charge in the amplification region. Whereas, in y direction the clusters are usually distributed among three pads. Thus the cluster position has a relatively high accuracy in y direction, as the signals of the three pads which compose a maximum are fit. The z coordinate of clusters is, however, always the middle of the pad, which is the best

[21] typical size of $0.725 \times 8.5\,\mathrm{cm}^2$

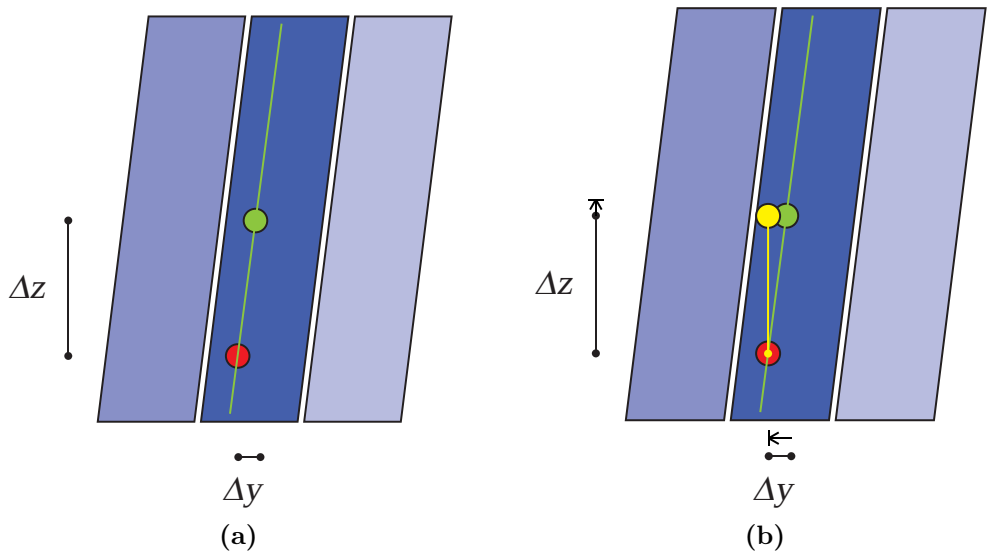


Figure 32: Correlation of the y and z coordinate of clusters due to the tilted pads (a) and how it is resolved in the QA (b). The green spot represents a cluster reconstructed at the optimal y position. However, it has a residual to the track (red spot) in y direction due to the pad tilt.

estimate. As the readout pads are tilted, the measurement of the y coordinate of the cluster is correlated with the (unknown) position in z direction along the pad. The tracklets, which are a straight line fit of clusters inside one readout chamber, have as consequence exactly the same problem. By fitting multiple tracklet of different chambers to build the track, this issue is resolved by the fact that the chambers of a stack have alternating pads tilt. Thus the effects of particular chambers cancel out.

In Figure 32a the principal problem of the correlation of the y and z coordinate is sketched. The red spot represents the position at which the particle hits the detector. The three pads which sense the ionization electrons of this particle are shown with their individual signal strength indicated by the intensity of the blue colour. By fitting the signals with the pad response function, the cluster is reconstructed along the green line. The best estimate of the z coordinate is the middle of the pads, thus the cluster is placed at the green dot. However, when calculating the y resolution of the clusters, this position is not meaningful. The position of the real track with respect to the pad is equally distributed along the pad. This means that even clusters which are perfectly positioned in y direction, have nevertheless a finite residual in this direction, which is only due to the uncertainty in the z coordinate. This effect must be eliminated before any quality assurance can be performed.

The measured distance of a cluster to the track shall be Δy and Δz and the angle of the pads' tilt α. There is per definition no distance in x direction as it is the independent variable. The correlation of the y and z coordinate is eliminated by a rotation of the reconstructed cluster (green) around the track position (red), see also Figure 32b:

$$\Delta y_{corr} = \Delta y \cdot \cos\alpha - \Delta z \cdot \sin\alpha \approx \Delta y - \alpha \cdot \Delta z$$

$$\Delta z_{corr} = \Delta y \cdot \sin\alpha + \Delta z \cdot \cos\alpha \approx \Delta z + \alpha \cdot \Delta y$$

Given the fact that α is very small (2°), and usually $\Delta y \ll \Delta z$, there is no significant loss in precision using the approximate formulas above.

Of course the same process must be done for the tracklet coordinates. The tracklets have two proposes: updating the Kalman track during tracking and for calibration. The update of the Kalman track is done using one point of the tracklet, the so called reference point of the tracklet. Thus the space residuals of the tracklets to the track are defined as the distances of the (tilt corrected) reference point of the tracklet to the track, at the x coordinate of the reference point. However, as tracklets are straight lines, it is not directly obvious how this point should be defined. There are two definitions used: one is for tracklets which are based on clusters from one pad row only, the other definition is used for tracklets crossing two pad rows. As the pads have a mean length of 8.4 cm, but the readout chamber has a height of only 3.7 cm, the former group includes most of the tracklets (about 75%). In this case the following definition is used: The reference point of the tracklet is that point along the tracklet, where the y uncertainty of the tracklet is minimal.

To see that this point indeed exists and how it is calculated, let's quickly recapitulate how a straight line fit is defined.
The general problem is defined by the equation

$$X\vec{b} = \vec{y} \quad \text{, where } \vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad , \vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \quad , X = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}$$

Here x_i and y_i are the coordinates of the n clusters based on which the tracklet is fit. The maximum number of clusters is determined by the number of time bins. The parameters of the straight line fit which describes the tracklet's spatial position are $b_1 = y_0$ and $b_2 = -\frac{dy}{dx}$. The solution is given by

$$\vec{b} = \left(X^T X\right)^{-1} X^T \vec{y}$$

The straight line fit is of the form

$$y\left(x\right) = y_0 - x \cdot \frac{dy}{dx}$$

The uncertainty of the parameters can be approximated[22] by

$$E = \begin{pmatrix} \sigma_{b_1}^2 & \operatorname{cov}\left(b_1, b_2\right) \\ \operatorname{cov}\left(b_2, b_1\right) & \sigma_{b_2}^2 \end{pmatrix} = \frac{S}{n-2}\left(X^T X\right)^{-1}$$

where S is the sum of squared residuals. This will however cancel out during following minimization, thus it can be disregarded here. Finally, the uncertainty of the tracklet is given by

$$\sigma_{y(x)}^2 = \sigma_{y_0}^2 + x^2 \sigma_{dy/dx}^2 + 2x \cdot \operatorname{cov}\left(y_0, \frac{dy}{dx}\right)$$

[22] this approximation does not include weights, in reality the weights are considered and are given by the inverse of the cluster variance

which has a minimum at

$$x^* = \frac{\mathrm{cov}\left(y_0, \frac{dy}{dx}\right)}{\sigma^2_{dy/dx}}$$

Since the cluster uncertainties increase with their distance to the anode wire, this minimum is usually in the first third of the drift region.

The described tracklet fit fits the tracklet in the xy plane. For tracklets which cross pad rows, an additional fit in the xz plane is performed. This leads to the coordinates at which the tracklet crosses the pad rows, which is finally the reference point of this sort of tracklets.

6.2 QA based on Simulated Data

All information needed for the QA is gathered by calculating residuals of specific quantities between different entities of the reconstruction to other such entities, or to the markers of the simulation. The entities are of course the reconstructed clusters, tracklets and tracks and the compared quantities are the attributes assigned to them during the reconstruction. How these measured differences lead to the resulting plot shall be explained in this Chapter.

The markers of the simulated data are the space points on the trace of the simulated particle, at which it intersects the boundaries of the TRD chambers. Thus for each particle there are two markers in each readout chamber it crosses. The straight line between two space points gives the notion of direction of the MC particle inside a chamber. All MC markers of one trace share a number which identifies their belonging, the MC label. During the reconstruction of simulated data, which includes these MC markers, clusters are assigned the MC label of the particle they originate from. The label of tracklets and tracks then calculated based on the clusters. A track might get reconstructed with clusters from different MC particles, in this case the majority of clusters defines the label of the track. Actually even clusters might originate from different MC particles, this is why clusters can be assigned up to three labels.

In order to get good statistics and thus reliable results a minimal amount of data must be processed. In this case a relatively small amount of data is already enough. About 10000 tracks, with their tracklets and clusters of a reconstruction session are necessary. The extraction of most quantities follows comparable principles, thus will be shown once using the example of the cluster resolution in y direction.

Cluster Resolution in y Direction

All resolution plots are based on the difference of an attribute of the reconstructed entity to the simulated markers. In the case of the y resolution of clusters, the entity is the cluster and its y coordinate is the central attribute. A histogram is filled containing the y distance of the cluster to the MC particle at the x position of the cluster. However, as we have seen in the previous Chapter, not the raw y distance must be observed, it must first be corrected for the pad tilt. The correction is based on the trace of the MC particle, as it is the reference for all measurements in this part of the QA.

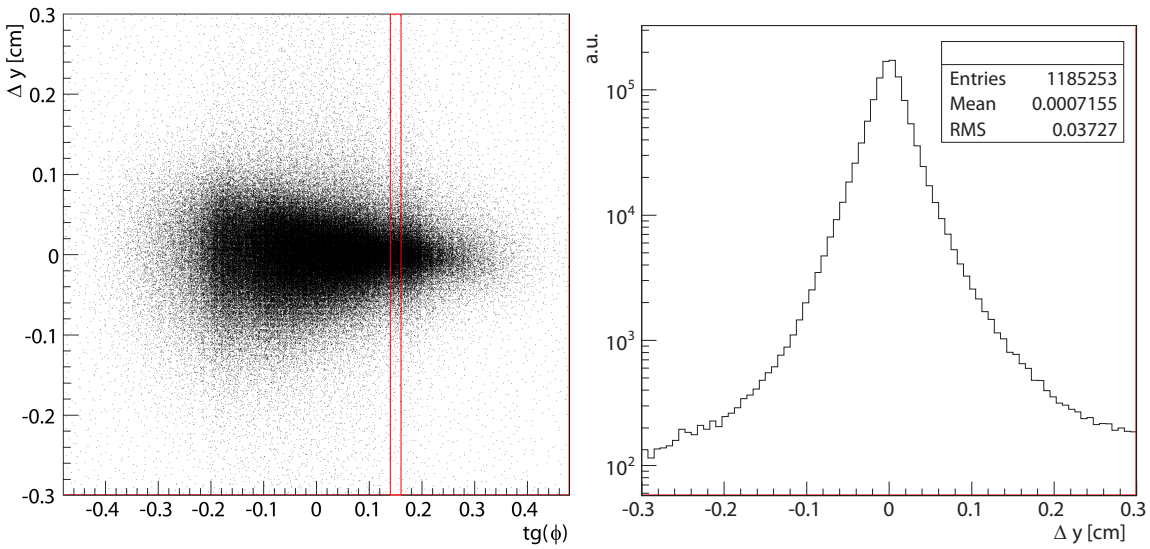


Figure 33: (a) Distribution of the tilt corrected distance of cluster to the MC particle. (b) The projection of the red marked slice and its Gaussian fit.

