EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

ALICE-INT-2012-xxx
December 3, 2012

EMCal Offline Documentation

EMCal collaboration

Email:alice-emcal-offline@cern.ch

Abstract

This document describes the EMCAL, it's offline geometry, the software to run a simulation or reconstruct data, the strategy to control the quality of the data, the trigger code and the analysis format.

Contents

1 Introduction

This document is addressed to those who want to work with the EMCal software. It explains the different steps to have the data taken ready to be analyzed. It is divided in 2 blocks: a first one with the description of the procedures needed to cook the data and a second one with the reconstruction and simulation offline code.

For a fast introduction on the code and how it works you can have a look to the EMCal for beginners guide [1]. Some other interesting references are the AliRoot primer [3], the offline AliRoot page [2], and the installation page from Dario Berzano [5].

1.1 Mechanical description of the EMCAL - Federico

The chosen technology is a layered Pb-scintillator sampling calorimeter with a longitudinal pitch of 1.44 mm Pb and 1.76 mm scintillator with longitudinal Wavelength Shifting Fiber (WLS) light collection. The full detector spans η = -0.7 to η= 0.7 with an azimuthal acceptance of $\Delta\phi$ 107° and is segmented into 12,288 towers, each of which is approximately projective in η and ϕ to the interaction vertex. The towers are grouped into super modules of two types: full size which span $\Delta\phi = 20°$ and 1/3 size which span $\Delta\phi$ = 6.67°. There are 10 full size and 2, 1/3-size super modules in the full detector acceptance (Fig. 1).

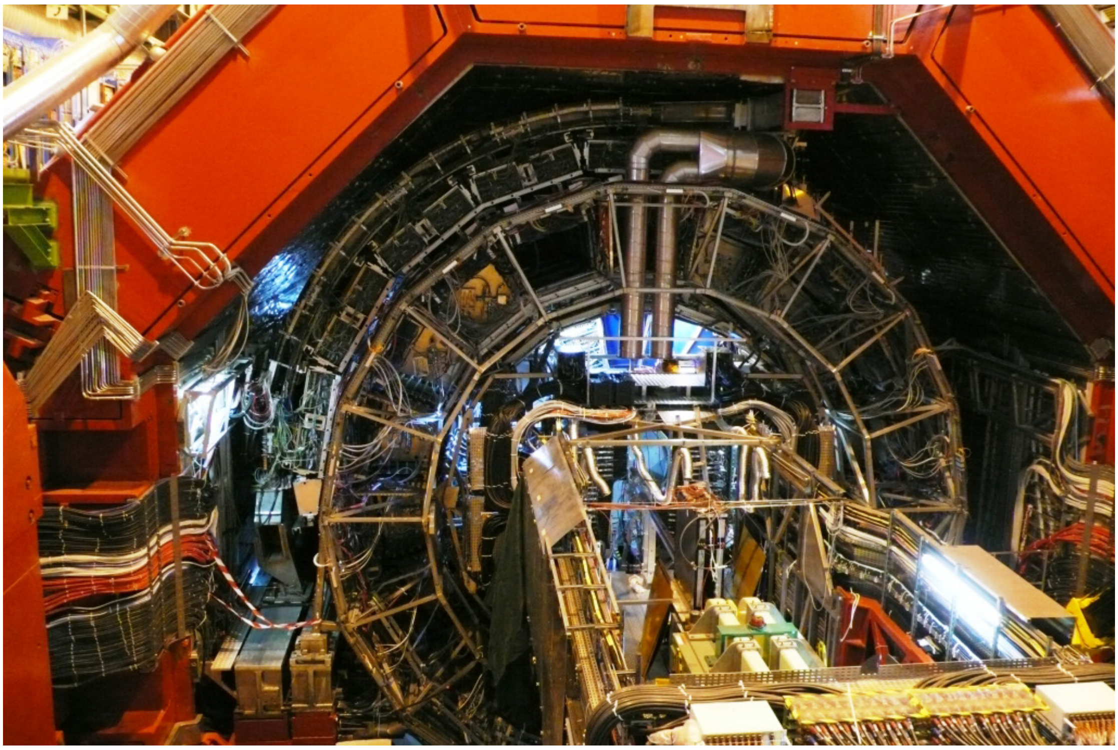

Fig. 1: Azimutha view from the A-side (opposite to the di-muon arm) of the full EMCal as installed into the ALICE detector. The two 1/3-size super-modules are visible at 9 o'clock position.

The super module is the basic structural units of the calorimeter. These are the units handled as the detector is moved below ground and rigged during installation.

Fig. 2 shows a full size super module with 12×24 modules configured as 24 strip modules of 12 modules each. The supporting mechanical structure of the super module hides the stacking into a nearly projective geometry which can be inferres by the differnt tilt of the strip modules going from the left to the right part of the picure. The electronics integration pathways are also visible. Each full size super module is assembled from $12 \times 24 = 288$ modules arranged in 24 strip modules of 12 modules each.

Fig. 2: View of one EMCal super-module during the installation into the ALICE detector. The cradle holds the 24 strip modules into a mechanically rigid unit. Each strip module holds 12 unit modules. On the right side the two electronics crates are visible.

Each module has a rectangular cross section in the ϕ direction and a trapezoidal cross section in the η direction with a full taper of 1.5°. The resultant assembly of stacked strip modules is approximately projective with an average angle of incidence of less than 2° in η and less than 5° in ϕ. An assembled strip module is shown in Fig. 3.

The smallest building block of the calorimeter is the individual module illustrated in Fig. 4 Each individual module contains $2 \times 2 = 4$ towers built up from 77 alternating layers of 1.44 mm Pb and 1.76 mm polystyrene, injection molded scintillator. White, acid free, bond paper serves as a diffuse reflector on the scintillator surfaces while the scintillator edges are treated with TiO2

Fig. 3: View of a fully assembled strip module. The photo shows the APD+CSP package and copper shielding monunted light guide fixture. On the right part of the photo the LED UV optical fiber distribution system is visible. Each strip module, is cabled via 3 T-Cards visible in the center of the assembly.

loaded reflector to provide tower to tower optical isolation and improve the transverse optical uniformity within a single tower. The Pb-scintillator stack in a module is secured in place by the static friction between individual layers under the overall load of 350 kg. The module is closed by a skin of 150 μm thick stainless steel screwed by flanges on all four transverse surfaces to corresponding front and rear aluminum plates. This thin stainless skin is the only inert material between the active tower volumes. The internal pressure in the module is stabilized against thermal effects, mechanical relaxation and long term flow of the Pb and/or polystyrene by a customized array of 5 non-linear spring sets (Bellville washers) per module. In this way, each module is a self supporting unit with a stable mechanical lifetime of more than 20 years when held from its back surface in any orientation as when mounted in a strip module.

All modules in the calorimeter are mechanically and dimensionally identical. The front face dimensions of the towers are 6×6 cm^2 resulting in individual tower acceptance of $\Delta\eta \times \Delta\phi = 0.014 \times 0.014$ at η=0. The EMCal design incorporates a moderate detector average active volume density of 5.68 g/cm^3 which results from a 1:1.22 Pb to scintillator ratio by volume. This results in a compact detector consistent with the EMCal integration volume at the chosen detector thickness of 20.1 radiation lengths.

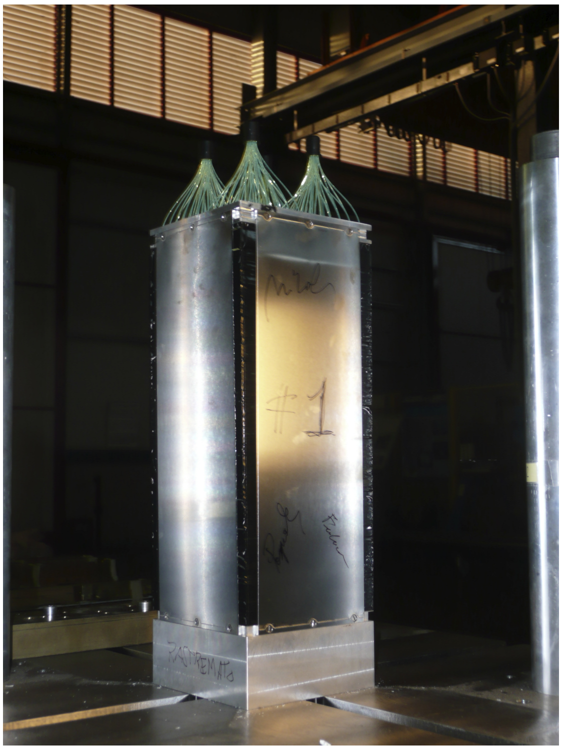

Fig. 4: The first 1.5° tapered module of the EMCal generation II prototype produced in EU shown. The module's internal compression is mantained by a set of 5 Bellville washers (non linear springs) acting between the top and bottom containment Al plates to prevent the delamination of the internal Pb-scintillator sandwich.

As described above, the super module is the basic building block of the calorimeter. Starting with 288 individual modules which are rather compact and heavy, the main engineering task is to create a super module structure which is rigid, with small deflections in any orientation yet does not require extensive, heavy external stiffening components that would reduce the volume available for the active detector. The solution adopted for the ALICE EMCal is to develop a super module crate which functions not as a box for the individual modules but rather an integrated structure in which the individual elements contribute to the overall stiffness. The super module crate is effectively a large I-beam in which the flanges are the long sides of the crate and the 24 rows of strip modules together. This configuration gives to the super module good stiffness for both the 9 o'clock and 10 o'clock locations. For the 12 o'clock location, the I-beam structure of the super module is augmented by a 1 mm thick stainless steel forward sheet (traction loaded), which controls the bending moment tending to open the crate main sides, and helps to limit deflection of strip modules. Ridges are provided on the interior surfaces of the crate to allow precision alignment of the strip modules at the correct angle. The stiffness given by this I-beam concept allows the use of non-magnetic light alloys for main parts of the super module crate. Parts of the super module crate will be made mainly from laminated 2024 aluminum alloy plates. The two main sides (flanges of the I-beam) of the crate will be assembled from 2 plates, 25 mm and 25 mm thick, bolted together and arranged so as to approximately follow the taper of the 20 degree sector boundary. Each of the 24 rows of a super module contain 12 modules as described above. Each of the modules is attached to a transverse beam by 3.4 mm diameter stainless steel screws. The 12 modules and the transverse beam form a strip module. The strip module is 1440 mm long, 120 mm wide, 410 mm thick. The total weight of the strip module is approximately 300 kg and like module, it is a self supporting unit. The transverse beam, which is the structural part of the strip module, is made from cast aluminum alloy with individual cavities along its length where the fibers emerging from towers are allowed to converge. The casting process is well suited to forming these cavities and the overall structure, saving considerable raw material and machining time.

In addition to functioning as a convenient structural unit which offers no interference with the active volume of the detector and forming the web of the I-beam structure of the super module, the transverse beam of the strip module provides protection for the fibers, a structural mount for the light guide, APD and charge sensitive preamplifier and a light tight enclosure for these elements.

1.2 Functional description of the EMCAL - Terry

**** need some additional info on PE APDs *********

Particles traversing the calorimeter, in particular photons and electrons, will deposit energy in different towers. The EMCAL reconstruction measures such energy per tower, forms clusters of cells produced by a given particle, and if possible matches them with particles detected by the tracking detectors in front of EMCAL (charged particles).

Scintillation photons produced in each tower are captured by an array of 36 Kuraray, Y-11, double clad, WLS fibers that run longitudinally through the Pb/scintillator stack. Each fiber terminates in an aluminized mirror at the front face end of the module and is integrated into a polished, circular group of 36 at the photo sensor end at the back of the module. The fiber bundles are pre-fabricated and inserted into the towers after the module mechanical assembly is completed. The 36 individual fibers are packed into a circular array 6.8 mm in diameter and held in place inside a custom injection molded grommet by Bicron BC-600 optical cement. An optical quality finish is applied to the assembled bundle using a diamond polishing machine. At the other end of the bundle, individual fibers are similarly polished and mirrored with a sputtered coat of aluminum thick enough to ensure the protection of the inner mirror. The response of the Al-coated fiber is considerably flatter with an overall increase in efficiency in the range of about 25% in the vicinity of shower maximum (i.e. the location of the highest energy deposition for an electromagnetic shower). This number accounts for material immediately in front of the detector; which ranges between 0.4 and 0.8 radiation lengths, and assumes 5.5 - 6.0 radiation lengths for shower maximum for 10 GeV photons. At this depth in the detector, the mirrored fiber response is very uniform does not contribute to the non-linearity of the detector as a whole.

