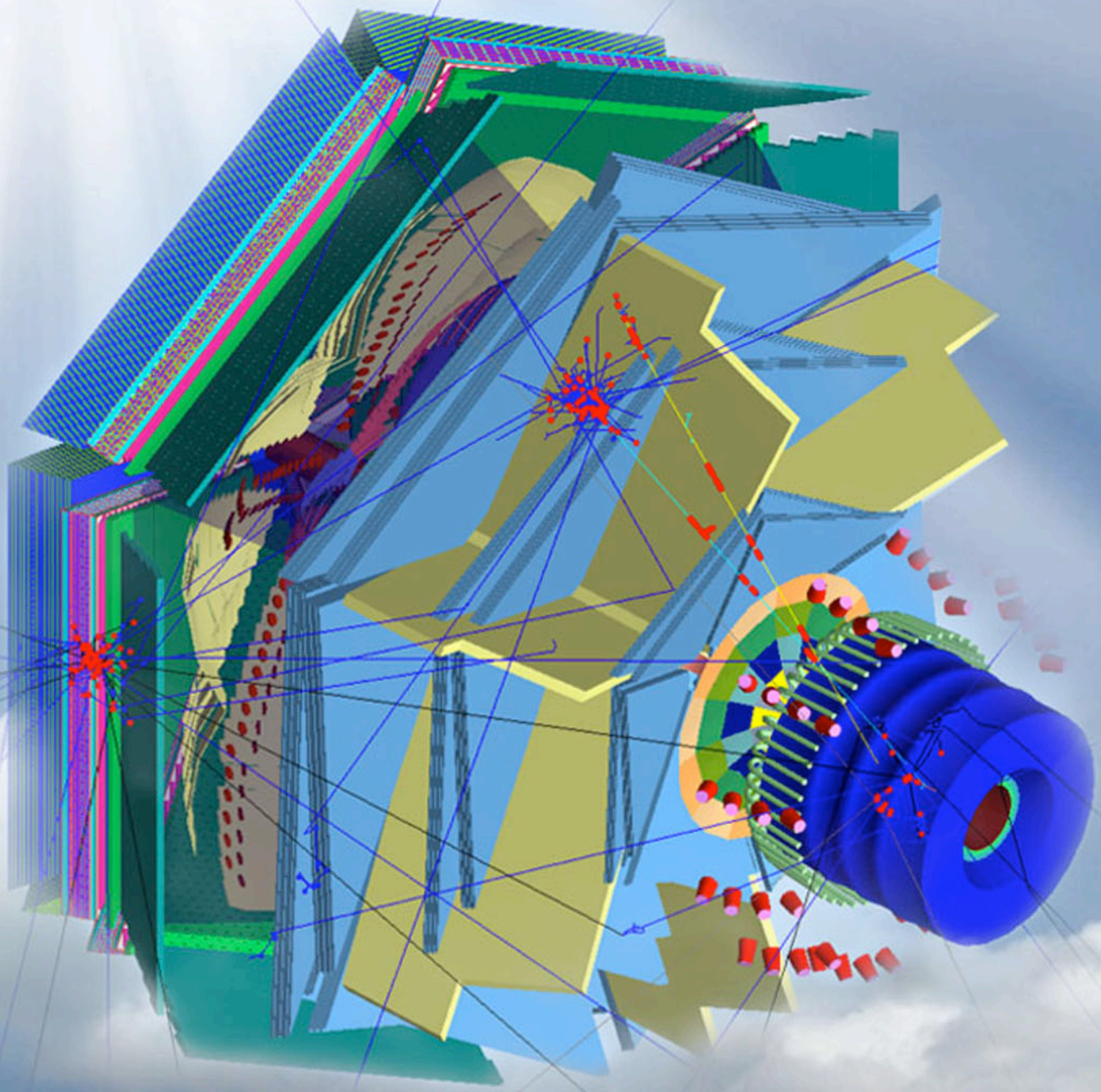
CLAS12 - Software Document

Hall B 12 GeV Upgrade

May 2012



CLAS12 Software Working Group

CLAS12 Software
Version 1.3

Editors: D.P. Weygand, V. Ziegler

May 30, 2012

Contents

1 The CLAS12 Detector and its Science Program at the Jefferson Lab Upgrade

1.1 Introduction

A brief overview of the CLAS12 detector is presented and a brief synopsis of the initial physics program after the energy-doubling of the Jefferson Lab electron accelerator. Construction of the 12 GeV upgrade project started in October 2008. A broad program has been developed to map the nucleon's 3-dimensional spin and flavor content through the measurement of deeply exclusive and semi-inclusive processes. Other programs include measurements of the forward distribution function to large $x_B \leq 0.85$ and the quark and gluon polarized distribution functions, and nucleon ground state and transition form factors at high Q^2. The 12 GeV electron beam and the large acceptance of CLAS12 are also well suited to explore hadronization properties using the nucleus as a laboratory.

1.2 Science Summary

The challenge of understanding nucleon electromagnetic structure still continues after more than five decades of experimental scrutiny. From the initial measurements of elastic form factors to the accurate determination of parton distributions through deep inelastic scattering (DIS), the experiments have increased in statistical and systematic accuracy. It was realized in recent years that the parton distribution functions represent special cases of a more general and much more powerful way of characterizing the structure of the nucleon, the generalized parton distributions (GPDs).

The GPDs describe the simultaneous distribution of particles with respect to both position and momentum. In addition to providing information about the spatial density (form factors) and momentum density (parton distributions), these functions reveal the correlation of the spatial and momentum distributions, *i.e.* how the spatial shape of the nucleon changes when probing quarks of different wavelengths.

The concept of GPDs has led to completely new methods of "spatial imaging" of the nucleon, either in the form of two-dimensional tomographic images, or in the form of genuine three-dimensional images. GPDs also allow us to quantify how the orbital motion of quarks in the nucleon contributes to the nucleon spin – a question of crucial importance for our understanding of the "mechanics" underlying nucleon structure. The spatial view of the nucleon enabled by the GPDs provides us with new ways to test dynamical models of nucleon structure.

The mapping of the nucleon GPDs, and a detailed understanding of the spatial quark and gluon structure of the nucleon, have been widely recognized as the key objectives of nuclear physics of the next decade. This requires a comprehensive program, combining results of measurements of a variety of processes in electron–nucleon scattering with structural information obtained from theoretical studies, as well as with expected results from

future lattice QCD simulations.

While GPDs, and also the more recently introduced transverse momentum-dependent distribution functions (TMDs), open up new avenues of research, the traditional means of studying nucleon structure through electromagnetic elastic and transition form factors, and through flavor- and spin-dependent parton distributions, must also be employed with high precision to extract physics on the nucleon structure in the transition from the regime of quark confinement to the domain of asymptotic freedom. These avenues of research can be explored using the 12 GeV cw beam of the JLab upgrade with much higher precision than has been achieved before, and can help reveal some of the remaining secrets of QCD. Also, the high luminosity available will allow researchers to explore the regime of extreme quark momentum, where a single quark carries 80% or more of the proton's total momentum. This domain of QCD has been hardly explored and provides challenges to the detection systems in term of kinematic coverage and high luminosity operation.

A large portion of the science program is dedicated to the measurement of exclusive processes, such as deeply virtual Compton scattering $ep \rightarrow ep\gamma$ or deeply virtual meson production, e.g. $ep \rightarrow ep\rho^0$ or $ep \rightarrow e\pi^+ n$, which require the detection of 3 or 4 particles in the final state. Another part of the program is based on the measurement of semi-inclusive processes, e.g. $ep \rightarrow e\pi^+ X$ or $ep \rightarrow e\pi^0 X$, where a high momentum charged or neutral pion must be detected in addition to the scattered electron.

Other programs require measurement of the full cm angular distribution of a more complex final state such as $ep \rightarrow ep\pi^+\pi^-$ to be able to identify the spin-parity of an intermediate baryon resonance.

The CLAS12 science program currently covers GDPs, hard-exclusive processes, semi-inclusive deep inelastic processes, single-spin asymmetry measurements, deep inelastic structure functions of the nucleon, experiments using the nucleus as a laboratory and hadron spectroscopy. It includes a broad program using polarized beams, longitudinally and transversely polarized targets, and measurements of outgoing recoil baryon polarization.

1.3 From CLAS to CLAS12

The CLAS detector was designed and built in the 1990s and became fully operational in 1998. It has been in operation since. CLAS has been operated with electron and photon beams with luminosities up to $L = 10^{34}$ sec^{-1}cm^{-2}. The driving motivation for CLAS was the nucleon resonance program, with emphasis on the study of resonance transition form factors and the search for *missing resonances*. Figure 1 shows two schematic views of CLAS. At the core of CLAS is a toroidal magnet consisting of six superconducting coils symmetrically arranged around the beam line. Each of the sectors is instrumented as an independent spectrometer with 34 layers of tracking chambers, allowing for the full reconstruction of the charged particle 3-momentum vectors. Charged hadron identification is accomplished by combining momentum and time-of-flight with the measured path length

from the target to the scintillation counters which surround the tracking detectors. The forward angular range from 10° to 50° is instrumented with gas Cerenkov counters for the identification of electrons.

The CLAS data acquisition system was designed to read out 35,000 drift chamber sense wires and over 2,500 channels of photomultiplier tubes.

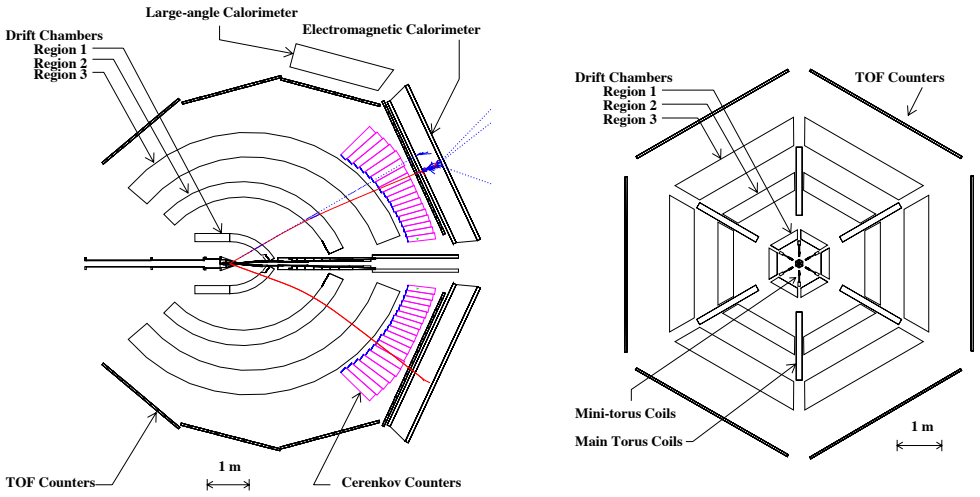


Figure 1: Two views of the CLAS detector system. Left panel: Longitudinal cut along the beam line showing the three different regions of drift chambers, the Cerenkov counters at forward angles, the time-of-flight (TOF) system, and the electromagnetic calorimeters. The simulated event shows an electron (upper) and a positively charged hadron. Right panel: Transverse cut through CLAS. The six superconducting coils provide a six sector structure with independent detectors.

1.4 The CLAS12 Detector

To meet the requirements of high statistics measurements for exclusive processes, the equipment at JLab will undergo major upgrades. In particular it will include the CLAS12 Large Acceptance Spectrometer [1]. A 3-dimensional rendition is shown in Fig. 2. CLAS12 has two major parts with different functions, the Forward Detector (FD) and the Central Detector (CD). The FD retains the 6-sector symmetric design of CLAS, which is based on a toroidal magnet with six superconducting coils that are symmetrically arranged around the beam axis. The main new features of CLAS12 include operation with a luminosity of 10^{35} cm^{-2}sec^{-1}, an order of magnitude increase over CLAS [2], and improved acceptance and particle detection capabilities at forward angles. In this section we present short

descriptions of the detector system.