A histogram of this distance distribution is shown in Figure 33a. The tilt corrected distance in y direction is plotted against $\tan\varphi$, where φ is the inclination of the MC particle in y direction. The histogram is then divided into $\tan\varphi$-slices, and each is projected onto the ordinate axis. The mean of this resulting distribution should be zero, as it is expected that (including all corrections during the reconstruction) the clusters are normally distributed around the particle trace. The root mean square (RMS), or sigma, of the distribution represents the resolution. These two quantities, the mean and the sigma, are finally shown in the resulting plot. Thus this contains the most important properties of the distribution.

Figure 34 shows the plot for the HLT reconstruction and the offline stand-alone TRD reconstruction. It can be observed that the data points are exactly on top of each other. This is expected, as the tiny quality-to-speed trade-off which has been included into the clusterizer during the HLT reconstruction can show up only at the tracklet level. The positions and errors of the clusters are calculated by the HLT including all higher order corrections.

The curve of the sigmas has a minimum at the Lorentz angle α_L. This is the angle at which clusters drift towards the pad plane, due to the interaction of the electric field of the TRD readout chamber and the magnetic field of the L3 magnet (Figure 35b). Electrons originating from particle traces inclined differently, have a reduced resolution due to the Landau distribution of the charge deposition along the ionizing particle trace. Around a big charge deposition other near smaller charges are shifted (Figure 35a). This effect is proportional to $\tan(\varphi - \alpha_L)$. In fact many effects concerning the resolution are proportional to the tangent of the inclination angles, both φ and θ. The reason is that the length of the projection of the particle's trajectory during the elapse of a time interval is proportional to the tangents of the inclination.

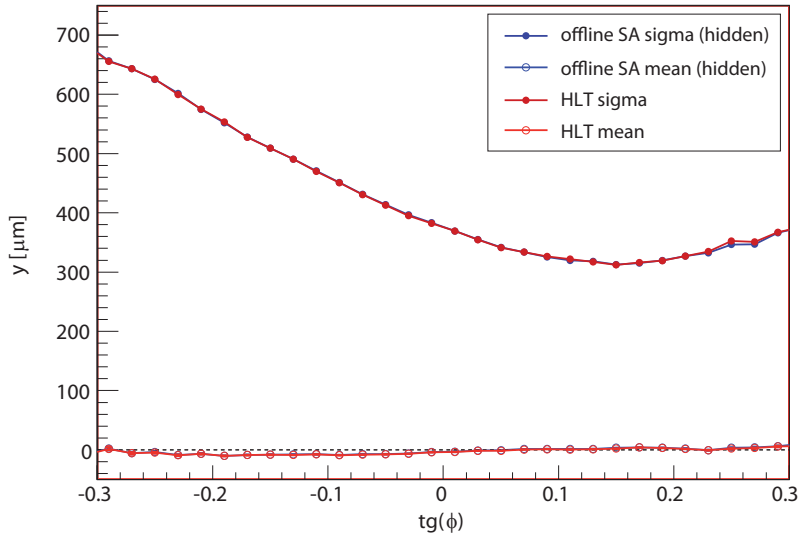


Figure 34: Resolution of the clusters in y direction, as compared with simulated data.

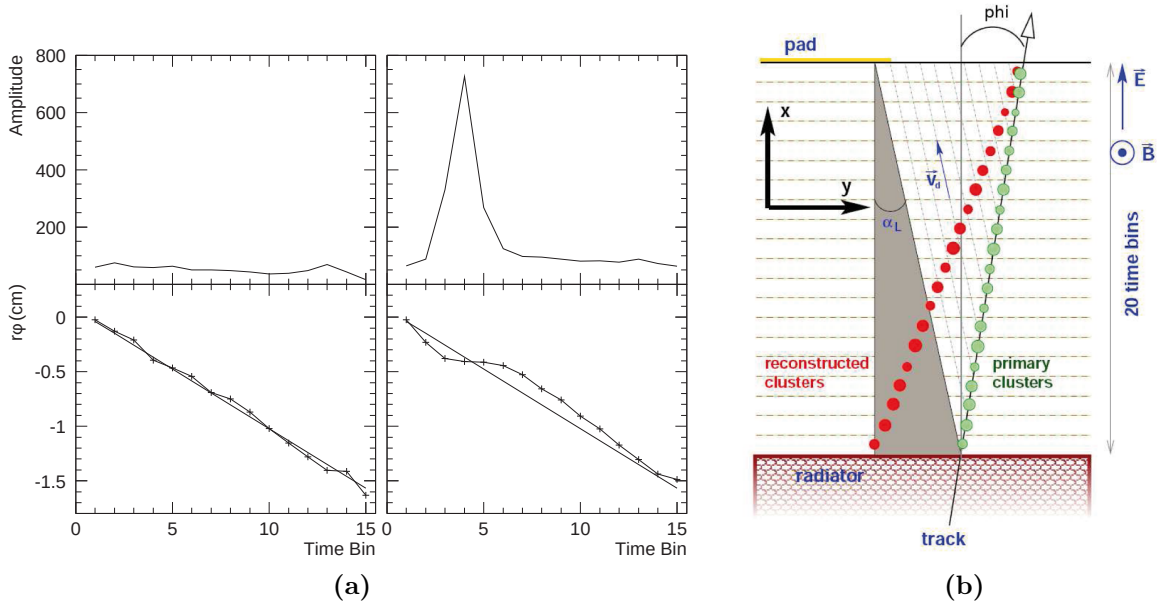
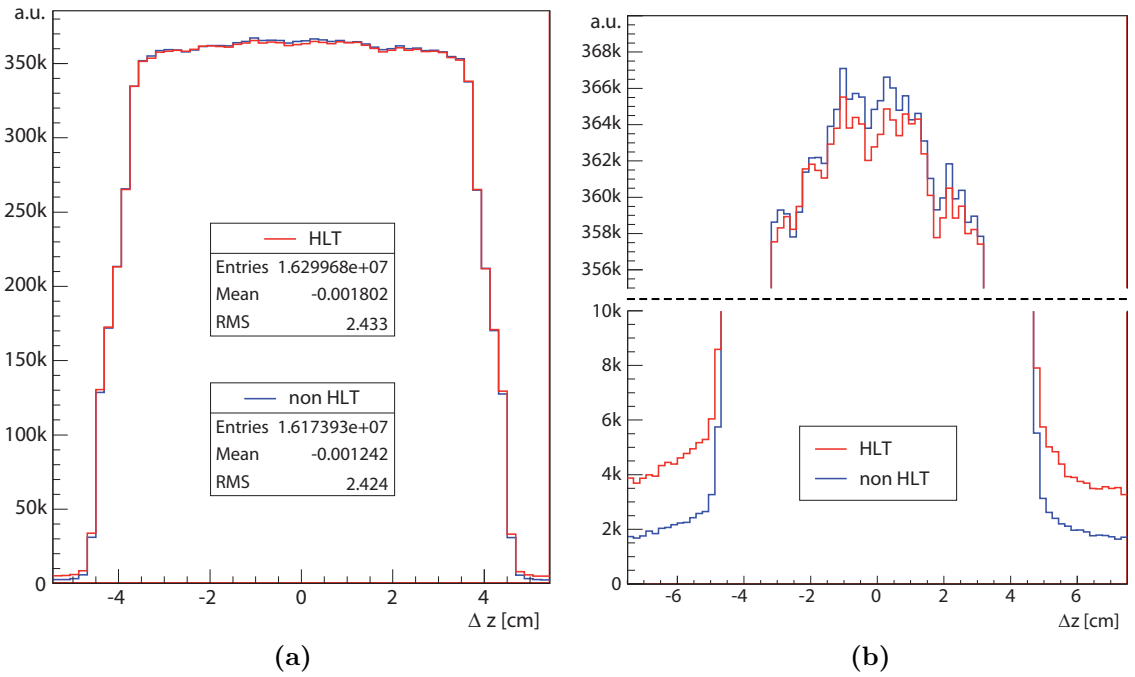


Figure 35: (a) Cluster charges are Landau distributed: big charges shift neighbours. [ATDR2] (b) Ionisation electrons are drifting at an angle 90°-α_L to the pad plane due to the interaction of the magnetic and electric field.



(a) **(b)**

Figure 36: Distribution of the cluster z coordinate residuals to the MC trace (a) and two zoom ins of the bottom and top part (b).

