BmnRoot

simulation and analysis framework
for the BM@N experiment

Start Guide

BM@N collaboration

1 March 2016

TABLE OF CONTENTS

1. GENERAL INFORMATION

1.1 The BM@N experiment at the NICA project

Heavy ion collisions at high energies provide a unique opportunity to study the nuclear matter under extreme density and temperature. These extreme conditions are well suited to the investigation of the compressibility of the nuclear matter, in particular, the stiffness of the nuclear equation-of-state (EOS). The theoretical models suggest different possible scenarios for these modifications, so that new experimental data with high resolution and statistics are needed in order to disentangle the different theoretical predictions. The research program on heavy-ion collisions at the Nuclotron of the Joint Institute for Nuclear Research includes investigation of the reaction dynamics and nuclear EOS, study of the in-medium properties of hadrons, production of (multi)-strange hyperons at the threshold and search for hyper-nuclei [1].

The BM@N [2] experiment, short from Baryonic Matter at the Nuclotron, presented in the figure 1, aims at studying collisions of the elementary particles and ions with a fixed target at energies (laboratory system) up to 4 GeV per nucleon (for Au^{79+}). The BM@N facility is one of the main elements of the first stage of the NICA collider development to study hot and dense matter in heavy ion collisions.

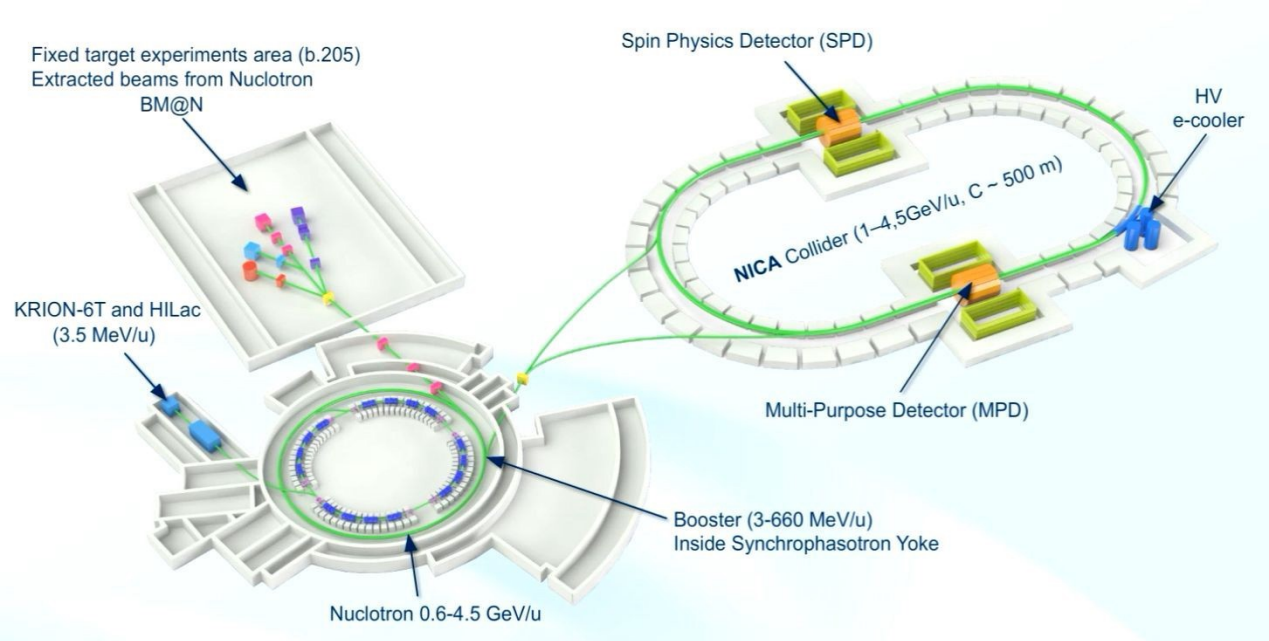

Figure 1. BM@N experiment at extracted Nuclotron beam

The figure 2 presents the three-dimensional scheme of the BM@N facility. It proposed to study the elementary reactions (p + p, p + n) and cold nuclear matter (p + A), the properties of dense baryonic matter in heavy ion collisions with fixed target, in-medium effects, hypermatter production, strangeness and hadron femtoscopy. Particle yields, ratios, transverse momentum spectra, rapidity and angular distributions, as well as fluctuations and correlations of hadrons will be studied as a function of the collision energy and centrality.

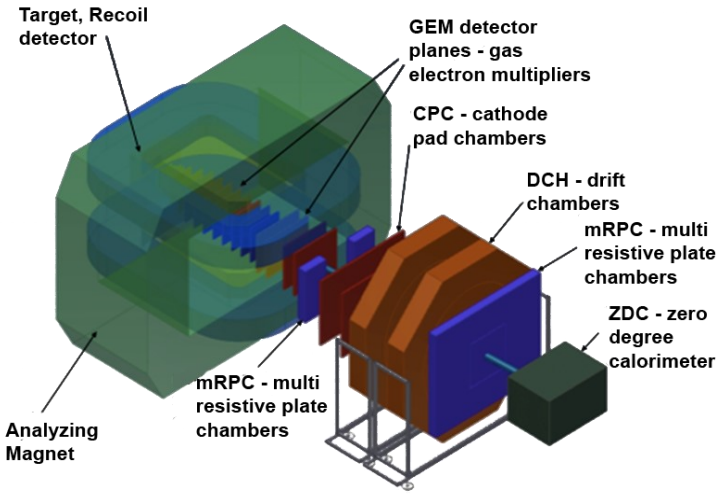

Figure 2. Three-dimensional view of the BM@N facility

In the gold ion collisions with a gold target at these energies, very high multiplicity can be reached, and the BM@N facility should identify the particles produced with high efficiency and estimate their parameters with high precision for the full study of this hot matter. For this purpose, BM@N combines high precision track measurements with the time-of-flight information for particle identification registered by detectors and total energy measurements for event characterization. The BM@N setup divides the detectors for particle identification into “near to magnet” and “far from magnet” to measure particles with low as well as high momenta.

The beam target is located inside the large-acceptance dipole magnet with the magnetic field up to 0.35 T for the Y projection. Intermediate coordinate detectors occupy the space between the magnet and the “far” detectors to increase the efficiency of particle identification. The Time-of-Flight detector is proposed to identify primarily hadrons and light nucleus, Zero Degree Calorimeter – to measure centrality of the collisions. The Recoil detector partially covering the backward hemisphere near the target is planned for the independent analysis of collision centrality by measuring the energy of the target fragments.

1.2 Purposes of the BmnRoot framework

The software and computing parts of the BM@N project is responsible for the activities including design, evaluation and calibration of the detector; storing, access, reconstruction and analysis of the data; and development and maintenance of a distributed computing infrastructure for physicists engaged in these tasks. To support the BM@N experiment, the software framework BmnRoot is developed. It provides a powerful tool for detector performance studies, event simulation, and development of algorithms for reconstruction and physics analysis of data of the fixed target events registered by the BM@N facility. The BmnRoot is implemented in the programming language C++ and based on the ROOT [3] environment and the object-oriented framework FairRoot [4] (for the FAIR experiment at GSI Institute).

The flexibility of the framework is gained through its modularity. The physics and detector parts could be written by different groups. In the applied framework, the detector response simulated by a package currently based on the Virtual Monte Carlo concept allows switching

simulation between Geant3, Geant4 and Fluka transport packages without changing the user code. For a realistic simulation of various physics processes, an interface to the event generators for nuclear collisions, e.g., UrQMD, Pythia and FastMC, is provided. One can easily choose between different modules, e.g., event generators. The same framework – BmnRoot is used to define the experimental setup, provides simulation, reconstruction, and physics analysis of simulated and experimental data. Using the same internal structure, the user can compare easily at any time the real data with the simulation results.

The BmnRoot environment also includes the tool for event navigation, inspection and visualization. The BM@N event display for Monte Carlo and experimental data is based on the EVE (Event Visualization Environment) package of the ROOT. The event display macro can be used to display both Monte Carlo points and tracks, and reconstructed hits and tracks, together with the BM@N detector geometry. The Event Manager implemented in the framework delivers an easy way to navigate through the event tree and to make cuts on energy, particle PDG codes, etc. in selected events.

1.3 Points of Contact

Dr. Gertsenberger Konstantin
Email: gertsen@jinr.ru

BM@N Collaboration (bmn.jinr.ru)
Laboratory of High Energy Physics
Joint Institute for Nuclear Research, Dubna

1.4 Organization of the Guide

1: General Information

This section contains general information about the BM@N experiment and the BmnRoot environment, including the purposes, points of contact, and acronyms and abbreviations.

2: Installation

The second section describes required packages and installation steps of the BmnRoot framework in details. The first subsection is devoted to the basic set of physics libraries and tools – FairSoft.

3: BmnRoot Structure

This section contains the information about structure of the BmnRoot framework, and briefly describes its directories, files and macros to be executed by the ROOT environment.

4: Simulation and Reconstruction Tasks

The fourth section details the simulation and reconstruction macros, their main parts, execution scheme. It lists possible event generators and transport packages which can be used in detector simulation.

5: Event display of the BM@N experiment

This section presents event display system intended to visualize the detector geometries and events of particle collisions with the fixed target, its structure, options and features as well as integration into the BmnRoot software.

6: Unified Database for BM@N offline data processing

The section describes the developed Unified Database designed as offline data storage of the BM@N experiment. The scheme of the Unified Database and its parameters are briefly presented. In addition, C++ interface developed for the access to the database are concisely described.

7: Distributed computing

The last section describes the design approaches and methods of cluster development for storing and processing of data obtained at the BM@N detector. The current cluster scheme and structure are presented; software for building data storage and parallelization of the BM@N data processing are noted.

1.5 Acronyms and Abbreviations

ROOT – modular scientific software framework, born at CERN, to deal with data processing, statistical analysis, visualization and storage in the researches on high-energy physics. Official site: root.cern.ch.

Macro – “small” C++/ROOT program code executed by ROOT CINT command interpreter line by line.

2. INSTALLATION

2.1 FairSoft Package

Many physics packages are necessary for the BmnRoot environment. Some of them can be installed using the package manager of the Linux distribution, but many others have to be installed from sources. To make the installation procedure as easy as possible an additional package called FairSoft is provided by FAIR collaboration. This package takes care of the installation of needed programs in the right order and with the right compilation flags. In the end, libraries and packages included in the FairSoft are installed in one directory.

The FairSoft package contains a configuration script, which checks if all the needed system packages are installed. If some of the system packages are missing, the configuration script will stop with a detailed error message. To simplify the installation procedure FairSoft package contains a set of scripts, which will automatically download, unpack, configure, build, and install all required software.

The FairSoft package includes the following packages:

- CMake (only installed if installed version is too old)
- gtest
- GSL
- Boost
- Pythia6(8)
- HepMC
- Geant4
- XRootD
- ROOT
- Pluto
- Geant321+_vmc
- VGM
- G4VMC
- MillePede
- ZeroMQ
- Protocoll Buffers
- Nano Message.

In case the python bindings are build the following additional packages will be installed:

- XercesC 3.1.2
- G4Py Version, which comes with Geant4.

The FairSoft package can be found on the GitHub repository.

2.2 BmnRoot Installation

You can find the short How-To installation manual on our site in the 'Software – HowTo' section. More instructions are given below in details.

The set of system packages are required to install the BmnRoot software. To install the needed packages on Red Hat-based Linux systems (e.g., Scientific Linux) you can execute in the terminal (bash):

$ *su*

$ *yum install subversion git make cmake gcc-gfortran binutils patch libX11-devel libXmu-devel libXpm-devel libXft-devel libXext-devel mesa-libGLU-devel libxml2-devel expat-devel zlib-devel postgresql-devel mysql-devel curl-devel automake libtool fftw3-devel*

To install the needed packages on Debian-based Linux systems (e.g. Ubuntu) you can execute in the terminal (bash):

$ *sudo apt-get install subversion git make cmake gcc gfortran binutils patch libX11-dev libXmu-dev libXpm-dev libXft-dev libXext-dev dpkg-dev xlibmesa-glu-dev libglew-dev libxml2-dev libexpat1-dev zlib1g-dev libpqxx3-dev libmysqlclient-dev libcurl4-openssl-dev automake libtool fftw3-dev*

Make sure that the packages above are installed on your system before installing BmnRoot.