Other factors which can significantly impact the electromagnetic performance of the calorimeter, include scintillator edge treatment and the density of the wavelength shifting fiber readout pattern and the material chosen for the interlayer diffuse reflector. For scintillator edge treatment and fiber density, advantage was taken from the extensive studies made by the LHCb collaboration for their ECAL. In particular, a diffuse reflector edge treatment was adopted, such as that obtained with Bicron Titanium Dioxide loaded white paint (BC622A) with a total fiber density of about one fiber per cm^2. In the case of the interlayer diffuse reflector, a white, acid free, bond paper was used in place of the Teflon based commercial TYVEK. While TYVEK produces slightly better surface reflectivity, its coefficient of friction is too low to permit its use in this design where the module's mechanical stability depends somewhat on the interlayer friction.

The 6.8 mm diameter fiber bundle from a given tower connects to the APD through a short light guide/diffuser with a square cross section of 7 mm $\times$ 7 mm that tapers slowly down to 4.5 mm $\times$ 4.5 mm as it mates (glued) to the 5 mm $\times$ 5 mm active area of the photo sensor. The 4 pre-fabricated fiber bundles are inserted into the towers of a single module.

The selected photo sensor is the Hamamatsu S8664-55 Avalanche Photo Diode ***********

This photodiode has a peak spectral response at a wavelength of 585 nm compared to an emission peak of 476 nm for the Y-11 fibers. However, both the spectral response and the quantum efficiency of the APD are quite broad with the latter dropping from the maximum by only 5% at the WLS fiber emission peak. At this wavelength, the manufacturer's specification gives a quantum efficiency of 80%.

2 EMCAL geometry software - Marco +++

This page is intended for a description of the EMCAL geometry and the methods to access it. *This is a very preliminary version that needs work.*

2.1 Classes description

The EMCAL geometry is implemented in several classes : (right now very brief description, it should be completed)

- AliEMCALGeoUtils: Steering geometry class. No dependencies on STEER or EMCAL non geometry classes. Can be called during the analysis without loading all aliroot classes.
- AliEMCALGeometry: Derives from AliEMCALGeoUtils, contains dependencies on other EMCAL classes (AliEMCALRecPoint).
- AliEMCALEMCGeometry: Does the geometry initialization. Does all the definitions of the geometry (towers composition, size, Super Modules number ...)
- AliEMCALGeoParams: Class container of some of the geometry parameters so that it can be accessed everywhere in the EMCAL code, to avoid "magic numbers". Its use has to be propagated to all the code.
- AliEMCALShishKebabTrd1Module: Here the modules are defined and the position of the modules in the local super module reference system is calculated

2.2 Accessing the geometry

One can get the geometry pointer in the following ways:

- If galice.root is available:

```
AliRunLoader *rl = AliRunLoader::Open("galice.root",AliConfig::
    GetDefaultEventFolderName(),"read");
rl->LoadgAlice();//Needed to get geometry
AliEMCALLoader *emcalLoader = dynamic\_cast<AliEMCALLoader*>(rl->
    GetDetectorLoader("EMCAL"));
AliRun * alirun = rl->GetAliRun();
AliEMCAL * emcal = (AliEMCAL*)alirun->GetDetector("EMCAL"); AliEMCALGeometry
     * geom = emcal->GetGeometry();
else, if galice.root is not available:
AliEMCALGeometry * geom = AliEMCALGeometry::GetInstance("EMCAL\_COMPLETE") ;
```

In this case you might need the file geometry.root if you want to access to certain methods that require local to global position transformations. This file can be generated doing a simple simulation, it just contains the transformation matrix to go from global to local.

The way to load this file is:

```
TGeoManager::Import("geometry.root");
```

The transformation matrices are also stored in the ESDs so if you do not load this file, you can have to load these matrices from the ESDs.

If you want to see different parameters used in the geometry printed (cells centers, distance to IP, etc), one just has to execute the method PrintGeometry().

2.3 Geometry configuration options

Right now the following geometry options are implemented:

- EMCAL_COMPLETE: 12 Super Modules (2 half Super Modules)
- EMCAL_FIRSTYEAR: 4 Super Modules (year 2010)
- EMCAL_FIRSTYEARV1: 4 Super Modules, corrected geometry (year 2010)
- EMCAL_COMPLETEV1: 10 Super Modules, corrected geometry (year 2011)
- EMCAL_COMPLETE12SMV1: 12 Super Modules (10+2/3), corrected geometry (year 2012)

Other options exists but need to be removed as they **should not be used**:

- EMCAL_PDC06: Old geometry, for reading old data (which do not exist anymore).
- EMCAL_WSU: Prototype geometry.

By default, the geometry is loaded with the EMCAL_COMPLETE12SMV1 configuration.

2.4 Mapping

The tower row/column mapping online and offline follows the alice numbering convention. Figures 5 to 7 display the position of the super modules from different points of view and the position of the tower index in them.

2.5 Tower index transformation methods

2.5.1 Absolute tower ID to Row/Column index

Each EMCAL supermodule is composed of 24x48 towers (phi,eta), grouped in 4x4 modules. Each tower (even each module) has a unique number assigned, called in the code "absolute ID" number (absId). This number can be transformed into a row (phi direction) or column (eta direction) index. The procedure to go from the absId to the (row, col) formulation or viceversa is as follow:

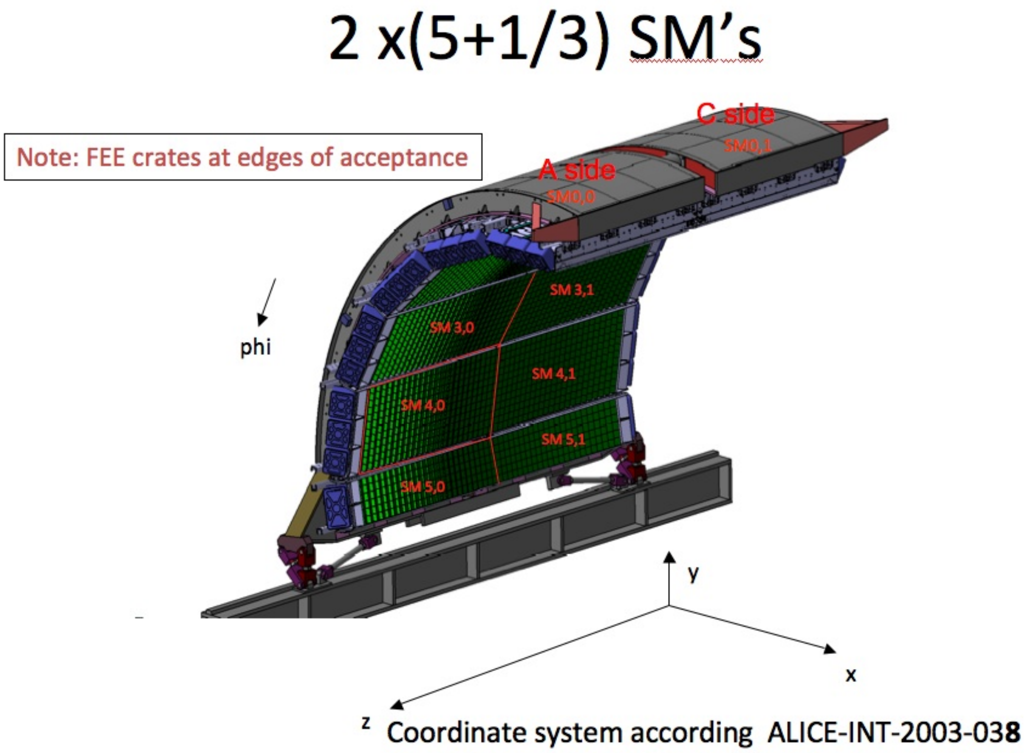

Fig. 5: Position of the super modules

– From absId to col-row:

```
Int\_t nSupMod, nModule, nIphi, nIeta, iphi, ieta;
//Check if this absId exists
if(!CheckAbsCellId(absId)) return kFALSE;
// Get from the absId the super module number, the module number and the eta-phi index
     (0 or 1) in the module
GetCellIndex(absId, nSupMod, nModule, nIphi, nIeta);
// Get from the  the super module number, the module number and the eta-phi index (0
     or 1) in the module the tower row (iphi) and column (ieta)
GetCellPhiEtaIndexInSModule(nSupMod,nModule,nIphi,nIeta, iphi, ieta);
```

– From col-row to absId, following the same notation as above:

```

absid = GetAbsCellIdFromCellIndexes(nSupMode, iphi, ieta);
```

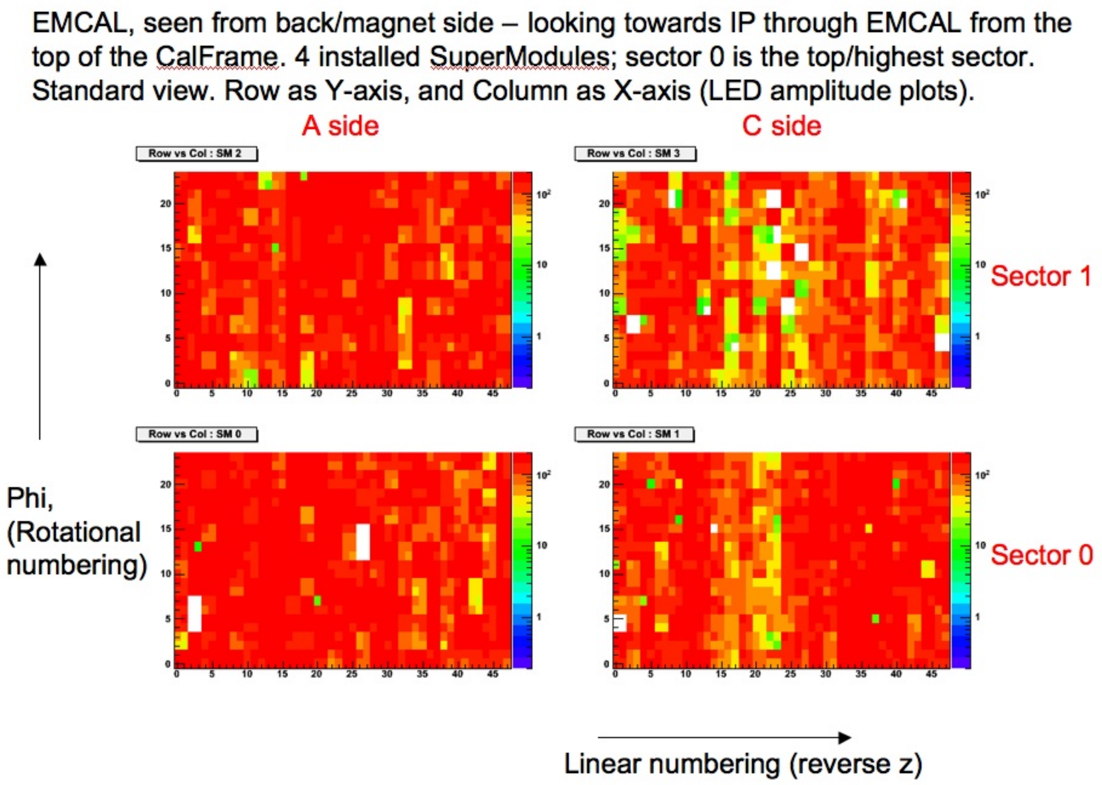

Fig. 6: EMCal seen from the magnet side with 4 SMs.

or

```
absid = GetAbsCellId(nSupMod, nModule, nIphi, nIeta);
```

– Other interesting method is

```
Int\_t  GetSuperModuleNumber(Int\_t absId)
```

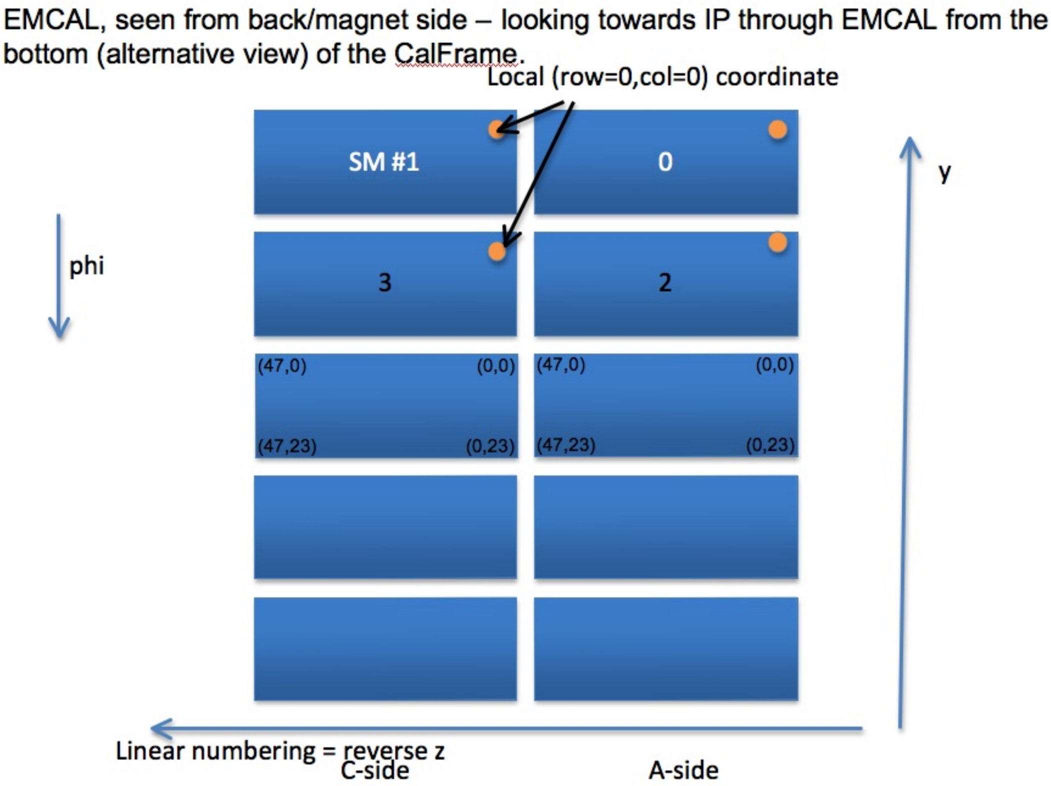

Fig. 7: EMCal geometrical numbering.