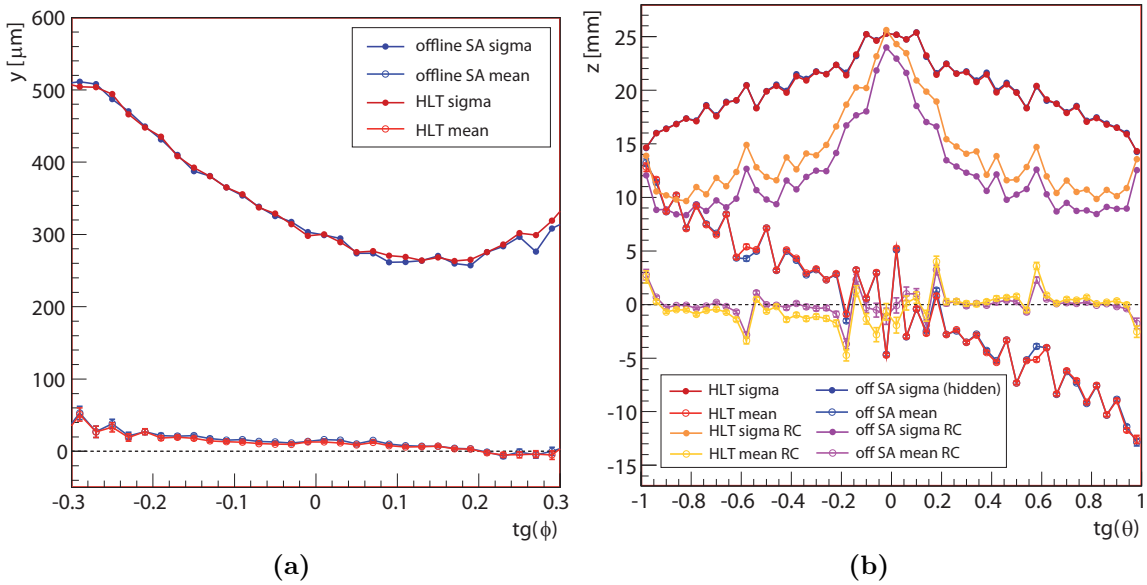
Cluster Resolution in z Direction

Due to the geometry, the resolution in z direction is very limited. For clusters, the resolution is given only by the pad length, which ranges from 7.5 to 9 cm with a mean value of 8.4 cm. Figure 36 shows the distribution of the distance of tilt corrected z coordinate of the cluster to the z coordinate of the MC particle. As in this case there is anyhow no dependency on $\tan(\theta)$, it is much more illustrative to see the raw distribution.

As expected, the residuals are equally distributed along the whole pad length (maximal deviation is about 2%), since clusters are reconstructed at the middle of the pad but the particle might hit the pad at any point. The RMS of the distribution is very well in line with the expected value, due to the mean pad length: $8.4\,\text{cm}/\sqrt{12} = 2.42\,\text{cm}$. The plot based on the HLT reconstruction differs slightly from the one of the offline stand-alone reconstruction, as seen in the zoomed in plots. This difference is due to the tracking. The MC label of the track is calculated out of the MC labels of all clusters, and then all distances of the clusters are calculated to this MC particle. The differences are due to clusters which are assigned to wrong tracks. Obviously the HLT reconstruction has a higher count of misassigned clusters.

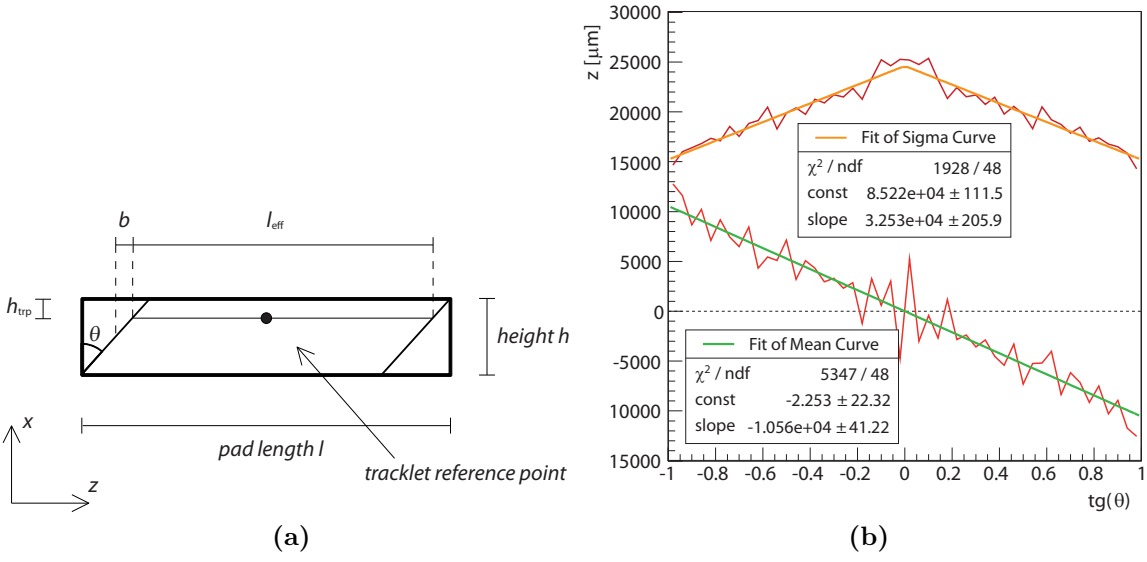
Tracklet Resolution in y Direction

The same procedure as it is used for clusters can be also applied to tracklets. Here, however, the distances are measured from the tilt corrected tracklet reference point to the MC trace. Thus Figure 37a shows the mean and sigma of the distribution



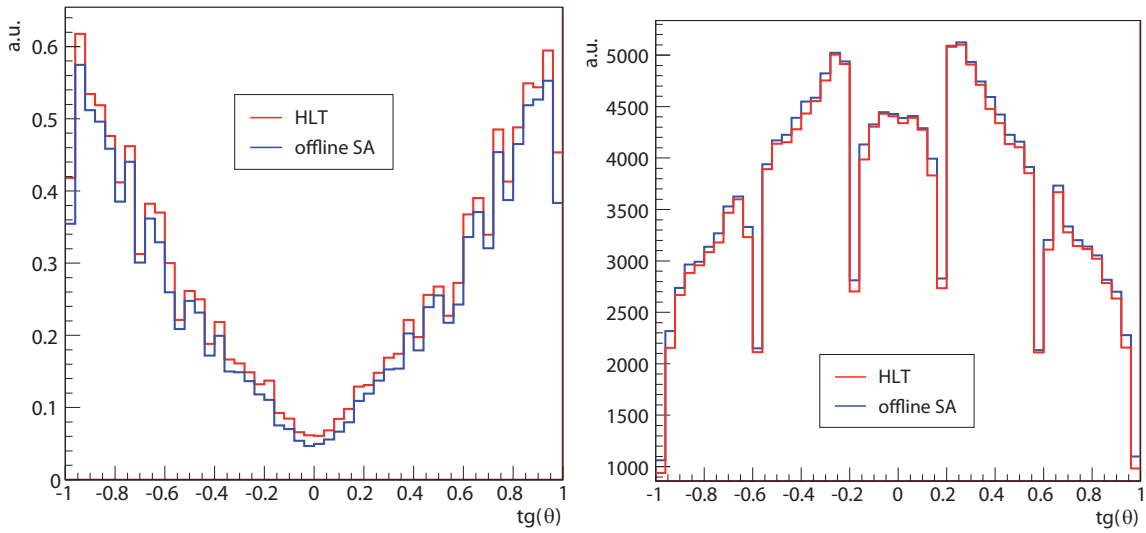
(a) **(b)**

Figure 37: Resolution of the tracklets as compared with simulated data in y direction (a) and in z direction (b).



(a) **(b)**

Figure 38: (a) Illustration of geometry in the x-z plane of a readout pad.
(b) Fits on the mean and sigma curves of Figure 37b for tracklets not crossing pad rows.



(a) **(b)**

Figure 39: (a) Probability of row crossing tracklets as function of $\tan\theta$.
(b) Number of reconstructed tracks as function of $\tan\theta$.

of this distance versus the tangent φ of the MC trace. As expected, the resolution of the tracklets is better than that of individual clusters, as the fit of one tracklet includes many clusters. The general look of the plot is also comparable, as tracklets from a $\tan\varphi$-region should include only clusters from the same region.

Tracklet Resolution in z Direction

As already mentioned, the tracklets are handled differently, depending on whether they cross the pad rows or not. The additional fit in the xz-plane, which is performed in case the pad row is crossed, increases the resolution of the z coordinate significantly, as it can be seen in Figure 37b. Of course, the probability of having a row crossing tracklet drops towards $\theta = 0$, this is shown in Figure 39a. Additionally, the running mean is also removed completely for those tracklets, the spikes that remain correspond to regions of low acceptance, which is shown in Figure 39b by the number of reconstructed tracks. The loss in precision of the HLT reconstruction in case of the row crossing tracklets, can be attributed mainly to the skipped iterative procedure of improving the tracklet quality. As explained in Chapter 5.3.3 the procedure is skipped since it is very expensive in terms of processing time.

The reason for the running mean of the tracklets not crossing pad rows is the position of the tracklet reference point. This point is usually in the first third of the drift region, and thus the z position is biased for inclined tracks. The systematic decrease of the sigma for these tracklets is also due to the inclination of the track in the readout chamber. The illustration in Figure 38a shows the situation. A track which is inclined at the angle θ and does not cross pad rows, cannot be outside of the length l_{eff}. Thus the resolution of the tracklet increases with increasing angle. However, the position of the tracklet reference point along the x axis is typically not in the middle of the drift region but in its first third, thus in proximity to the amplification region. In the z direction the tracklet reference point is along the

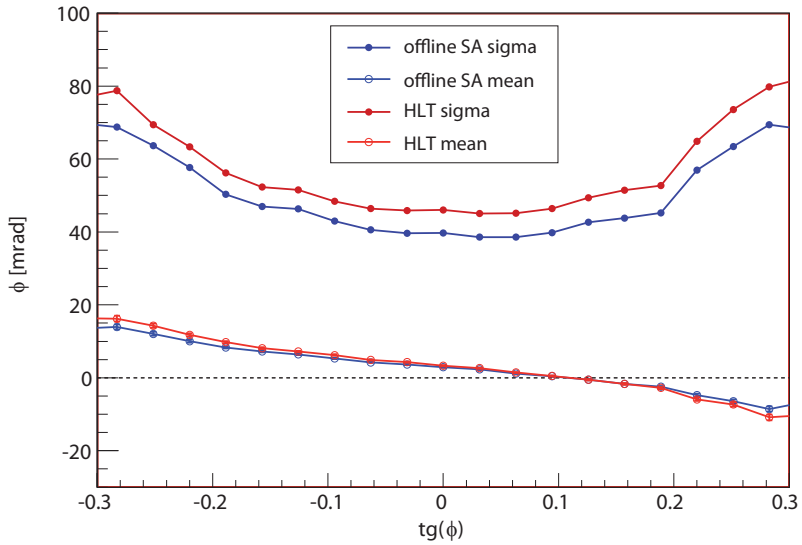


Figure 40: Resolution of the tracklet inclination, as compared with simulated data.