The next step is to install FairSoft package (described in the Section 2.1) containing the required external libraries and tools for the BmnRoot environment. We are using March 2015 release of FairSoft now. To download it from the GitHub repository to '/opt' directory, please execute the following commands under root privilege:

$ *cd /opt*

$ *git clone https://github.com/FairRootGroup/FairSoft.git fairsoft*

$ *cd fairsoft*

$ *git checkout mar15p2*

To install the downloaded FairSoft package, please, execute under root:

$ *cd /opt/fairsoft*

$ *./configure.sh*

Then follow the instructions on the screen. It is preferred to install binaries and libraries to "/opt/fairsoft/install" directory. The proposed choices for configuring FairSoft (*configure.sh* script):

compiler to compile the external packages > *1) GCC (Linux, and older versions of Mac OSX)*
to compile with or without debug info? > *1) No Debug Info*
to install ROOT 6 instead of ROOT 5? > *2) No*
to install Simulation engines and event generators? > *1) Yes*
to install the additionally data files for the Geant4? > *2) Internet*
to install the python bindings for ROOT and Geant4? > *2) No*
define a directory for the installation > */opt/fairsoft/install*
is /opt/fairsoft/install the correct path? > *2) Yes*

The next step is to actually install the BmnRoot software. At first, please, select a directory where the BmnRoot will be installed, and go to this directory, e.g. for your home directory:

$ *cd ~*

Now you need to download the BmnRoot package from the GitLab repository. If you are developer of the BmnRoot, please, register (or login) on the site https://git.jinr.ru with JINR (@jinr.ru) email and then add SSH key to your profile. After registration, you can clone the BmnRoot by the following command:

$ *git clone --recursive git@git.jinr.ru:nica/bmnroot.git*

If you cannot get access to the GitLab repository, please, download the BmnRoot as zip archive:

$ *wget https://git.jinr.ru/nica/bmnroot/repository/archive.zip*

$ *unzip archive.zip*

$ *mv bmnroot-* bmnroot*

Then follow these installations steps to configure and build BmnRoot:

$ *cd bmnroot*

$ *mkdir build*

$ *. SetEnv.sh*

$ *cd build*

$ *cmake ..*

$ *make*

In order to use the BmnRoot (and other packages of the FairSoft, such as Geant and ROOT), please, set environment variables every time you run bash (terminal) by the following command executing in the build directory:

$ *. config.sh*

Our site mpd.jinr.ru also has documentation on the use of the GIT version control system at the 'Software – HowTo – How to use GIT' section.

3. BMNROOT STRUCTURE

3.1 BmnRoot Directories

The BmnRoot is based on the ROOT and an object-oriented framework, FairRoot of the FAIR experiment at GSI Institute, and was developed to use it in other high-energy physics (HEP) experiments. The BmnRoot uses the FairRoot classes to simplify simulated and experimental data processing. The FairRoot framework contains the following directories:

base – common classes of the FairBase environment, it contains base classes to implement another classes and macros for BM@N data processing, such as: FairDetector, FairField, FairHit, FairRun (FairRunSim, FairRunAna) and many others;

cmake – CMake scripts to compile the BmnRoot software, CMake generates native Makefiles for all supported platforms;

cuda – templates and example to start working with CUDA toolkit in order to parallelize data processing via graphic processors (NVidia video cards), FairCuda is a class which wraps CUDA implemented functions so that they can be used directly from ROOT CINT or compiled code;

fairtools – additional tool classes: FairLogger, FairMonitor, FairSystemInfo;

geane – tool for particle track propagation via TGeant3 package (Geane calculates the average trajectories of particles through dense materials and the transport matrix as well as the propagated errors covariance matrix in a given track representation);

generators – classes of the event generators: UrQMD, Shield, Pluto, etc.;

geobase – classes to construct detector geometries, such as: FairGeoTube, FairGeoConen FairGeoMedia and many others;

parbase – classes to work with parameters stored in output ROOT files;

passive – classes to build passive volumes of the BM@N geometry;

trackbase – classes to store particle tracking parameters of the Geane package.

The BmnRoot environment contains a set of directories, some were taken from the FairRoot and CbmRoot, and adapted to the BM@N experiment.

bmnbase – base classes of the BmnRoot environment, such as: CbmPrimaryVertexFinder, CbmTrackMerger, etc.;

bmndata – common classes of the BmnRoot designed primarily to store data of the experiment, it includes classes of digits, hits, points and tracks of the BM@N subdetectors, e.g. CbmPsdDigi, CbmPsdHit and CbmPsdPoints;

bmnfield – classes to work with magnetic fields;

eventdisplay – classes for visualization and monitoring of collision events and detector geometries (event display), the event display is described in the Section 5;

gconfig – macros with cuts and settings of particle transport packages (Geant, Fluka...) for event simulation;

geometry – files with geometries of the BM@N subdetectors in ROOT (.root) or ASCII (.geo) formats;

input – text files with different BM@N parameters and detector mappings;

macro – ROOT macros for simulation, reconstruction and physical analysis tasks of the BM@N experiment;

parameters – text files with detector parameters;

QA – QA-histograms to check simulation, reconstruction and physical analysis tasks of the BMNRoot environment;

uni_db – C++ database interface with ROOT macros, examples and documentation to use it for getting data of the BM@N experiment, the Unified Database is described in the Section 6.

Classes for reconstruction of particle tracks in collisions of the BM@N experiment are located in the following directories:

bmnKalmanFilter – classes of Kalman filter to reconstruct particle tracks;

cat – classes of L1 algorithm of track reconstruction used in CBM experiment;

globaltracking – classes to combine reconstructed track segments of different BM@N subdetectors into global particle track;

KF – Kalman filter's classes.

Every subdetector of the BM@N facility is presented by separate directory with the same name:

bd – classes to work with Barrel detector (BD);

dch – classes to work with Drift Chambers (DCH);

ecal – classes to work with Electromagnetic Calorimeter (ECAL);

gem – classes to work with Gas Electron Multipliers (GEM);

mwpc – classes to work with Multi Wire Proportional Chambers (MWPC);

psd – classes to work with Zero Degree Calorimeter (ZDC), this implementation is based on classes for PSD detector of the CBM experiment;

recoil – classes to work with Recoil detector;

sts – classes to work with Gas Electron Multiplier, the implementations is based on classes for STS detector of the CBM experiment;

tof – classes to work with Time-of-Flight detector (TOF-700) located at a distance about of 7 m from the target;

tof1 – classes to work with Time-of-Flight detector (TOF-400) located at a distance about of 4 m from the target;

zdc – classes to work with Zero Degree Calorimeter.

The top directory of the BmnRoot framework includes the following files:

CMakeLists.txt – the main CMake script to compile BmnRoot;

COPYING – general license of the BmnRoot;

INSTALL – short manual to install the BmnRoot environment;

README – short description of the BmnRoot software;

SetEnv.sh – bash script for setting the FairSoft environment to compile the BmnRoot (see Section 2.2).

3.2 BmnRoot Macros

Macro is a main executable element of the BmnRoot environment. Usually, it has *.C* extension. How to run macro by ROOT interpreter (CINT), you can find in the ROOT documentation. The most of BmnRoot macros are located in *macro* directory. It includes the following subdirectories:

dch – DCH macros, such as *recoRunDch.C* for reconstruction of particle tracks registered by Drift Chambers;

ecal – macros for Electromagnetic Calorimeter;

gem – macros for Gas Electron Multipliers;

mwpc – macros for Multi Wire Proportional Chamber;

tof_700 – macros for Time-of-Flight detector (TOF-700);

zdc – macros for Zero Degree Calorimeter;

eventdisplay – includes event display macro to graphically present BM@N geometry and collision events with a fixed target;

geometry – macros for creating files with subdetector geometries in ROOT format to use them in BM@N simulation and data processing, e.g. *create_rootgeom_TOF1.C*, *magnet.C*, etc.;

magfield – contains *BuildMagField.C* macro to generate file with extrapolated magnetic field map;

mpd_scheduler – source files of scheduling system to distribute data processing tasks on the cluster nodes or multi-core processors;

raw – macros to convert and process experimental raw data files to the ROOT format;

run – main macros of the BmnRoot environment for BM@N event simulation and reconstruction.

The main macros and subdirectories of *macro/run* directory are listed below:

geometry_run – macros to generate full facility geometry for a set of BM@N runs;

qa – contains macro to create QA-histograms for BM@N simulation and reconstruction tasks;

bmnloadlibs.C – macro to load all libraries required for data processing (FairSoft and BmnRoot libraries), it's usually used at the first steps in another macros;

geometry.C – macro to construct BM@N detector geometry from a set of geometry files (*.geo or *.root) corresponding to the subdetectors;

run_sim_bmn.C – macro to simulate BM@N events for selected event generator;

run_reco_bmn.C – macro to reconstruct BM@N events for simulated data;

recoRun1.C – macro to reconstruct events for experimental data.

One can start to learn BmnRoot with *run_sim_bmn.C* and *run_reco_bmn.C* macros. These simulation and reconstruction macros are described in the next section in details.

4. SIMULATION AND RECONSTRUCTION TASKS

4.1 Virtual Monte Carlo

Basically, there are two ways to simulate the behavior of a system: either one can describe its evolution analytically, in which case computers are used to find the solution of the dynamic equations, or a probabilistic approach is used, in which at each step pseudo-random numbers are used to select one among different physics processes. Example of the first approach is the computation of charged particle paths inside a magnetic field. When dealing with the interactions of particles with matter, the second approach is usually followed, because of the variety of possible physics processes and of their discrete nature. Because such approach is based on pseudo-random numbers, it is usually called a "Monte Carlo" method.

TVirtualMC class of the ROOT environment provides a virtual interface to Monte Carlo applications, allowing the user to build a simulation independent of any actual underlying Monte Carlo implementation itself. A user will have to implement a class derived from the abstract Monte Carlo application class. The concrete Monte Carlo implementation such as Geant3, Geant4, Fluka, is selected at run time – when processing a ROOT macro where the concrete Monte Carlo object is instantiated. This allows for comparison between different engines (necessary to estimate the theoretical uncertainties) using a single application. The concept of Virtual Monte Carlo has been originally developed by the ALICE [5] Software Project.

Monte Carlo simulations always have to describe the input particles, together with their interactions, and the detector (geometry, materials and read-out electronics). The definition of particles, available interactions and detector is carried on during the initialization phase. The main body of the application is then a loop over all particles that are traced through all materials until they exit, stop or disappear (by decay or annihilation). The tracing is done in a discrete fashion: at each step, the detector volume is found in which the particle is located and pseudo-random numbers are used to "draw" one among possibly several physical processes, to simulate the interaction of the particle with the matter. If an interaction occurs, the energy lost by the particle is computed and subtracted from its kinetic energy. When the latter reaches zero, the particle stops in such volume, otherwise a new step is performed.

Having computed the energy lost by all particles inside the detector, one has to simulate the behavior of the read-out electronics. This is usually done later, with another program that receives the energy lost in different locations as input, but it can also be done by the very same application that is performing the particle tracing inside the detector. Usually, the simulation of the read-out electronics also involves some use of pseudo-random generators, at least to simulate the finite resolution of any real measuring device.

The Virtual Monte Carlo (VMC) allows to run different simulation Monte Carlo without changing the user code and therefore the input and output format as well as the geometry and detector response definition. It provides a set of interfaces, which completely decouple the dependencies between the user code and the concrete Monte Carlo. The implementation of the TVirtualMC interface is provided for two Monte Carlo transport codes, Geant3 and Geant4. The implementation for the third Monte Carlo transport code, Fluka, has been discontinued. The VMC is now fully integrated with the ROOT geometry package TGeo, and users can easily define their VMC application with TGeo geometry definition (figure 3).

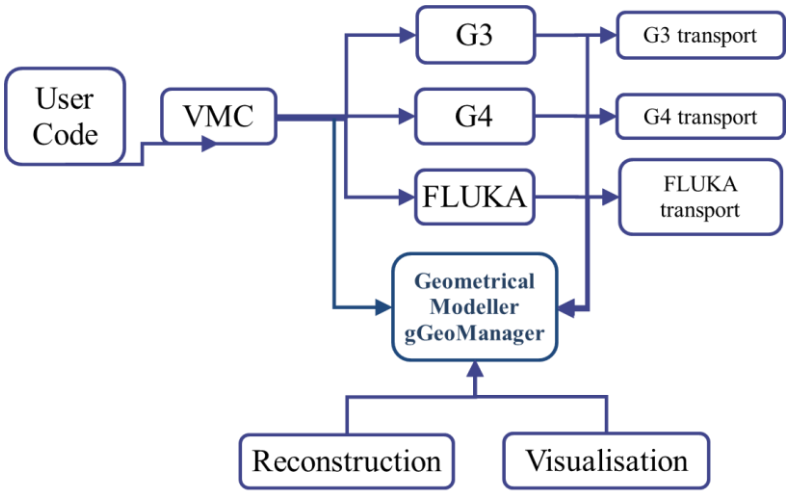

Figure 3. Geometrical Modeller (gGeoManager) for the MC simulation and subsequent use in reconstruction and visualization

4.2 Event generators

Event generators are software libraries that randomly generate high-energy particle physics events. Monte Carlo (because they exploit Monte Carlo methods) event generators are essential components of experimental analyses today and are also widely used by theorists and experiments to compare of experimental results with theoretical predictions and to make predictions and preparations for future experiments.

Monte Carlo generators allow to include theoretical models, phase space integration in multiple dimensions, inclusion of detector effects, and efficiency and acceptance determination for new physics processes. All event generators split the simulation up into a set of phases, such as initial-state composition and substructure, the hard process, parton shower, resonance decays, multiple scattering, hadronization and further decay. As a result, event generator produces the final-state particles, which feed into the detector simulation, allowing a precise prediction and verification for the entire experimental setup.

The BmnRoot framework supports an extended set of event generators (physics models) for particle collisions, which are commonly used in HEP experiments, such as:

- Ultrarelativistic Quantum Molecular Dynamics (UrQMD)
- Quark Gluon String Model (QGSM, LAQGSM) } Nuclear fragments
- Shield
- Parton Hadron String Dynamics (PHSD, HSD)
- Pluto
- Hybrid UrQMD } Femtoscopy
- EPOS
- 3 Fluid Dynamics (for baryon stopping)

Flows

The figure 4 shows the place of the event generators in the data processing chain and the further steps of simulation in particle physics.