2.6 Tower index to local / global reference system position

2.6.1 Local coordinates

To correlate the tower index and its position in local coordinates, the following methods are available:

```
Bool\_t AliEMCALGeoUtils::RelPosCellInSModule(Int\_t absId, Double\_t \&xr,
    Double\_t \&yr, Double\_t \&zr) const;

Bool\_t AliEMCALGeoUtils::RelPosCellInSModule(Int\_t absId, Double\_t loc[3])
    const;

Bool\_t AliEMCALGeoUtils::RelPosCellInSModule(Int\_t absId, TVector3 \&vloc)
    const;
```

To which the input is the absId and the output are the coordinates of the center of towers in the local coordinates of the Super Module. This method gets the column and row index of the cell from the absId, independently of the Super Module (like above), and gets the center of the cell from 3 arrays (x,y,z) filled with such quantities. Such central positions are calculated during the initialization of the geometry, where the arrys are filled, in the method :

```
AliEMCALGeoUtils::CreateListOfTrd1Modules()
```

««<Someone else should explain how it works»»>

In case we calculate the cluster position, things are a bit different.

««< This explanation should go to the clusterization section»»

This is done in

```
void AliEMCALRecPoint::EvalLocalPosition()
```

First we calculate the cell position with the method

```
AliEMCALGeometry::RelPosCellInSModule(Int\_t absId, Int\_t maxAbsId, Double\_t
    tmax, Double\_t \&xr, Double\_t \&yr, Double\_t \&zr)
```

The calculation of the cell position done here is different in the "x-z" but the same in "y".

««< ««<Someone else should explain how it works»»>

In this particular case the position calculation per tower depends on the position of the maximum cell, and the sum of the energy of the cells of the cluster. The maximum depth (tmax) is calculated with the method

```
Double\_t AliEMCALRecPoint::TmaxInCm(const Double\_t e){

    // e: energy sum of cells

static Double\_t ca = 4.82;// shower max parameter - first guess; ca=TMath::Log(1000./8.07)

    static Double\_t x0 = 1.23; // radiation lenght (cm)

    static Double\_t tmax = 0.; // position of electromagnetic shower max in cm

    tmax = TMath::Log(e) + ca+0.5;

    tmax *= x0; // convert to cm

}
```

After the cells position of the cluster is accessed, the position of the cluster is calculated averaging the cell positions with a logarithmic weight:

```
w(cell i) = TMath::Max( 0., logWeight + TMath::Log( energy[cell i] / summed\
    _cluster\_cell\_energy ));
```

where the logWeight was chosen to be 4.5 (this value was taken from PHOS, never optimized as far as I know)

So in the end the position, is

```
f = Sum(f(i) * w(i))/Sum(w(i))
```

where f=x,y,z.

2.6.2 Global coordinates

To transform from local to global we have the methods

```
void GetGlobal(const Double\_t *loc, Double\_t *glob, int ind) const;

void GetGlobal(const TVector3 \&vloc, TVector3 \&vglob, int ind) const;

void GetGlobal(Int\_t absId, Double\_t glob[3]) const;

void GetGlobal(Int\_t absId, TVector3 \&vglob) const;
```

These methods take the local coordinates and transform them into global coordinates using the transformation matrix of the Super Module.

```
TGeoHMatrix* m = GetMatrixForSuperModule(nSupMod);

if(m) m->LocalToMaster(loc, glob);
```

GetGlobal is called in the following useful methods in the geometry class:

– Return the eta and phi angular position of the cell from the AbsId

```
void  EtaPhiFromIndex(Int\_t absId, Double\_t \&eta, Double\_t \&phi) const
    ;
void  EtaPhiFromIndex(Int\_t absId, Float\_t \&eta, Float\_t \&phi) const;
```

– Print information of the cells. For "pri>0" returns more information. "tit" has not much use, this value is printed.

```
void PrintCellIndexes(Int\_t absId, int pri, const char *tit)
```

2.7 Geometry Alignment

AliRoot contains a frame for the correction of the misplacement of geometry objects with respect to the ideal positions which are kept in the STEER/ directory of the following classes:

```
AliAlignObj
AliAlignObjMatrix
AliAlignObjParams
AliAlignmentTracks
```

The class AliEMCALSurvey creates the corrections to the alignable objects. The class AliEMCALSurvey was established to take the survey parameters from OCDB, calculate the shift in position of the center of the end faces of the supermodules from the nominal position, and convert this to a transformation matrix for each supermodule which is applied to correct the global position of the supermodules. All calculations of global positions would then use these corrected supermodule positions to determine their locations within the ALICE global coordinate system.

3 EMCal OCDB/OADB - Marcel

OCDB is the Offline data base. It contains the different parameters used for simulation or reconstruction of the detectors or even the LHC machine parameters that might change for the different run conditions.

OADB is the Offline Analysis data base.

The EMCAL OCDB (and other detectors OCDB) is divided in 3 directories that can be found in

```
$ALICE_ROOT/OCDB/EMCAL
```

- Calib: Very different type of information, from hardware mapping to calibration parameters.
- Align: Survey misplacements in geometry.
- Config: Detector configuration, temperatures

Inside these directories you will find other subdirectories with more specific types of parameters. Each of the directories contains a file named in this way:

```
Run(FirstRun)_(LastRun)_v(version)_s(version).root
```

being the default and what you will find in the trunk

```
Run0_999999999_v0_s0.root
```

What is actually used for the real data reconstruction can be found in alien here:

```
/alice/data/20XX/OCDB/EMCAL
```

There are different repositories for different years (20XX). For the simulation productions, there is another repository on the grid:

```
/alice/simulation/2008/v4-15-Release/XXX/EMCAL
```

which is divided into 3 other repositories: Ideal, Full and Residual. Each one is meant to reproduce the detector with different precision. For EMCAL, right now these 3 repositories contain the same parameters.

The following section explain the elements stored and how to read and fill OCDB parameters.

3.1 Accessing a different OCDB

In the simulation/reconstruction macro a default OCDB needs to be specified if different from

```
$ALICE_ROOT/OCDB.
```

When running on the grid, one needs to set for example in a reconstruction of simulated data:

```
reco.SetDefaultStorage("alien://Folder=/alice/simulation/2008/v4-15-Release/
    Residual/");
```

If one or several OCDB files have been modified, the following line has to be added in the simulation or reconstruction macro:

```
reco.SetSpecificStorage("EMCAL/Calib/Pedestals","local://your/modified/local/OCDB
    ");
```

The file with the calibration coefficients needs to be stored in the directory :

```
/your/modified/local/OCDB/EMCAL/Calib/Data
```

If more of the OCDB files are modified, add the following line :

```
reco.SetSpecificStorage("EMCAL/Calib/","local:/your/modified/local/OCDB");
```

with all the directories inside

```
\begin{lstlisting}
/your/modified/local/OCDB/EMCAL/Calib/
```

3.2 Energy calibration

Calibration Coefficients tower by towers are stored in the following directory :

```
EMCAL/Calib/Data
```

What is stored is an object of the class AliEMCALCalibData which is a container of gains and pedestals per tower. These coefficients are used in:

- Simulation: during the digitization, in AliEMCALDigitizer::Digitizer(), when calling AliEMCALDigitizer::DigitizeEnergy(), to transform the deposited energy into ADC counts.

- Reconstruction: in AliEMCALClusterizerv1::Calibrate() called in AliEMCALClusterizer::MakeClusters(), when forming the cluster, to get the final cluster energy.

The macro

```
$ALICE_ROOT/EMCAL/macros/CalibrationDB/AliEMCALSetCDB.C
```

is an example on how to set the calibration coefficients per channel, or how to read them from the OCDB file. This macro can set all channels with the same selected value or with random values given a uniform or gaussian smearing of a selected input value. A simple example that shows how to print the parameters is PrintEMCALCalibData.C

All channels in the simulation have the same value for the gains (0.0153 GeV/ADC counts) and pedestal (set to 0 since the calorimeter works with Zero Suppressed data).

3.3 Bad channels

Storage for the bad channels map found in hardware are here :

```
EMCAL/Calib/Pedestals
```

The object stored is from the class AliCaloCalibPedestal used for monitoring the towers calibration and functionality. This class has the data member TObjArray *fDeadMap which consists of an array of 12 TH2I (as many as Super Modules), and each TH2I has the dimension of 24x48 (number of towers in $\phi \times \eta$ direction), each bin corresponds to a tower. The content of each entry in the histogram is an integer which represents the possible status:

```
enum kDeadMapEntry{kAlive = 0, kDead, kHot, kWarning, kResurrected,
    kRecentlyDeceased, kNumDeadMapStates};
```

Right now only the status kAlive, kDead, kHot and soon kWarning (soon, not yet) are set but, the code is basically skipping all the channels that are kDead and kHot. The bad channel map is used in the reconstruction code in 3 places:

- AliEMCALRawUtils::Raw2Digits() : Before the raw data time sample is fitted, the status of the tower is checked, and if bad (kHot or kDead), the fit is not done. This avoids trying to fit ill shaped samples. This step is optional though, right now default is to skip the bad channels here. With the RecParam OCDB we can select to use it or not.
- AliEMCALClusterizerv1::Calibrate(): once the cluster is formed, to get the cluster energy from its cells.
- AliEMCALRecPoint::EvalDistanceToBadChannels(): Evaluate the distance of a cluster to the closest bad channel. During the analysis we may want to skip clusters close to a bad channel. This time a bad channel is whatever is not kAlive.

The macro

```
$ALICE_ROOT/EMCAL/macros/PedestalDB/AliEMCALPedestalCDB.C
```

is an example on how to set the bad channel map and how to read it from a file. When executed, it displays a menu that allows to set randomly as bad a given % of the towers. It also allows to set the map from an input txt file, with the format like $ALICE_ROOT/EMCAL/macros/PedestalDB/map.txt (this map file is the one used in the last mapping in the raw OCDB). It can also read the OCDB file and display the 12 TH2I histograms on screen.

3.4 Reconstruction parameters

The storage of the parameters used in reconstruction is done in

```
EMCAL/Calib/RecoParam
```

What is stored is an object of the class AliEMCALRecParam which is a container for all the parameters used. There are different kind of parameters, we can distinguish them depending on which step of the reconstruction are used as explained below.