clusters, thus exactly in the middle of the pad. This combination of being in the middle of the pad along z but not in the middle of the chamber along x, generates a bias b which linearly depends on the tangents of track inclination. The resolution curve can be described by

$$\sigma(\theta) = \frac{l - h \cdot |\tan\theta|}{\sqrt{12}}, \tag{3}$$

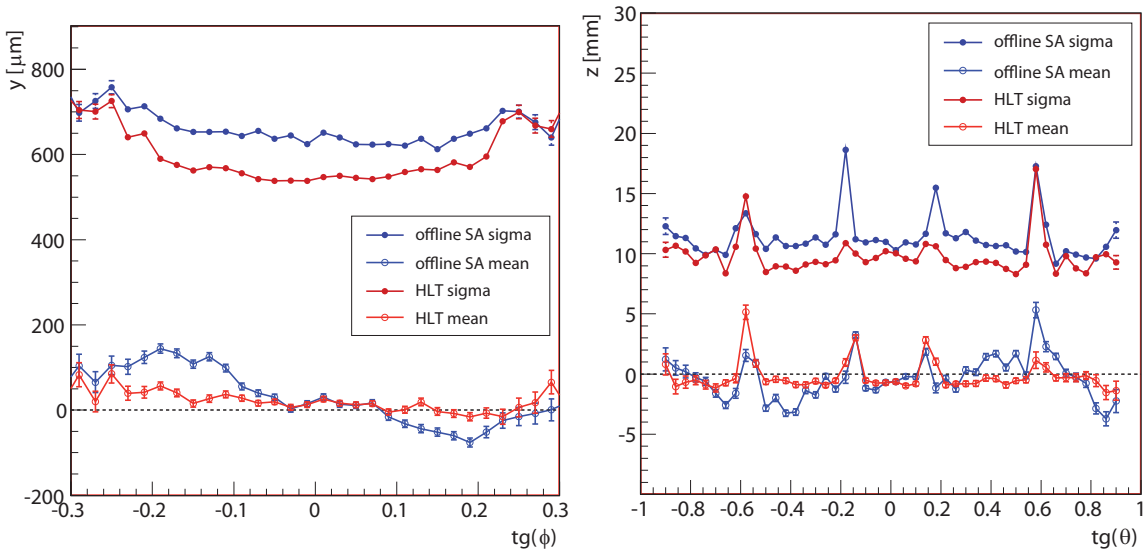
where l is the pad length, h is the height of the drift region. The length l_{eff} shown in Figure 38a equals to $h \cdot |\tan\theta|$. The fit of Equation 3 on the sigma curve is shown in Figure 38b. The return value of 8.522 cm for the pad length matches the value of 8.4 cm for the mean pad length quite well. The return value of 3.253 cm represents the height of the drift region, which is in reality 3 cm. The linear fit of the mean curve, which is also shown in Figure 38b returns value of 1.056 cm for the bias for tracks with an inclination of 45 degree. With these values, it is possible to calculate the mean x coordinate of the tracklet reference point using this expression

$$h_{trp} = \frac{h}{2} - b \tag{4}$$

This gives a value of 0.5705 ± 0.0113 cm for the mean height of the tracklet reference point above the amplification region. This is indeed inside the first third of the drift region.

Tracklet φ-resolution

Increased uncertainties of the y coordinate of the clusters not only lead to increased uncertainties in the y coordinate of the tracklets, but also in the inclination angle along this coordinate: the φ angle. Thus the systematic increase of the sigma to the edges of Figure 40. The visible loss in resolution for the HLT reconstruction has many causes, but the most important one is again the skipping of the multiple



(a) **(b)**

Figure 41: Resolution of the tracks at the last readout chamber as compared with simulated data in y direction (a) and in z direction (b).

tracklet fits. Additionally, the simplified error propagation of the clusters towards the tracklet also has an effect.

What may also have a small effect is the skipped count of active pads neighbouring the clusters. As explained in Chapter 5.3.2, this number is used by the tracklet fit to artificially decrease the weight of the respective cluster if it has more than five active pads around. This is based on the knowledge that such configurations are extremely unlikely and could indicate a detection flaw. For simulated data this is of course not the case, but still such clusters might be originating from two crossing traces. In this case a charge sharing of the two neighbouring clusters is taken into account, however the clusters might still influence each other.

The running mean is a feature, which is not understood. But since offline stand-alone and HLT behave the same, it is not due to the altered HLT reconstruction.

Track Resolution in y Direction

The QA procedure for tracks is in principle exactly the same as for clusters or tracklets. However, for tracks no tilt correction is needed, as the alternation of tilting angles in subsequent chambers of a stack should cancel out. Each track is compared in each chamber to the MC data separately. Thus for tracks there is an output plot for each of the (mostly) six chambers the particle passed through. The resolution at chambers higher in the stack is better, since in the stand-alone reconstruction the tracklets are updating the Kalman filter beginning with the lowest chamber. This is why in the full offline reconstruction tracks are fit multiple times, from inward to outward and back.

As explained in Chapter 5.3.3 the helix fit based on the tracklets is different in the HLT reconstruction. There, a helix constrained to the origin of the ALICE coordinate system is used. This is due to the fact, that the bulk of the tracks is

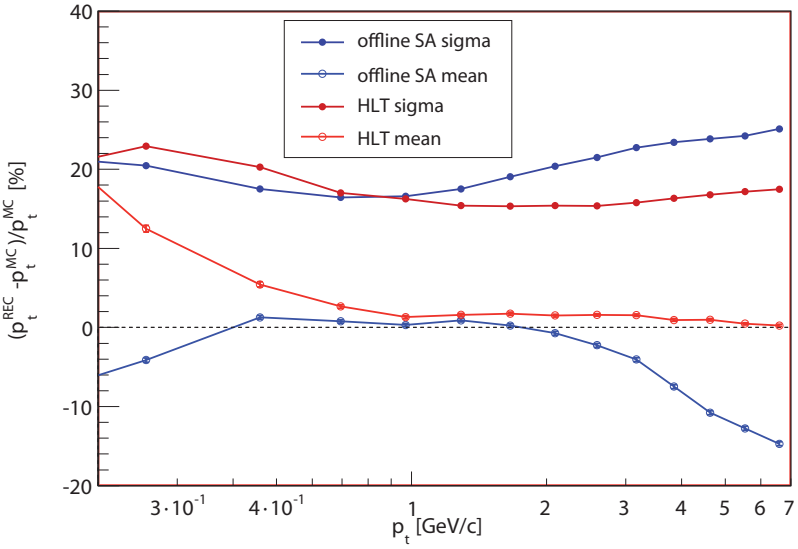


Figure 42: p_T-resolution of the tracks, as compared with simulated data.

coming directly from this direction. Only tracks originating in conversions inside the TRD have no prolongations to the primary vertex, and these are not of high interest. Thus this approximation does not influence the physics result too much.

Looking at Figure 41a, however, clear differences can be seen. The unconstrained helix fit used in the offline reconstruction seems to cause a deterioration of the resolution.

Track Resolution in z Direction

Figure 41b shows the resolution of the z coordinate of the track. As expected, it is better than the resolution of single tracklets. The position of the spikes corresponds to regions of low acceptance (Figure 39b). Obviously on track level the running mean of the tracklets which do not cross pad rows (shown in Figure 37b) can be compensated by the track fit.

The fact that the HLT has again a better resolution than the offline stand-alone reconstruction is also due to the different helix fits.

Track p_T-resolution

An important measurement of the TRD is the transverse momentum p_T. In Figure 42 it is again clearly visible, that the two helix fits behave differently. In the very low p_T region, the resolution of the offline stand-alone reconstruction performs slightly better. Above $p_T \approx 1\,\mathrm{GeV/c}$ the HLT reconstruction with its constrained helix fit starts to be superior, and over about $p_T \approx 3\,\mathrm{GeV/c}$ the offline stand-alone fit degrades substantially. However, for the offline reconstruction including all detectors the TRD stand-alone tracker is used only for clusters which were not tracked during the barrel tracking. Thus all clusters belonging to tracks with prolongations into the inner detectors have been removed at this stage, and only tracks originating in conversions inside the TRD remain to be tracked. These tracks usually do not have a high p_T and thus the impact of this issue is not very big.

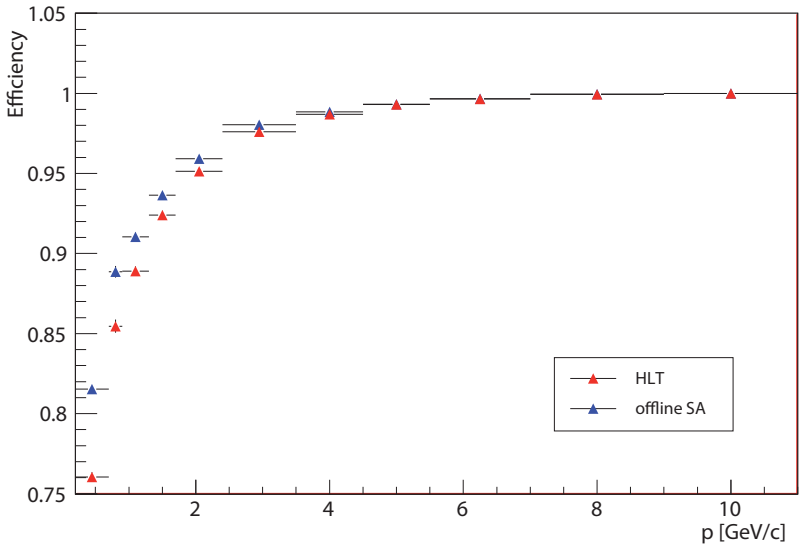


Figure 43: Tracking efficiency.