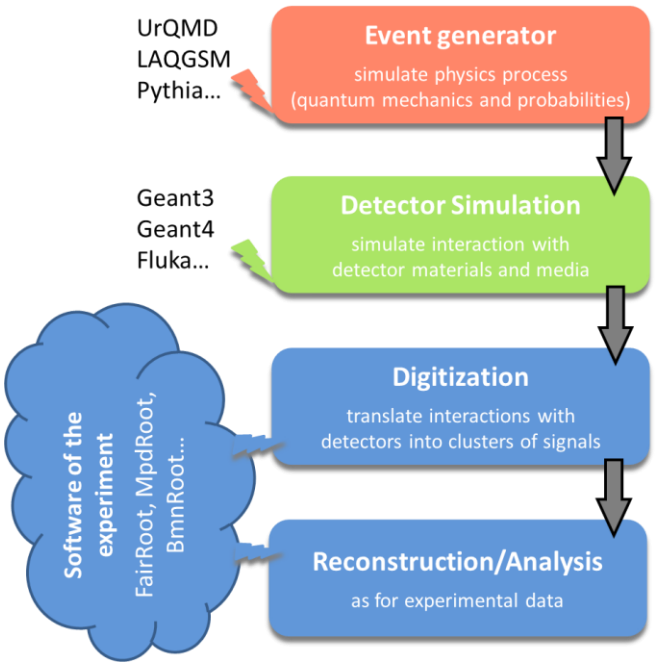

Figure 4. Simulation and analysis steps of high energy physics experiment

Partly due to historic reasons, many event generators are written in Fortran, newer ones are usually implemented in C++ language. Most particle collision (event) generators produce files of own format, which is then used in simulation of the experiment. The figure 5 presents the steps above with file storage levels of the BmnRoot in more details.

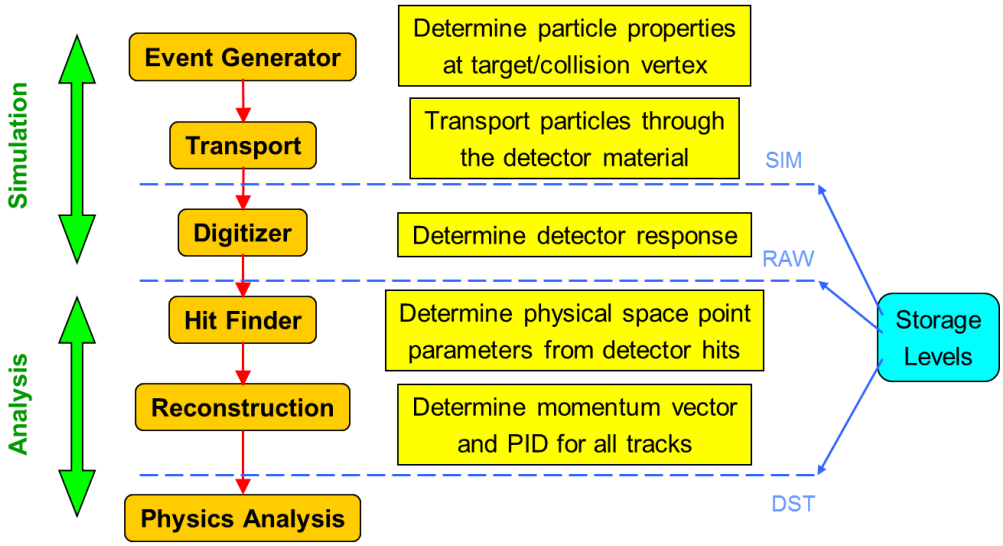

Figure 5. Data processing steps and storage levels of the BM@N experiment

4.3 Simulation of the BM@N experiment

BM@N simulation includes interactions and particles of interest, geometry of the system, materials used, generation of test events of particles, interactions of particles with matter and

electromagnetic fields, response to detectors, records of energies and tracks, analysis of the full simulation at different detail and visualization of the detector system and collision events. As the events are processed via the simulation, the information is disintegrated and reduced to that generated by particles when crossing a detector.

Simulation in high energy physics experiments uses transport packages to move the particles through experimental setup from initial state or origin files produced by event generators, such as UrQMD or QGSM described above. Geant (short from "geometry and tracking") transport package is developed at CERN and most commonly used today. Geant3 was implemented in Fortran programming language whereas Geant4 is developed in C++.

Transport includes detailed description of detector geometries and propagates all particle tracks through detector materials and media. The detector geometries are described by Geant and native geometrical models. During tracking of all particles through detectors, Geant forms the detector responses (hits) which are used in reconstruction task. In order to evaluate the software and detector performance, simulated events are processed through the whole cycle and finally the reconstructed information about particles is compared with the information taken directly from the Monte Carlo generation.

To simulate BM@N events of particle collisions with a fixed target, *run_sim_bmn.C* macro is used. Function parameters of the macro:

TString inFile – path of the input file with generator data if required;

TString outFile – path of the result file with MC data, default value: bmnsim.root;

Int_t nStartEvent – number (start with zero) of the first event to process, default: 0;

Int_t nEvents – number of events to transport (0 – all events in the input file), default: 3;

Bool_t flag_store_FairRadLenPoint – whether enables radiation length manager to estimate radiation length data, default: kFALSE;

Bool_t isFieldMap – kFALSE corresponds to the constant field, kTRUE corresponds to the field map stored in the file (it is used by default).

The main macro lines are described below. On the first step (after creating a timer for execution profiling), the simulation macro creates *FairRunSim* (run simulation) class and chooses a transport package for it. By default, Geant3 is chosen:

```
fRun->SetName("TGeant3");
```

This function also automatically runs ROOT macros from *gconfig* directory to set configuration of selected transport package. On the next step, the BM@N geometry is loaded by *geometry.C* macro:

```
gROOT->LoadMacro("$VMCWORKDIR/macro/run/geometry.C");
geometry(fRun);
```

It parses media definition and creates passive volumes, such as cave, magnet, target, and subdetectors: Recoil, GEMs, TOF-400, DCH1, DCH2, TOF-700, and ZDC. Then *run_sim_bmn.C* macro defines event generator, which will be used to simulate BM@N events.

```
FairPrimaryGenerator* primGen = new FairPrimaryGenerator();
fRun->SetGenerator(primGen);
```

One can choose and tune one of the following event generators: UrQMD, Particle Generator, Ion Generator, Box Generator, HSD/PHSD or LAQGSM/QGSM Generator. To select event generator, *AddGenerator* function of *FairPrimaryGenerator* class is used.

Files of different event generators are located on the NICA cluster in the directory */eos/nica/bmn/sim/gen*. If you do not have access to the cluster, you can use, for example, a simple particle BOX generator to propagate selected particles to the desired directions. BOX generator is used without any input files for the simulation macro. To choose BOX generator for BM@N simulation, change the line "#define URQMD" to "#define BOX" in *run_sim_bmn.C*.

On the next step, the output file name is specified:

```
fRun->SetOutputFile(outFile.Data());
```

The magnetic field set by map (ASCII or ROOT) file or constant field is chosen in the next lines of the macro. In the BM@N experiment the transition from a constant magnetic field to real field map, interpolation of the field between map nodes and extrapolation of the field map to out-of-magnet region were made.

The following line enables or disables radiation length manager:

```
fRun->SetRadLenRegister(flag_store_FairRadLenPoint);
```

The manager determines the particle fluency through a certain boundary (surface) and deduces a map. Knowing the volume and density of the object of interest and the specific energy loss, doses can be estimated. Therefore, FairRadLenManager class estimates radiation length data and can answer the questions: “What energy dose will be accumulated during a certain time of operation?” and “How to create physical volumes with correct material assignment?”.

The next function initializes BM@N simulation, i.e. Virtual Monte Carlo classes and tasks (if they are used in simulation): *fRun->Init()*. To run event simulation, the following line is executed: *fRun->Run(nEvents*).

The figure 6 presents the main modules of the BmnRoot framework and their relationship during the simulation.

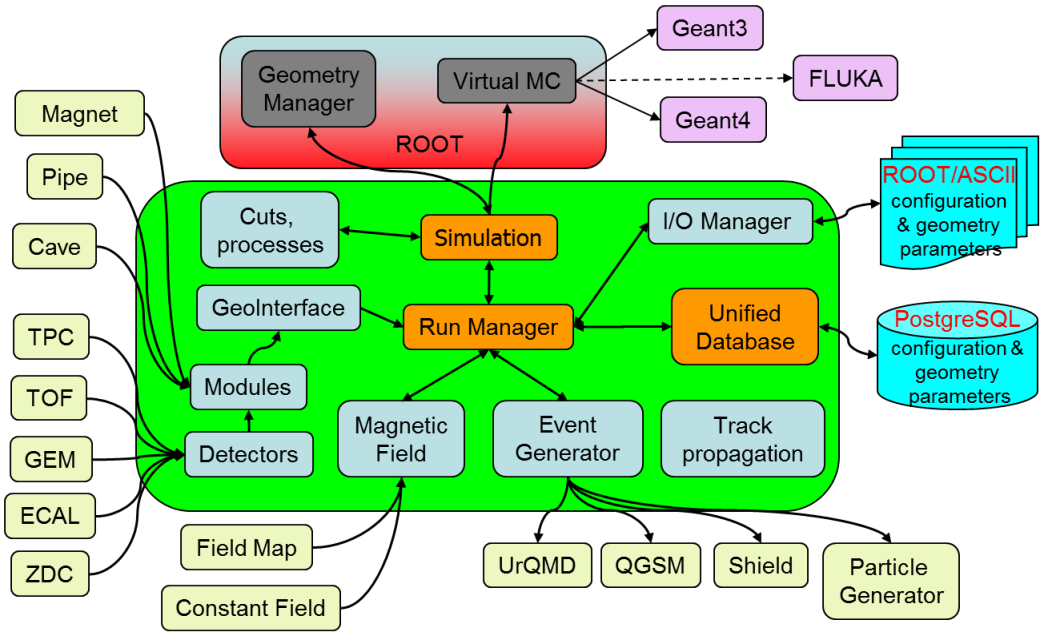

Figure 6. The scheme of the BmnRoot modules in the simulation

Input/Output manager (shown on the figure as I/O manager) called the Runtime Parameter Manager is the manager class for all parameter containers and based on ROOT TFolder and TTree classes. The Runtime Manager includes container factory, list of parameter containers and list of runs. The parameters are stored in the containers, which are created in the Init() function of the tasks via the container factory or produced from ASCII or ROOT files in the macro.

Subdetector geometries can be defined using different input formats: ASCII or ROOT files (Root Geometry format presented in the form of TGeoVolumes tree) or defined directly in the source code. The ASCII input format for detector geometry consists of volumes defined by the following sequence:

	Example
Volume name	target
Mother volume name	pipe_vac_1
Shape name	TUBE
Medium name	Gold
Shape parameters	0. 0. -0.25 0. 2.5 0. 0. 0.25
Positioning of the volume in a mother node: position and rotation matrix	0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1.

The simulation result is *bmnsim.root* (default name) file with Monte Carlo data such as simulated track points and particle tracks in the BM@N detector. It also includes BM@N detector geometry stored in the *FairBaseParSet* container and presented by ROOT geometry manager

(TGeoManager class) – its global pointer *gGeoManager*. To get geometry from this file, you can execute in a stand-alone ROOT macro the following lines:

```
gROOT->LoadMacro("$VMCWORKDIR/macro/run/bmnloadlibs.C");
bmnloadlibs();
TFile* f = new TFile("bmnsim.root");
f->Get("FairBaseParSet"); // and then use gGeoManager
```

ROOT Geometry package is described in the Section 5 of the Start Guide. ROOT files are created via the object streaming mechanism (information about ROOT Streamers you can find in the ROOT documentation) and are presented by a tree of objects saved as byte array.

4.4 BM@N event reconstruction

Event reconstruction is the process of interpreting the electronic signals produced by the detector to determine the original particles that passed through, their momenta, directions, and the primary vertex of the event. Event reconstruction consists of the following main steps:

- Hit reconstruction in subdetectors;
- Track reconstruction usually composed of:
  - Searching for track candidates in main tracker
  - Track propagation, e.g., using Kalman filter
  - Matching with other detectors (global tracking)
- Vertex finding;
- Particle identification.

The tasks for charged-track reconstruction in experimental high energy physics are pattern recognition (i.e., track finding) and track fitting. The common approach to the track reconstruction problem is based on the Kalman filtering technique. The Kalman filtering method provides a mean to do pattern recognition and track fitting simultaneously. The multiple scattering can also be handled properly by the method. The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. The algorithm starts from track candidates (“seeds”), for which vectors of initial parameters and covariance matrices are evaluated. Then each track is propagated to some surface (detector or intermediate point). The new covariance matrix can be obtained using the Jacobian matrix of the transformation, i.e., the matrix of derivatives of propagated track parameters with respect to current parameters. The BmnRoot environment also uses another method of track reconstruction – L1 (CBM) tracking.

Tracks are usually defined by specifying their integer identification number, the particle code (integer code also known as the "PDG code" from the list by the Particle Data Group which maintains a standard for designating Standard Model particles and resonances), the parent track (if any), and a particle object. In addition, each track has a (possibly empty) list of daughters that are secondary tracks originating from it.

Precise knowing of the primary events vertex position essentially improves the momentum resolution and secondary vertices finding efficiency. The primary vertex is found by extrapolating all primary tracks reconstructed back to the origin, and its resolution is found as the RMS of the distribution of the primary track’s extrapolation at the origin. The global average of this distribution is the vertex position.

The figure 7 shows the place of event reconstruction in the processing chain for experimental and simulated data.