1.4.1 The Forward Detector

The doubling of the beam energy from 6 GeV to 12 GeV requires improved electron-pion separation at higher momentum. This is achieved with a threshold gas Cerenkov counter with a pion momentum threshold of about 5 GeV. The new high threshold Cerenkov counter (HTCC) is positioned in front of the superconducting toroidal magnet, and has to present as little material to the charged particles as practical to limit the multiple scattering contributing to the vertex resolution. This requires use of low mass composite material for the mirror system.

The HTCC is followed by a toroidal magnet for the momentum analysis of tracks with scattering angles from 5° to 40°. Similar to CLAS, the new toroidal magnet has six superconducting coils symmetrically arranged around the beam line, and provides six sectors for charged particle detection. Each sector has its own tracking and particle identification detectors. Tracking is accomplished with a set of 3 regions of drift chambers with 12 layers of hexagoal drift cells arranged at stereo angles of ±6°. This arrangement provides good angular resolution both in polar and in azimuthal angles. The drift chamber system will provide up to 36 measurements for a single charged track and has sufficient redundancy for pattern recognition and track reconstruction.

The torus magnet and the drift chamber system are followed by the low-threshold Cerenkov counter (LTCC) that provides charged pion identification for momenta greater than 3 GeV. Following the LTCC are two arrays of plastic scintillators for charged particle identification. The first layer is 6 cm thick and has 6 cm wide bars. It provides timing information of $\delta T < 100$ psec. The second layer contains 23 bars of 5 cm thick and 15 cm wide scintillator and provides timing information of 120 to 180 psec, depending on the length of the bars. A combined average resolution of 80 psec is expected. For equal pion, kaon, and proton yields, this will enable a 4σ π/K separation up to 3 GeV, and a K/p separation up to 4.5 GeV from time-of-flight measurements alone. A future upgrade of the particle identification system is under consideration. One or more of the LTCC sectors would be replaced with RICH detectors, allowing for a significant improvement in the identification of pions, kaons and protons at high momentum where time-of-flight measurements are less effective.

Large parts of the physics program require the identification of single high energy photons and separation from $\pi^0 \to \gamma\gamma$ up to 9 GeV. The granularity of the existing electromagnetic calorimeter (EC) will be improved by adding a preshower calorimeter (PCAL) of 5-6 radiation lengths in front of the EC. The PCAL is expected to provide a factor of 2.5 improvement in spatial resolution and two-photon separation up to 10 GeV momenta. At forward angles, below 6°, a lead-tungstate inner calorimeter (IC) consisting of 420 crystals with an average cross section of 15 mm x 15 mm and 22 rad. length thick, provides photon and π° identification for momenta up to 10 GeV.

A quasi-real photon forward tagging system (FT) will measure electrons at scattering angles from 2° - 4° using the IC for energy measurements, a segmented scintillator hodoscope for triggering, and micromegas tracking chambers.

1.4.2 The Central Detector

The Central Detector (CD) is based on a compact solenoid magnet with maximum central magnetic field of 5 Tesla. The solenoid magnet provides momentum analysis for polar angles greater than 35°, protection of the tracking detectors from background electrons, and acts as a polarizing field for dynamically polarized solid state targets. All three functions require high magnetic field. The overall size of the solenoid is restricted to 2000 mm in diameter, which allows a maximum warm bore for the placement of detectors of 780 mm. To obtain sufficient momentum resolution in the limited space available requires high field and excellent position resolution of the tracking detectors. The central field in the target region must also be very uniform at $\Delta B/B < 10^{-4}$ to allow the operation of a dynamically polarized target. To achieve a sustained high polarization for polarized ammonia targets requires magnetic fields in excess of 3 T. Magnetic fields of 5 T have been most recently used for such targets with polarization of 80% - 90% for hydrogen. In addition, the solenoidal field provides the ideal guiding field for keeping the copiously produced Möller electrons away from the sensitive detectors and guide them to downstream locations where they can be passively absorbed in high-Z material. The production target is centered in the solenoid magnet. A forward micromegas tracker (FMT) consisting of 6 stereo layers is located just downstream of the production target and in front of the HTCC. The FMT aids the track reconstruction in the FD and covers the polar angle range of 5° to 35° and improves vertex and momentum resolution of high momentum tracks.

Tracking in the CD is provided by a silicon vertex tracker (SVT) and a barrel micromegas tracker (BMT). Each of the two detectors provide 3 space points for the tracks in the angle range of the CD. The combined SVT and BMT cover polar angles from 35° to 125°.

The central time-of-flight scintillator barrel (CTOF) consists of 48 bars of fast plastic scintillator equipped with 96 photomultiplier tubes that provide 2-sided light read-out. The scintillator light is brought to an area of reduced magnetic field where fast PMTs with dynamical shielding arrangement can be employed. A timing resolution of less than 60 psec has been achieved for this system in prototype tests. The very short flight path available in the CD allows for particle identification in a restricted momentum range of up to 1.2 GeV and 0.65 GeV for pion-proton and pion-kaon separation, respectively.

Also under development by a French-Italian collaboration is an additional detector that will add neutral particle detection capabilities. This detector will fill the gap between the CTOF and the solenoid cryostat.

With these upgrades CLAS12, will be the workhorse for exclusive and semi-inclusive electro-production experiments in the deep inelastic kinematics.

1.5 Data Acquisition and Trigger

CLAS12 will typically operate at design luminosities of 10^{35} $cm^{-2}sec^{-1}$ on liquid hydrogen or on polarized targets. Most experiments require the presence of an electron track at scattering angles greater than 5° with momentum above 1.5 GeV. The event trigger will be provided by the HTCC and the combination of clustered energy deposited in the PCAL and the EC to select scattered electrons with high enough energy. The estimated rate for this trigger selection is approximately 5 kHz, with average event sizes of 7 kB, or a data rate of 35 MB/sec, which is comfortably below the DAQ design capabilities of 10 kHz and 100 MB/sec. For measurements where the forward tagger is used, hadronic energy is required in the forward and central detector to record the event. The rate of this type of event is very high and will be pre-scaled to remain below the acceptable acquisition rate.

The base detector system will be instrumented with a total of 4,336 photomultiplier tubes, read-out with pipeline TDCs and flash ADCs. The number of read-out channels in the drift chamber system is 24,000. The SVT has a total of 30,000 read-out strips, and the micromegas gas detectors add another 20,000 channels, resulting in a total of 78,336 channels.

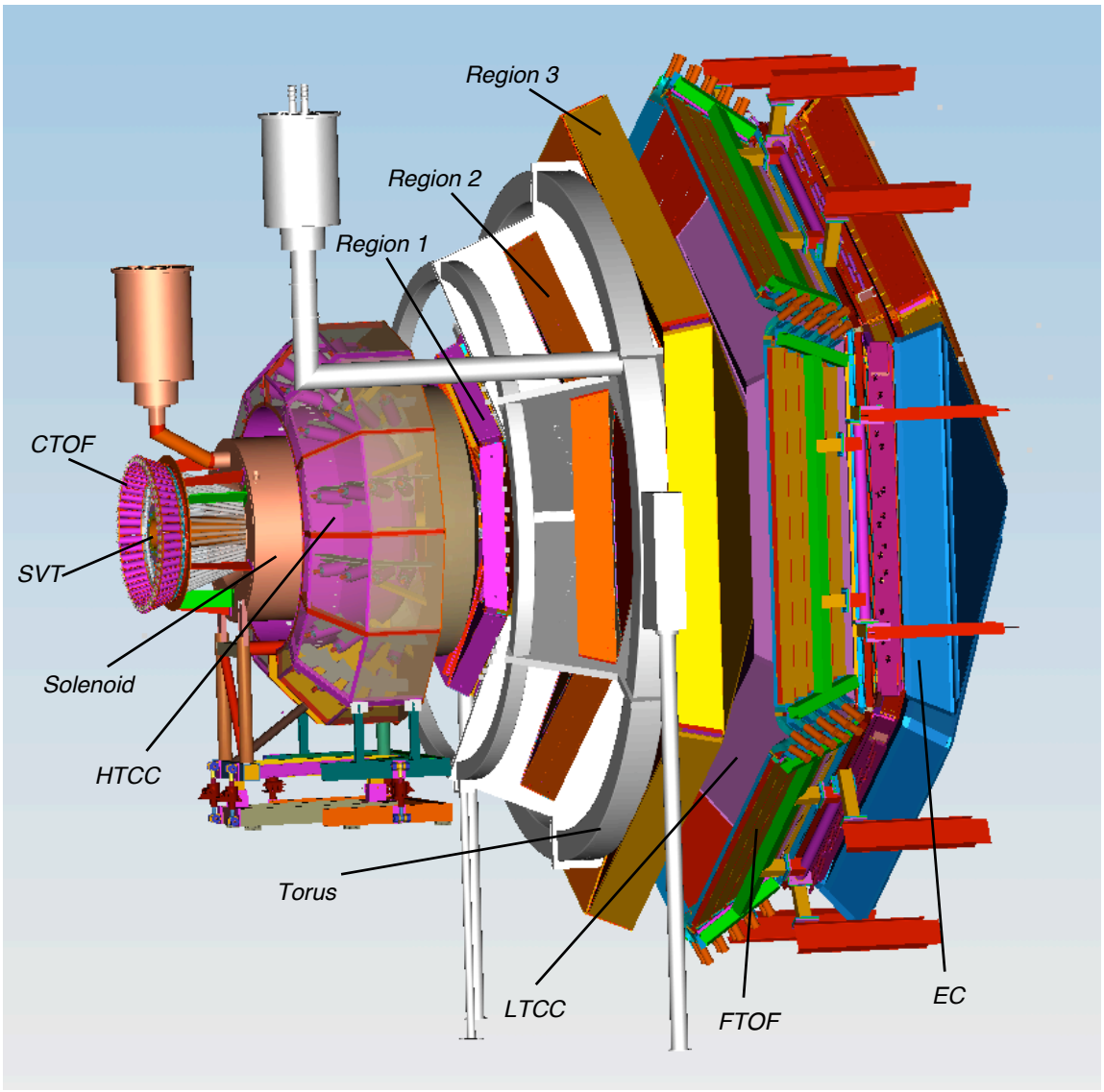


Figure 2: 3D view of the CLAS12 detector. The beam comes from the left. The target is located inside the superconducting solenoid magnet.

1.6 Calibration and Commissioning of CLAS12

The CLAS collaboration has established a four member team to organize the calibration and commissioning effort of all CLAS12 detector components and the CLAS12 system as a whole. Procedures, requirements and plans have been developed in conjunction with the respective detector groups. The second draft document (V 2.2) is available [3]. Here we give a brief summary of this work. The document covers all aspects of the detector construction, from quality assurance measures of procured components, to detector checkout during construction, to the testing of the fully assembled detectors with cosmic rays, to final tests after initiallization with cosmic rays, to final calibration and commissioning plans with beam.

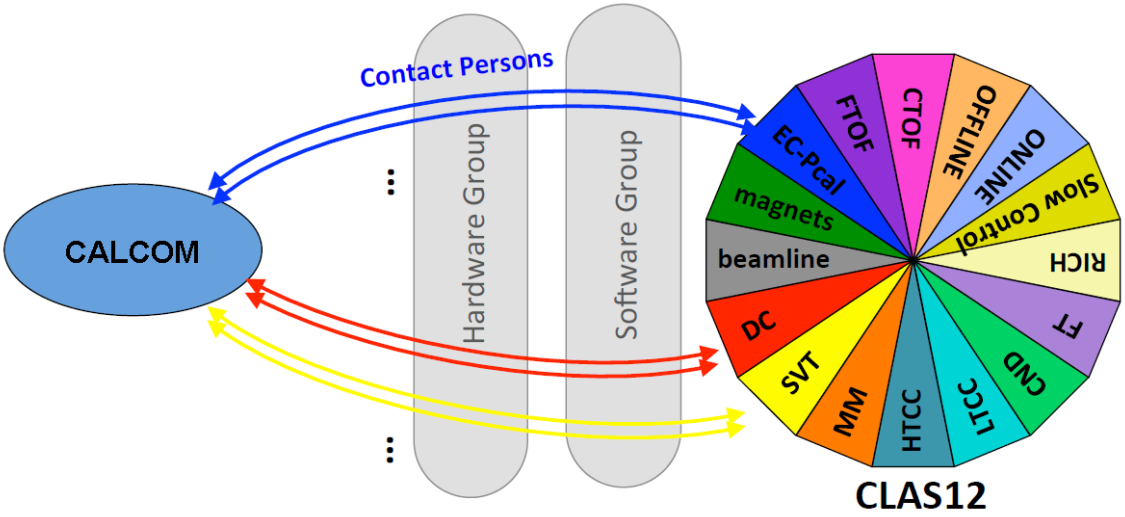


Figure 3: Communication between detector groups and the CALCOM team proceeds through two contact persons who act as liaisons for each detector system.