Tracking Efficiency

An algorithm must not only be precise, it must also be efficient. For the reconstruction this means that not only the quality of the tracks is of importance but these tracks must really be found with high efficiency. Fake tracks, i.e. tracks which are reconstructed even though there was actually no particle in the detector, or tracks which are not found at all are very hard or impossible to handle afterwards on event by event basis.

Tracking efficiency is defined by *found tracks / findable tracks*. Only tracks meeting the following criteria are considered:

- At least 4 of the 6 chambers are hit
- total momentum > 0.2 GeV/c
- φ, $\theta < 50°$

A track is declared found if it has a MC label assigned which exists for the corresponding event. The result is shown in Figure 43. The total dependency on the total momentum p is expected, as tracks with low momentum are very tight helices and these are of course harder to find. The decisive quantity is not p_T, as for example tracks with a low p_T but a high total momentum are much straighter inside one TRD stack than tracks with the same p_T but a low total momentum. The reason that the HLT reconstruction has a lower efficiency is a combination of the following performance increasing measures: the lower number of used seeding configurations, the skipped refits of tracklets, and the thus skipped quality sorting of the seeding candidates. (for more details, see Chapter 5.3.3).

Particle Identification

The discrimination of pions and electrons by making use of the emitted transition radiation of the electrons is the unique feature of the TRD. The design specification

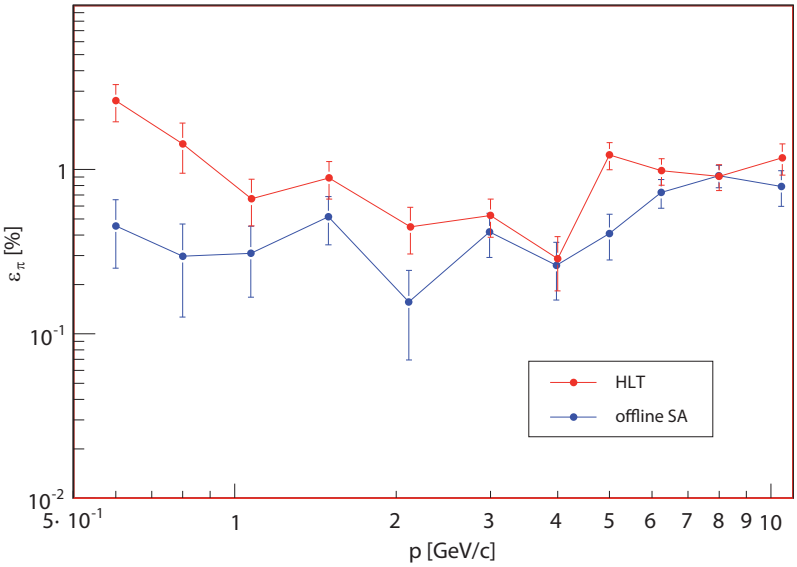


Figure 44: Pion contamination at an electron electron efficiency of 90%.

of the TRD was defined as being 1% pion contamination at an electron identification efficiency of 90%. The actual result of the online and stand-alone offline TRD reconstruction can be seen in Figure 44. The observable degradation of the particle identification can be explained by the less accurate tracking. The multiple refits of tracklets and track candidates are very time consuming and have to be skipped. The procedure providing the particle identification is not changed, but already a few mismatched clusters due to the less accurate tracking can harm.

6.3 QA based on real data

All quality assurance plot shown up to now are only producible with simulated data. To be able to check the quality of real data, only information available there may be used. Thus only residuals between the reconstructed entities themselves can be calculated, which means that thereby only relative measurements are possible. However, by comparing these relative measurements of simulated and real data, and including the information based on the MC markers of the simulated data, the quality of the real data can be estimated.

What follows is a comparison of plots extracted from the same simulated data as in the previous chapter, with real data from the pp run 126405.

Cluster to Tracklet Residuals in y Direction

Clusters are also for the real data analysis the starting point. As clusters are the first reconstruction entity being calculated, anything else is based on them. Additionally, the information needed by the clusterizer to calculate the clusters is from a very small region of the detector, thus the calculations are very straight forward and should be less prone to errors than for the other entities.

The residuals of clusters to tracklets can reveal problems of the tracklet fit and, since this is a relative measurement, also of the clusters themselves. Figure 45 shows

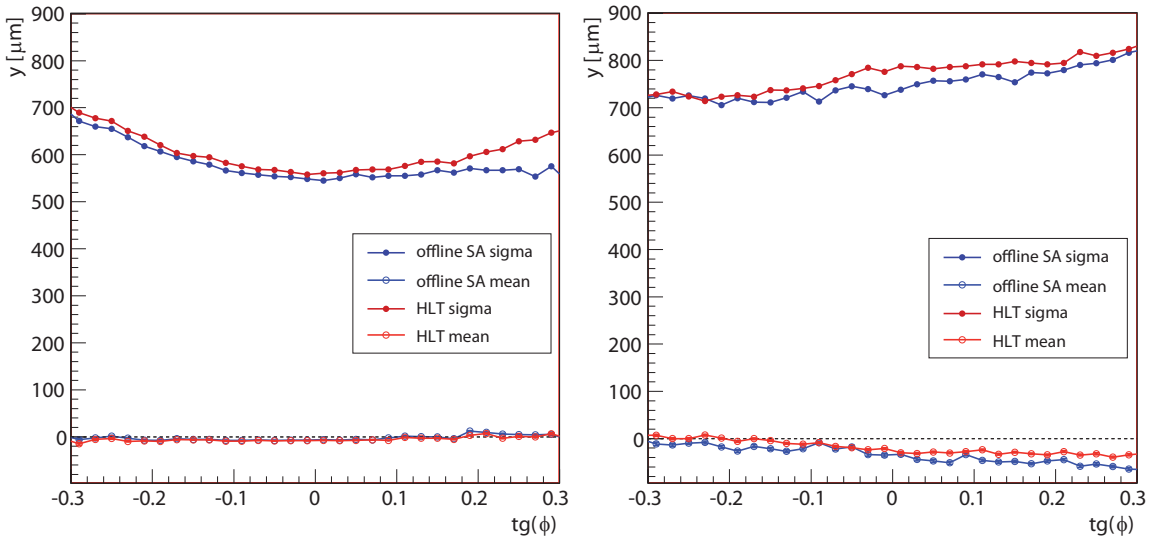


Figure 45: Residuals of the y coordinate of clusters to tracklets, for simulated data (a) and real data (b).

the results for the simulated data and the real data. The first obvious difference is the running direction of the sigma curve. This difference is due to the magnetic field which, in this run, had the inverse polarity. This curve is also somewhat higher for the real data, which means that the distribution of the residuals is wider. Additionally the dip of the curve at the Lorentz angle is less pronounced in the real data plot. This could be due to an increased mixing of clusters from one tan φ-region to tracklets of another tan φ-regions, i.e. misassignment of clusters to tracklets.

In general, the plot of the real data is comparable to the plot of the simulated data, thus, most probably, no dramatic errors are introduced during the tracklet fit of the real data.

The difference of the sigma curves of the HLT and offline reconstruction in case of the simulated data is surprising. As Figure 34 and Figure 37a show, that neither clusters nor tracklets have a worse resolution in the HLT reconstruction. This feature can be seen on all following plots and could be due to an increased misassignment of clusters to tracklets and subsequently of tracklets to tracks.

Cluster to Track Residuals in y Direction

The measurement of the residuals of clusters to tracks is very important, since tracks are the most important entity. As tracks are based on multiple readout chambers, the cluster coordinates must be corrected for the pad tilt before any calculation is done. The reference for this correction is the reconstructed track.

Figure 46 shows the resulting properties of the distribution of the tilt corrected residuals of clusters to tracks. First of all, the general height of the sigma curves is comparable, but the dip of the curve is not very pronounced in the real data plot. In fact this seems plausible, as this matches with the observation of the residuals of clusters to tracklets. However, looking at the mean curves a real problem shows up

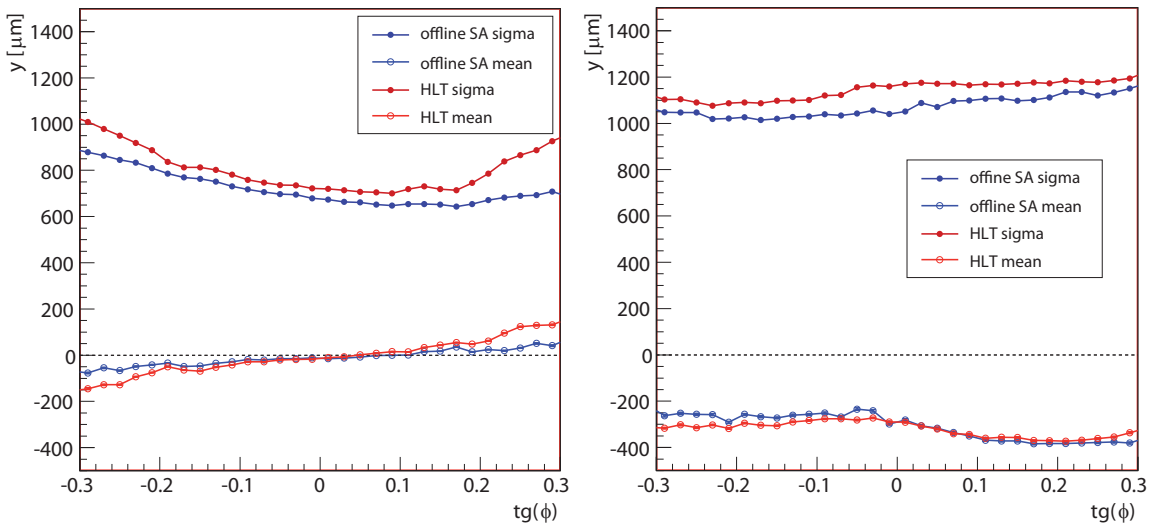


Figure 46: Residuals of the y coordinate of clusters to tracks, for simulated data (a) and real data (b).

for the real data here: the mean is systematically shifted. By the fact that there is no hint about such a serious problem in the cluster to tracklet residuals, it could have been argued that the errors are introduced during the track fitting. But we will see that this is improbable. The residuals of tracklets to tracks will give more information on this issue.