Raw data fitting and mapping

- Double_t fHighLowGainFactor; // gain factor to convert between high and low gain
- Int_t fOrderParameter; // order parameter for raw signal fit
- Double_t fTau; // decay constant for raw signal fit
- Int_t fNoiseThreshold; // threshold to consider signal or noise
- Int_t fNPedSamples; // number of time samples to use in pedestal calculation
- Bool_t fRemoveBadChannels; // select if bad channels are removed before fitting
- Int_t fFittingAlgorithm; // select the fitting algorithm
- static TObjArray* fgkMaps; // ALTRO mappings for RCU0..RCUX

Clusterization

- Float_t fClusteringThreshold ; // Minimum energy to seed a EC digit in a cluster
- Float_t fW0 ; // Logarithmic weight for the cluster center of gravity calculation
- Float_t fMinECut; // Minimum energy for a digit to be a member of a cluster
- Bool_t fUnfold; // Flag to perform cluster unfolding
- Float_t fLocMaxCut; // Minimum energy difference to consider local maxima in a cluster
- Float_t fTimeCut ; // Maximum difference time of digits in EMC cluster
- Float_t fTimeMin ; // Minimum time of digits
- Float_t fTimeMax ; // Maximum time of digits

Track Matching

- Double_t fTrkCutX; // X-difference cut for track matching
- Double_t fTrkCutY; // Y-difference cut for track matching
- Double_t fTrkCutZ; // Z-difference cut for track matching
- Double_t fTrkCutR; // cut on allowed track-cluster distance

- Double_t fTrkCutAlphaMin; // cut on ’alpha’ parameter for track matching (min)
- Double_t fTrkCutAlphaMax; // cut on ’alpha’ parameter for track matching (min)
- Double_t fTrkCutAngle; // cut on relative angle between different track points for track matching
- Double_t fTrkCutNITS; // Number of ITS hits for track matching
- Double_t fTrkCutNTPC; // Number of TPC hits for track matching

PID

- Double_t fGamma[6][6]; // Parameter to Compute PID for photons
- Double_t fGamma1to10[6][6]; // Parameter to Compute PID not used
- Double_t fHadron[6][6]; // Parameter to Compute PID for hadrons
- Double_t fHadron1to10[6][6]; // Parameter to Compute PID for hadrons between 1 and 10 GeV
- Double_t fHadronEnergyProb[6]; // Parameter to Compute PID for energy ponderation for hadrons
- Double_t fPiZeroEnergyProb[6]; // Parameter to Compute PID for energy ponderation for Pi0
- Double_t fGammaEnergyProb[6]; // Parameter to Compute PID for energy ponderation for gamma
- Double_t fPiZero[6][6]; // Parameter to Compute PID for pi0

The macro

```
$ALICE_ROOT/EMCAL/macros/RecParamDB/AliEMCALSetRecParamCDB.C
```

is an example on how to set the parameters. There are different event types that we might record, and each event type may require different reconstruction parameters. The event types that are now defined in STEER/AliRecoParam.h are:

```
  enum EventSpecie\_t {kDefault = 1, kLowMult = 2, kHighMult = 4, kCosmic = 8,
      kCalib = 16};
```

The default event species that we have is kLowMult (low multiplicity). For AliRoot versions smaller than release 4.17 it was set to be kHighMult (high multiplicity). Today, the code is as follow :

```
kDefault=kLowMult=kCosmic=kCalib.
```

kHighMult differs only from the other two in 2 clusterization parameters, for low multiplicity they are fMinECut=10 MeV and fClusteringThreshold=100 MeV and for high multiplicity they are fMinECut=0.45 GeV and fClusteringThreshold=0.5 GeV.

A simple example that shows how to print the parameters for the different event species is PrintEMCALRecParam.C

3.5 Simulation parameters

The parameters used in the simulation are stored in EMCAL/Calib/SimParam. What is stored is an object of the class AliEMCALSimParam which is a container of all the parameters used. There are different kind of parameters depending on the step of the simulation :

SDigitization

- Float_t fA ; // Pedestal parameter
- Float_t fB ; // Slope Digitizition parameters
- Float_t fECPrimThreshold ; // To store primary if Shower Energy loss > threshold

Digitization

- Int_t fDigitThreshold ; // Threshold for storing digits in EMC = 3 ADC counts
- Int_t fMeanPhotonElectron ; // number of photon electrons per GeV deposited energy = 4400 MeV/photon
- Float_t fPinNoise ; // Electronics noise in EMC = 12 MeV
- Double_t fTimeResolution ; // Time resolution of FEE electronics = 600 ns
- Int_t fNADCEC ; // number of channels in EC section ADC =

The macro $ALICE_ROOT/EMCAL/macros/SimParamDB/AliEMCALSetSimParamCDB.C, is an example on how to set the parameters. A simple example that shows how to print the parameters is PrintEMCALSimParam.C

3.6 Alignment

4 Simulation code

The class AliSimulation manages this part. An example is here : "$ALICE_ROOT/EMCAL/macros/TestEMCALSimulation.C". The simulation consists of different steps: geometry and event definition, particle generation, transport of the particle in the material (GEANT) and finally digitization. Note that the final output from the digitization process is different from the processing of real experimental Raw Data. The process of converting the digitized data to Raw Data is discussed in Sec. 4.2. Sec. 4.4 gives the recipe to do all the steps of the simulation.

4.1 Event generation and particle transport: Hits

Once the generator is executed, the generated particles are transported in the detector material with the Monte Carlo code, GEANT3 by default. Other options are GEANT4 or FLUKA[1]. All the generated particles are kept in a file called **Kinematics.root**. After the particle transport is executed, the objects **Hits** are created. They contain the energy deposited in the sensitive material of the detector by the generated particle, their position, impact time (after collision) and the identity of the original particle. Hits are stored in a file called **DETECTOR.Hits.root**, in the calorimeter case: **EMCAL.Hits.root.**

4.2 Digitization: SDigits and Digits - Evi

We want to generate events which look like the real data collected by the experiment. In the end, we want to have an amplitude in ADC counts and a time (when particle traverse a cell) per each cell (tower) of the calorimeter. In the code for calorimeters, it is done in the following steps:

1. **SDigit** objects are created, they consist of the sum of deposited energy by all Hits in a cell (a particle can create Hits in different cells but only one in a single cell), so there is only one SDigit per fired cell.

2. **Digit** objects are created, they are like the SDigits but the energy in the cell is transformed into the ADC amplitude units, the electronic noise is added and Digits whose energy does not pass an energy threshold (3 ADC counts) are eliminated. SDigits and Digits are stored in the files **EMCAL.SDigits.root** and **EMCAL.Digits.root**, respectively.

4.3 Raw data - David

The experiment does not record Digits directly but a time samples of ADC counts per cell. These samples are called **Raw Data**. The samples have a shape, more complicated than a Gaussian distribution, which is fitted offline. With real data, Digits amplitude is just the maximum of the distribution obtained with the fit to the sample. The Digit time (defined by the time the particle hits the active volume of the detector) is the time bin when the signal begins to rise. There is a method to go from Digits to Raw and vice versa AliEMCALRawUtils class: Raw2Digits and Digits2Raw, respectively. For the reconstruction step Digits are needed. The generation of Raw

[1] There may be some license problems with FLUKA right now which could explain why it cannot be used at the moment

Data is optional during simulations and the generated data can be reconstructed directly from Digits, but Raw data will be the initial step when reconstructing real data.

4.4 How to make a simulation

TestEMCALSimulation.C is a very simple macro where we specify all the simulation parameters and proccess the simulation. Below is a similar but a bit more elaborated macro:

```
void TestEMCALSimulation() {

TString detector=``EMCAL TPC''; // Define in this variable the detectors that you want to be included in the simulation for the digitization . They can be less detectors than the detectors defined in the Config.C file , imagine that you want all the detectors in front of EMCal present to consider the conversion of particles but you are not really interested in the output from these detectors .
// Option detector=``ALL'' makes all detectors .

AliSimulation sim ; //Create simulation object

// Generation and simulation

sim.SetRunGeneration(kTRUE) ; //Default value is kTRUE, make generation
// For some reason we may want to redo the Digitization , without redoing the generation , in this case it must set to kFALSE

// Making SDigits
sim.SetMakeSDigits(detector) ; //We want to make SDigits
// set no detectors if SDigits are already made

// Making Digits
sim.SetMakeDigits(detector) ; //We want to make Digits
// set no detectors if SDigits are already made

//Merging
//sim.MergeWith(``bgrd/galice.root'') ; // If we want to merge a signal and a background, the merging is done at the SDigit level . The background must be located in the repertory defined in the method.

//Write Raw Data, make Raw data from digits
//sim.SetWriteRawData(detector) ;
//sim. SetConfigFile (`` somewhere/ConfigXXX.C'');//Default is Config.C

//Make the simulation
sim.Run(3) ; // Run the simulation and make 3 events
```

5 Reconstruction code

The energy deposited by the particles in the towers produces scintillating light that is propagated with optic fibers through the different layers to APD placed at the base of the cells. The APDs amplify the signal and generate an electronic pulse shape that is stored in the raw data format. From this pulse shape, we extract the signal amplitude and the arrival time. The pulse shape is fitted during the reconstruction via a parametrized function and TMinuit, and those 2 values are extracted.

A particle produces signals in different towers (electromagnetic shower expands more than its Molière radius which is a cell size). The next step is the formation of clusters of cells that belong to the same particle, although depending on the energy, granularity, clusterization algorithm or event type, those clusters might have contributions from different particles. The default algorithm in pp collisions is a simple aggregation of neighboring cells until there is no more cells above a certain energy threshold (named *clusterizer V1*). In case of Pb-Pb collisions environment, where particle showers merge quite often, we apply another algorithm that aggregates cells to the clusters until reaching a cell with more energy than the precedent (named *clusterizer V2*). Depending on the analysis type, one might want to use one or the other clusterization type. For this reason, a re-clusterization is also possible at the analysis level. A last clusterizer is implemented, which makes 3x3 clusters. It has been used in jet analysis for instance in order to avoid biasing jet reconstruction where one is interested in the energy flow over a large area without explicit reconstruction of photon showers and where the driving consideration is that the wide clusterizer does not interfere with the jet finder. For π^0, η, and direct γ analyses, *V2* is most likely preferable).

Once the cluster is defined, we calculate cluster parameters, shower shape parameters, that will help at the analysis level to identify each cluster as one particle type. Also, we compare the cluster position information with the propagation of tracks measured in the central barrel to the EMCAL surface, to identify the clusters generated by charged particles.

The final analysis objects, ESDs and AODs, contain all the cluster and cell basic informations allowing to redo the clusterization if needed at the analysis level.

5.1 Offline data base access

How to create explained OCDB/OADB section.

5.1.1 Energy calibration

5.1.2 Bad channels - Marie, Alexis

5.1.3 Alignment - Marco

5.2 Raw data fitting: from ADC sample to digits - David

`AliEMCALRawUtils`, `AliCaloRawAnalyzer*`, `AliCalo*`, `AliEMCALDigit`.

5.3 Clusterization: From digits to clusters - Constantin, Adam

AliEMCALClusterizer*, AliEMCALRecPoint

5.4 Cluster-Track matching - Rongrong, Shingo, Michael

Even though EMCal is intended to measure the energy of particles that interact with EMCal via electromagnetic showering, e.g. photons and electrons, charged hadrons can also deposit energy in EMCal, most commonly via minimum ionization, but also via nuclear interactions generating hadronic showers. In the analysis where the distinction between hadroic and electromagnetic showers is necessary, cluster-track matching is often used to meet this requirement.

The main method used to extrapolate tracks in the ALICE software framework is:

```
static Bool_t PropagateTrackToBxByBz(AliExternalTrackParam *track, Double_t x,
    Double_t m,Double_t maxStep, Bool_t rotateTo=kTRUE, Double_t maxSnp=0.8,Int_t
     sign=0, Bool_t addTimeStep=kFALSE);
```

which takes the following arguments: *"track"* stores all the information of the starting point for the extrapolation; *"x"* is the coordinate of the destination plane in the local coordinate system; *"m"* is the mass assumption for the track; *"maxStep"* is the step size used in the extrapolation. This method extrapolates the track trajectory to a destination plane in a magnetic field, taking into account the energy loss of the tracks when going through detector materials. However, the energy loss model is tuned for charged hadrons, so it does not work very well for electrons or positions whose primary energy loss process is bremsstrahlung.