The calibration and commissioning plan:

- should address all detector systems with a coherent plan from the construction and installation phase to in-beam commissioning;
- should include hardware, software and manpower requirements;
- should ensure all necessary data for the checkout and calibration of the CLAS12 spectrometer are collected in an efficient and timely manner.

Much of the commissioning includes specialized calibration procedures that require the collaboration to

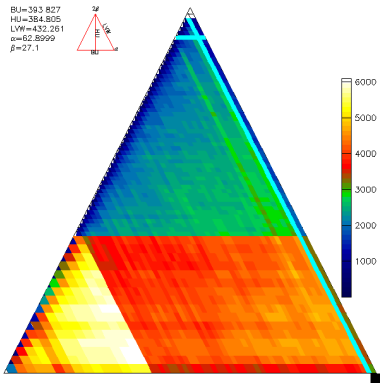
- develop and test procedures for each detector system;
- define input data (data type and statistics) for the calibration algorithms;
- develop and test the necessary software tools;
- evaluate manpower and computing resource needs;
- provide all relevant parameters to perform full reconstruction and to evaluate the detector performance.

An important mechanism in this process is the CLAS12 Service Work Committee to which all collaborating institutions provide manpower resources: graduate student, postdocs, and senior researchers who pledge to devote part of their time in general support of the collaboration's goals through software or hardware contributions. The CALCOM group receives requests from the detector groups via two liaison persons for support with calibration, commissioning, and maintenance activities. The CALCOM group can then request resources from the Service Work Committee to assign the necessary resources to support a specific detector group. Figure 3 illustrates schematically the relationship between the detector groups and the CALCOM committee. The detector groups develop their own software needed for the calibration and commissioning tasks, much of which will be re-used after the detector installation in CLAS12, including during beam operation. Representative figures of the PCAL calibration procedure during cosmic ray testing are shown in Fig. 4.

PCAL Checkout and Calibration

Full checkout/calibration of first sector module with cosmic rays

- PMT gain matching
- Light attenuation
- Light Yield (in progress)



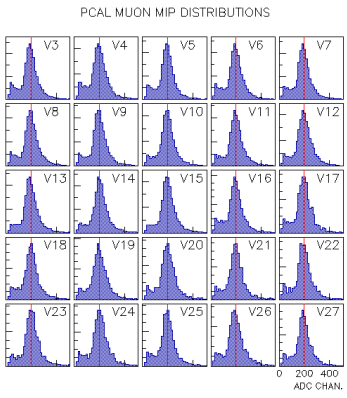


Figure 4: Calibration procedures during the cosmic ray testing of one of the six pre-shower calorimeter modules. The PMT signals are analyzed with the online software to determine the attenuation length of each scintillator stack and for gain matching. The ADC spectra show the fully calibrated and gain-matched detector response.

2 The CLAS12 Offline Software

2.1 Introduction

Modern high energy and nuclear physics experiments require significant computing power to keep up with large experimental data volumes. To achieve quality physics data analysis, intellectual input from diverse groups within a large collaboration must be brought together.

Experimental physics data analysis in a collaborative environment has historically involved a computing model based on self-contained, monolithic software applications running in batch-processing mode. This model, if not organized properly, can result in inefficiencies in terms of deployment, maintenance, response to program errors, update propagation, scalability and fault-tolerance. We have experienced such problems during the fifteen years of operation of the CLAS on- and off-line software. Even though these challenges are common to all physics data processing (PDP) applications, the small size of the CLAS offline group magnified their effect. Experimental configurations have become more complex and compute capacity has expanded at a rate consistent with Moore's Law. As a consequence, these compute applications have become much more complex, with significant interaction between diverse program components. This has led to computing systems so complex and intertwined that the programs have become difficult to maintain and extend.

In large experimental physics collaborations it is difficult to enforce policies on computer hardware. For example, some groups might use whatever computing resources they have at their home institutions. In turn, these resources evolve as new hardware is added. Additional software and organizational effort must often be put in place to provide the maintenance needed to update and rebuild software applications for proper functionality in specific hardware or OS environments.

In order to improve productivity, it is essential to provide location-independent access to data, as well as flexibility of the design, operation, maintenance and extension of physics data processing applications. Physics data processing applications have a very long lifetime, and the ability to upgrade technologies is therefore essential. Thus, software applications must be organized in a way that easily permits the discarding of aged components and the inclusion of new ones without having to redesign entire software packages at each change. The addition of new modules and removal of unsatisfactory ones is a natural process of the evolution of applications over time. Experience shows that software evolution and diversification is important, and results in more efficient and robust applications. New generations of young physicists doing data analysis may or may not have the programming skills required for extending/modifying applications that were written using older technologies. For example, JAVA is the main educational programming language in many universities today, but most of the data production software applications are written in C++ and some even in FORTRAN.

The offline software of the CLAS12 project aims at providing tools to the collaboration

that allow design, simulation, and data analysis to proceed in an efficient, repeatable, and understandable way. The process should be designed to minimize errors and to allow cross-checks of results. As much as possible, software-engineering-related details should be hidden from collaborators, allowing them to concentrate on the physics.

2.2 Ways to Increase Productivity

There are well-known practices leading to improved software productivity and quality. These include software modularity, minimized coupling and dependencies between modules, simplicity and operational specialization of modules, technology abstraction (including high level programming languages), and most importantly, rapid prototyping, deployment and testing cycles. It is also important to take into account the qualifications of software contributors: the physicist best understands the physics process and algorithms, and the computer scientist/programmer has advanced skills in software programming. An environment that encourages collaboration, with code development responsibilities clearly separated and established, can increase the quality and number of physics data analysis contributions. The CLAS12 software group has studied different PDP frameworks and has searched for contemporary approaches and computing architectures best suited to achieve these goals. The CLAS12 framework design was inspired to a great extent by the GAUDI framework [4] which was adopted by the LHCb experiment. The CLAS12 framework includes design concepts based on the GAUDI framework's data centricity, its clear separation between data and algorithms, and its data classifications. However, the GAUDI framework is based on an Object Oriented Architecture (OOA), requiring compilation in a self-contained, monolithic application that can only be scaled in batch or grid systems. This approach usually requires that a relatively large software group be involved in the development, maintenance and system operation.

2.3 Choice of Computing Model and Architecture

We have researched among the most advanced emerging computing trends and our attention was caught by the cloud-computing model. This model promises to address our computing challenges. It has reached its maturity level and many scientific organizations, including CERN, are moving in its direction. The cloud computing model is based on a Service Oriented Architecture (SOA). SOA is a way of designing, developing, deploying and managing software systems characterized by coarse-grained services that represent reusable functionality. In SOA, service consumers compose applications or systems using the functionality provided by these services through a standard interface. SOA is not a technology and is more like a blueprint for designing and developing computational environments. Services usually are loosely coupled, depending on each other minimally. Services encapsulate and hide technologies, as well as the functional contexts used inside a service.

2.4 ClaRA: CLAS12 Reconstruction and Analysis Framework

The goal of the ClaRA project [5] is to develop a framework which can be applied to a wide range of physics data processing applications written for the CLAS12 experiments. The framework shall cover all stages of the physics data processing, including physics and detector simulations, high-level software triggers, reconstruction, analysis and visualization applications.

Building a physics data processing application is a collaborative process that involves the development of a framework and basic software components by computer scientists and the implementation by physicists of specific algorithms for simulation, calibration, reconstruction and physics analysis purposes. The quality and productivity of physics results depends on the efficiency of data processing and the ability to perform detailed checks at each processing stage. The unprecedented scale and complexity of the physics computing environment requires substantial software engineering skills from the end user, resulting in a reduction in the number of qualified data processing physicists, and therefore poorer physics outcome.

The CLAS12 computing environment must keep up with fast growing computing technologies. Our evaluation of the CLAS12 research and software design requirements led us to conclude that an SOA would provide a computing environment capable of addressing our requirements to allow for high, efficient and quality physics data processing outcomes. This choice of architecture involves technology abstraction, and includes multilingual support, the lack thereof would alienate many potential contributors. The CLAS12 reconstruction and analysis framework was developed to support both traditional and cloud computing models by providing single process or distributed deployment choices. These deployment choices do not require any additional programming effort, which is a very important advantage of this framework.

ClaRA identifies the entire data processing application as a composite application. A composite application is both assembled and orchestrated. Assembling is the process of combining many different pieces into a workable unit. The orchestration consists of insuring that all the assembled pieces work together collaboratively to solve a given problem within a specific scenario. Composite applications have been established to be more efficient, robust and flexible. The composites of a software application, called *software services* are specialized, easily maintainable, replaceable and modifiable. Hence, software applications which are composed of services are extremely flexible, robust and adaptable to address different data processing needs. The SOA provides the foundation of creating, deploying, maintaining and governing the software services. The ClaRA framework is an implementation of an SOA.

2.4.1 The Attributes of ClaRA Services

The ClaRA software modules or services contain special data processing algorithms. Within an application, services communicate data through a pub-sub messaging system (for dis-

tributed deployment) and/or through shared memory (for single process deployment). Figure 5 is a block diagram illustrating a CLAS12 event reconstruction application which uses two data communication channels.

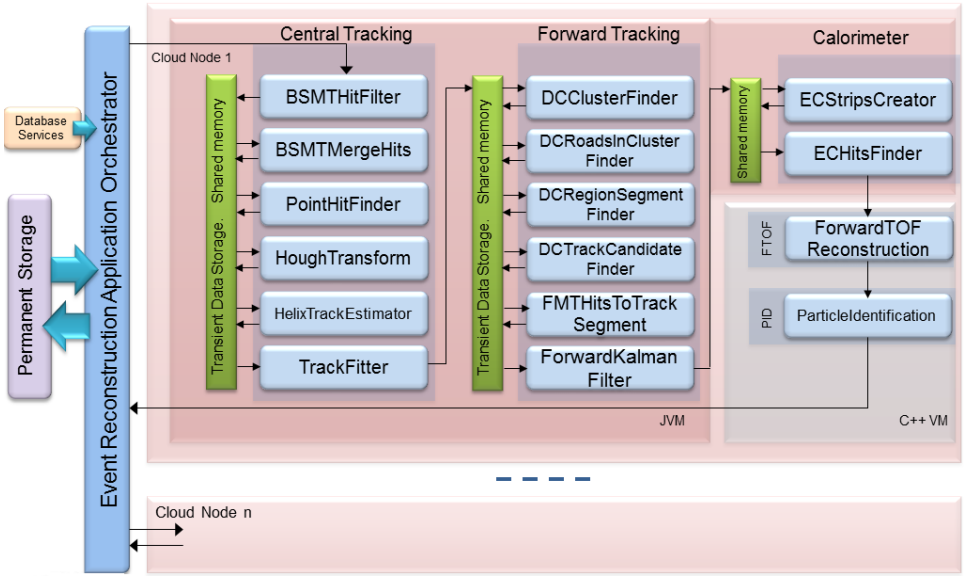


Figure 5: CLAS12 event reconstruction application.

Services are self-contained with no visible dependencies on other services. ClaRA services have temporal continuity. They are deployed once and are always available. The re-deployment of a service is not an expensive process and can be performed at runtime. For an application user or a designer the location of a service is transparent.

2.4.2 A New Programming Approach

A PDP application will be designed and composed using the available services from the ClaRA service inventory pool without requiring the knowledge of their programming details. The process of composition of a PDP application does not require programming skills and can be done graphically or by means of a script. During the process of design and composition, the focus isn't on the logical algorithmic flow of the application as in traditional programming, but rather on the data that gets passed between services. In this programming approach, the application design logic follows the data transformation across services.