Assuming that the reconstruction precision is unchanged for individual clusters, the comparable height of the sigma curves of the real and simulated data indicates that the precision of the reconstructed tracks is also comparable (apart from the systematic shift, of course).

Cluster to Track Residuals in z Direction

In the context of the problems with the residuals of the y coordinate, it is worth looking at the residuals in the z coordinate. However, Figure 47 gives no evidence for any kind of problems in the z direction, as there are no big differences of the plots for simulated and real data. Especially important is that the mean curves have no systematic shift. The sigma curves are all higher than the expected 2.4 cm, which means that clusters of different pad rows are wrongly assigned to tracks. The dip in the middle is expected, as tracks exactly at mid-rapidity have the possibility to hit all chambers at the same pad row. In this case the track will be reconstructed exactly along the clusters and then the residuals are zero. The offline reconstruction of the simulated data, however, does not show this sudden drop of the sigma curve. This feature is not understood up to now.

The distributions for the real and simulated data are very similar, thus the achieved precision of the real data reconstruction should also be similar.

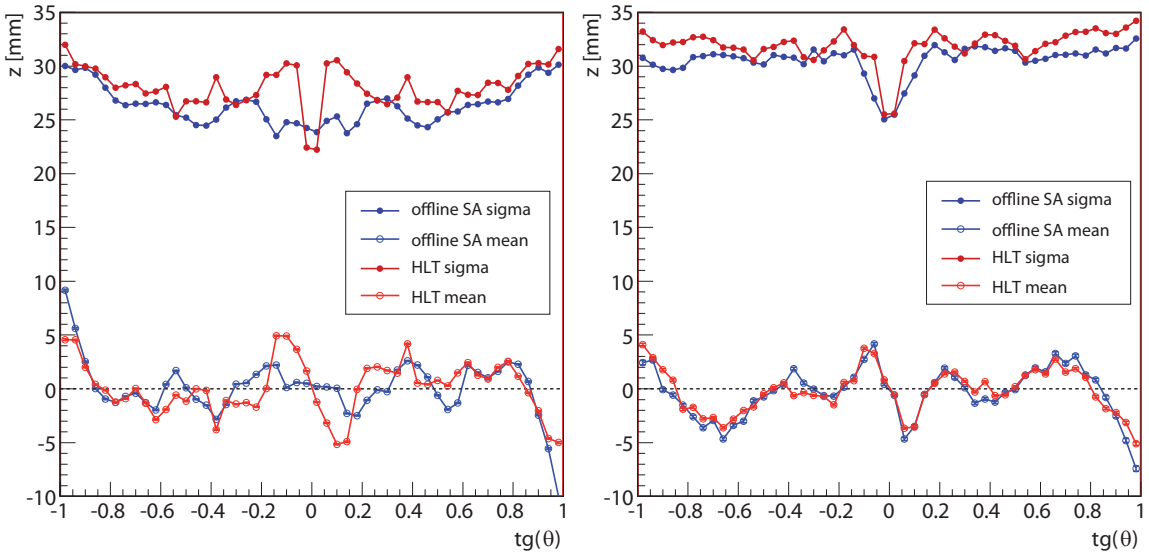


Figure 47: Residuals of the z coordinate of clusters to tracks, for simulated data (a) and real data (b).

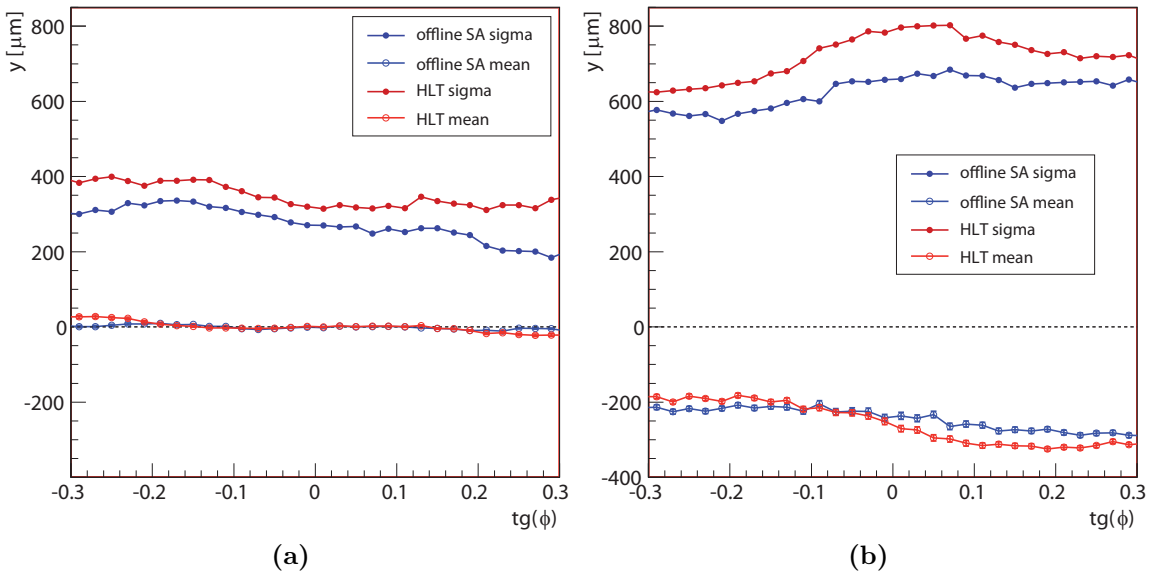
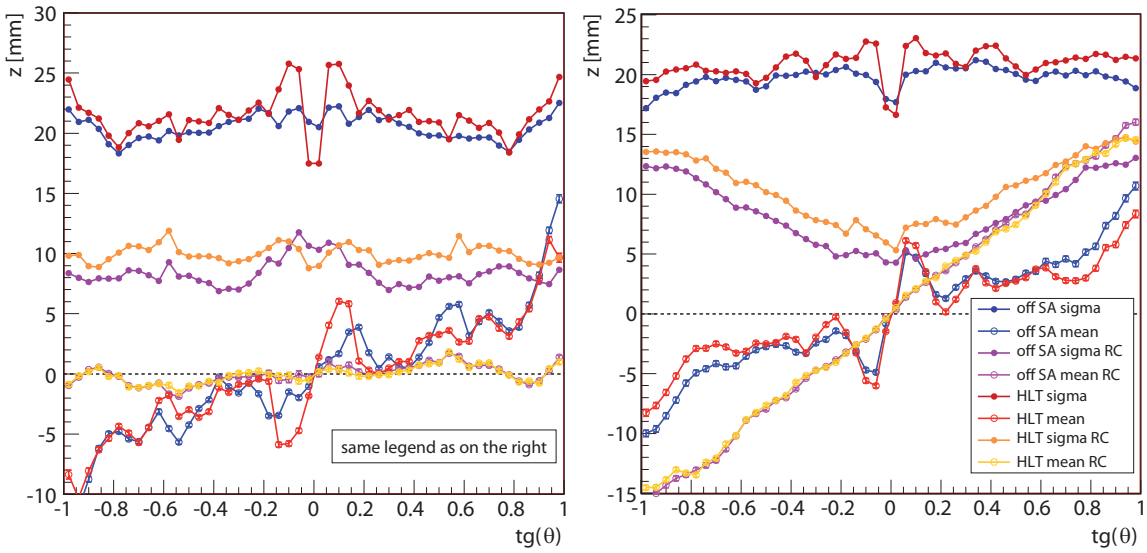


Figure 48: Residuals of the y coordinate of tracklets to tracks, for simulated data (a) and real data (b)



(a) **(b)**

Figure 49: Residuals of the z coordinate of tracklets to tracks, for simulated data (a) and real data (b)

Tracklet to Track Residuals in y Direction

Concerning the problem of the y residuals of clusters to tracks in case of the real data reconstruction, the measurement of the residuals of tracklets to tracks can give valuable information. Figure 48 reveals that the same amount of systematic shift as previously seen in Figure 46 is present. This could be another hint that the actual error is introduced during the fit of the track. But this possibility is mainly ruled out by the findings on the φ angle.

Tracklet to Track Residuals in z Direction

Figure 49 shows the properties of the distribution of the tracklet to track residuals in the z coordinate. The improvement in precision for the pad crossing tracklets can be clearly seen by comparing the sigma curves.

The running mean of the tracklets which do not cross pad rows is the result of the compensation of the running mean in Figure 37b at the track level. The running mean for the pad crossing tracklets of the real data reconstruction is, however, surprising. Differential analysis has shown that this issue has no connection to the problem of the shifted mean of the y distributions. The effect of this issue to the track is also uncertain, since Figure 47 does not show any signs of problems.