For EMCal, the track-cluster matching is done by default in the reconstruction chain and the code is implemented in:

```
AliEMCALTracker::PropagateBack(AliESDEvent* esd)
```

The logic of the matching procedure is the following:

- Find all the EMCal clusters in the event. See AliEMCALTracker::LoadClusters().
- Find all the good tracks in the event. See AliEMCALTracker:: LoadTracks(). Several cuts are applied to select good tracks
  - Minimum p_{T} cut, which can be set during the reconstruction.
  - Cut on number of TPC clusters, which can be set during the reconstruction.
  - $|\eta| < 0.8$ and $20^\circ < \varphi < 120^\circ$. These fiducial cuts are hard coded since tracks out of this range should never make it to EMCal.
- For each good track, find the nearest cluster as matched if their residuals fall within the cuts. See AliEMCALTracker::FindMatchedCluster(), which follows the following steps:

- Get the starting point: if the *friendTrack* is available, use the last point on the TPC. Otherwise, use the point at the inner wall of the TPC.
- Extrapolate tracks to the EMCal surface at 430 cm, and apply fiducial cuts on the extrapolated points: $|\eta| < 0.75$ and $70^\circ < \varphi < 190^\circ$. The step size in the extrapolation can be set in the reconstruction, and the default value is 20 cm.
- Extrapolate tracks further, with 5 cm step size, to the positions of all the EMCal clusters which are in the vicinity of the extrapolated points from last step. Then the distance between extrapolated tracks to the clusters are calculated, and the nearest cluster is assigned as matched if the residuals fall within cuts. By fitting the distributions of the residuals using Gaussian functions, we can choose to cut on $N\sigma$ of the residuals.To further improve the matching performance, p_T and charge dependent cuts can be used.

5.5 How to execute the reconstruction

Executing the reconstruction is very similar to the simulation case, see the macro TestEMCAL-Reconstruction.C (a bit more detailed than the one in $ALICE_ROOT/EMCAL/macros) :

```
void TestEMCALReconstruction() {

TString detector=``EMCAL TPC'';//Same function as in Simulation.C

AliReconstruction rec; //Create reconstruction object

//Making Tracking
rec.SetRunTracking(detector) ;

// Particle  Reconstruction . Make Rec Points
rec.SetRunReconstruction(detector);

//read RAW data. Give directory where raw data is stored
//rec . SetInput (``RawDataDirectory/raw.root");

//Make vertex finder
rec.SetRunVertexFinder(kFALSE) ; //false only if the tracking detectors are not included .

// Fill ESD file with RecPoints information .
rec.SetFillESD(detector) ;

//Run Reconstruction
rec.Run() ;
}
```

6 Calibration and detector behavior

6.1 Calibration

This section describes how different correction factors are obtained: the energy calibration (MIP, π^0, run by run), the time calibration and the bad channel mask.

All these correction factors or masks are stored in the OCDB but also the OADB. Since these calibration parameters do not arrive before the full ALICE data reconstructions of the first periods are completed, the parameters are stored not only in the OCDB but also in the OADB so that the clusters can be corrected at the analysis level. For the moment we do not store the time calibration and run by run correction factors in OCDB just in OADB.

6.1.1 Energy calibration: MIP calibration before installation - Julien

First, the calibration is done on cosmic measurements before installing the SuperModules at P2, but the accuracy obtained using MIPs is not good enough.

6.1.2 Energy calibration: π^0 - Catherine

The energy calibration relies during data taking on the measurement of the π^0 mass position per cell. Each tower has a calibration coefficient. In what follows, a calibration parameter is equal to the result of the fitted mass over the PDG mass value, where the fitted mass denotes the mass given by a gaussian fit on the π^0 invariant mass peak distribution in a given tower (plus a combinatorial background, fitted by a 2nd degree polynomial). About 100-200 M events EMCAL (L0) triggered (trigger threshold at 1.5-2 GeV) allow to calibrate a majority of the towers. The towers located on rows 0 and 23 of each super modul (SM) and those behind the support frame (about 5 columns per SM) have much fewer statistics and would need a minimum of 150 Mevts (probably more). It is to be noted that the run-to-run temperature variations change the towers' response in a non-uniform way, i.e. the width of the π^0 peak increases, and the mean π^0 mass is shifted differently for the various towers. Also the π^0 mass shifts to lower values for the towers with material in front, due to photoconversion close to the EMCAL surface.

A few iterations on the data, obtaining in each iteration improved calibration coefficients, are needed to achieve a good accuracy (1-2%). Since the online calibration has a strong effect on the trigger efficiency, the voltage gains of the APDs are varied after each running period, to get a uniform trigger performance. Still, some towers are difficult to calibrate because they are behind of a lot of material (TRD support structures). For those MIPs or J/Ψ measurements could help.

π^0 Calibration Procedure

Since π^0s decay into 2 gammas, their invariant mass is calculated from the energy of 2 clusters (and angle between the clusters). The position of the invariant mass peak of a tower therefore doesn't depend only on its response and calibration coefficient, but also on an average of the responses and calibration coefficients of all the other towers of the SM, weighted by how often they appear in combination with a cluster in the considered tower. The 2nd effect, of weaker magnitude maybe, originates from the fact that a cluster most often covers more than the considered tower. To simplify the calibration process, the calibration coefficient is calculated as if the whole energy of the cluster was contained in the tower of the cluster which has the largest signal. So the position of the invariant mass peak of a tower also depends on an average of the responses and calib coeffs of its neighbouring towers. For these reasons, the calibration of the calorimeter with the π^0 is an iterative procedure :

- Set all calib coeffs to 0 in OCDB.
- Reconstruct the π^0's with these OCDB coeffs.
- Run the analysis code on this data to produce the analysis histograms and a 1st version of the calib coeffs.
- Look at the fits on the towers invariant mass histograms and discard the value (or set it by hand) of the calib coeff of the towers for which the fit can't be trusted.
- Create a 1st set of OCDB coeffs.
- Reconstruct the π^0's with these OCDB coeffs.
- Run the analysis code on this data to produce the analysis histograms and a 2nd version of the calib coeffs.
- Look at the fits on the towers invariant mass histograms and discard the value (or set it by hand) of the calib coeff of the towers for which the fit can't be trusted.
- Create a 2nd set of OCDB coeffs.
- Etc..., until the invariant mass is satisfactory in all the towers.

When the statistics is enough, 4 iterations should be enough to finalize the calibration (in practice, more are needed, due to outliers or studies that are needed).

There are 3 sets of codes :

- Reco code : reads the data, reconstructs the π^0 inv mass distrib in each tower after it applies some cuts on the clusters and π^0 parameters. The output is a root file with invariant mass histograms (per tower, and summed-up per SM, per pT-bin).
- Analysis code : reads the file produced by the reco code and analyses the histograms to produce the calib coeffs. This code is the one I present in what follows.
- A code which reads the calib coeffs and writes them into a format that is loadable to OCDB.

The code is located in EMCAL/calibPi0/ :

- macros/ : contains the various macros.

– input/ : contains the root files produced by the analysis code for the various iterations ("passes"). It has subdirectories "pass0/", "pass1/", etc... with, in each dir, the root file.

– output/ : contains the various files produced by the analysis code for the various passes. It has subdirectories "pass0/", "pass1/", etc... with, in each dir, the various output files related to the pass.[2]

The cuts which must be put in the reconstruction are :

– Bad towers masked.

– Both clusters in the same SM (to avoid misalignment effects).

– Cut the 1-tower clusters out.

– 20 ns timing cut.

– Non-linearity correction (for the cluster energy)– from beam test AFAIK.

– No asymetry cut.

– $E_{cluster} > 0.8$ GeV, or 0.7 GeV if there is little statistics. Tests showed that to remove the residual non-linearity (the pi_0 invariant mass rises with p_T), tightening the cut on $E_{cluster}$ was more efficient than requiring symetric decays (both gamma's of similar energy) (e.g. $asym < 0.5$ with $E_{gamma} > 0.5$ GeV).

It has the possibility to mask some areas. This is useful to disentangle the zones which have more material in front of them from those which don't. In the invariant mass distributions, the π^0 candidates kept are only those for which both clusters belong to the non-masked zones. In 2011, we considered masking the zones behind the support frame (in all the SMs or only in the SMs with TRD modules in front of them, i.e. SM 6-9 that year), plus additionnal problematic zones, to avoid taking clusters in these zones for the calculation of the average invariant mass in the towers with less material. (NB : not used for final calibration results, but for studies).

The analysis code has 3 input files :

– the root file f05 with inv mass histograms produced by the reconstruction code,

– a file txtFileIn (output_calibPi0_parameters.txt) that contains the values of various parameters of the fit for each tower, at the previous pass,

– a file txtFilePrevCalib (output_calibPi0_coeffs_clean.txt) that contains the value of the calibration coefficient for each tower, at the previous pass (and after the hand-made corrections).

[2]Note that it wouldn't necessarily help to set-up a code that automatically reads and writes the pass number to avoid the hardcoded directories in the code, because it happens to do several times the same pass with various parameters (e.g. cuts in the reconstruction, or more statistics, or various masked zones, or hand-customization of a few calib coeffs, etc...).

The 2 last files are therefore useless for the "pass0". To run the code for "pass0" (1st iteration), put the name of a valid file (e.g. one of last year) and just ignore the plots (red colour, in the last section – see below).

There are 4 output files, that are written in the current directory (calibPi0/) : be careful not to overwrite an existing file ! After the code has been run, simply move those files to the relevant passXX directory := output/passXX/=.

- a postscript file psfile (output_calibPi0.ps) with the plots described below,
- a root file rootFileOut (output_calibPi0.root) that contains the same plots in root format,
- a file txtFileOut (output_calibPi0_parameters.txt) that contains the values of various parameters of the fit for each tower, for the current pass,
- a file outputFile (output_calibPi0_coeffs.txt) that contains the value of the calib coeff for each tower, for the current pass. Once the code has been run and the output files copied to the relevant output directory, I copy output_calibPi0_coeffs.txt to output_calibPi0_coeffs_clean.txt, and modify the latter by hand to put the desired calib coeffs where we estimate that they can't be trusted.

9 parameters are defined to qualify the invariant mass distribution in each tower : the distribution is fitted by a gaussian + pol2 for the combinatorial background. The parameters are :

- amplitude of gaussian fit,
- mean of the gaussian fit,
- sigma of the gaussian fit,
- c, b and a parameters of the combinatorial background fit ax^2+bx+c, I (histo integral),
- I-S, S (integral of the gaussian fit). Minimal and maximal cut values are hardcoded (and to be changed at each iteration) for each parameter.

When the value of all the parameters lie between both extremes, the tower (i.e. the fit values, hence the mean, hence the calculated calib coeff) is "trusted". If one or more parameter has a value beyond the max cut value or below the min cut value, the tower is "untrusted". Because these cut values can't be guessed in advance, the analysis code must be run twice per pass. The 1st time, so as to get the distributions of all 9 parameters, and decide on the basis of those distributions what are the suitable cut values to separate the towers to be trusted and those not to be trusted. The values are plugged in the code, and the code is then run a 2nd time, for real this time. The macro (currently called DrawJulienFullEMCAL6.C) runs with 1 parameter in argument (set to 10 by default) : choice, which sets the number of SMs that one desires to include in the analysis. The values are either 4 (for the older SMs), or 6 (for only the newer SMs), or 10 (for the whole EMCAL). Here is the code. The macro is run this way :

```
aliroot -b -q 'macros/DrawJulienFullEMCAL6.C++(10)'
```

There are various places where things must be customized before running the code ; they can be spotted by searching for this line : //CUSTOMIZE customize :.

- testChoice : this variable is a flag that allows to shorten the execution time for tests. 0 = not a test ; 1 = runs with only the 2 first columns of each SM ; 2 = runs with only the 2 first columns of the first SM,
- the root input file f05,
- the text input file txtFilePrevCalib (in principle not the name, only the path),
- the text input file txtFileIn (in principle not the name, only the path),
- if necessary : the min and max range values for the parameter histograms : tabMin and tabMax,
- the min and max cut values for the parameters cutMin and cutMax,
- if necessary : the number of bins in pT (for the 1st section, see below) nbPtBins and their range tabPtBins.
- Text output on the standard output ("printf's") :

Finally, the first iteration needs the recalibration factors. This file is made running macros/RecalibrationFactors_TextToHistoJulien_mult_2012.C on the output_calibPi0_coeffs.txt file. Once the RecalibrationFactors.root file is created it needs to be linked properly to re-run the reconstruction.

6.1.3 *Energy calibration: Run by run temperature gain variations - Evi, David*

The SuperModules calibration depends on the temperature dependence of the different towers gains. We observe that from one period to other, where the T changes, the π^0 peak positions also changes. There are 2 ways to correct for this effect : either measure the mean T per run, and get the gain curves per tower a calculate the corresponding correction; or use the calibration LED events to quantify the variation from one reference run. Each of those 2 procedures have problems, poor or lack of knowledge of the gain curves of some towers or bad performance of the LED system in certain regions.