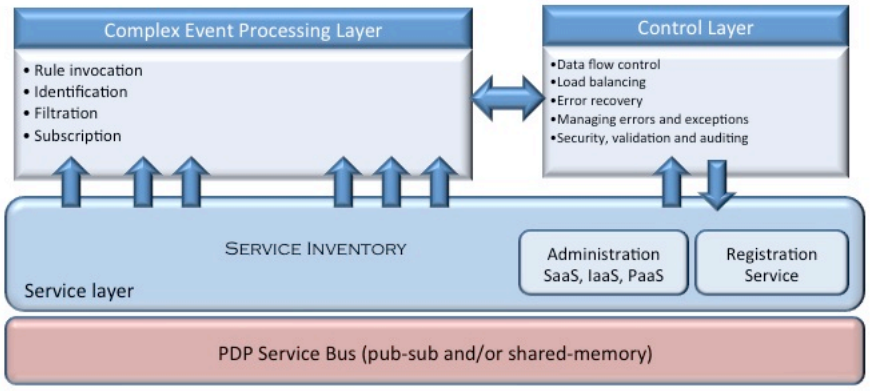


Figure 6: ClaRA design architecture.

2.4.3 The ClaRA Design Architecture

The ClaRA architecture consists of four layers (see Fig. 6). The first layer is the PDP service bus. This layer provides an abstraction of the cMsg publish subscribe messaging system [6]. Services or components from the orchestration or control layers communicate via this bus which acts as a messaging tunnel between communicating parties. Such an approach has the advantage of reducing the number of point-to-point connections between services required to allow services to communicate in the distributed ClaRA platform. By standardizing communication between services, adapting a PDP system to changes in one of its components becomes easier and simplifies the data transfer security implementation (for example by deploying a specialized access control service). The service layer houses the inventory of services used to build the PDP applications. An administrative registration service stores information about every registered service in the service layer, including address, description and operational details. The orchestration of service-based physics data analysis applications is accomplished with the help of an application controller component resident in the orchestration layer of the ClaRA architecture. Components from the orchestration layer are designed to subscribe and analyze event data in real-time. This enables immediate insightful responses to changing conditions in the active orchestrated service-based PDP. An orchestrated layer component can subscribe data from different (parallel running) services and/or service chains, which can make high-level decisions by

correlating multiple events (for example particle identification, triggers, etc.).

2.4.4 The ClaRA Distribution Environment

ClaRA can have one or many federated clouds. A ClaRA cloud that is also a ClaRA platform is a PaaS (Platform as a Service - a category of computing services that provide a computing platform as a service). Each ClaRA platform itself contains multiple ClaRA computing nodes as IaaS (Infrastructure as a Service). Each platform has a master node that runs administrative and registry services. Every cloud node has a master node that runs administrative and registry services. Every cloud node runs computing node monitoring services as well as local pub-sub servers for data exchange between services and communication across language barriers. Each ClaRA node has JAVA and C++ service containers. A container is a Virtual Machine (e.g. JVM for JAVA), which provides a complete runtime environment for ClaRA SaaS (Software as a Service) services, and allows several ClaRA services to run concurrently in the same environment (see Fig. 7). The ClaRA C++ service containers provide a runtime environment for C++ SaaS.

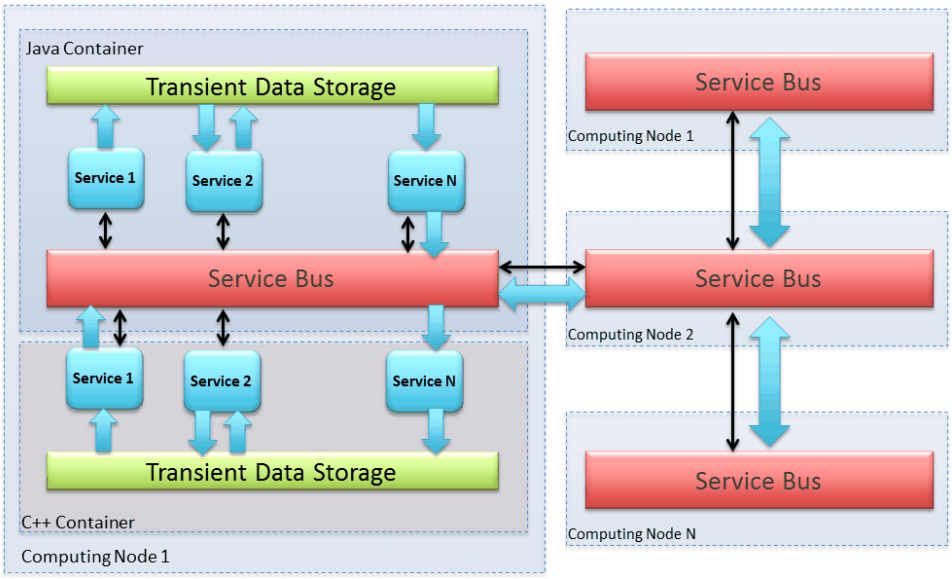


Figure 7: ClaRA cloud organization.

2.4.5 Performance

The track reconstruction application (SOT) services were deployed in ClaRA running on a 17 node Xeon 2×6 Westmere CPU cluster. The simulated dataset contained events with

at least one charged track in the CLAS12 detector. Performance measurements were performed as a function of the number of cores. The performance was tested by reconstructing from 1 up to 12 events at a time in a single ClaRA container utilizing up to 12 cores of a node. The result, shown in Fig. 8 indicates a linear dependence between the average processing time and the number of cores. Similarly, Fig. 9 represents the performance of the SOT application distributed over 17 ClaRA containers and shows that tracking performance scales linearly with the number of nodes.

Figure 8: Tracking performance in a single node cloud.

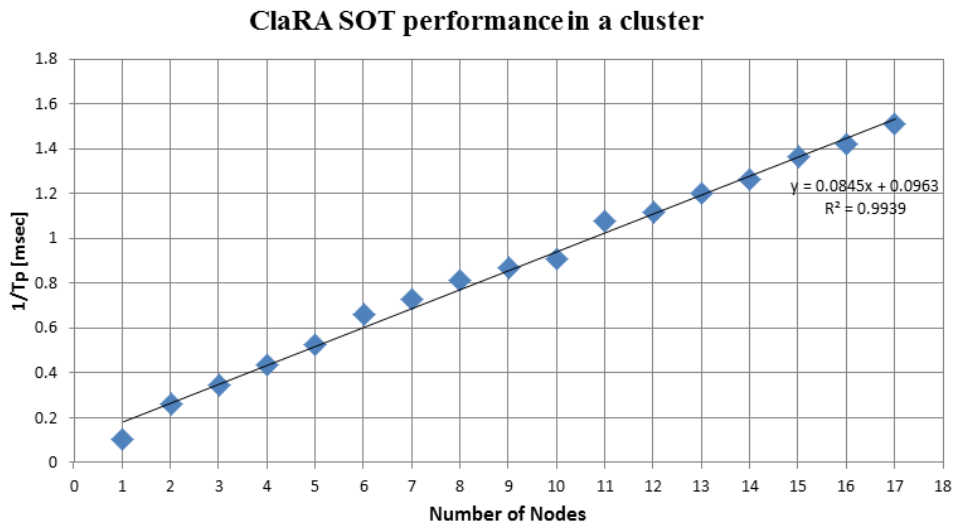


Figure 9: Tracking performance in a 17 node cloud.

2.5 Services

This section details the various services of the ClaRA framwork.

2.5.1 Detector Geometry

The Geometry service is an example of a detector data service. This is the front-end of the CLAS12 geometry data, encapsulating data access and data management details from the service consumers. All the ClaRA algorithm services, including simulation, reconstruction, alignment, and calibration will access the geometry service using standard service communication protocols, provided by the framework. Currently geometry information is stored in a mySQL database and will be part of the calibration database.

2.5.2 Event Display

The single event display in the CLAS12 framework will be both a service consumer (a client) and a service provider (see Fig. 10). Implementing the event display as a service consumer provides all the benefits discussed in the section on Service Oriented Architecture. Additionally it allows us to use the *thin client* model favored by modern software architects. *Thin clients*, sometimes called *smart clients*, split their application code between data models and processing, performed as much as possible in a different thread, process or CPU, and visualization. This shrinks the pure visualization code, allowing it to be deployed remotely or updated frequently without incurring a huge download penalty. The most familiar and successful such model is probably Google Earth. A thin client is downloaded and runs quickly, but complicated imagery processing is performed remotely.

An SOA is one way to implement thin clients. A visualization process such as acquiring detector geometry, is performed by a service and the thin client obtains the geometry by sending a request.

In the case of event display, the client uses a variety of services such as geometry, magnetic field, event streaming, file system access, running metadata, and analysis. A display service provides pictures of random on-line events that will be used by remote users for diagnostic purposes. Another more sophisticated service will be to answer a request for an image of a specific event viewed in a specified view (for example a zoomed view of region one in sector three of the drift chamber).

Another feature of the event display is that it employs a plug-in architecture based on the reflection capabilities of the JAVA language. This will provide a simple yet powerful extendibility feature for users who would like to use the event display to visualize and debug their new analysis and simulation code. Reflection allows JAVA applications to examine all the classes in their path. The event display looks for all classes that inherit from a specific abstract base class. Once found, the application creates an object from that class and then provides a set of services for the object, such as notifying it that a new event has arrived.

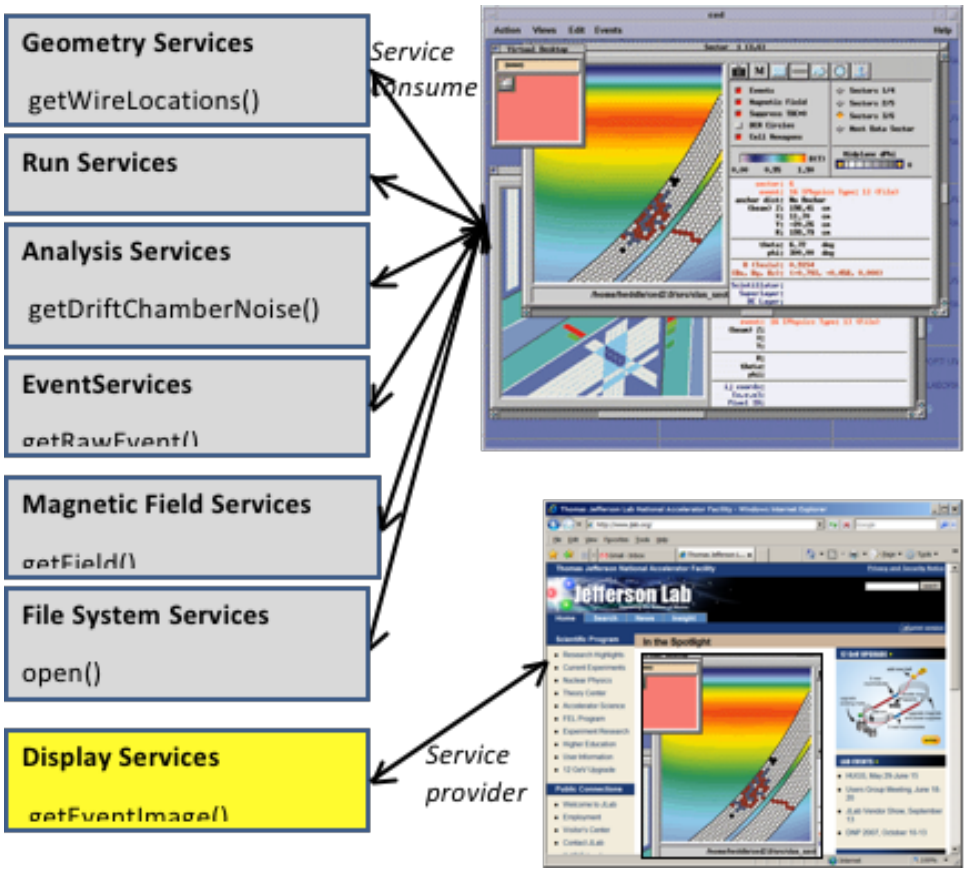


Figure 10: The event display consumes a number of CLAS12 services. It will also provide a display service.