Tracklet to Track Residuals in φ

Figure 50 shows the results for the tracklet to track residuals for the angle φ. As for the plots of the y residuals, the mean of the distribution for the real data has a worrying systematic shift. The root of both issues is most probably the same. The shift of about 50 mrad is of the same order as the 8 degree of the Lorentz angle, thus

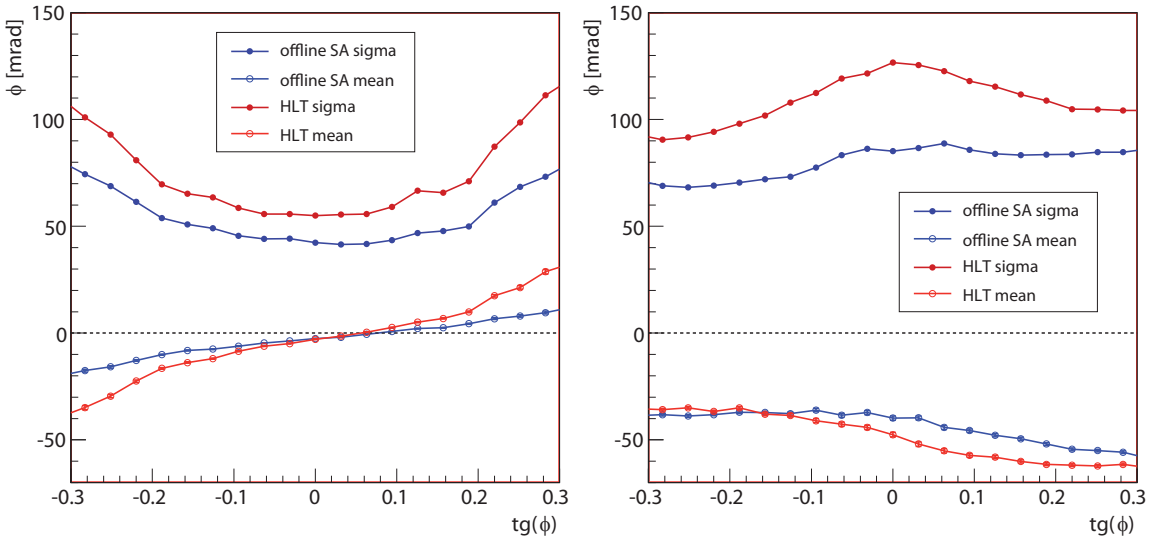


Figure 50: Residuals of the φ angle of tracklets to tracks, for simulated data (a) and real data (b)

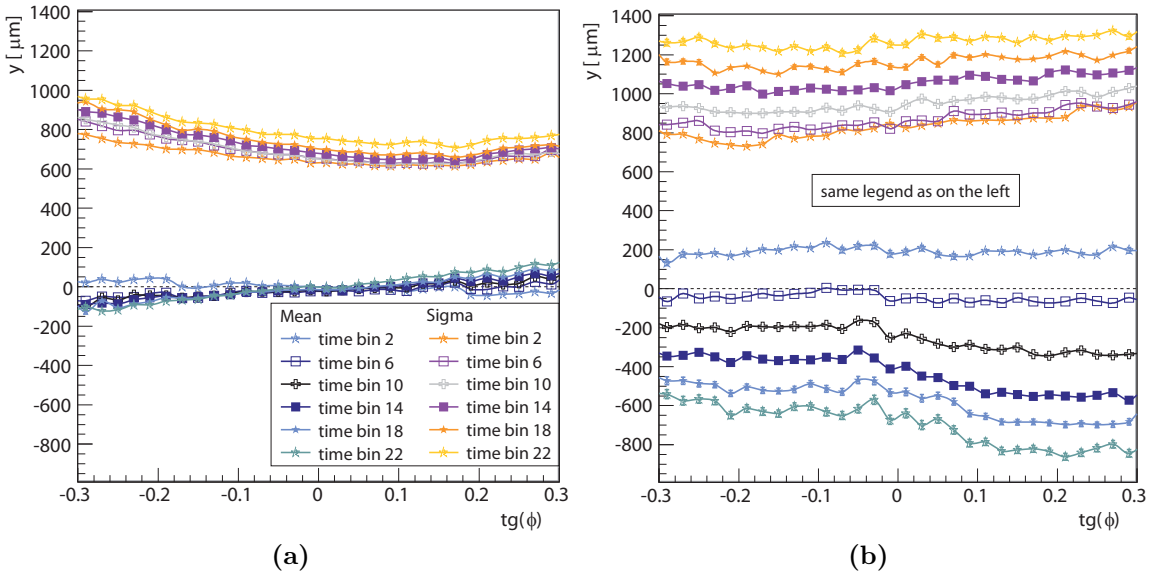


Figure 51: Residuals of the y coordinate of clusters to tracks per time bin, for simulated data (a) and real data (b). As the plots for the HLT reconstruction and for the offline standalone reconstruction look almost identical, only the plots of the offline reconstruction are shown.

the first step to find the reason for this problem, was to double check the correction for the Lorentz angle.

If the correction is not applied, the expectation would be that clusters at different time bins have different residuals to the track (Figure 35b). Indeed, exactly this symptom was found, as shown in Figure 51. This figure corresponds to Figure 46, but here an independent histogram was generated for each time bin. It can be seen that the mean systematically moves depending on the time bin. This finding means that the problem is very likely on the cluster level. It would be possible to reconstruct a track with a systematic shift of 200 μm but it is very unlikely that a track crossing a whole stack has a systematic shift of 8 degree. However, no evidence of a wrong calculation during the real time reconstruction was found, until now.

Apart from this problem, which both, the stand-alone TRD reconstruction and the HLT TRD reconstruction are affected of, the HLT reconstruction seems to have a good quality. The plots for the HLT reconstruction are usually very similar to the ones of the stand-alone online reconstruction. This conclusion is also backed by the good results of the online calibration (Figure 28).

7 Summary

Current experiments at the LHC try to find answers for very fundamental questions about our universe, by endeavouring the sub microscopic environment. In particular, ALICE searches for the properties of the QGP, which is the state our universe was in during its first microsecond. This will be done in ion-ion collisions at energies never reached before in controlled surroundings. The low probability of many signatures demands for high collision rates, which leads to huge amounts of data. For being able to cope with this, experiments include multiple trigger levels, activating detectors and data taking only for interesting events.

The ALICE experiments provides an additional triggering layer, called HLT. Here a full online reconstruction of the data enables online monitoring of the data, and online calibration of the detectors. Based on an online analysis of the reconstructed data, the HLT can steer the data taking and add additional information to provide a pre organisation.

This thesis presented the development of the code, interfacing the TRD offline reconstruction and calibration algorithms to the HLT framework. For being able to follow with the speed of the detectors, the TRD offline algorithms have been significantly accelerated, such that the processing can be performed on a reasonable number of processors. For this dedicated tools were used to detect bottlenecks.

During the process of increasing the processing performance the resulting quality needed to be monitored closely, as the physical information must not be lost. The good online calibration results of the proton-proton runs show that this goal was mainly reached.

There are still efforts to be done for understanding the problem revealed by the Quality Assurance for the real data, and, additionally, preparations of the setup for the lead-lead runs are to be finished.

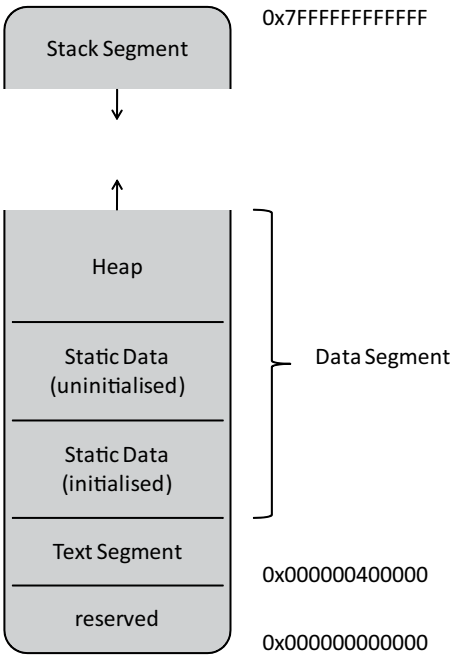


Figure 52: User-level memory layout in Unix-like operating systems

8 Appendix

8.1 Memory Management on Modern Computers

For the actual implementation of the code, and thus for the success of the project, the understanding of memory and how it is used by the hardware was a major necessity, thus the appendix is devoted to this subject. It is an aggregate of [PICA, LPE] and my own experiences with the GCC compiler (versions 3.4 to 4.4).

8.1.1 Static and Dynamic Memory Allocation

The memory of computers as seen by the running executed software is not a monolithic block, but it is substructured. As simplification let's first assume that the computer runs only one programme at a time. Figure 52 shows the typical user-level memory layout typically used in current Unix-like operating systems. Listing 4 shows examples for the memory allocation procedures which are discussed in the following.

Processors process data using instructions telling them how to process that data. The *text segment* contains the instructions describing the executed programme. From here the processor is fed with the next instructions to perform. Data is saved within the *data segment*, which again has a substructure.

Variables, which are being defined within the scope of the currently processed instructions, are saved on the *stack*. The stack is a Last-In-First-Out (LIFO) structure. It starts at the maximal address and grows downwards. By entering a new scope, the local variables of this scope are *pushed* onto the stack. Leaving the scope is accompanied with *pulling* the scope's data out of the stack. Push and pull actually only reserve or free memory and update the processor's stack pointer. This

```
{                // enter the scope and push variable i onto the stack
  int i = 123;   // initialise the variable by the immediate 123
}                // leave the scope and pull variable i out of the stack

{                // enter the scope and push arrays arr and brr onto the stack
  int arr[3] = { 0, 1, 2 }; // copy the immediates 0, 1, 2 one by one
  int brr[1000] = { 0, 1 , ... }; // copy the predefined array from static memory
  static int crr[3] = { 0, 1, 2 }; // nothing to do, crr points to static memory
}                // leave the scope and pull arr and brr out of the stack

{                // enter the scope and push pointers arr and brr onto the stack
  int* arr = new int[10]; // demand for heap memory
  int* brr = new int[10]; // demand for heap memory
  delete [] arr;          // free the heap memory used by arr
}                // leave the scope and pull arr and brr out of the stack
                 // warning: the heap used by brr was not freed
                 //          and thus a memory leak was produced
```

Listing 4: Examples of memory allocation during run-time

points to the *top of stack*, which is either the last full or the first empty address at the lower edge of the stack. The variables pushed, however, usually also need to be initialised to some values needed. In the simplest case, the variable shall be initialised to a number known at compilation time. This number is called an *immediate* and is saved directly together with the instruction used for initialising the variable, thus ending up in the text segment.

When a local variable describes a data array with known size at compilation time, the whole array is pushed onto the stack, when entering the scope. However, the array is not initialised, thus the values are arbitrary. Predefined arrays may be saved either in the *static data segment*, or as immediates in the text segment. The compiler will choose; typically the decision will be based on the size of the array. Up on entering the scope, and after pushing the array onto the stack, the array of the static data segment is copied from there onto the stack or each element is initialised by its own using the immediates. This copying may or may not be desired, depending on the need. In case no copying is needed, the data array of the static memory may be used directly, thus also no change of the stack is needed.

The static data segment contains all global variables. It, however, differentiates between initialised data, like the predefined array from the previous example, or uninitialised data: This is, for example, a global array that is not predefined.