6.1.4 *Time calibration - Marie*

The time of the amplitude measured by a given cell is a good candidate to reject noisy towers, identify pile up events, or even identify heavy hadrons at low energy. The average time is around 650 ns. The aim of the time calibration is to move this mean value to 0, with as small spread as possible (negative values are unavoidable for the moment).

6.2 Alignment - Marco

CERN provides survey measurements of the position of different EMCAL Supermodules points at the beginning of the running period (and on request?). As soon this information is available, the ideal EMCAL positions used in the reconstruction by default, are corrected with special position matrices calculated from the measurements. Finally, once the data is reconstructed, the accuracy of the alignment is cross checked with track matching and π^0 mass measurements, since those values change depending on variations on the positions of the SuperModules.

6.3 Bad channel finding - Alexis

The analysis is done on the output of QA histograms. The idea is to check distributions over the cells of:

- average energy (criteria 1) and
- average number of hit per event (criteria 2) (average computed for E > Emin)
- Shape criteria : χ^2/ndf (criteria 3), A (criteria 4) and B (criteria 5) which are parameters from the fit of each cell amplitude (the fit function is $A * e^{-B*x}/x^2$ and the fit range is from Emin to Emax).

Each criteria is ran once, at each step, and the marked cells are excluded (above nsigma from mean value) to compute the next distribution. [3]

The typical nsigma used is 4 or 5. The min energy considered is 0.1 GeV -0.3 GeV. And the max energy for the fit depends on the data. Bad/warm channels are not detected automatically. The distinction is made by a visual check, so it is at some point subjective. (??????)

The cells are then marked bad or warm and passed through OCDB, in the reconstruction pass, the bad ones are excluded.

[3] For each criteria we have some parameters Emin (min energy) Emax, (max energy for the Energy distribution fit), and nsigma, nb of sigma we use for excluding the cell;

7 Trigger

7.1 L0 - Jiri

Documented in [10]. Add Summary or more info here.

7.2 L1 - Rachid

7.3 L0-L1 simulation - Rachid

8 The EMCal HLT online chain - Federico

The EMCal L0 or L1 hardware trigger decisions provide the input for a dedicated on-line event processing chain running on the HLT cluster, where further refinement based on criteria using the full event reconstruction information is performed. In fact, the detector optical link transports the raw data to the Read-Out Receiver Card (RORC) in the local data collector of the data acquisition system, which sends a complete copy of the readout to a set of specialized nodes in the HLT cluster (FEP or Front End Processors). Each FEP node is equipped with RORC cards in analogy to the collector nodes used by the data acquisition. The FEP nodes are physically linked to the detector hardware and reflect the geometrical partitioning of each ALICE sub-system. The 10 full-size super-modules are read out using 2 Read-Out Control Units (RCUs) for a total of 20 optical links running into the HLT FEPs. The reduced-size super-modules were installed prior to the 2012 LHC run and are not discussed in the present report. In addition to the 20 links from the super-module readout, the HLT receives also a copy of the L0/L1 trigger data stream via an additional optical link from the EMCal jet trigger unit (STU) data collector. The different stages of data processing are then performed by the software analysis chain executed on the HLT cluster: a set of general purpose nodes (Computing Nodes or CNs) perform the higher level operations on the data streams which have been already pre-processed on the FEPs at the lower level. The EMCal software components form a specialized sub-chain executed at run time together with all other ALICE sub-systems participating in the HLT event reconstruction.

The functional units of the EMCal HLT online chain are presented in Figure 8 where the online reconstruction, monitoring, and trigger components are shown together with their relevant data paths. The lower-level EMCal online component (*RawAnalyzer*) is fed by the detector front end electronics and performs signal amplitude and timing information extraction. Intermediate components (*DigitMaker*) use this information to build the digitized data structures needed for the clusterizer components to operate on the cell signals. Alternatively, the digitized signals can be generated via monte carlo simulations (*DigitHandler*).

At the top of the EMCal reconstruction chain, the digits are summed by the *Clusterizer* component to produce the cluster data structures. The calorimeter clusters are then used to generate the different kinds of EMCal HLT trigger information: a single shower trigger (γ) with no track matching, an electron trigger using the matching with a corresponding TPC track, and a jet trigger also using the TPC tracks information and the V0 multiplicity dependent threshold.

The trigger logic generated by the EMCal chain is evaluated (together with the outputs of the HLT trigger components coming from other ALICE detectors) within the HLT Global Trigger

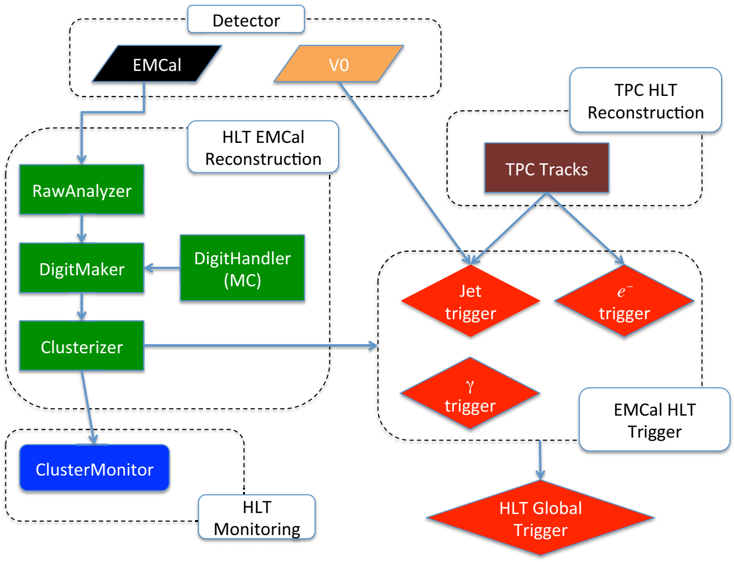

Fig. 8: Functional diagram of the EMCal online reconstruction components (signal processing, data structure makers, and clusterizers) shown in green. The EMCal chain is fed by the detector raw data. Trigger components are shown in red. EMCal-specific triggers operate on the calorimeter clusters and perform TPC track-matching when needed (electron and jet triggers). Monitoring components are shown in blue and live in a separate monitoring chain. The EMCal triggers are evaluated within the Global Trigger which is aware of the full HLT trigger logic of the other ALICE detectors.

which produces the final high level decision based on the reconstructed event. The ALICE data acquisition system will then discard, accept or tag the event according to the HLT decision.

For performance and stability reasons, the full on-line HLT chain contains only analysis and trigger components. On the other hand, monitoring components typically make heavy use of histogramming packages and ESD objects, hence they are kept in a separate chain. The isolation of the monitoring from the reconstruction chain gives additional robustness since a crash in a monitoring component will not affect the reconstruction chain and the data taking.

8.1 Reconstruction components

As shown in Figure 8 the EMCal HLT analysis chain provides all the necessary components to allow the formation of a trigger decision based on full event reconstruction. The following subsections are devoted to a detailed discussion of each processing stage, starting from the most basic, i.e. signal extraction, to the highest stage: the HLT trigger decision.

8.1.1 RawAnalyzer

The *RawAnalyzer* component extracts energy and timing information for each calorimeter cell. Extraction methods implemented in the offline code (AliRoot) typically use least squares fitting algorithms, and cannot be used in online processing for performance reasons. Conversely, the HLT signal extraction is done without need of fitting using two possible extraction methods. The first method, referred to as *kCrude*, simply produces an amplitude using the difference between the maximum and the minimum values of the digitized time samples and associates the time bin of the maximum as the signal arrival time. The *kCrude* method was used during the 2011 data taking: it has the advantage of being extremely fast and fully robust since no complex algorithms are used. On the other hand, it produces a less accurate result than the processing of the full signal shape. An alternative method (*kPeakFinder*) evaluates the amplitude and peak position as a weighted sum of the digitized samples. This approach is not as fast as *kCrude* but is a few hundred times faster than least squares fitting.

8.1.2 DigitMaker

The *DigitMaker* component essentially transforms the raw cell signal amplitudes produced by the *RawAnalyzer* into digit structures by processing the cell coordinates and by the application of dead channel maps and the appropriate gain factors (low and high-gain).

8.1.3 Clusterizer

The *Clusterizer* component merges individual signals (digits) of adjacent cells into structures called clusters. At transverse momenta $p_T > 1$ GeV/c most of the clusters are associated to electromagnetic showers in EMCal from π^0 and η mesons decays. Other sources of electromagnetic showers are direct photons and electrons from semi-leptonic decays of *c* and *b* hadrons. Since the typical cluster size in the EMCal can vary according to the detector occupancy due to shower overlap effects, which are much different for *pp* and heavy-ion collisions, clustering algorithms with and without a cutoff on the shower size are available (both in offline and in the HLT) to optimize the cluster reconstruction for the different cases. Events originating from *pp* collisions tends to generate smaller, spherical and well-separated clusters in the EMCal, at least up to 10 GeV/c. At higher transverse momenta, overlapping of the showers requires a shape analysis to extract the single shower energy. Above 30 GeV/c the reconstruction can be performed only with more sophisticated algorithms such as isolation cuts to identify direct photons.

The identification of an isolated single electromagnetic cluster in the EMCal can be performed using different strategies: summing up all the neighboring cells around a seed-cell over threshold until no more cells are found or adding up cells around the seed until the number of clustered cells reaches the predefined cutoff value.

The first approach is more suitable for an accurate reconstruction. A further improvement to this clustering algorithm would be the ability to unfold overlapping clusters as generated from the photonic decay of high-energy neutral mesons, however this procedure usually requires computing intensive fitting algorithms.

Such performance penalty must be avoided in the online reconstruction so the cutoff technique is preferred. In the EMCal HLT reconstruction a cutoff of 9 cells is used (according to the geometrical granularity of the single cell size), so the clusterization is performed into a square of 3×3 cells. The cutoff and non-cutoff algorithms are referred to as $N \times N$ and $V1$, respectively.

In *pp* collisions the response of the two methods is very similar since the majority of clusters are well separated, while in *PbPb* collisions, especially in central events, the high particle multiplic-

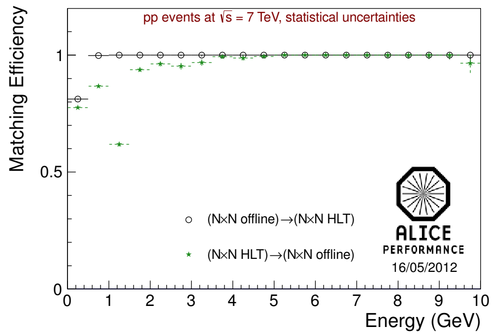

Fig. 9: Reconstruction efficiency for the $N \times N$ algorithm (cutoff) in offline and HLT. The notation $(A) \to (B)$ indicates the fraction of clusters found using method A that are also found using method B (data from run 154787, period LHC11c).

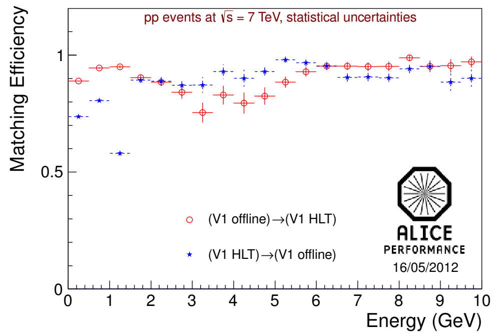

Fig. 10: Reconstruction efficiency for the $V1$ algorithms (no cutoff) in offline and HLT. The notation $(A) \to (B)$ indicates the fraction of clusters found using method A that are also found using method B (data from run 154787, period LHC11c).

ity requires the use of the cutoff (or unfolding in offline) to disentangle the cluster signals from the the underlying event to avoid the generation of artificially large clusters.

The quality of the EMCal online clusterizer algorithms implemented in the HLT chain were checked against offline, as shown in Figures 9 and 10 where it can be seen that the performance is in a reasonable agreement in all cases. The low point at 1.25 GeV is due to bad towers, which are assigned an energy of 1 GeV. Bad clusters are removed in later stages of the analysis, but that is not yet reflected in Figures 9 and 10. This effect leads to an excess of clusters that are found by the HLT clusterizer, but not by the offline clusterizer.

Since the EMCal HLT reconstruction is mainly targeted for triggering, a small penalty in the accuracy of the energy reconstruction of the clusters is accepted as a trade off in favor of faster performance, and for this reason the cutoff clustering method was used, especially in *PbPb* collisions.