In this scheme, no recompilation or restart of the event display is required. The developer extending the event display creates the class, drops it into the path, and the class will be automatically plugged into the application. If a particular plug-in proves of general use, it can be placed in the path of the shared event display (e.g., a run in the counting house) so that all users can access it. On the other hand a class file that posseses undersirable or buggy features can simply be deleted. Adding and removing features will occur with no change to the code or recompilation of the base event display.

2.5.3 Data and Algorithm Services

The CLAS12 software design separates data and algorithm services. An algorithm service, in general, will accept and process an output data object from a data service and will then produce a new data object.

Data objects are resident on the disk, whereas data services manipulate the data objects in memory. The algorithm services should be independent of the technology used for data object persistency. This will allow the future replacement of persistency technology without affecting the user-produced algorithm services. The separation of persistent and transient data services is aimed at achieving a higher level of optimization by targeting inherently different optimization criteria for persistent and transient data storages. For example, the optimization should target I/O performance, data size, avoid multiple I/O requests, etc., for data objects on the disk, and execution performance, API and usage simplicity, etc., for transient data in memory.

We foresee three major categories of data objects:

- Event data, such as raw, simulated, or reconstructed data.
- Detector data, describing a detector apparatus needed to interpret the event data. Examples of detector data are geometry, calibration, alignment, and slow control data.
- Statistical data, such as histograms, and n-tuples.

Specific data services are provided for each of these categories (see Fig. 11).

2.5.4 Calibration Services

Calibration services are currently being written for the FTOF, EC and PCAL detectors.

The calibration services for the EC and PCAL will be based on existing legacy code that utilizes cosmic muon data for online calibration and physics data for offline calibration monitoring and final adjustments[1]. The present intention is to perform energy-weighted

[1]It is observed that minimum ionizing muons for instance have an energy deposition profile that is uniform and *localizable* as opposed to electromagnetic showers that have a non-uniform energy deposition function.

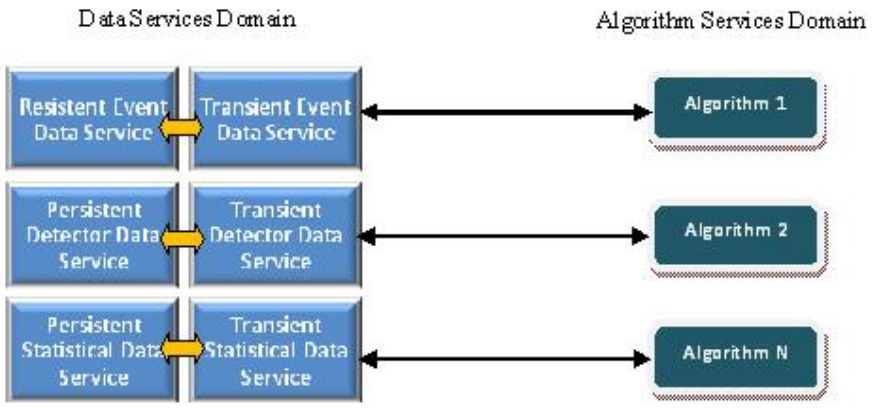


Figure 11: The data services of the ClaRA framework. Algorithm services deal with transient data services only.

cluster finding in the EC/PCAL trigger front-end electronics, which will require determination of constants for the trigger firmware, consistency of calibrations of the EC and PCAL, and the online monitoring of the constants. The use of cosmic muons runs prior to beam turn-on, which do not require sophisticated hit reconstruction or pre-existing calibrations, will allow hardware gain-matching of PMTs and determination of attenuation corrections needed for uniformity of trigger response. The online live monitoring of cosmic events will permit quick diagnosis of miscalibrated or malfunctioning PMTs. While refinements of the online muon calibration are possible offline, it has been shown adequate to meet basis physics analysis requirements.

Offline monitoring of PMT gains and adjustments of the energy calibration will make use of all physics data: MIP pions with $p > 0.6$ GeV/c, two-photon events for π^0 invariant mass, run-by-run PMT gain monitoring using electron E/p and cross-checks of the online cosmic muon calibrations. The EC calibration algorithms have been used for 15 years with good results and are currently being further developed using cosmic testing of the PCAL modules as they are built. Their future developments include a possible rewrite of the algorithms in JAVA and integration into the new calibration database and slow controls services. All calibration services are developed to be multi-threaded and fully integrated within the ClaRA framework.

The FTOF calibration suite currently reads data in EVIO format and performs automated calibrations on ADC pedestals and TDC offsets. This provides read-back histograms and editable tables of constants; the calibration tables and monitoring histograms are then saved to disk.

The primary purpose of the FTOF is to provide the necessary timing ingredients for

particle identification in the forward detector. The raw output data provided by the FTOF will be recorded as channels from ADC and TDC for PMTs attached to the scintillator bars. This means that before basic particle identification can be achieved, an intermediate bank of timing information must be combined with tracking information and this timing bank must provide the time of particle interaction at the paddle along with the paddle ID/sector. The time of particle interaction at the paddle is calculated simply by taking the average of the times at the PMTs, adjusted for signal propagation time through the scintillator material. The coupling of a scintillator paddle hit and the corresponding track that produced the hit is achieved by "swimming" the track (through a tracking service) to an FTOF scintillator bar identified through the CLAS12 geometry database. This intermediate FTOF timing bank is already defined and in use in the Service Oriented Tracking (SOT) package as well as the simulation package GEMC.

The secondary purpose of the FTOF is to provide energy deposition information calculated from FTOF ADC signals. This particular conversion has yet to be written for CLAS12 specifically, but will follow the standard procedure that was used in CLAS. (In fact, the panel-1a scintillator bars will be salvaged directly from the CLAS TOF detector). It should be noted that the expected energy deposition for a track interacting with an FTOF scintillator bar is currently simulated in GEMC.

The CLAS12 calibration suite is a JAVA-based calibration program that is being written as a platform for calibrating the many sub-systems of CLAS12. The design goals are to produce an agile and modular software system that is fully ClaRA integrated and both user and programmer friendly. These goals reflect the clear advantages to future users by having a single unified and well maintained program for calibration of all sub-systems, as opposed to the common and unfortunate practice of many different subsystem calibration programs being written by a variety of authors in different software languages and then often being left unmaintained with the software source hard to find. The program structure has been kept as simple and lightweight as possible: An orchestrator service acts as a bridge between the main GUI and the calibration parent-containers. The parent-containers are modular groups of calibration services that can be written and plugged directly into the orchestrator with relative ease and minimal changes to the orchestrator or GUI code.

The Calibration Suite accepts EVIO input through the JEvio reader service. The suite can be run "offline", analyzing a predetermined data set, or it can be run "online" continuously accepting data and analyzing it when the user requests. The GUI provides a histogram readout specific for each calibration type which allows the user to see the quality of the performed calibrations. To display histograms, our program uses the histogram libraries from AIDA's freeHEP project, a JAVA-based and open sourced high energy physics toolset maintained at SLAC. The output from the calibration routines is also displayed in an in-GUI table which allows the user to inspect and alter the calibration results. The user can then push the results to disk in an ASCII format file, or send the results directly to a calibration database through the CLAS12 database service.

The first set of calibration algorithms has been implemented for the FTOF, allowing

the Calibration Suite to be bug-tested successfully using real CLAS data. The current goal for the Calibration Suite is to have complete integration into the ClaRA software framework. Version 1.0 (non-ClaRA) is now frozen while the ClaRA-ready Version 2.0 is developed. A set of tools is also being developed to help assist the user in flagging and managing problematic calibration results.

2.5.5 The Magnetic Field Service

The magnetic field service is implemented in C++. It makes extensive use of the standard library's map construct. The entire field map is divided into individual maps corresponding to CLAS12 magnets, which comprise the solenoid and main torus. Each of these maps inherit all capabilities of `std::map`. In addition, it has methods to get the magnetic field at a certain point, check for consistency within the map, and interpolate values inside the defined grid spacing.

Each map is tailored to the field it holds. For instance, the main torus is defined in cylindrical coordinates, with $\phi \in [0, \frac{\pi}{6}]$. The class holding the main torus map has an algorithm to calculate the field at any point $\phi \in [0, 2\pi]$. However, this is only good for ideal fields. For measured fields, the class can be easily extended to handle a case where the field is known precisely in the entire region. Furthermore, the dimensions and coordinates of the map's position and field need not be same. Inside the solenoid field for instance, the position is stored in (r, ϕ) while the magnetic field is stored in (x, y, z).

A separate *mother* class is responsible for loading in and storing the maps from a database or file. This class is aware of the volumes of the individual maps and sums the fields where appropriate.

The final layer on this system is the magnetic field service. This is where the mother class is initialized and held in memory. Several mother classes can be held; i.e. one for the ideal fields, one for the measured fields, and one mix of these two. The service registers itself with the ClaRA system and can provide the various field maps in several formats depending on the consumer's preference. As an example, the service can be polled for an entire map or for an individual position.

2.6 The CLAS12 Calibration Constants Database

The normal operating procedure for CLAS12 experiments will be a series of short "runs" which will contain data where the configuration is reasonably constant throughout. For reconstruction and simulation purposes, there will be several sets of constants kept on a run-by-run basis including the geometry and calibration parameters. To this end, a database design was created for the GlueX experiment in Hall D (called "CCDB") and a version was implemented for CLAS12. This database is intended to store all constants for geometry and calibration, and possibly the magnetic field maps as well.

Each set of constants is identified by the combination of run number, variation and time. The variation is a string name or tag to allow several versions of the same constants to exists for the same run. The time in this case is the time the constants were added to the database. A running history of changes made to the database is kept and nothing is ever deleted. Also, one could substitute the time the data was taken for run number.

Within the ClaRA framework, the CCDB software comes with an interface in C++, JAVA and Python. Constants are obtained by contacting the appropriate service, either geometry or calibration. The geometry service provides the sets of parameters that are needed by the simulation (GEMC), track reconstruction (SOT) or other calibration services. These parameters are all derived from a unique set of detector-specific "core" numbers. The calibration service is similar, though generally no intermediate calculations are needed and it acts as a simple interface to the numbers in the database.

3 Simulation

3.1 Introduction

A parametric Monte Carlo and a GEANT3-based model of CLAS12, described below, were developed in order to validate design decisions for the upgraded detector. At the same time we started to build a modern simulation package able to support the engineering project and meant to be used for the whole lifetime of the CLAS12 program. The result of this effort is an entirely new object-oriented design C++ framework called GEMC.

3.1.1 Parametric Monte Carlo

One of the fundamental algorithmic challenges in the design of CLAS12 is the problem of track reconstruction in a non-uniform magnetic field. Not only does the torus produce an inhomogeneous field in the tracking volume, but charged particles emerging from the solenoid must be tracked as they traverse the fringe field of that magnet. Since no analytic form for the particle trajectories exist, they must be calculated by "swimming" the particles numerically through a map of the magnetic field. Track fitting then becomes very expensive in terms of CPU time. One way to finesse the problem is to linearize it by parameterizing the trajectory as small deviations from a reference trajectory. The reference trajectory must come from a "swim", but subsequent "trial" trajectories, with different starting parameters (momentum, direction), can be computed by a simple matrix inversion. Position resolution is put in at a set of idealized detector planes. It is also possible to incorporate multiple Coulomb scattering in this model. This technique has already been used to estimate momentum resolution for CLAS12. Results appear in other sections of this document. The method cannot give information on some things, such as the effect of accidentals, track reconstruction efficiency or confusion due to overlapping tracks.

3.1.2 CLAS Software with CLAS12 Geometry

The current CLAS system consists of over a half a million lines of FORTRAN, C and C++ code contained in about 2,500 source code files. It represents a large investment by the CLAS collaboration over many years. CLAS, with its toroidal magnetic field also presents the difficulty of tracking in a non-homogeneous field and that problem has been solved in this body of code. Recently, the geometry of crucial detector elements was changed to reflect the CLAS12 design, both in simulation and in reconstruction. The resulting system can now do a full GEANT3-based simulation and reconstruction of CLAS12 events, in particular charged particle tracking in the forward drift chambers. Studies using this system have been carried out to verify momentum resolution results from the parametric Monte Carlo and to estimate the effect of accidental Møller scattering background on track reconstruction. More of the details of the detector subsystems and beam line components

of the upgraded configuration are being added to extend the range of these and similar studies.