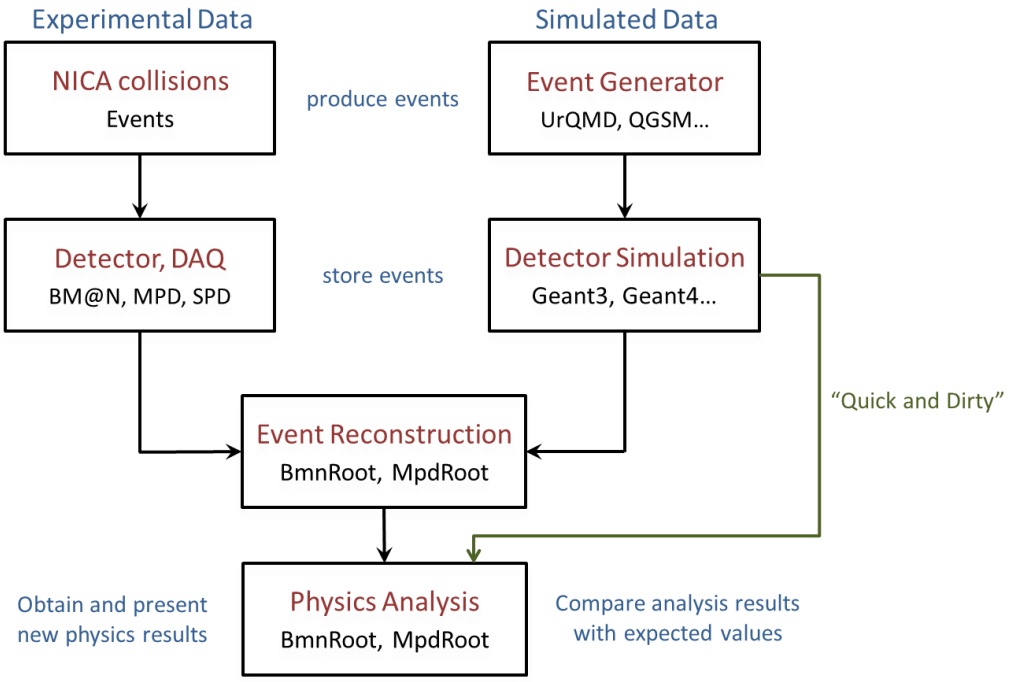

Figure 7. The chain of event processing for experimental and simulated data

To reconstruct BM@N events of particle collisions with a fixed target registered by the BM@N facility, *run_reco_bmn.C* macro is used. Function parameters of the macro:

TString inFile – path of the input file with MC data, default value:bmnsim.root;

TString outFile – path of the result file with reconstructed DST data, default: mpddst.root;

Int_t nStartEvent – number (start with zero) of the first event to process, default: 0;

Int_t nEvents – number of events to process (0 - all events of given file), default: 100.

The main macro lines are described below. On the first steps the reconstruction macro creates *FairRunAna* (run analysis) class. To set input file with simulated or experimental data and output file with reconstructed data in DST format, the following lines are executed:

```
fRun->SetInputFile(inFile);
fRun->SetOutputFile(outFile);
```

The data reconstruction and analysis in the BmnRoot is organized by a list of analysis tasks (tasks can also be used in the simulation with *FairRunSim* class). Each task participating in event processing is inherited from the base class FairTask (TTask). Two main functions of tasks can be noted. *Init()* function initializes task and its variables and is executed inside *FairRunAna::Init* function. *Exec(Option_t* option)* function contains the main function code of the task and is called by *FairRunAna::Run* function. The scheme of task initialization and execution is presented on the figure 8.

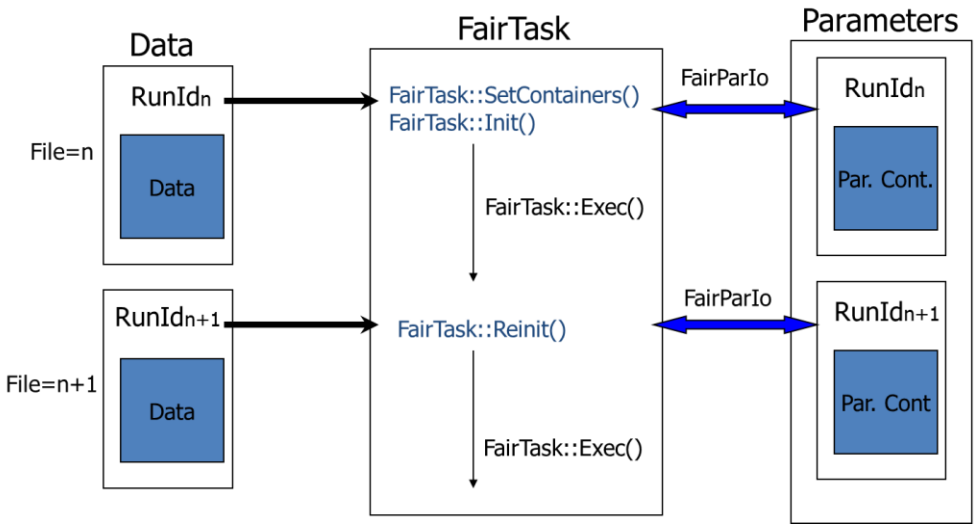

Figure 8. The scheme of task initialization and execution

The following example demonstrates task definition and adding to the task list in *run_reco_bmn.C* macro:

```
BmnTof1HitProducer* tof1HP = new BmnTof1HitProducer();
fRun->AddTask(tof1HP);
```

In general, a task can be added not only to the *FairRunAna* instance but also to another task, i.e., all tasks can be organized into a hierarchy. If you want to get detector geometry from the given input file in your analysis task (if gGeoManager has not been known yet), you can execute:

```
FairRunAna* ana = FairRunAna::Instance();
FairRuntimeDb* rtdb = ana->GetRuntimeDb();
FairBaseParSet* baseParSet = (FairBaseParSet*)rtdb->getContainer("FairBaseParSet"); //and then use gGeoManager
```

The figure 9 presents the main modules of the BmnRoot framework and their relationship during the reconstruction.

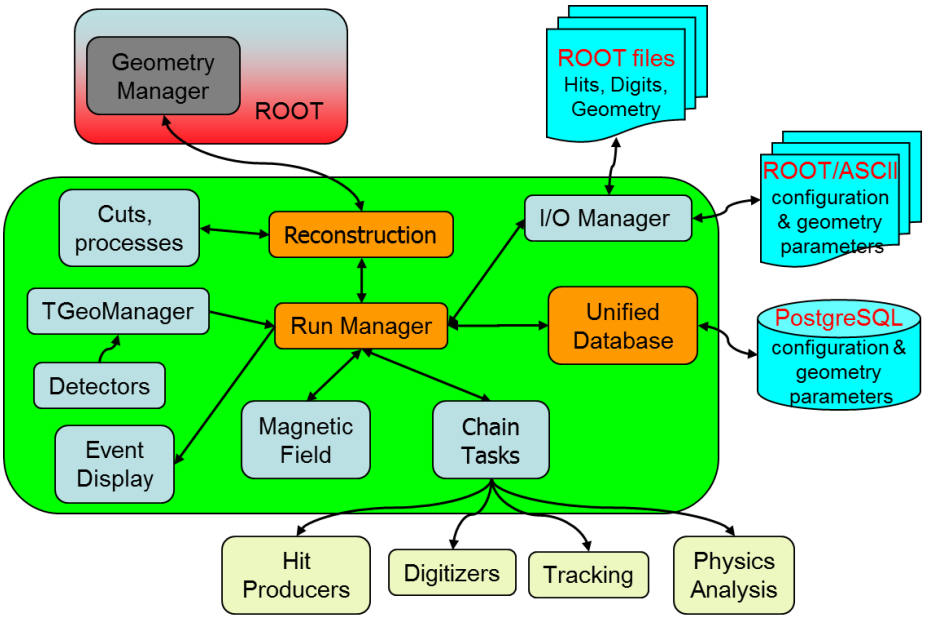

Figure 9. The scheme of the BmnRoot modules in the reconstruction

The result DST file contains reconstructed (physical and geometrical) data about particles and particle tracks of collision events registered by the BM@N facility.

5. EVENT DISPLAY OF THE BM@N EXPERIMENT

5.1 ROOT Geometry Package

The ROOT geometry package is a tool to build, browse and visualize detector geometries. It is independent from any tracking Monte Carlo engine, though it has been designed to optimize particle transport in correlation with simulation packages as Geant3, Geant4 and Fluka.

The building blocks of any geometry are the volumes that may contain other volumes. The biggest one, called the “cave”, contains all other volumes and provides the master reference system (MARS) in which the others are positioned. Each volume needs to be associated with a medium that can be a mixture of different materials (whose weights are the relative densities). An example of creating of the simplest setup having a single empty box in the ROOT environment (CLING) is presented below.

```
root[] gSystem->Load("libGeom"); // load the ROOT geometry library
root[] TGeoManager* mng = new TGeoManager("cave", "the simplest geometry");
root[] TGeoMaterial* mat = new TGeoMaterial("Vacuum", 0, 0, 0);
root[] TGeoMedium* med = new TGeoMedium("Vacuum", 1, mat);
root[] TGeoVolume* top = gGeoManager->MakeBox("Top", med, 10., 10., 10.);
root[] gGeoManager->SetTopVolume(top);
Info in <TGeoManager::SetTopVolume>: Top volume is Top. Master volume is Top
root[] gGeoManager->CloseGeometry();
Info in <TGeoManager::CheckGeometry>: Fixing runtime shapes...
Info in <TGeoManager::CheckGeometry>: ...Nothing to fix
Info in <TGeoManager::CloseGeometry>: Counting nodes...
Info in <TGeoManager::Voxelize>: Voxelizing...
Info in <TGeoManager::CloseGeometry>: Building cache...
Info in <TGeoManager::BuildCache>: --- Maximum geometry depth set to 100
Info in <TGeoManager::CloseGeometry>: 1 nodes/ 1 volume UIDs in title
Info in <TGeoManager::CloseGeometry>: ----------modeler ready----------
root[] top->SetLineColor(kMagenta);
root[] mng->SetTopVisible(); // the TOP is generally invisible
root[] top->Draw();
--- Drawing       1 nodes with 3 visible levels
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```

The last command pops up a new window displaying a canvas with our magenta cube. It is possible to move, rotate and zoom the camera interactively by clicking and dragging the mouse.

Complex geometries can be built in a hierarchical way, through the concept of containment: one has to define and position some volumes inside other ones. Positioning is done with spatial transformations with respect to the "mother reference system" (i.e., the system defined by the containing volume). Complex volumes are built using basic or primitive shapes, already defined by ROOT (e.g., box, tube, cone, etc.), through operations like join, subtract or intersect. Finally, a given volume can be positioned several times in the geometry or it can be divided accordingly to user-defined patterns, automatically defining new contained volumes.

Once a geometry has been created, it can be saved into a ROOT file or as C++ macro with the *Export(...)* method of TGeoManager. Loading the geometry is done with its *Import(...)* method.

In addition, individual volumes can also be saved into a ROOT file. Finally, ROOT provides a graphical user interface to edit or build geometry. The editor can be opened with the *Edit(...)* method of TGeoManager.

The possibility to visualize 3D objects is very important: in ROOT, the rendering engine is provided by the OpenGL library that provides advanced visualization features and real-time animations. Event display applications are an important application of 3D visualization. EVE is a ROOT module based on experiment-independent part of the ALICE event displays, using the ROOT GUI and the OpenGL rendering engine.

5.2 Event Display of the Experiment

One of the problems to be solved in modern high-energy physics experiments on particle collisions with a fixed target is the visual representation of the events during the experiment run. Event display system can be used at the design stage of the detectors for:

- model and reconstruction algorithm checking and debugging by developers;
- analysis of algorithms for event data processing by experts;
- visualization of data reconstruction and physics analysis with better understanding of the detectors and collision event structure by users;
- demonstration of running and presentation of results of an experiment.

In addition, event display is required at the run stage for online visual monitoring of selected events during the experiment run with the aim of visual control and debugging of current events. Visual monitoring of physical processes and data is very important, for example, for visual estimation of the multiplicity of current events, which depends on the impact parameters of the particles.

The event display package is a part of BmnRoot environment [6]. The event visualization uses data of the experiment as follows (figure 10). The data (types of particles produced, their momenta and other kinematic parameters) obtained by event generators such as UrQMD and QGSM are passed to simulation macro transferring the particles through the detectors by particle transport packages (Geant, Fluka). The designed detector geometry in a special text format is converted by the macro into a corresponding hierarchy of class instances of the ROOT geometric shapes, transformation matrix and other graphics information that is stored in the ROOT file. After the simulation has been completed, the resulting file contains both the Monte Carlo data and the BM@N geometry.

The next step is the reconstruction of particle data, tracks and other parameters that are written to DST file. The reconstruction algorithms restore the information about the particle trajectory and identity from the information contained in the simulated (or raw) detector data. The MC and DST data is stored in ROOT files in a hierarchical tree view. The last step of the data processing on the figure is physical analysis of the reconstructed data to evaluate efficiency of the simulated detector or to investigate physical properties.

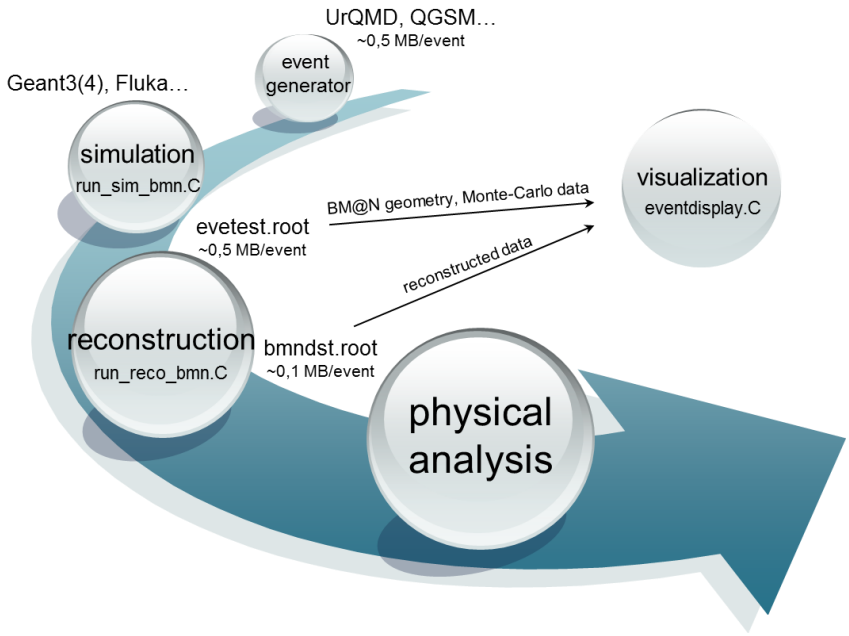

Figure 10. Input data for BM@N event display

For graphical representation of the experiment, the developed *eventdisplay.C* macro uses both the file with the simulated data together with the BM@N geometry and the DST file with the reconstructed data. The event display is integrated with the BmnRoot software to combine different stages of the event processing with a graphical representation of these events with the aim to evaluate and check the correctness of reconstruction and physical analysis algorithms. In addition, it enables one to take into account adding the subdetectors to the facility structure and changing the current configuration of the BM@N setup.

To display detector geometry and events of the BM@N experiment the high-level graphics package EVE is used. The ALICE collaboration developed their own base classes included in the ROOT environment as a new package EVE to provide graphical representation and management of visualization objects for high energy physics. The package uses ROOT GUI environment and OpenGL library to display three-dimensional objects and its two-dimensional projections. The EVE classes developed primarily for the creation, management of event objects such as raw data, hits, clusters, points (TEvePointSet), and particle tracks (TEveTrack), allow selecting and highlighting different elements, configuring its parameters.

To visualize events of the BM@N experiment new classes were developed for the management of different physical objects and their graphical representation, e.g., hits (BmnHitDraw class), simulated and reconstructed points and particle tracks, calorimeter towers. These classes are located in '*eventdisplay*' directory. FairEventManager main class (inherits from TEveEventManager) was developed to perform the following functions: read events directly from ROOT tree, select events for display, apply cuts to events, event navigation (next event, previous event and go to event number), read and display the detector geometry.

The event display macro (*macro/eventdisplay/eventdisplay.C*) provides users with control and graphical representation of detector geometry, MC tracks and their points, as well as reconstructed hits and tracks. It reads detectors' geometry, simulated and reconstructed event data from the generated ROOT files and displays the objects by the EVE package. All necessary options

for graphical representation of the physical objects are included in the macro, set by default, and can be changed.

The main function parameters of the *eventdisplay.C* macro:

char* *sim_geo_file* – input file with MC data and/or detector geometry
char* *reco_file* – input file with reconstructed data for simulation or experimental events
int *data_source* – data source type, default: 0
0 – event display for simulatated data
1 – event display for experimental data
bool *is_online*: true - use online mode (continuous view events), false - use offline mode (manual switching of events), default: false.

The possible options of *eventdisplay.C* run are presented below.

1. Display the simulated and corresponding reconstructed data.

Parameters: datasource – 0, file with simulated data and detector geometry – bmnsim.root, file with reconstructed data – bmndst.root. The task chain:

```
FairMCPointDraw *RecoilPoint = new FairMCPointDraw("RecoilPoint", kRed, pointMarker);
fMan->AddTask(RecoilPoint);
FairMCPointDraw *MWPC1Point = new FairMCPointDraw("MWPC1Point", kRed, pointMarker);
fMan->AddTask(MWPC1Point);
FairMCPointDraw *MWPC2Point = new FairMCPointDraw("MWPC2Point", kRed, pointMarker);
fMan->AddTask(MWPC2Point);
FairMCPointDraw *MWPC3Point = new FairMCPointDraw("MWPC3Point", kRed, pointMarker);
fMan->AddTask(MWPC3Point);
FairMCPointDraw *DCH1Point = new FairMCPointDraw("DCH1Point", kRed, pointMarker);
fMan->AddTask(DCH1Point);
FairMCPointDraw *DCH2Point = new FairMCPointDraw("DCH2Point", kRed, pointMarker);
fMan->AddTask(DCH2Point);
FairMCPointDraw *TOF1Point = new FairMCPointDraw("TOF1Point", kRed, pointMarker);
fMan->AddTask(TOF1Point);
FairMCPointDraw *TofPoint = new FairMCPointDraw("TofPoint", kRed, pointMarker);
fMan->AddTask(TofPoint);
FairMCModuleDraw *PsdPoint = new FairMCModuleDraw("PsdPoint", kRed, pointMarker);
fMan->AddTask(PsdPoint);
FairMCPointDraw *StsPoint = new FairMCPointDraw("StsPoint", mcPoi kRed ntColor, pointMarker);
fMan->AddTask(StsPoint);

FairMCTracks* GeoTrack = new FairMCTracks("GeoTracks");
fMan->AddTask(GeoTrack);

FairHitPointSetDraw *BmnGemHit = new FairHitPointSetDraw("BmnGemStripHit", kBlack, pointMarker);
fMan->AddTask(BmnGemHit);
FairHitPointSetDraw *TOF1Hit = new FairHitPointSetDraw("TOF1Hit", kBlack, pointMarker);
fMan->AddTask(TOF1Hit);
FairHitPointSetDraw *BmnDch1Hit = new FairHitPointSetDraw("BmnDch1Hit0", kBlack, pointMarker);
fMan->AddTask(BmnDch1Hit);
FairHitPointSetDraw *BmnDch2Hit = new FairHitPointSetDraw("BmnDch2Hit0", kBlack, pointMarker);
fMan->AddTask(BmnDch2Hit);
FairHitPointSetDraw *BmnTof2Hit = new FairHitPointSetDraw("BmnTof2Hit", kBlack, pointMarker);
fMan->AddTask(BmnTof2Hit);

BmnTrackDraw* BmnGlobalTrack = new BmnTrackDraw("GlobalTrack");
```

```
fMan->AddTask(BmnGlobalTrack);
```

The *FairMCPointDraw* class (inherited from *FairPointSetDraw : FairTask*) was developed to display simulated track points on the screen which are presented by *TClonesArray* class containing array of *FairMCPoint* objects.

The *FairMCModuleDraw* class (inherited from *FairTask*) was designed to fill array items corresponding the number of ZDC towers: 0 – particles didn't pass through the tower, 1 – particles passed through the tower.

The *FairMCTracks* class (inherited from *FairTask*) was implemented to display simulated tracks on the screen stored in the *GeoTracks* branch as *TClonesArray* of *TGeoTrack* objects containing all points through which it passes.

The *FairHitPointSetDraw* class (inherited from *FairPointSetDraw*) was developed to display reconstructed hits and tracks points on the screen, presented by *TClonesArray* of *FairHit* instances. It is also possible to graphically present hits and track points in the form of parallelepipeds using *FairHitDraw* class (inherited by *FairBoxSetDraw* class drawing *FairBoxSet* class – an array of little parallelepipeds).

The *BmnTrackDraw* class (inherited from *FairTask*) was implemented to display reconstructed tracks of the simulated events in the BmnRoot. It uses a specialized algorithm to pass the branches with reconstructed points and tracks of the DST file.

When flag is_*online* is set to true value, event display continuously shows all events of *bmnsim.root* and *bmndst.root* files by pressing the "Start online display" button.

2. Display the reconstructed points of BM@N events for "raw" experimental data.

Parameters: datasource – 1, file with detector geometry – evetest_run3.root, file with experimental raw data – bmn_run0607_digit.root. The task chain:

```
BmnDigitDraw* MwpcDigit = new BmnDigitDraw("bmn_mwpc_digit", 1, pointColor, pointMarker);
fMan->AddTask(MwpcDigit);
BmnDigitDraw* DchDigit = new BmnDigitDraw("bmn_dch_digit", 2, pointColor, pointMarker);
fMan->AddTask(DchDigit);
```

The *BmnDigitDraw* class (inherited from *FairTask*) was developed to reconstruct the "raw" data – points stored in *BMN_DIGIT* tree and to display the reconstructed points saved as *TClonesArray* of *FairHit* objects. The input file is opened during task initialization.

When flag is_*online* is set to true value, event display continuously shows all events of *bmn_run0607_digit.root* files by pressing the "Start online display" button.

3. Display the reconstructed points and tracks (built by points) of experimental events stored in the ROOT file.

Parameters: datasource – 1, file with detector geometry – evetest_run3.root, file with reconstructed experimental data – bmndst_run688.root. The task chain:

```
BmnHitDraw* MwpcHit = new BmnHitDraw("BmnMwpcHit", pointColor, pointMarker);
fMan->AddTask(MwpcHit);
BmnHitDraw* DchHit = new BmnHitDraw("BmnDchHit", pointColor, pointMarker);
fMan→AddTask(DchHit);

BmnExpTrackDraw* MwpcTrack = new BmnExpTrackDraw("MwpcMatchedTracks", "BmnMwpcHit");
fMan->AddTask(MwpcTrack);
```

```
BmnExpTrackDraw* DchTrack = new BmnExpTrackDraw("DchTracks", "BmnDchHit");
fMan->AddTask(DchTrack);
```

The *BmnHitDraw* class (inherited from *FairTask*) was developed to display reconstructed track points presented by *TClonesArray* of *FairHit* objects. The input file is opened during task initialization.

The *BmnExpTrackDraw* class (inherited from *FairTask*) displays reconstructed tracks presented by *TClonesArray* of *CbmTrack* objects. The input file is opened during task initialization. The tracks are built on the points stored in the same ROOT file.

When flag is_*online* is set to true value, event display continuously shows all events of *bmndst_ run688.root* files by pressing the "Start online display" button.

4. Display the reconstructed tracks (built by Geane package) of experimental events stored in the ROOT file.

Parameters: datasource – 1, file with detector geometry – evetest_run3.root, file with reconstructed experimental data – bmndst_run688_gl.root. The task chain:

```
BmnHitDraw* MwpcHit = new BmnHitDraw("BmnMwpcHit", pointColor, pointMarker);
fMan->AddTask(MwpcHit);
BmnHitDraw* DchHit = new BmnHitDraw("BmnDchHit", pointColor, pointMarker);
fMan→AddTask(DchHit);