There are cases were the size of an array is unknown at compilation time. In this case the procedure using the stack cannot work, as push and pull are controlled by the compiler during the compilation. However, in this case it is unknown how many push or pull instructions to use. Thus, this sort of arrays must be dynamically allocated, and this is done using the so called *heap*. The management of this memory segment is not controlled by the kernel and is thus highly dependent on the used programming language. However, independent of the actual implementation, there are general problems concerning dynamic memory allocation. Firstly, it is possible to call for too much memory during run time, more than is physically available.

This can happen in case of *memory leaks*. Secondly, the frequent allocation and deallocation of memory with different sizes can lead to a fragmentation of the heap. Eventually, although enough space was available for the next allocation, it could fail in case free space was not contiguous. Even if the heap management includes *garbage collection* or *memory defragmentation*, it is useful to have the dynamically allocated arrays as fixed as possible during the run time.

The data, which is needed to be loaded in memory at the startup of the executable, must be saved in the executable file. Thus the executable file contains all information strictly needed: It has a text section, where the text segment is saved, and a data section, containing the initialised static data segment. The other segments are not saved on disk.

8.1.2 Data Alignment

The smallest amount of information is saved in one bit. This is what a boolean effectively saves: yes or no. However, modern processors cannot access data bit-wise, the addresses are byte-wise. Thus the smallest addressable datum has a size of one byte, which is typically 8 bits. Retrieving a datum with a size of one byte from the memory to the processor is done by one dedicated instruction. Modern processors can, however, process data of bigger sizes, up to 64 bits. Thus there are additional instructions for loading data with sizes of 16, 32 or 64 bits from memory. Depending on the computing architecture, these instructions require that the loaded data is saved at addresses divisible by the size to be loaded. This is than called an aligned memory access. Unaligned memory accesses can lead to poorly performing code, or even run-time errors.

8.1.3 Virtual Memory

Computers can run multiple processes at a time. For example, the same executable may be started twice, and both processes will be processed concurrently. However, both executables will use the same addresses for their global data, as these addresses where hard coded during the linking process. If the addresses used by the programmes were the physical addresses of the memory, the processes would mutually overwrite their data. This could lead to corruption and would be a giant security flaw. Obviously there is a mechanism to prevent this.

The operating system provides each process with its own address space, which is mapped onto physical memory. The mapping is done such, that addresses to memory of all running processes are disjunct[23]. This memory management is called *virtual memory*. Thus, to remain at the example of the two processes, each process accesses its global data at the same address, but these addresses are mapped onto different physical memory regions[23].

The concept of virtual memory has the advantage, that the process' address space can also be mapped onto disk storage, or any other device. Additionally, in case not enough memory is available the operating system can decide to remap the address space of a certain process, from memory to the disk storage.

[23] at the very least, this is true for writeable memory after processes try to alter it

8.1.4 Inter-Process Communication

As mentioned in the previous chapter; by concept, virtual memory cuts direct inter-process communication through common memory space. As this is, however, an important feature, Unix systems provide operating system assistance. There are multiple solutions for many different circumstances. The ones which are used by the HLT are: *shared memory* (shm), and *named pipes* (a.k.a. FIFO's).

The usage of pipes is widespread in the Unix-like operating systems. To make an example:

```
$ ls | grep m
```

The "|" is called a pipe, and forwards the output from "ls" to "grep". Pipes can be given unique names. In this case these pipes are called *named pipes*. Thus otherwise unrelated processes may attach to these pipes and use them for communication. Like that, the descriptors are exchanged between HLT components[TSD].

Shared memory provides a block of memory, dedicated for the data exchange between processes, which deliberately connect to this block. Additionally the memory block is protected by access permission, set during its creation. This is how HLT components exchange all data except the descriptors [TSD].

List of Figures

List of Tables

Listings

References

[AHI] S.R. Bablok et al. ALICE HLT interfaces and data organisation. Computing in High Energy Physics Conf. 2006 (CHEP06), Mumbai, India, 2006. http://cdsweb.cern.ch/record/1066629

[ALE] S Bagnasco et al. AliEn: ALICE environment on the GRID. J. Phys. Conf. Ser. 119 062012, 2008

[ATDR1] The ALICE Collaboration. Technical Design Report of the Trigger Data Acquisition High-Level Trigger and Control System. CERN–LHCC–2003–062 ALICE TDR 10 7 January 2004

[ATDR2] The ALICE Collaboration. Technical Design Report of the Transition Radiation Detector. CERN /LHCC 2001-021 ALICE TDR 9 3 October 2001

[ATDR3] The ALICE Collaboration. Technical Design Report of the High Level Trigger. ALICE TDR 10, CERN–LHCC–2003–062

[CER] CERN-Brochure-2009-003

[DEM] W. Demtröder. Experimentalphysik 3/4

[GAF] V. L. Ginzburg and I.M. Frank, Zh. Eksp. Teor. Fiz. 16 (1946) 15

[GAJ] P. Goldsmith and J.V. Jelley, Philosoph. Mag. 4 (1959) 836

[GSI] http://www-alice.gsi.de

[HIJ] M. Gyulassy and X.-N. Wang, LBL-34246, 1997.

[JPS] A. Andronic et al. Statistical Hadronization of Charm in Heavy-Ion Collisions at SPS, RHIC and LHC. Phys. Lett. B571, 36 (2003)

[JSN] B.I. Abelev et al. Indications of Conical Emission of Charged Hadrons at the BNL Relativistic Heavy Ion Collider. Phys. Rev. Lett. 102:052302, 2009

[KAL] P. Billoir and S. Qian Nucl. Instr. and Meth. A294 (1990), p. 219.

[KBK] K. Bethge. Kernphysik - Eine Einführung, 978-3540745662

[LHCDR] LHC Design Report
http://lhc.web.cern.ch/lhc/LHC-DesignReport.html

[LOS] J. Wagner et al. Lossless Data Compression for ALICE HLT - ALICE-INT-2008-020 version 1.0

[LPE] A. Robbins. Linux Programming by Example: The Fundamentals, 978-0131429642

[MRD] M. Richter, Doctroal Thesis, Development and Integration of on-line Data Analysis for the ALICE Experiment

[OBS] O. Busch. Nucl. Instr. Meth. A 522 (2004) 45

[ORS] Oana Ristea. Study of the chemical freeze-out in nucleus-nucleus collisions - Romanian Reports in Physics. Vol. 56, No 4, P. 659-666, 2004

[QGP1] Cheuk-Yin Wong. Signatures of Quark-Gluon Plasma Phase Transition in High-Energy Nuclear Collisions. Nucl.Phys. A681 (2001) 22-33

[QGP2] J. Harris, B Müller. The Search for the Quark-Gluon Plasma. Annu. Rev. Nucl. Part. Sci. 1996. 46:71–107

[QGP3] K. Yagi et al. Quark-Gluon Plasma. Cambridge University Press, 2005

[PDG] C. Amsler et al. (Particle Data Group). Phys. Lett. B667, 1 (2008) and 2009 partial update for the 2010 edition.

[PER] D. Perkins. Introduction to High Energy Physics

[PICA] D. Page. Practical Introduction to Computer Architecture, 978-1-84882-255-9 / 978-1-84882-256-6

[PPR1] The ALICE Collaboration. Physics Performance Report, Volume I, J. Phys G: Nucl. Part. Phys. 30 (2004) 1517

[RB] R. Bailhache. private contact

[SLI] F. Sauli. Principles of operation of multiwire proportional and drift chambers. CERN Report 77-09, 1977.

[TAL] T. Alt. HLT Heavy-Ion Preparation at CERN 09/09/2010

[TPC] J. Berger et al. TPC data compression, Nucl. Instr. Meth. A 489 (2002) 421

[TUK] B. Povh, K. Rith, C. Scholz, F. Zetsche. Teilchen und Kerne, Eine Einführung in die physikalischen Konzepte. Springer-Verlag, 1993.

[TSD] T. Steinbeck et at. A Software Data Transport Framework for Trigger Applications on Clusters - CHEP03, La Jolla, Ca., USA, March 24-28, 2003

Danksagung

Zunächst möchte ich mich bei meinem Betreuer Prof. Dr. Harald Appelshäuser bedanken, für die Möglichkeit an einem derart interessanten und hochaktuellem Experiment teilhaben, und einen kleinen Teil beitragen zu können.

Ich bedanke mich bei Dr. Henner Büsching für die moralische und praktische Unterstützung bei Präsentationen und Berichten und das Lesen dieser Diplomarbeit.

Konstantin Antipin danke ich für die exzellente Einführung in das gesamte Themengebiet und seine Vorarbeit. Linux und Emacs möchte ich seitdem nicht mehr missen.

Für die unglaublich schnellen Updates des SVN trunks und für seine Hilfe bei den vielen Details des TRD Offline Codes bedanke ich mich bei Prof. Dr. Christoph Blume.

Desweitern möchte ich mich bei Dr. Matthias Hartig bedanken, für seine Idee einen schnellen Testrechner zu akquirieren, der in zügiger Art und Weise seine aufgetragene Arbeit zu verrichten wusste.

Werner Amend danke ich für seine Unterstützung bei den Drucker- und Netzwerkproblemen, aber auch für seine positive Art die einen einfach immer umstimmt.

Nicht zuletzt möchte ich meinen Arbeitszimmerkollegen Hermes León Vargas und Jason Ulery für die hervorragende, aber trotzdem nie ablenkende Arbeitsatmosphäre danken.

Bei Raphaelle Bailhache bedanke ich mich für ihre Hilfe bei der Implementierung der Kalibrieung und für die Tests, die sie mit Jason unternommen hat.

Im allgemainen möchte ich der ganzen ALICE Kollaboration danken, für die Hilfestellungen und das aufgebrachte Vertrauen.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, keine anderen als die angegebenen Hilfsmittel verwendet und sämtliche Stellen, die benutzten Werken im Wortlaut oder dem Sinne nach entnommen sind, mit Quellen- bzw. Herkunftsangaben kenntlich gemacht habe.

Frankfurt am Main, den 29. September 2010

Theodor B. Rascanu