3.2 GEANT4 Monte Carlo: GEMC

GEMC is a software framework that interfaces with the GEANT4 libraries. It is driven by two design principles:

1. Use object oriented design that includes the C++ Standard Template Library. The GEANT4 solid, logical physical volumes, magnetic fields, sensitivities, etc., are abstracted into general classes that can be built from the database of user choice. For example, one can use MySQL, GDML or ClaRA to define the geometry.

2. All simulation parameters must be stored in a database external to the code. In other words, there are no hardcoded numbers in GEMC. In fact GEMC is agnostic on the detector: choosing what configuration to use is a matter of choosing what database to use, a choice that can be done trivially at run time.

3.2.1 Main GEMC Features

- The users do not need to know the C++ programming language or the GEANT4 interface to build detectors. A simple API takes care of the database I/O. This allows users to concentrate on making the geometry as realistic as possible.

- Upon database upload (which is an instantaneous MySQL table upload) the geometry is available to all GEMC users - literally across the globe - without having to recompile the code or even download files.

- Many additional parameters are selectable at run time: displacements, step size, magnetic field scaling, physics list, beam luminosity etc. This is done by either command-line options or a configuration file called "gcard."

- Rigorous object oriented implementation allows code flexibility and painless, simple debugging.

3.2.2 GEMC Detector Geometry

The GEANT4 volumes are defined as follows:

- Shapes, dimensions, boolean operations of shapes.

- Material, magnetic field, visual attributes, identity, sensitivity and hit process.

- Placement(s) in space: position, rotation, copy number.

These parameters are stored in MySQL tables, one table per detector (i.e. HTCC, EC, DC, etc). Below is an example of the API that loads the parameters into the database:

```
for(my $n=1; $n<=$NUM_BARS; $n++)
{
    # element name, mother, description
    $detector{"name"}        = "CTOF_Paddle_$pnumber";
    $detector{"mother"}      = "CTOF" ;
    $detector{"description"} = "Central TOF Scintillator $n";

    # positioning, rotation, color
    $detector{"pos"}        = "$x*cm $y*cm $z*cm";
    $detector{"rotation"}   = "90*deg $theta2*deg 0*deg";
    $detector{"color"}      = "66bbff";

    # Solid type, dimension, materials, style
    $detector{"type"}       = "Trd";
    $detector{"dimensions"} = "$dx1*cm $dx2*cm $dy*cm $dy*cm $dz*cm";
    $detector{"material"}   = "Scintillator";
    $detector{"style"}      = 1;

    # Magnetic Field
    $detector{"mfield"}     = "clas12-solenoid";

    # Sensitivity: This will associate the volume to the Sensitive Detector
    $detector{"sensitivity"} = "CTOF";

    # Hit Process: This will associate the volume to the Hit Process routine
    $detector{"hit_type"}    = "CTOF";
}
```

At run time, GEMC reads the gcard, an XML file that specifies which detector to include in the simulation, including possible tilts and displacements from the original positions. An example of gcard syntax is given below:

```
<sqltable name="LH2target"/>    Includes the Liquid Hidrogen Target
<sqltable name="BST"/>          Includes the Barrel Silicon Vertex Tracker
<sqltable name="FST"/>          Includes the Forward Silicon Vertex Tracker
<sqltable name="CTOF"/>         Includes the Central TOF
<sqltable name="beamline"/>     Includes the beamline
<sqltable name="DC"/>           Includes the Drift Chambers

<detector name="BST">
 <position x="0*cm"  y="0.1*cm" z="0*cm" />  Displaces the BST by 1 mm in the Y axis
 <rotation x="1*deg" y="0*deg"  z="0*deg"/>  Tilts the BST by 1 degree around the X axis
</detector>
```

3.2.3 CLAS12 Geometry Implementation

Particular attention is paid in implementing the geometry details of each detector with adequate accuracy. In Fig. 12 the SVT GEMC representation is shown, while in Fig. 13 is shown the GEMC implementation of the various central detectors. A multiple track event in the central and forward part of CLAS12 can be seen in Fig. 14.

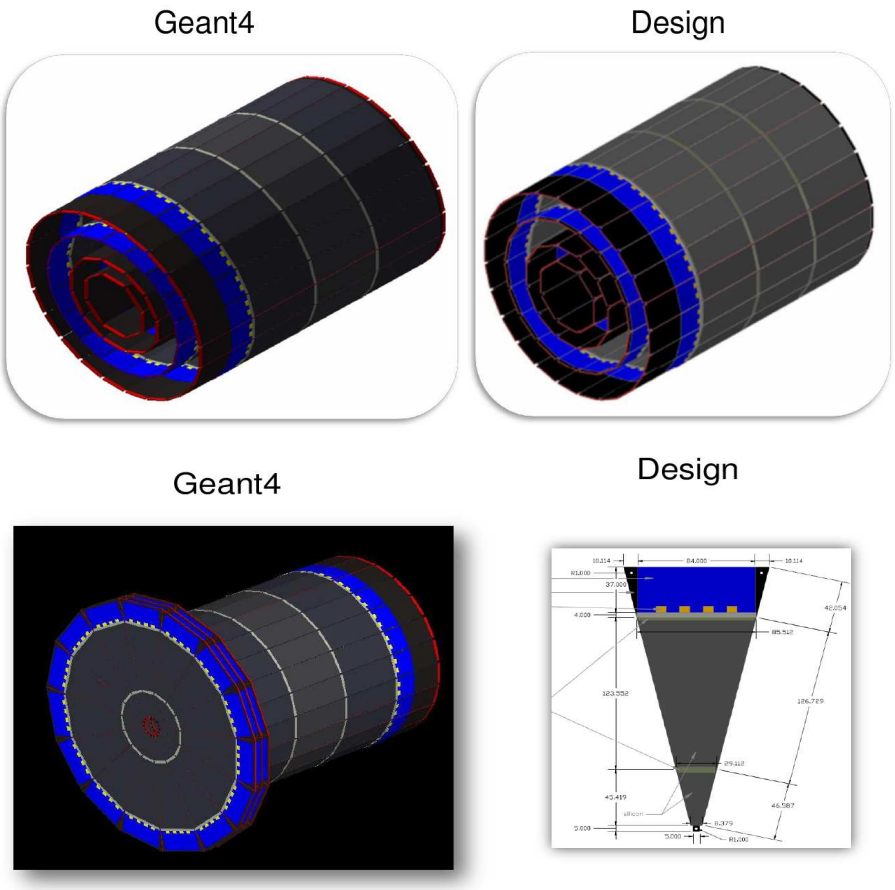


Figure 12: The Silicon Vertex Tracker in GEANT4. Upper left: the GEMC BST. Upper right: the CAD model. Lower Left: the complete GEMC implementation of the BST+FST. Lower right: an FST module. All components (including supports, wirebonds and chips) and dimensions in GEMC reproduce exactly the design.

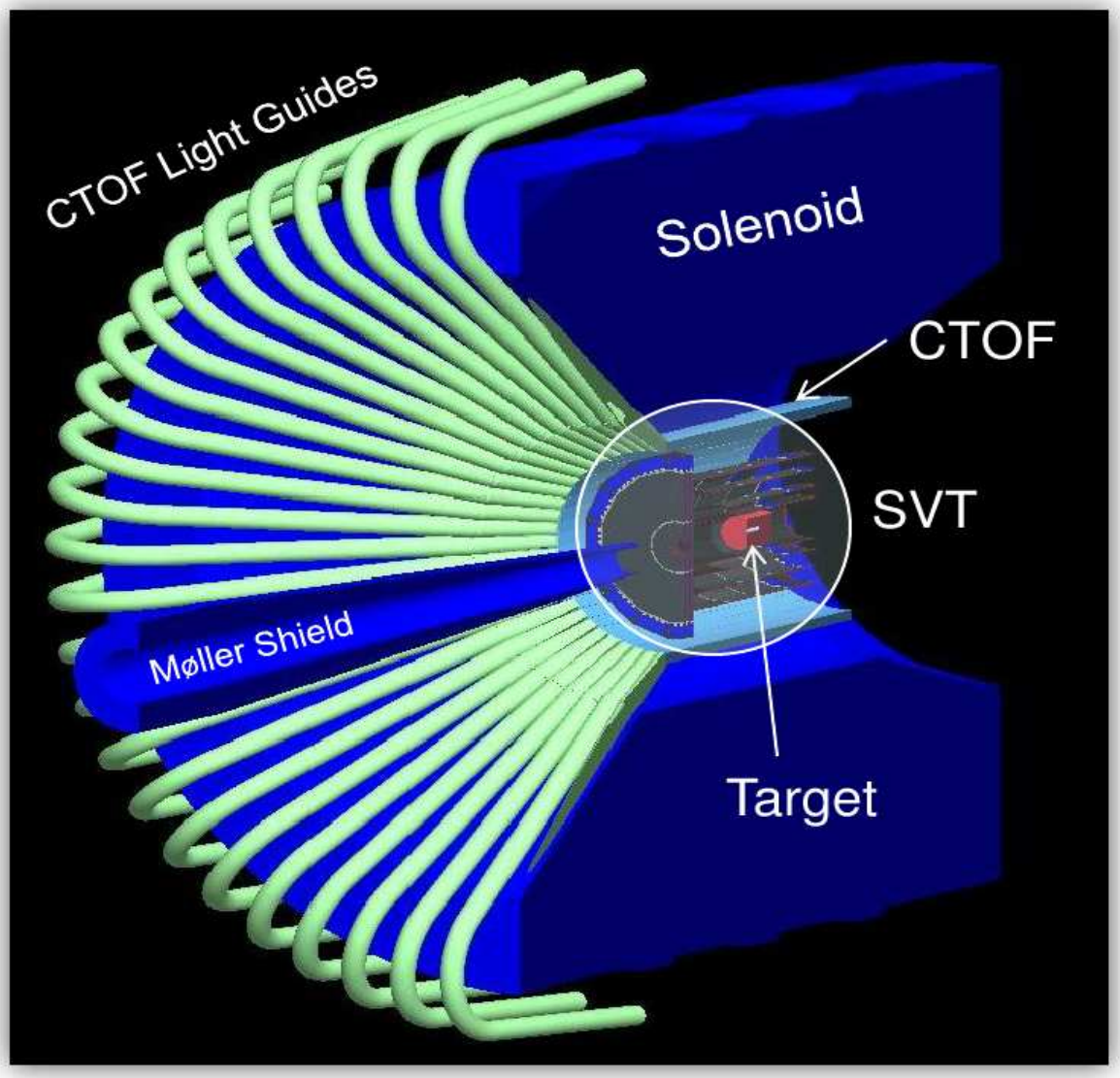


Figure 13: The CLAS12 Central Detector. The target (white) is at the CLAS12 center, surrounded by the scattering chambers (red). The BST and FST constitute the silicon vertex tracker. The CTOF paddles (cyan) are connected to light guides (light green) that wrap around the Solenoid (blue). The Möller shield is also visible (blue).