8.2 Trigger components

The online HLT chain is capable of producing trigger decisions based on full event reconstruction. In terms of EMCal event rejection the following relevant trigger observables have been implemented:

– neutral cluster trigger

– electron and jet trigger

8.2.1 Cluster trigger

The single shower triggering mode is primarily targeted to trigger on photons and neutral mesons. In all collision systems, the high level trigger post-filtering can improve the hardware L0 and L1 trigger response by using the current bad channels map information and calibration factors (which could be recomputed directly in the HLT).

8.2.2 Electron trigger

For this trigger the cluster information reconstructed online by the EMCal HLT analysis chain is combined with the central barrel tracking information to produce complex event selection as a single electron trigger (matching of one extrapolated track with an EMCal cluster. Performance and accuracy studies of the track matching component developed for this purpose have been done using simulated and real data taken during the 2011 LHC running period. Results are shown in Figures 11 and 12 where the cluster - track residuals in azimuth and pseudo-rapidity units are to be compared with a calorimeter cell size of 0.014×0.014.

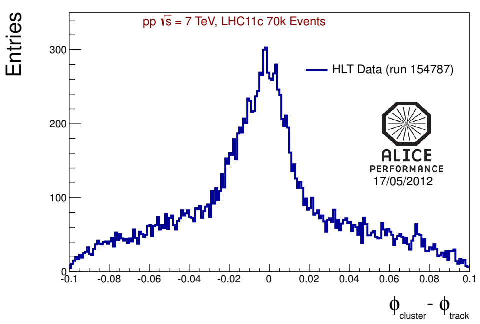

Fig. 11: Distribution of the residuals in azimuth ($\Delta\phi$) for the EMCal cluster and central barrel tracks obtained using the HLT online chain for run 154787 (LHC11c), 70 k events reconstructed.

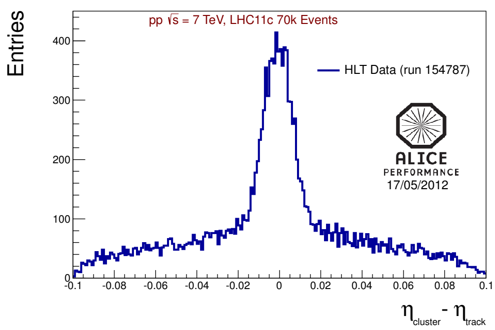

Fig. 12: Distribution of the residuals in pseudo-rapidity ($\Delta\eta$) for the EMCal cluster and central barrel tracks obtained using the HLT online chain for run 154787 (LHC11c), 70 k events reconstructed.

In addition to the extrapolation of the track from the central barrel to the EMCal interaction plane and the matching with a compatible nearby cluster, the electron trigger component must finally perform particle identification to issue a trigger decision. The selection of electron candidates is done using the E/pc information where the energy is measured from the EMCal cluster and the momentum from the central barrel track. The trigger component is initialized with default values for the cut of $0.8 < E/pc < 1.3$. The default cuts are stored in the HLT conditions database and can be overridden via command line arguments at configuration time (usually at start of run).

The performance of the electron trigger was studied using *pp* minimum bias data at 7 TeV with embedded J/Ψ events. Figure 13 shows the good agreement of the E/pc distributions obtained with the track extrapolation - cluster matching performed using the online algorithms compared to the ESD-based tracking (red).

To determine the possible improvement of the event selection for electrons with energies above

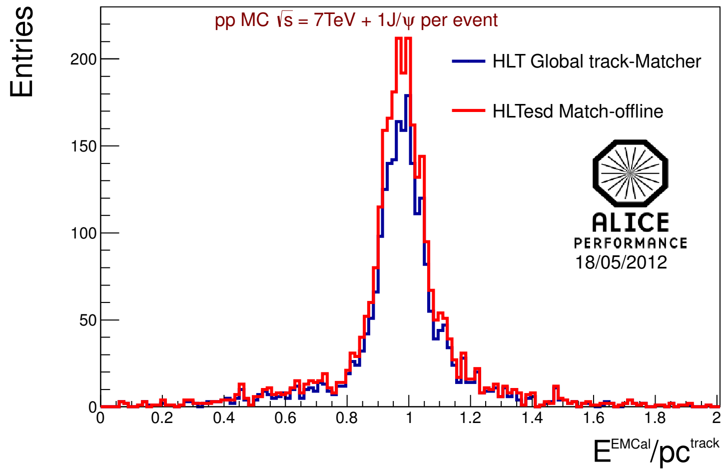

Fig. 13: E/pc distributions obtained with the track extrapolation - cluster matching via the online algorithms compared to the ESD-based tracking (red).

1 GeV, AliRoot simulations of the HLT chain using LHC11b10a *pp* minimum bias data at 2.76 GeV and the EMCal full geometry (10 super-modules) have been used. These studies have shown that at least a factor 5 to 10 in event selection can be gained compared to the single shower trigger, as shown in Figure 14.

8.2.3 Jet trigger

The EMCal online jet trigger component was developed to provide an unbiased jet sample by refining the hardware L1 trigger decisions. In fact, the HLT post-processing can produce a sharper turn on curve using the track matching capabilities of the online reconstruction chain. In addition, a more accurate definition of the jet area than the one provided by the hardware L1 jet patch, can be obtained choosing a jet cone based on the jet direction calculated online. The combination of the hadronic and electromagnetic energy provides a measurement of the total energy of the jet by matching the tracks identified as part of the jet with the corresponding EMCal neutral energy.

The use of the HLT jet trigger also allows a better characterization of the trigger response as a function of the centrality dependent threshold by re-processing the information from the V0 detector directly in HLT.

Performance considerations, due to the high particle multiplicity in *PbPb* collisions, impose that the track extrapolation is done only geometrically without taking into account multiple scattering effects introduced by the material budget in front of the EMCal. The pure geometrical extrapolation accounts for a speedup factor of 20 in the execution of the track matcher component with respect to the full-fledged track extrapolation used in *pp* collisions.

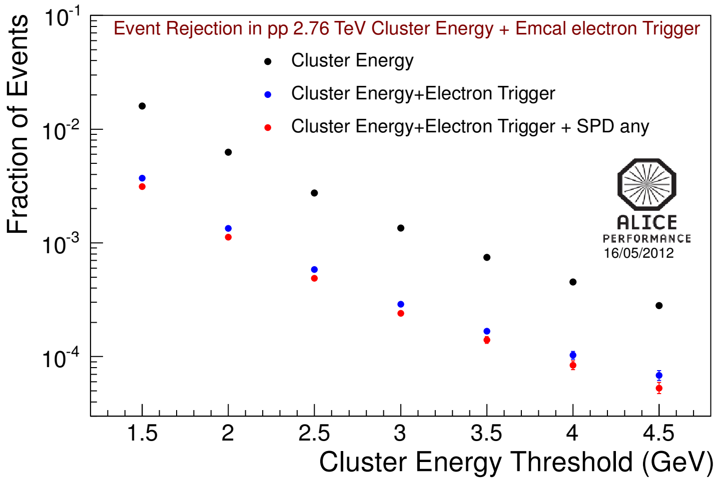

Fig. 14: Improvement in the event selection for $E_{e^-} > 1$ GeV from AliRoot simulation (anchor to LHC11b10a) with minimum bias *pp* at $\sqrt{s} = 2.76$ TeV (EMCal full geometry). The red points are obtained with the requirement of one hit in one of the silicon pixel (SPD) layers to reject a higher fraction of photon conversions.

The identification of the jet tracks is performed using the anti-k_T jet finder provided by the FastJet package.

The EMCal jet trigger was only partially tested during the 2011 data taking period and will be fully commissioned for the LHC *pPb* run period in 2012.

8.3 Monitoring components

The role of the EMCal HLT reconstruction in *pp* collisions is targeted mainly on the monitoring functions since the expected event sizes are small enough for the complete collision event to be fully transferred to permanent storage.

In this respect, two monitoring components have been developed and deployed in the online chain. The first component currently monitors reconstructed quantities, such as the cluster energy spectra and timing, the cluster position in the η and ϕ coordinates, and the number of cells per cluster as a function of the cluster reconstructed energy as shown in Figure 15.

The second component re-evaluates the EMCal hardware trigger decisions by recalculating the cluster energy spectrum for all the clusters with the L0 trigger bit set as shown in Figure 16. The L0 turn on curve can then be calculated online as the ratio between the triggered and the reconstructed cluster spectra and monitored for the specific run.

No recalculation of hardware L1 trigger primitives was possible during the 2011 data taking since the optical link from the EMCal L1 trigger unit could only installed during the 2011-2012 winter shutdown of the LHC hence the software development for the L1 trigger monitoring is

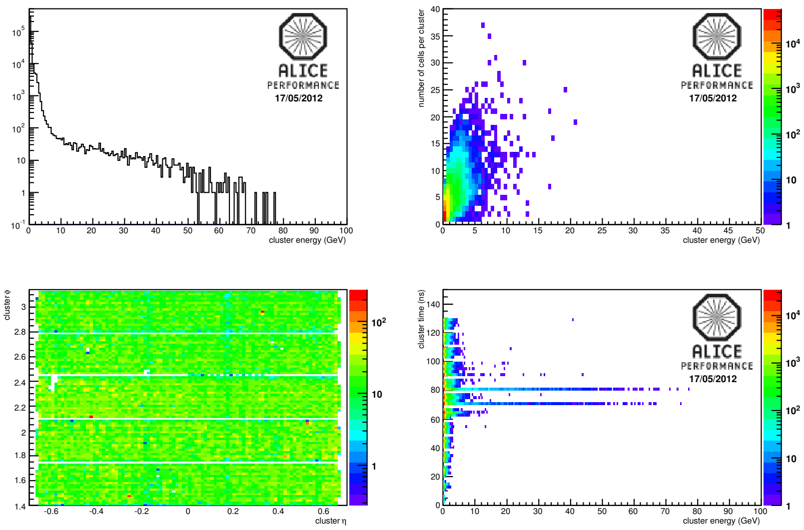

Fig. 15: Output from the EMCal HLT monitoring component. Top left: cluster energy spectra as a function of the reconstructed cluster energy; bottom left: cluster position in η and ϕ coordinates; bottom right: cluster time distribution; top right: number of cells per cluster vs cluster energy. LHC11b period, $\sqrt{s} = 7$ TeV *pp* data, 10 kEvent analyzed.

still underway.

Documented in [11]. Add Summary or more info here.

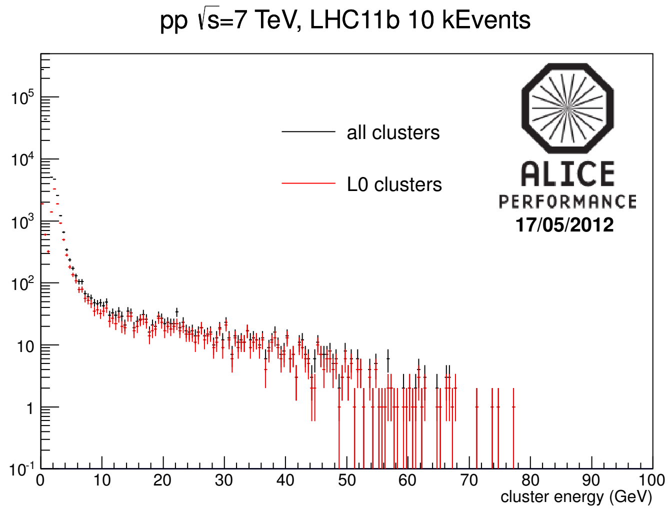

Fig. 16: Energy spectrum for all clusters reconstructed by the EMCal (black points) superposed with the triggered cluster spectrum (i.e. clusters reconstructed which also carry the L0 hardware trigger bit set, red points).

9 Analysis format and code

All the reconstructed particles of all the detectors are kept in a file called **AliESDs.root**. The detectors must store there the most relevant information which will be used in the analysis. Together with the AliESDs.root file, another file is created with some reference tags of the simulated events, containing for example the number of events per run. This file is named **Run0.Event0_1.ESD.tag.root** (1 means that only 1 event was simulated).

In order to do the analysis with the data contained in the ESDs, the only file needed is **AliESDs.root** in the local directories or a grid collection. No other files are needed in the working directory (such as galice.root nor EMCAL.*.root) unless one needs to access the primary particles generated during the simulation. In that case, the files **galice.root** and **Kinematics.root** are needed locally. Also, if one want to access to some information of the detector geometry, the **geometry.root** file is needed.

There are other data analysis containers created from the ESD, the AOD (Analysis Object Data) with smaller quantity of data for most of the subsystems but for the calorimeters, where we copy all the information[4].