BmnExpTrackDraw* MwpcTrack = new BmnExpTrackDraw("MwpcMatchedTracks", "BmnMwpcHit");
fMan->AddTask(MwpcTrack);
BmnExpTrackDraw* DchTrack = new BmnExpTrackDraw("DchTracks", "BmnDchHit");
fMan->AddTask(DchTrack);
```

The event display shows the BM@N geometry and events in different projections and views, and has a multiview mode. A user can select the tab with a convenient graphical representation, set light sources, configure physical objects' parameters, for example, specify a desired color of the object or a background color. The event display offers full interactivity, supporting on-line rotations and picking of objects. Three methods of geometry coloring were released in the visualization system: default by ROOT, hierarchical and pre-selected for detectors. In the case of hierarchical coloring, a user can choose colors for the different levels of the BM@N geometry tree. If the pre-selected mode is chosen, a color can be set for each subdetector of the facility. The figure 11 presents visualization of the BM@N detector in the multiview mode: 3D view together with XY and XZ projections.

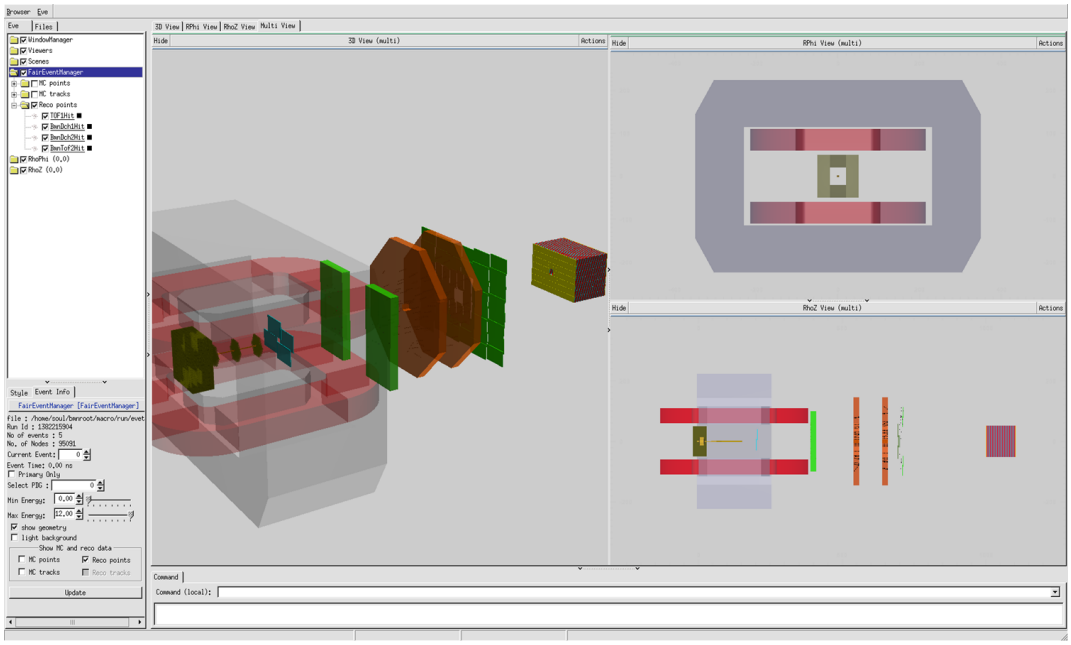

Figure 11. The views and projections of the BM@N facility in the event display

The figure 12 presents MC tracks (simulated by UrQMD generator) of the central event in collisions of gold ion beam with a gold target at the energy 4 GeV per nucleon. Tracks of different particles are highlighted in different colors. The EVE package provides a browser (on the left side of the window) to all the displayed objects. The object browser includes both geometry and event information presented in a hierarchical tree. Users can select in the tree individually which objects to draw, change object colors and set other visualization settings. In addition, the event display provides quick access to the following settings of event representation:

- selecting an event number displayed on the screen;
- showing all or only primary particles, particles with given PDG codes;
- setting energy range of particles to be displayed;
- changing the background of BM@N views and projections;
- choosing objects to be presented on the screen: simulated hits and tracks, reconstructed points and particle tracks.

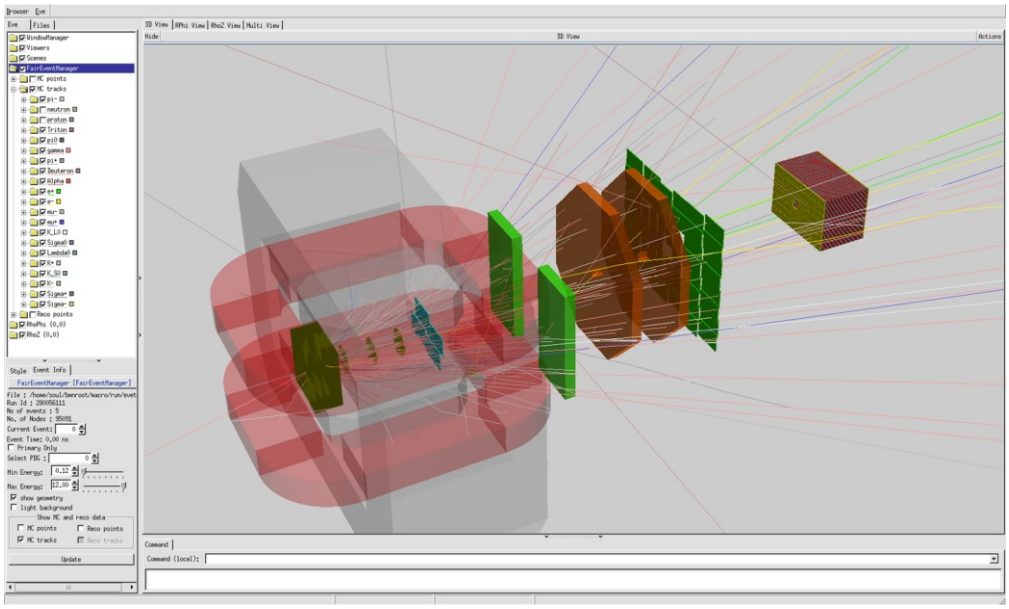

Figure 12. BM@N event visualization of MC tracks in Au – Au collision at 4 GeV/nucleon

Therefore, the event display system allows one to compare the simulated and reconstructed data visually to verify the correctness of developed algorithms. In order to clearly compare hits and particle tracks the display option was implemented to hide the entire BM@N geometry on the screen. Additionally, the option for high transparency of detector geometry was also developed (figure 13). It sets the different transparency coefficients appropriated for different BM@N subdetectors. Besides, it is also appropriate to show only the active detector towers hit by particles and not all track points covered by the Zero Degree Calorimeter.

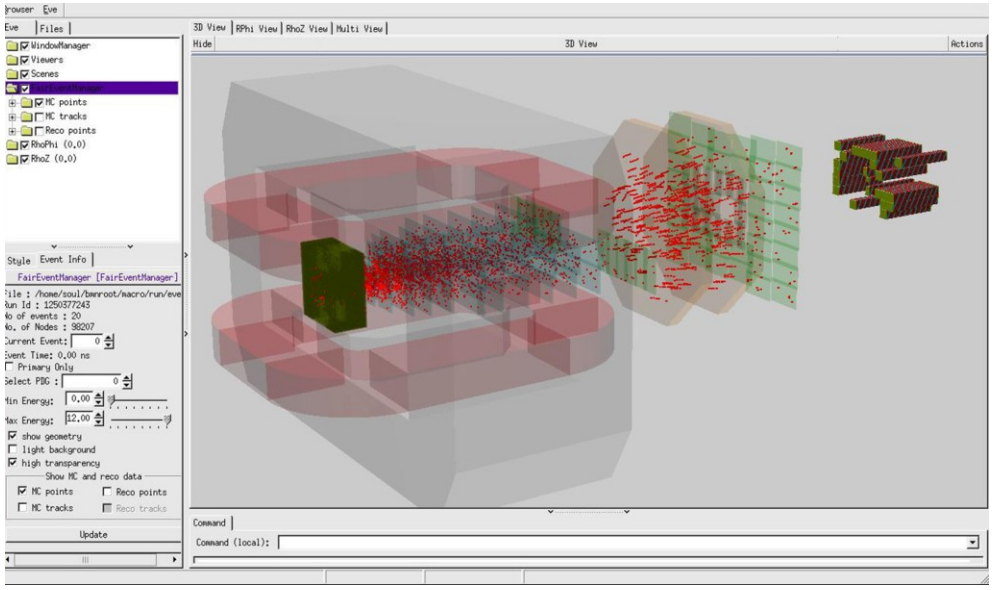

Figure 13. Event display for Au – Au collisions in the high transparency mode

6. UNIFIED DATABASE FOR BM@N OFFLINE DATA PROCESSING

6.1 The Unified Database of the Experiment

Today the use of databases is a prerequisite for qualitative management and unified access to the data of modern high-energy physics experiments. If you use file storing approach for experimental data, you can encounter with the following main problems:

- The usage of multiple files and arbitrary formats (binary, xml, html, excel, text), duplication of information in different files lead to data isolation, redundancy and inconsistency.
- The data of the experiment is distributed between many subdivisions and it can be difficult to find necessary absent information for the other subdivisions.
- Sequential, non-indexed file access and search are not efficient.
- No mechanism exists for relating data between these files.
- It is difficult to access and manipulate the data: one needs some dedicated programs.
- Uncontrolled concurrent access by multiple users also often leads to inconsistencies.

The Unified Database designed as comprehensive relational data storage for offline data analysis in the high-energy physics experiments offers solutions to the problems above. It provides unified access and data management for all collaboration members, correct multi-user data processing, ensuring the actuality of the information being accessed (run parameters, detector geometries and positions, technical and calibration data, etc.), data consistency and integrity, excluding the multiple duplication and use of outdated data. Furthermore, it provides automatic backup of the stored data to ensure that data of the experiment will not be lost due to software or hardware failures.

The parameter data being stored in the Unified Database can be classified into 4 groups:

1. Configuration data is concerned with the detector running mode, i.e., voltage settings as well as some programmable parameters for frontends electronics.
2. Calibration data describes the calibration and the alignment of the different subdetectors. Usually, quantities are evaluated by running dedicated offline algorithms.
3. Parameter data presents the state of detector subsystems. They include a variety of detector settings including the geometry and material definitions.
4. Algorithm data is used to control the way algorithms operate. It includes, for example, cuts for selection and production paths of files.

The Unified Database is developed primarily for the fixed target experiment BM@N and MPD experiment of the NICA project. As already mentioned, the BmnRoot has a runtime parameter manager which writes and initializes parameters from ASCII or ROOT files but it cannot work with a database system.

Entity-relationship diagram of the Unified Database is shown in the figure 14. The database includes the following tables: run periods, runs, shifts, detector geometries, detectors and its parameters, parameter values, and simulation files. Tango parameters (‘tango_parameter’ virtual table) of slow control system are not corresponding to any table of the Unified Database, but its interface was implemented as C++ object class. The detailed description of the database tables and their attributes is given in the Unified Database User’s Guide (*The Unified Database User’s Guide.pdf* file in *uni_db/docs* directory).

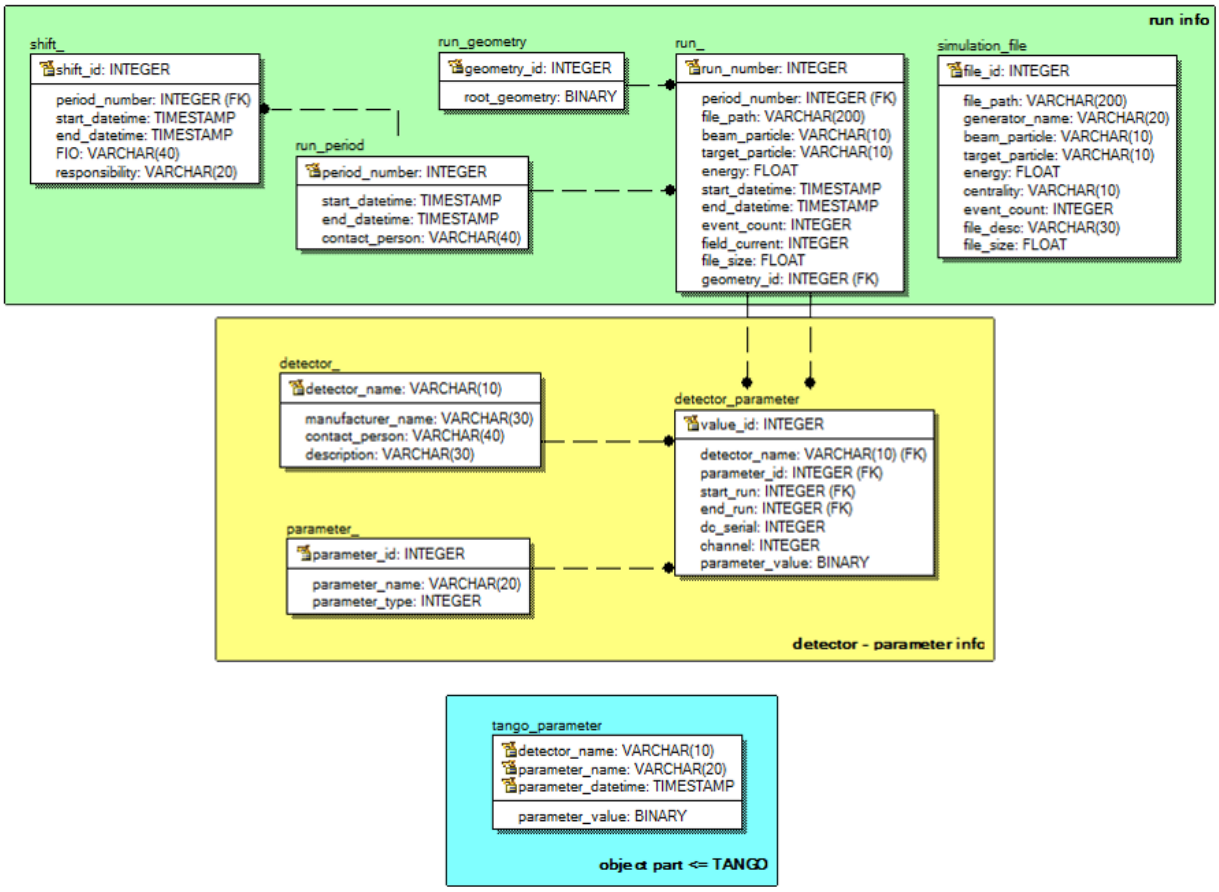

Figure 14. ER-diagram of the Unified Database

The Unified Database implemented on the PostgreSQL DBMS (database management system) provides user access to the actual information of the experiment run parameters, detector geometries changing during the runs, experimental data obtained, etc. The SQL code used for deploying the database was saved to the 'scheme/*uni_db.sql*' file.

The BM@N database was deployed on the NICA cluster. By default, users have a read-only access to the Unified Database. One can set own login and password to get other privileges in the corresponding 'db username' and 'db password' fields in the file 'uni_db/db_settings.h'. To test connection to the database, you can run test_db.C macro with ROOT CINT by the following command: "root –q uni_db/macros/test_db.C".

The common scheme of the Unified Database is presented in the figure 15.

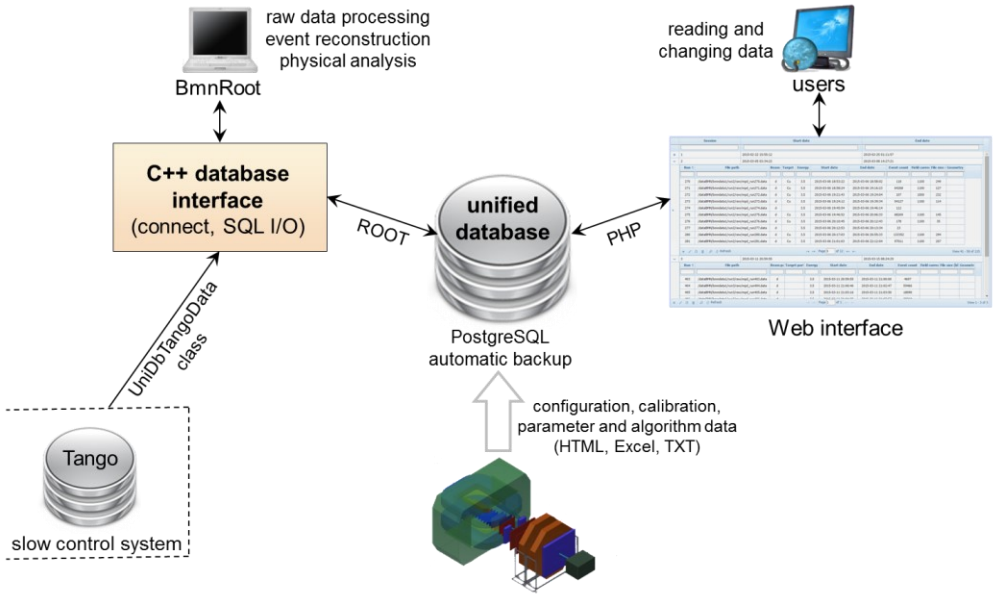

Figure 15. The common scheme of the Unified Database

The figure shows that the BM@N Unified Database can parse experimental data from multiple text files. The BmnRoot environment uses C++ database interface to get data for raw data processing, event reconstruction and physical analysis tasks. The BmnRoot framework also has the access to hardware data of slow control system Tango by the developed *UniDbTangoData* class. The development of Web-interface is required to easily read and change the data of the BM@N experiment by users.

6.2 C++ Interface of the Unified Database

To process data of the experiment, C++ database interface was implemented as a set of C++ class wrappers for all database tables with many specific functions. It allows one to work with the data of the Unified Database without any SQL statements. The structure of the Unified Database main directory (‘*uni_db*’) and classes for reading and writing data to the Unified Database are described in the Unified Database User’s Guide (*The Unified Database User’s Guide.pdf* file in *uni_db/docs* directory) in details. These classes allow to manage information about run periods, shifts, runs, detectors, parameters and its values, detector geometries and generated simulation files:

UniDbRunPeriod describes run periods (a set of runs) of the BM@N experiment;
UniDbRun – run parameters (number, time, energy, beam, target, magnetic field, file path, etc.);
UniDbRunGeometry contains BM@N geometry in the ROOT format;
UniDbDetector – detectors of the BM@N experiment (detector dictionary);
UniDbParameter – common information about detectors’ parameters presented on the previous slides and stored in the database (parameter dictionary);
UniDbDetectorParameter – values of detector parameters for experiment runs;
UniDbShift contains information about shifts of the experiment;
UniDbSimulationFile describes a set of simulation files produced by different event generators, such as UrQMD and QGSM.

The classes have main static functions to manage corresponding whole objects: *Create*, *Delete, Get*, *Search*, *PrintAll,* and non-static functions to work with their attributes: *Getters* and *Setters* functions, *Print*. The examples of use of the database interface are located in *uni_db/examples* directory.

Some additional classes of C++ database interface were developed. UniDbConnection class serves to open and close database connections, UniDbGenerateClasses class – to generate skeletons of the class wrappers for all tables of the database (it was used to generate classes of the C++ interface stored in the *db_classes* directory), UniDbParser class – to convert (parse) existing data of the experiment from text (txt, html, xml, excel) files with parameters and write them to the Unified Database (it's used by *parse_data_to_db.C* macro), UniDbSearchCondition class – to form criteria for selection of runs and detector parameters in the database, UniDbTangoData class – to get hardware data from the Tango ('slow' control system) database. In addition, the interface includes ROOT macros to execute the tasks presented by the classes above, numerous examples (ROOT macros) of using C++ interface to work with the database and documentation (Reference Manual and User's Guide).

The interface to the database of the slow control system is implemented as C++ object class – *UniDbTangoData*. *UniDbTangoData* class is used to read hardware electronics data from the Tango control system based on MySQL DBMS for selected time range (run number), detector and parameter defined by name. Main function of the class is *Tango_Data* GetTangoParameter(char* detector_name, char* parameter_name, char* date_start, char* date_end)*. The figure 16 presents an example of using *GetTangoParameter* function to get the high voltage of ZDC for a selected period.

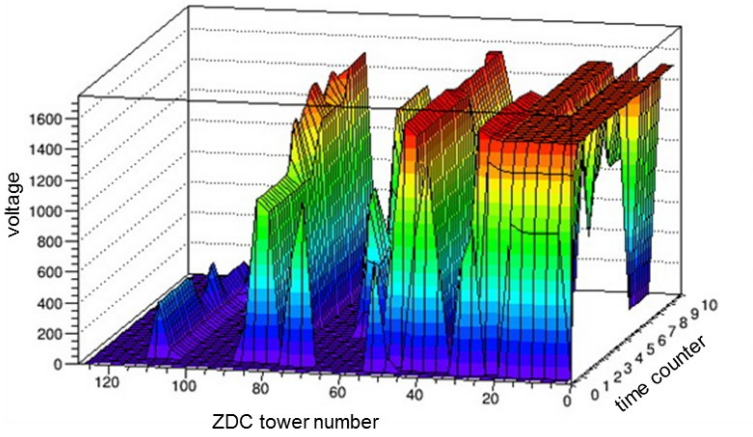

Figure 16. The High Voltage of Zero Degree Calorimeter

The Web-interface of the Unified Database is being developed to simplify reading and changing data of the experiment over the Web page.

7. DISTRIBUTED COMPUTING

7.1 NICA Cluster

The development of the distributed cluster for the experiments of the NICA project is required primarily for the following reasons: high interaction rate (up to 100 KHz for BM@N) and particle multiplicity. An MPD event of a central Au+Au collision at the energies of the NICA collider will register up to 1000 charged particles. As a result, one event reconstruction takes about ten of seconds now, and then sequential processing of one million events can take several months. Furthermore, a large data stream is expected from NICA, which is estimated at 10-20 PB of raw data per year. The figure 17 shows the scheme of data flow processing for the experiments of the NICA collider.

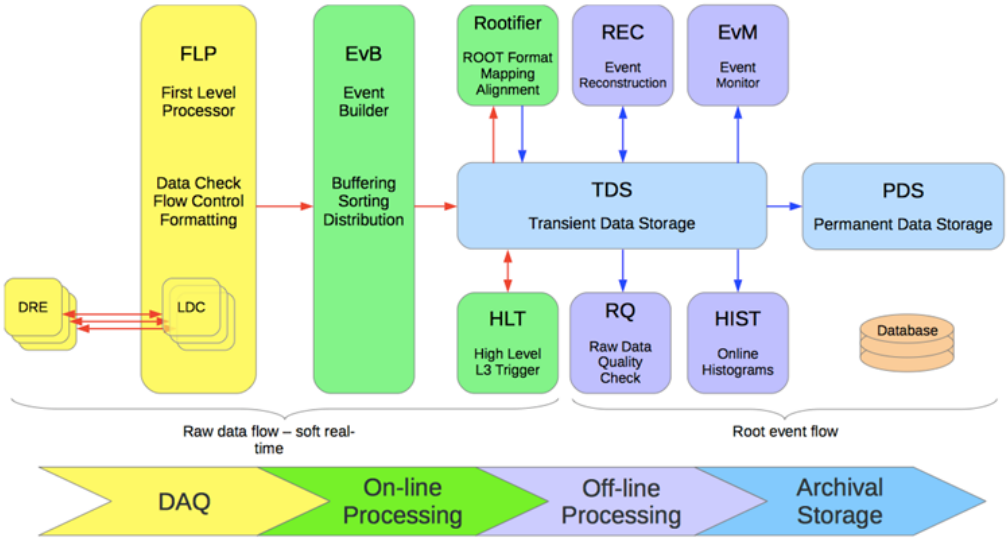

Figure 17. Data processing pipeline

The reasons given above led to the distributed NICA cluster development. Creation of the distributed NICA cluster based on the computer farm of the Laboratory of High Energy Physics was started in 2013. Now it consists of about 8 000 processor cores connected by Ethernet network with a bandwidth up to 100 Gb/s (figure 18). The NICA cluster consists of two parts: the interactive part, which can be connected to and interactively used by users, and the batch part where the user jobs are performed in parallel by scheduling system and other parallelizing tools.

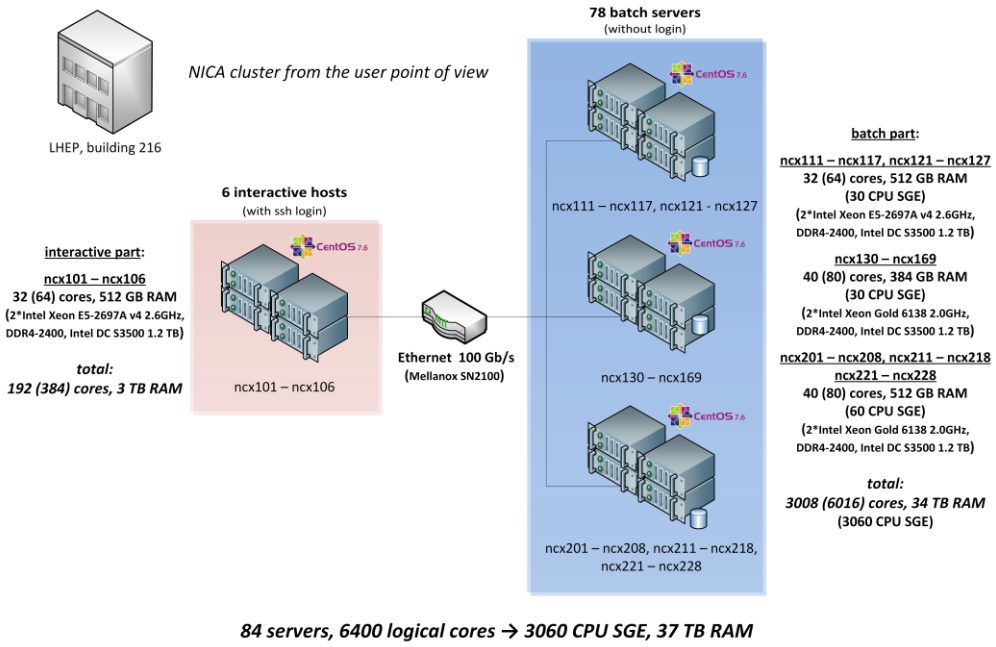

Figure 18. The current structure of the NICA cluster at LHEP

The interactive machines of the NICA cluster are available for the users of our software frameworks. To login, use your account for **ncx** machines (*ncx[101-105].jinr.ru*), e.g.:

```
ssh -X nc101.jinr.ru -l your_username
```

Users connecting to any interactive machine of the NICA cluster are directed to the same home directory. Home directory space is limited to 50 GB only. You can use EOS cluster space for your data.

Scientific Linux 7 is used on the cluster nodes now. All external packages and libraries (including FairSoft at */opt/fairsoft/bmn/pro* on each machine) for the experiments were installed and configured. Two main directions of the cluster development are data storage development for the experiments and organization of parallel event processing.

7.2 Distributed Data Storage

EOS [7] distributed file system is used to organize the data storage at the NICA cluster. It aggregates existing file systems in a common distributed file system. TCP/IP protocol or InfiniBand RDMA can be used to interconnect data nodes on EOS. Automatic replication and self-checking services working as background processes make it possible to prevent data loss and to restore files in case of hardware or software failure.

The current scheme of the distributed data storage based on the EOS is presented on the figure 19. The existing file systems were joined on the cluster machines to the */eos/nica/* shared volume of 8 PB. The shared partition */eos/nica/bmn/user/* was created for user directories, so that users connecting to any machine of the NICA cluster are directed to the same home directory. All the amounts are replicated on hard disks of the different machines.

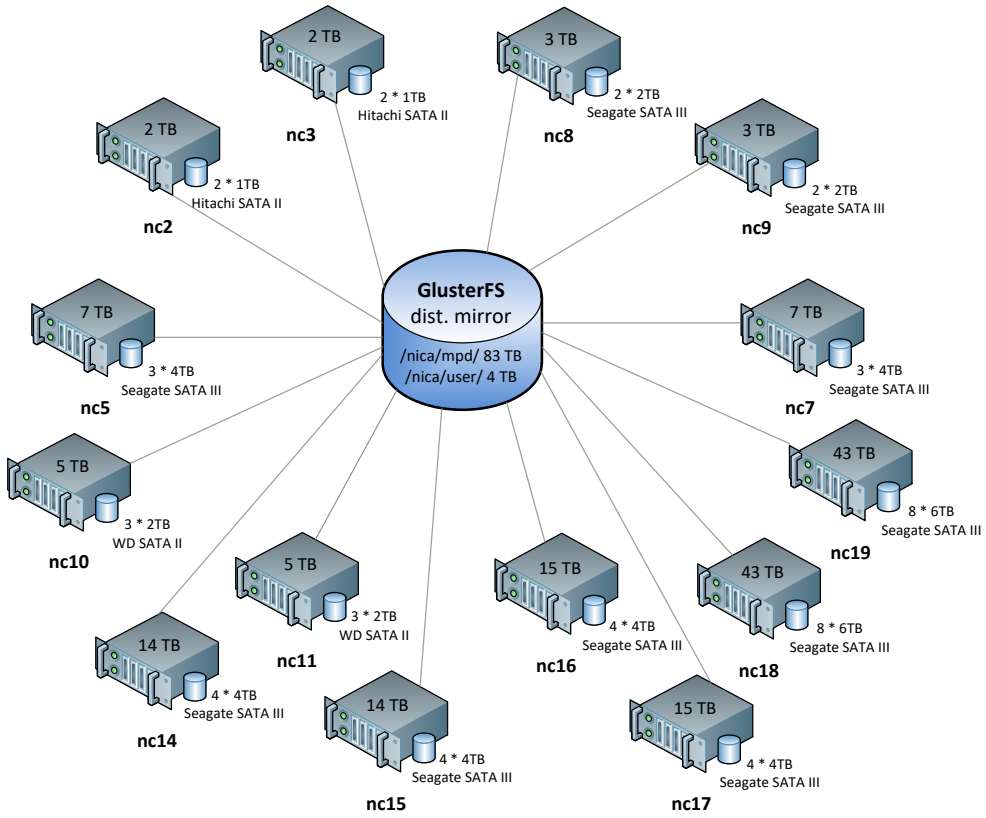

Figure 19. Distributed data storage on the NICA cluster

To parallelize the data processing on the NICA cluster two methods were implemented: using PROOF [8] software tool to parallel data processing in ROOT macros on the parallel architectures and developing scheduling system for the task distribution on the cluster nodes.

7.3 PROOF Parallelization of Event Processing

You can essentially accelerate the data processing by your macros with Parallel Root Facility (PROOF). The PROOF system is a part of the ROOT environment and included in ROOT distribution that is why one do not need to install additional software. It uses data independent parallelism based on the lack of correlation of events to process different events in parallel that leads to good scalability. One of the important PROOF properties is transparency: the same program code can execute both sequentially and concurrently. PROOF orients on three parallel architectures. PROOF-Lite parallelizes the data processing on one multiprocessor or multicore machine. PROOF parallelizes processing on heterogeneous computing clusters and in Grid system.

PROOF support was added to the BmnRoot software and reconstruction code was rewritten according to the PROOF rules and classes. The last new parameter of the reconstruction: *run_type* has default ‘local’ value for sequential processing, i.e., without PROOF. There are two possibilities to use PROOF in the BmnRoot software: on user multi-core local machine or on the NICA cluster.

To parallel BM@N event processing on one multicore machine with PROOF-Lite user can use ‘proof’ string value and can limit the threads number by “workers” number, e.g.

'proof:workers=3'. To speed up the data processing on the NICA cluster with PROOF server the last parameter is used and also can limit the number of the workers, e.g. 'proof:mpd@nc10.jinr.ru:21001:workers=$N_{workers}$'. Additional information on the use of the PROOF system in the BmnRoot you can find on our website in the *PROOF parallelization* Section.

The graph in the figure 20 presents the speedup of the reconstruction for one hundred events with PROOF-Lite on one machine having four processor cores.

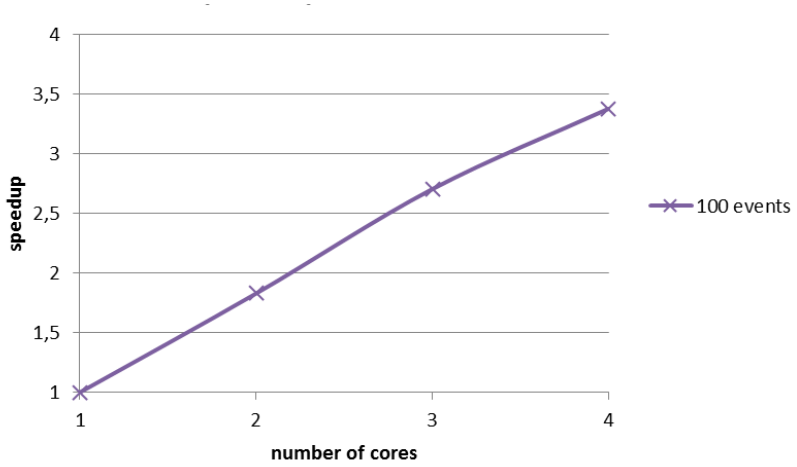

Figure 20. Speedup of the reconstruction on 4-cores machine

PROOF works on the NICA cluster as described below. A client sets the macro parameter to run it on the PROOF cluster in parallel. PROOF sends this task to PROOF On Demand server. The server runs the macro on the worker nodes and each assigned node receives one event from the input file to process. If worker node finishes the processing of the current event, it receives the next one. When all the events have been processed, the result is merged on the master server and is sent to the client or to the output file. It is important that operations on the objects in output lists must be independent on the order in which they executed.

Proof on Demand (PoD) farm was deployed on the NICA cluster. Now 64 logical processors of the batch cluster part are allocated for the PROOF farm to distribute event processing. The graph in the figure 21 presents the reconstruction speedup on the NICA cluster with PROOF On Demand for one thousand MPD events.

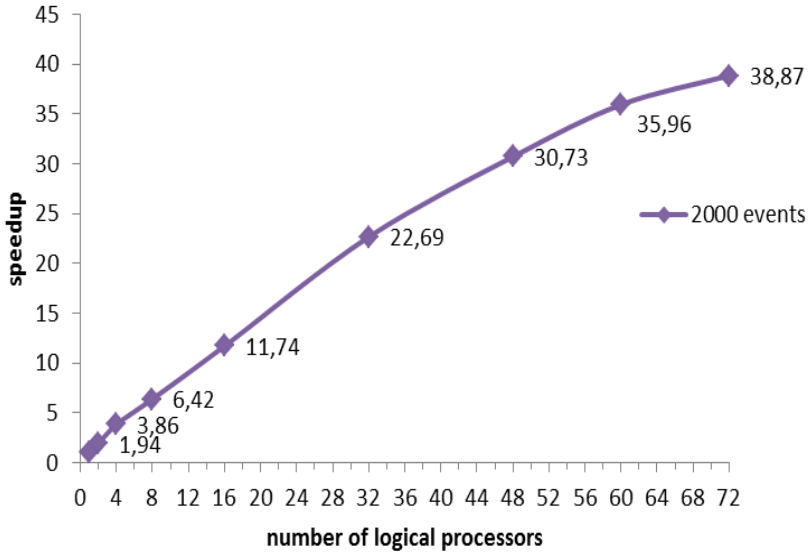

Figure 21. Speedup of the reconstruction on the NICA cluster with PROOF

7.4 Batch System for Distributed Job Execution

If you have many tasks or many files to process, you can use MPD Batch System on the NICA cluster to essentially accelerate your work. To distribute user's jobs on the NICA cluster and run it in parallel NICA-Scheduler system was developed. NICA-Scheduler is a part of BmnRoot framework. It was implemented in C++ language with ROOT classes' support. It uses the Sun Grid Engine (SGE) [9] scheduling system to distribute user jobs on the NICA cluster and simplifies parallel job executing without knowledge of SGE and *qsub* command. SGE combines batch cluster machines into the pool of worker nodes with 3 060 logical processor.

If you know how to work with Sun Grid Engine system, you can use *qsub* command on the NICA cluster by yourself. Otherwise, you can use NICA-Scheduler to run your tasks in parallel. Jobs for distributed execution on the NICA cluster are described and passed to NICA-scheduler as XML file (e.g., to run in bash, execute $ *nica-scheduler my_job.xml*).

The description starts and ends with tag <*job*>. Tag <*macro*> sets information about macro being executed by BmnRoot, its path and arguments. Tag <*file*> defines files to process by the macro. It can set absolute file path or define a file list from the database. Tag <*run*> describes run parameters and allocated resources for the job. If you want to run non-ROOT (arbitrary) command by MPD-scheduler, use <*command*> tag instead of <*macro*> with argument line – command line for distributed execution by the scheduling system. In addition, NICA-scheduler can be used to parallel event processing on one user multicore machine in local mode.

To execute user's jobs on the NICA cluster NICA-Scheduler parses the job description and runs shell scripts by *qsub* command of the Sun Grid Engine environment. Sun Grid Engine defines free workers of the cluster and runs the event data processing in parallel. When the worker finishes its part of the processing, the status of this worker will be changed to free value, so it can execute another user job. NICA-scheduler has the possibility to merge the result files in the mode of partial file processing.

The speedup for reconstruction of 72 simulated files on the NICA cluster is presented on the figure 22.

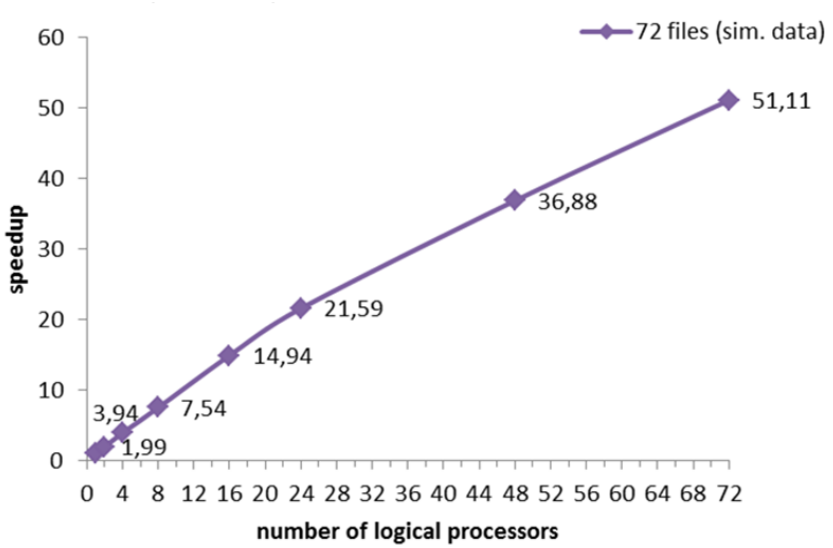

Figure 22. Reconstruction of 72 input files on the NICA cluster

More detailed manual to use the NICA-scheduler is located on our website at the *Batch Processing* Section. The directory *macro/nica_scheduler/examples* of the BmnRoot software contains different XML-examples for the developed scheduling system.

8. FREQUENTLY ASKED QUESTIONS

8.1 How to get full geometry (gGeoManager) of the BM@N facility?

1. **In a stand-alone macro from MC file with geometry (e.g. bmnsim.root):**