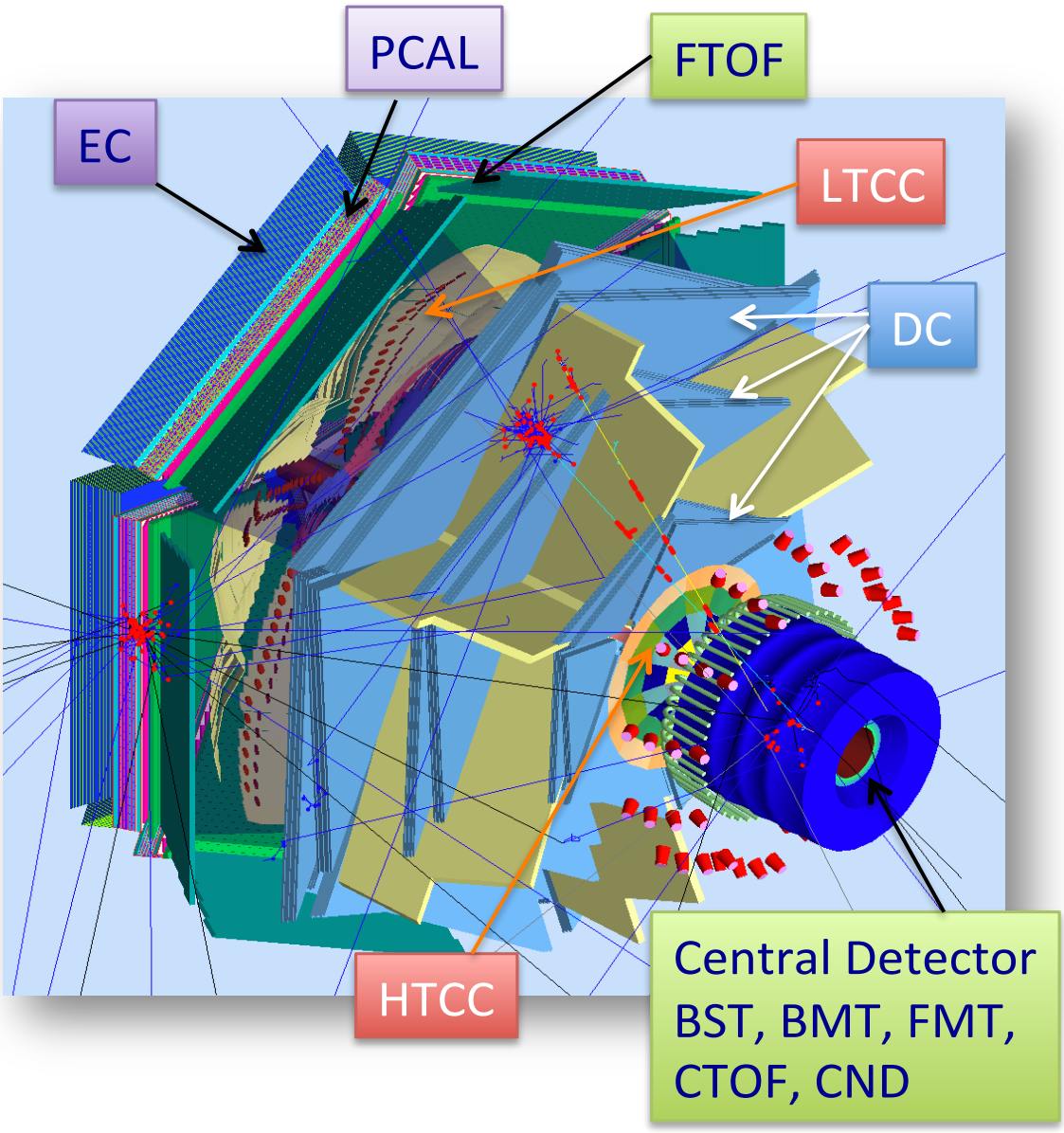


Figure 14: The CLAS12 central and forward detectors. In the forward part: the HTCC, 3 regions of drift chambers (DC), the LTCC, three panels of time-of-flight and the two electro-magnetic calorimeters PCAL and EC. Two simulated tracks produced hits (in red) in the various detector. Photons are the blue straight tracks.

3.2.4 Primary Generator

In GEMC there are two ways to define the primary event.

1. `GEMC internal generator` (command line or `gcard`): with this method the user defines the primary particle type, momentum range, vertex range. Example:

```
BEAM_P="proton, 0.8*GeV, 80*deg, 10*deg"    (particle type, momentum, theta and phi)
SPREAD_P="0.2*GeV, 40*deg, 40*deg"          (momentum, theta, phi ranges)

BEAM_V="(0.0, 0.0, -10.0)cm"        (x,y,z) of the primary vertex
SPREAD_V="(0.1, 2.5)cm"             (z, radius ranges) i=for the primary vertex

results in:

Primary Particle:  Proton
momentum: 800 +- 200 MeV
theta: 80 +- 40 deg
Phi: 10 +- 40 deg
Vertex: ( +- 1 mm , +- 1mm +- 2.5cm)
```

2. `external input file`: with this method the user defines (command line or `gcard`) the format of the input file and the file name. Example:

```
INPUT_GEN_FILE="LUND, dvcs.dat"    (LUND file format, filename:  dvcs.dat)
```

The various file formats are registered in GEMC by a *factory method* that allows the user to derive new formats from the GEMC C++ pure virtual methods defined for the input and to choose the desired format at run-time.

3.2.5 Beam(s) Generator

In addition to the primary particles, two additional *luminosity beams* can be defined to add realistic background to the simulation in the form of N beam particles per event. The user defines (command line or `gcard`) the beam particle type, the number of beam particles per event and the time structure of the beam. Example:

```
LUMI_P="e-, 11*GeV, 0, 0"         Beam Particles: 11 GeV e- along z-axis
LUMI_V="(0, 0, -10)cm"            Beam vertex is at z=-10 cm
LUMI_EVENT="60000, 124*ns, 2*ns"  60,000 particles per event, spread in 124 ns, 2ns per bunch
```

3.2.6 Hit Definition

The GEMC hit definition is illustrated in Fig. 15. A `Time Window` (TW) is associated with each sensitive detector. In the same detector element, track signals within the TW form one hit, while tracks separated in time by more than the TW will result in multiple hits.

An energy sharing mechanism is in place as well. For each sensitive detector element users can decide how much of the energy deposited will be detected in that element. Users can also decide whether a sensitive element triggers other sensitive elements even if they are not directly touched by a track, and how much energy is shared between the elements. A good example of this is the charge sharing mechanism in strips in a silicon detector.

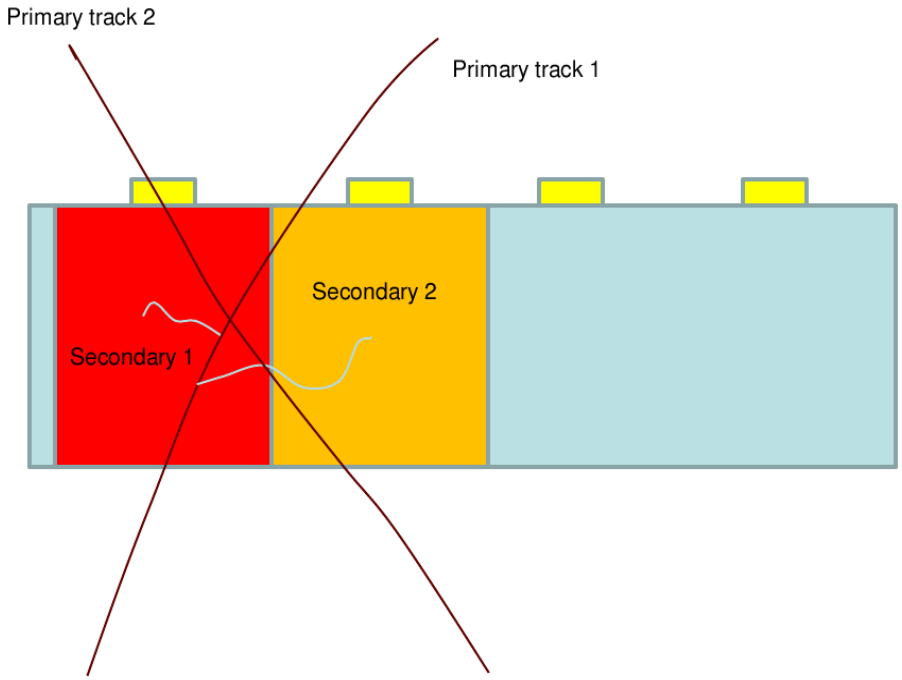


Figure 15: Hit definition illustration: In the picture two different detector elements are shown in different colors (red and yellos). All tracks within the same TW and the same cell constitute one hit for that cell. If any track has enough time separation from an existing hit, it will form another, separate hit.

3.2.7 Hit Process Factory

Each detector has a user-defined Hit Process Routine (HPR) associated with it, derived from an abstract class with pure virtual methods. All HPRs are registered and loaded at run-time by a factory method.

The input to all HPRs is a `GEMC hit`. This stores, for each step in the hit, the following information:

- Hit position (global coordinates)
- Hit position (relative to the volume in which the step occurs)
- Deposited energy
- Time of the hit
- Momentum of the track
- Energy of the track
- Primary vertex of the track
- Particle ID
- G4Track ID
- Identity
- Detector hit
- Mother particle ID
- Mother G4Track ID
- Mother primary vertex of the track
- Energy threshold of the sensitive detector

Each HPR processes the `GEMC hit` and produces Standard Template Library (STL) vectors of double (raw informations) and integers (digitized informations). Each STL vector corresponds to a MySQL entry in the bank table corresponding to the HPR.

Below are three examples of such database entries.

- A partial list of variables used to store "raw" information, usually common to all detectors:

name	index	comment
ETot	1	Total Energy Deposited
<x>	2	Average x position
<y>	3	Average y position
<z>	4	Average z position
<t>	8	Average time
pid	9	Particle ID
E	13	Energy of the track at the entrance point

- A partial list of variables used to store DC information:

name	index	comment
LR	18	Left/Right: -1 (right) if the track is below the wire
doca	19	distance of closest approach
sdoca	20	smeared doca
time1	21	doca / (50 um/ns)
stime1	22	sdoca / (50 um/ns)
sector	23	CLAS12 Sector
superlayer	24	DC Superlayer
layer	25	DC Layer
wire	26	DC Wire

- A partial list of variables used to store BST information:

name	index	comment
layer	20	BST layer
sector	21	BST sector
strip	22	BST strip

3.2.8 Elements Identity

In order to correctly identify and process the correct detector element at run-time, the class `identifier` is used. The following scenarios can happen:

1. For detectors where each element corresponds to a unique volume, the identifier is defined in the geometry. An example is the CTOF:

```
element          identifier
paddle 4              4
paddle 16            16
```

2. For detectors where each element corresponds to a unique volume that is copied, the identifier copy number is determined at run-time. An example is the FTOF, where each sector is copied:

```
     element           identifier: sector copy is determined at run time.
paddle 4,  sector 3                          4, 3
paddle 16, sector 5                         16, 5
```

3. For detectors where different elements exist within the same GEANT4 volume, the identifier is processed at run-time by the identifier method of the Hit Process Routines. For example in the drift chambers implementation, the single cells are not individual GEANT4 volumes but part of layers of gas volumes. At run-time, the cell is identified by the HPR based on the track position in each layer:

```
           element                identifier: sector, wire are determined at run-time
wire 52, layer 3, SL 2, sector 4                     52, 3, 2, 4
wire 87, layer 4, SL 5, sector 1                     87, 4, 5, 1
```

4. For detectors where one element will share energy with other elements, the identifier is processed at run-time by the identifier method of the Hit Process Routines and will return a vector of elements with the percentage of energy shared in each element. Example:

```
element                identifier vector with percentage of shared energy
strip 12                      12 94    11 3    13 3
```

3.2.9 Output

The file formats for the simulation output stream are registered in GEMC by a factory method. New files types can be derived from the C++ pure virtual methods defined in GEMC. The main registered formats, selectable at run-time, are:

- txt: readable from any editor or shell. Bank names and variables are printed out.
- EVIO: this is the format used by the CODA system chosen to be the default CLAS12 format for the output stream. Banks and variables are identified by integers (called tag and num) defined in the MySQL tables.