9.1 Calorimeter information in ESDs/AODs

The basic calorimeter information needed for analysis is stored in the ESDs or AODs in the form of CaloClusters and CaloCells (cell = EMCal Tower or PHOS crystal). Also there is some information stored in the AOD/ESD event classes, it will be detailed more in the lines below. Both AOD and ESD classes derive from virtual classes so that with a similar analysis code and access methods, we can read both kind of data formats.

9.1.1 AliVEvent (AliESDEvent, AliAODEvent)

Those are manager classes for the event information retrieval. Regarding the calorimeters they have the following access information (getters) methods[5]:

– AliVCaloCluster *GetCaloCluster(Int_t i) : Returns a CaloCluster listed in position "i" in the array of CaloClusters. It can be either PHOS or EMCal (PHOS list of clusters is before the EMCal list).

– TClonesArray *GetCaloClusters() : Returns the array with CaloClusters PHOS+EMCAL, Only defined for AODs

– Int_t GetEMCALClusters(TRefArray *clusters) ; Int_t GetPHOSClusters(TRefArray *clusters) : Returns an array with only EMCal clusters or only with PHOS clusters.

– Int_t GetNumberOfCaloClusters(): Returns the total number of clusters PHOS+EMCAL.

– AliVCaloCells *GetEMCALCells(); AliESDCaloCells *GetPHOSCells() : Returns the pointer with the CaloCells object for EMCal or PHOS.

[4]until half 2012 everything but the time of the cells was stored

[5]There are the equivalent setters just have a look to the header file of the class

- AliVCaloTrigger *GetCaloTrigger(TString calo) : Access to trigger patch information, for calo="PHOS" or calo="EMCAL"
- const TGeoHMatrix* GetPHOSMatrix(Int_t i); const TGeoHMatrix* GetEMCALMatrix(Int_t i): Get the matrices for the transformation of global to local. The transformation matrices are not stored in the AODs.

9.1.2 AliVCaloCluster (AliESDCaloCluster,AliAODCaloCluster)

They contain the information of the calorimeter clusters. Note that PHOS and EMCAL CaloClusters are kept in the same TClonesArray (see above). The information stored in each CaloCluster is :

- General
  - Int_t GetID(): It returns a unique identifier number for a CaloCluster.
  - Char_t GetClusterType():It returns kPHOSNeutral (kPHOSCharged exists but not used) or kEMCALClusterv1. Another way to get the origin of the cluster:
  - Bool_t IsEMCAL(); Bool_t IsPHOS().
  - void GetPosition(Float_t *pos) : It returns a x,y,z array with the global positions of the clusters in centimeters.
  - Double_t E() : It returns the energy of the cluster in GeV units.
  - void GetMomentum(TLorentzVector& p, Double_t * vertexPosition): It fills a TLorentzVector pointing to the measured vertex of the collision. It also modifies the cluster global positions to have a vector pointing to the vertex, this has to be corrected. Assumes that cluster is neutral. To be used only for analysis with clusters not matched with tracks.
- Shower Shape
  - Double_t GetDispersion(): Dispersion of the shower.
  - Double_t Chi2(): Not filled.
  - Double_t GetM20() Double_t GetM02() : Ellipse axis.
  - UChar_t GetNExMax() : Number or maxima in cluster. Not filled.
  - Double_t *GetPID(): PID weights array, 10 entries corresponding to the ones defined in AliPID.h
  - enum EParticleType kElectron = 0, kMuon = 1, kPion = 2, kKaon = 3, kProton = 4, kPhoton = 5, kPi0 = 6, kNeutron =7, kKaon0 = 8, kEleCon = 9,kUnknown = 10; : PID tag numbers, corresponding to the PID array
  - Double_t GetDistanceToBadChannel() : Distance of the cluster to closest channel declared as kDead, kWarm or kHot.
  - Double_t GetTOF() : Measured Time of Flight of the cluster.

- Track-Cluster matching
  - TArrayI * GetTracksMatched(): List of indexes to the likely matched tracks. Tracks ordered in matching likeliness. If there is no match at all, by default it contains one entry with value -1. Only in ESDs.
  - Int_t GetTrackMatchedIndex(Int_t i): Index of track in position "i" in the list of indices stored in GetTracksMatched(). Only in ESDs
  - Int_t GetNTracksMatched() : Total number of likely matched tracks. Size of GetTracksMatched() array.
  - Double_t GetEmcCpvDistance() : PHOS method, not used anymore. Use instead those below.
  - Double_t GetTrackDx(void), Double_t GetTrackDz(void): Distance in x and z to closest track.
  - TObject * GetTrackMatched(Int_t i): References to the list of most likely matched tracks are stored in a TRefArray. This method retrives the one in position "i". Tracks are listed in order of likeliness. The TObject is a AliAODTrack. Only for AODs
- MonteCarlo labels:
  - TArrayI * GetLabels(): List of indexes to the MonteCarlo particles that contribute to the cluster. Labels ordered in energy contribution.
  - Int_t GetLabel(): Index of MonteCarlo particle that deposited more energy in the cluster. First entry of GetLabels() array.
  - Int_t GetLabelAt(UInt_t i): Index of MonteCarlo particle in position i of the array of MonteCarlo indices.
  - Int_t GetNLabels() : Total number of MonteCarlo particles that deposited energy. Size of GetLabels() array.
- Cluster cells
  - Int_t GetNCells() : It returns the number of cells that contribute to the cluster.
  - UShort_t *GetCellsAbsId(): It returns the array with absolute id number of the cells contributing to the cluster. Size of the array is given by GetNCells().
  - Double32_t *GetCellsAmplitudeFraction(): For cluster unfolding, it returns an array with the fraction the energy that a cell contributes to the cluster.
  - Int_t GetCellAbsId(Int_t i) : It returns the absolute Id number of a cell in the array between 0 and GetNCells()-1.
  - Double_t GetCellAmplitudeFraction(Int_t i) : It returns the amplitude fraction of a cell in the array between 0 and GetNCells()-1.

9.1.3 AliVCaloCells (AliESDCaloCells, AliAODCaloCells)

They contain an array with the amplitude or time of all the cells that fired in the calorimeter during the event. Notice that per event there will be a CaloCell object with EMCAL cells and another one with PHOS cells.

- Short_t GetNumberOfCells(): Returns number of cells with some energy.
- Bool_t IsEMCAL(); Bool_t IsPHOS(); Char_t GetType(): Methods to check the origin of the AliESDCaloCell object, kEMCALCell or kPHOSCell.
- Short_t GetCellNumber(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the absolute cell number (from 0 to NModules*NRows*NColumns - 1).
- Double_t GetCellAmplitude(Short_t cellNumber): Given absolute cell number of a cell (from 0 to NModules*NRows*NColumns - 1), it returns the measured amplitude of the cell in GeV units.
- Double_t GetCellTime(Short_t cellNumber): Given absolute cell number of a cell (from 0 to NModules*NRows*NColumns - 1), it returns the measured time of the cell in second units.
- Double_t GetAmplitude(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the amplitude of the cell in GeV units.
- Double_t GetTime(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the time of the cell in second units.
- Double_t GetCellMCLable(Short_t cellNumber): Given absolute cell number of a cell (from 0 to NModules*NRows*NColumns - 1), it returns the index of the most likely MC label.
- Double_t GetCellEFraction(Short_t cellNumber): Given absolute cell number of a cell (from 0 to NModules*NRows*NColumns - 1), it returns the fraction of embedded energy from MC to real data (only for embedding)
- Double_t GetMCLabel(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the index of the most likely MC label.
- Double_t GetEFraction(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the fraction of embedded energy from MC to real data (only for embbedding)
- Bool_t GetCell(Short_t pos, Short_t &cellNumber, Double_t &amplitude, Double_t &time, Short_t &mclabel, Double_t &efrac); : For a given position of the list of cells, it fills the amplitude, time, mc lable and fraction of energy.

9.1.4 AliVCaloTrigger (AliESDCaloTrigger, AliAODCaloTrigger) - Rachid)

9.2 Macros

You can find example macros to run on ESDs or AODs in

```
$ALICE_ROOT/EMCAL/macros/TestESD.C or TestAOD.C
```

All the ESDs information is filled via the AliEMCALReconstructor/AliPHOSReconstructor class, in the method FillESD(). The AODs are created via the analysis class

```
$ALICE_ROOT/ANALYSIS/AliAnalysisTaskESDfilter.cxx,.h
```

and as already mentioned, for the calorimeters it basically just copies all the information from ESD format to AOD format.

Below is a description of what information is stored and how to retrieve it. The location of the corresponding classes is

```
$ALICE_ROOT/STEER
```

9.3 Code example

The analysis is done using the data stored in the ESD. The macro

```
$ALICE_ROOT/EMCAL/macros/TestESD.C
```

is an example of how to read the data for the calorimeters PHOS and EMCal (just replace where it says EMCAL by PHOS in the macro to obtain PHOS data). For these detectors we have to use the ESD class AliESDCaloCluster or AliESDCaloCells to retrieve all the calorimeters information. For the tracking detectors, the class is called AliESDtrack, but the way to use it is very similar (see "$ALICE_ROOT/STEER/AliESDtrack.*" and "$ALICE_ROOT/STEER/AliESDCaloCluster*" for more details). In AliESDCaloCluster we keep the following cluster information: energy, position, number of Digits that belong to the cluster, list of the cluster Digits indeces, shower dispersion, shower lateral axis and a few more parameters. In AliESDCaloCells we keep the following tower information: amplitude (GeV), time (seconds), absolute cell number.

The structure of the ESD testing macro (TestESD.C) is the following:

- Lines 0-29: This macro is prepared to be compiled so it has "includes" to all the Root and AliRoot classes used.
- Lines 30-36: This macro prints some information on screen, the kind of information is set here. We print by default clusters information and optionally, the cells information, the matches information, the cells in the clusters information or the MonteCarlo original particle kinematics.
- Lines 40-64: Here are the methods used to load AliESDs.root , geometry or kinematics files. Also loop on ESD event is here.

- Lines 65-66 Gets the measured vertex of the collision.
- Lines 69-78 Loops on all the CaloCell entries and prints the cell amplitude, absolute number and time.
- Lines 84- end: We access the EMCAL AliESDCaloCluster array and loop on it. We get the different information from the CaloCluster.
- Lines 111-130: Track Matching prints. Access to the matched track stored in AliESD-track.
- Lines 133-159: Cells in cluster prints
- Lines 161 - end: Access the stack with the MC information and prints the parameters of the particle that generated the cluster.

9.4 Advanced utilities : Reconstruction/corrrections of cells, clusters during the analysis

9.4.1 AliEMCALRecoUtils

9.4.2 Tender : AliEMCALTenderSupply

10 Run by run QA, how to and code

10.1 Online - Francesco, Michael

DQM, etc

10.2 Offline - Marie

Analysis code, what we control, how

10.3 Event display

10.4 Logbook tips

References

[1] EMCal for beginners, https://twiki.cern.ch/twiki/pub/ALICE/EMCalOffline/EMCalBeginners.pdf

[2] AliRoot: ALICE offline project, http://aliceinfo.cern.ch/Offline/

[3] AliRoot Documentation, http://aliceinfo.cern.ch/Offline/AliRoot/Manual.html

[4] AliRoot Installation, http://aliceinfo.cern.ch/Offline/AliRoot/Installation.html

[5] AliRoot Installation from Dario Berzano, http://newton.ph.unito.it/~berzano/w/doku.php?id=alice:compile-any&redirect=1

[6] AliEn web page, http://alien.cern.ch/twiki/bin/view/AliEn/Home

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

[8] EMCAL documentation, http://aliceinfo.cern.ch/Offline/Detectors/EMCALOffline.html

[9] EMCal Offline twiki, https://twiki.cern.ch/twiki/pub/ALICE/EMCalOffline

[10] EMCAL L0 http://www.sciencedirect.com/science/article/pii/S0168900212007681#

[11] EMCAL HLT http://arxiv.org/abs/1209.3647

[12] ALICE Collaboration, ALICE EMCal Technical Design Report, CERN/LHCC 2008-014

[13] Casalderrey-Solana and C. A. Salgado, Introductory lectures on jet quenching in heavy ion collisions, Acta Phys. Polon. B 38 (2007) 3731

[14] ALICE Collaboration, ALICE: Physics Performance Report, Volume I .J. Phys. G, 30 (2004) 1517-1763

[15] ALICE Collaboration, ALICE: Physics Performance Report, Volume II. J. Phys. G, 32 (2006) 1295-2040

[16] P. H. Hille, Fast Signal Extraction for the ALICE Electromagnetic Calorimeter, in preparation.