```
TFile* f = new TFile("bmnsim.root");
f->Get("FairGeoParSet");
gGeoManager…
```

2. **In any analysis task connected to FairRunAna (if gGeoManager is not yet known):**

```
FairRunAna* ana = FairRunAna::Instance();
FairRuntimeDb* rtdb = ana->GetRuntimeDb();
FairGeoParSet* geoParSet =  (FairGeoParSet*) rtdb->getContainer("FairGeoParSet");
gGeoManager…
```

3. **For existing BM@N run from the Unified Database (full example is presented in uni_db/examples/get_root_geometry.C):**

```
UniDbRun::ReadGeometryFile(period_number, run_number, “save_geometry_file.root”);
TFile* geoFile = new TFile(“save_geometry_file.root”, "READ");

TList* keyList = geoFile->GetListOfKeys();
TIter next(keyList);
TKey* key = (TKey*)next();
TString className(key->GetClassName());
if (className.BeginsWith("TGeoManager")) key->ReadObj();
gGeoManager…
```

8.2 How to get magnetic field value from the simulated data?

The field information is stored in container “FairBaseParSet”. From this container you can get field parameters. But you should not use it directly, because it is already done on the level of the “FairRunAna” class in the method “Init()”. You can access the magnetic field directly:

```
FairField *field = fRun->GetField();
```

The value of the magnetic field in the space point can be obtained by any of the following methods:

```
field->GetBx(x,y,z);
field->GetBy(x,y,z);
field->GetBz(x,y,z);
Double_t point[3], b[3];
field->GetFieldValue(point, b);
```

8.3 How to check if current MC track created hit into active volume of your detector?

You can use information about your detector placed into *MCTrack* branch of the simulated data. Each detector has the flag (bit value: 0 or 1), which indicate if this detector has hits for this *TrackID*. You can use the function:

```
Int_t FairMCTrack::GetNPoints(DetectorId detId);
```

for such a check, for example:

```
FairRootManager* ioman = FairRootManager::Instance();
pMCTracks  = (TClonesArray*) ioman->GetObject("MCTrack");

mcTrack = (FairMCTrack*) pMCTracks->UncheckedAt(mcTrackIndex);
if (mcTrack->GetNPoints(kTOF))
{
    // do your analysis of this track
}
```

The constant numbers (DetectorId) of all BM@N detectors are described in the header:

```
bmnroot/bmndata/CbmDetectorList.h
```

REFERENCES

1. Kapishin M., Kolesnikov V., Vasendina V., Zinchenko A. Study of baryonic matter with the BM@N experiment at the Nuclotron // JINR NEWS, No.3, 2014.
2. BM@N Collaboration. BM@N — Baryonic Matter at Nuclotron. Conceptual Design Report. JINR, 2013.
3. CERN ROOT. https://root.cern.ch.
4. Al-Turany M., Bertini D., Karabowicz R. and other. The FairRoot framework // Journal of Physics: Conference Series. Vol. 396, Part 2. 2012. 10 p.
5. CERN. A Large Ion Collider Experiment. http://aliceinfo.cern.ch/Public/Welcome.html.
6. Gertsenberger K., Merz S., Chepin E. Event visualization of the MPD experiment at the NICA collider for the monitoring system // Scientific Visualization. 2014. Vol. 6, No. 1. pp. 1–19.
7. GlusterFS Developers. Gluster File System 3.3.0. Administration Guide. Using Gluster File System. Gluster, 2012. 134 p.
8. Hayrapetyan A., Vala M. ROOT and PROOF Tutorial. GridKa School, Karlsruhe Institute of Technology, 2012. 58 p.
9. Haas A. Sun Grid Engine 6.1 + DRMAA interface. Sun Microsystems, 2007. 33 p.