The EVIO output can be parsed with the utility evio2xml, that outputs event in XML format. As illustration, an example of an EVIO event dumped with evio2xml:

```
<event_n>   3  </event_n>

<particle_generator>
 <generated_particle_1>
   2212 1.0+03 0.0 1.7321e+03  - particle id, 3-momentum
   0.0  0.0  0.0               - vertex
 </generated_particle_1>
</particle_generator>

<DC>    DC has 3 hits, so each variables has 3 entries
  <sector>      2              2              2   </sector>
  <SuperLayer>  3              3              3   </SuperLayer>
  <Layer>       1              2              3   </Layer>
  <Wire>       71             70             71   </Wire>
  <Edep>   1.4498e-04   1.1808e-03   1.0493e-03   </Edep>
</DC>
```

3.2.10 Magnetic Fields

The magnetic fields parameters are stored in a MySQL database. They are defined by their name (unique ID), and contain their definitions (including map file format), symmetry (if any) and swim method. Example:

```
name: clas12-torus
data type: mapped txt
symmetry: phi-segmented
filename (if map) clas12_torus_fieldmap.dat
n points, range, units in each dimention: 61 0 30 deg   126 0 500 cm    126 100 600 cm
shift from (0,0,0): 0 0 0 cm
field units: kilogauss
swim method: RungeKutta
description: CLAS12 Torus MAP
```

3.2.11 Parameters Factory

In order to avoid hardcoded numbers in the various software components (simulation, reconstruction, event display, analysis, etc) some "mother numbers" parameters are stored in a MySQL database. These parameters can be accessed by the GEMC geometry API to build the GEANT4 volumes via hash map "parameters":

```
my $NUM_BARS = $parameters{"ctof_number_of_bars"};
```

They can also be accessed by the hit-process routines to determine the identifiers and to perform the digitization via an STL map <string, double> "gpars":

```
double ec_tdc_time_to_channel = gpars["EC/ec_tdc_time_to_channel"];
```

3.2.12 Material Factory

The GEANT4 materials definitions, including optical properties, have been abstracted to a material factory. At the command line users can decide to load the materials definitions from the traditional GEANT4 C++ implementation or from a MySQL database. A third option will allow for GDML definitions. Example of MySQL definition:

```
BusCable  |  1200 | 2  | G4_Cu 24 G4_POLYETHYLENE 76
```

defines the material BusCable with density 1.2 g/cm^3, composed of 2 materials: copper (24%) and polyethylene (76%).

3.2.13 Geometry Factory

The current implementation of GEMC builds the GEANT4 solids, logical and physical volumes, and sensitive detectors from a MySQL database. The "detector" class will be abstracted to a factory to expand its input to:

- Traditional C++ GEANT4 detector construction
- MySQL DB (current implementation)
- ClaRA: a service will provide detector constructions
- GDML Format: a GEANT4 standard XML syntax, extended for sensitivity and output format.
- HDDS Format: XML syntax, extended for sensitivity and output format.

3.2.14 Results

GEMC has been used extensively to validate and assist the CLAS12 engineering design. In Fig. 16 an example of one event at 1/10 of the nominal CLAS12 luminosity in the central detector is shown. Full luminosity studies have been crucial in maximizing shield design, minimizing the background radiation dose to the most sensitive detector elements and in choosing the optimal energy threshold for the various electronics. In Fig. 17 proton acceptances and threshold rejection factor studies are shown.

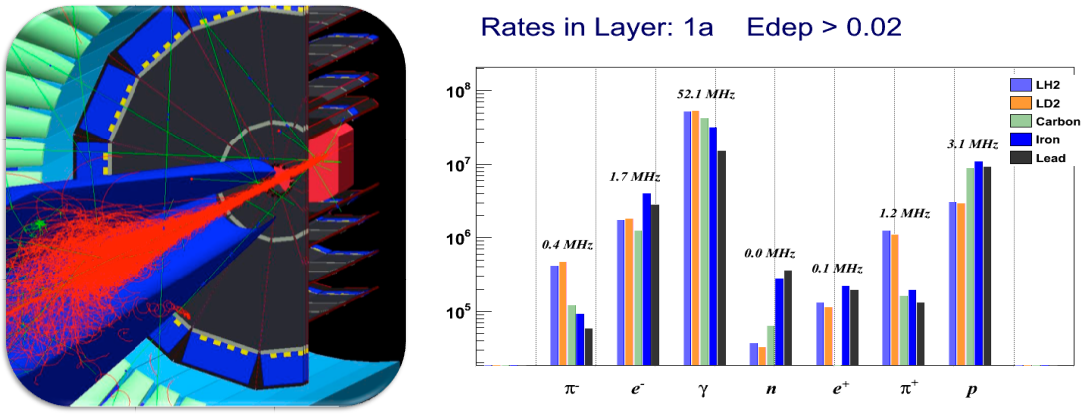


Figure 16: Left: Background simulation in the central detector. An electron beam corresponding to $10^{34}\text{cm}^{-2}\text{s}^{-1}$ (1/10 of CLAS12 luminosity) is sent to a 5 cm liquid hydrogen target. Secondary interactions include Möller electrons and photonuclear hadronic production. The Möller (red lines) are the main source of background and are aligned by the 5 T solenoid field near the target, but will spread to a "sausage" after the forward tracker. A carefully designed tungsten shield (in blue in the picture) will contain most of them. Right: background rates for different targets.

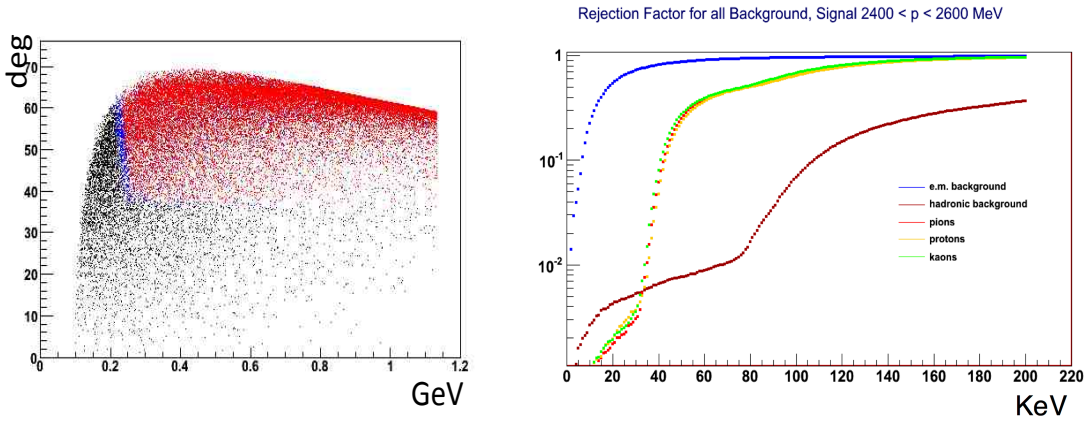


Figure 17: Left: Proton acceptance calculation in the BST as a function of their momenta and angles. The generated events are in black; three (four) BST layers required in reconstruction are in blue (red). Right: the rejection factor as a function of threshold energy cut. One can see that at about 30 keV most of the e.m. background is kept while at the same time rejecting only 0.5% of tracks of interest.

3.2.15 Documentation

The main portal to the GEMC documentation is the JLab website:

`http://gemc.jlab.org`

Various how-tos, tutorials, quick guides, and step by step installation guides can be found on the website.

The C++ code is documented with doxygen. Classes and methods are defined with inline comments and described in detail. The doxygen documentation is generated nightly. The documentation can be found at:

`http://clasweb.jlab.org/clas12/gemc_doxygen`

A mailing list

`gemc_software@jlab.org`

is used as the main channel for news and communication.

3.2.16 Bug Report

GEMC uses Mantis [7], a web-based bug-tracking system. Mantis is written in the PHP scripting language and works with MySQL. Mantis can be browsed from most computers and phones. Bugs are tracked, solutions and debugging are logged, and all the conversation is automatically sent by email to the interested parties. The JLab GEMC Mantis Bug Report system can be found at:

`https://clasweb.jlab.org/mantisbt`

Two screenshots of the website can been seen in Fig. 18.

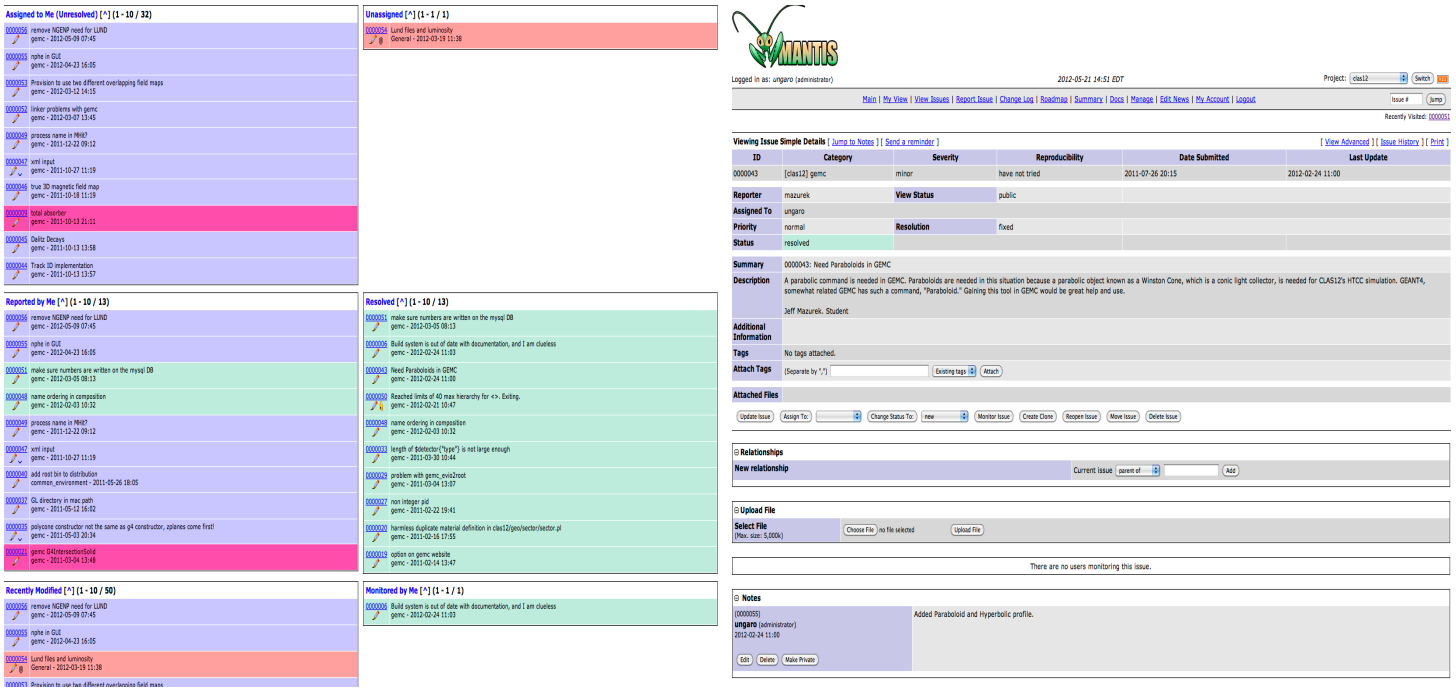


Figure 18: Left: Screenshot of the GEMC Mantis Bug Report / Feature Request. Right: A screenshot of one resolved issue. Email communication is sent to the proper party and archived.

3.2.17 Project Management, Code Distribution and Validation

The software is released on the subversion repository, and it is maintained and tested across several platforms:

- Darwin macosx 10.7 i386 gcc 4.2.1
- Darwin macosx 10.6 x86_64 gcc 4.2.1
- Darwin macosx 10.7 x86_64 gcc 4.2.1
- Linux CentOS 5.3 i686 gcc 4.1.2
- Linux CentOS 5.3 x86_64 gcc 4.1.2
- Linux RHEL5 i686 gcc 4.1.2
- Linux RHEL6 i686 gcc 4.4.6
- Linux Fedora15 x86_64-gcc4.6

At JLab, GEMC and its dependencies (CLHEP, XERCESC, QT, GEANT4, MySQL) are installed and available on the CUE machines (RHEL 5 and 6, CentOS 5) and on the JLab farm (CeontOS 5). JLab users have been using these distributions since 2008.

M. Ungaro, the author of the code, supervises all aspects of development and has in place code validation and testing. Cron jobs runs periodically (from as often as 15 minutes to daily) to check the geometry, parameters and material databases, code compilation, etc. For example, if a change is introduced that will cause compilation errors for a particular platform, his pager will alarm. This system will be expanded in the summer of 2012.

Physics events labelled by the GEMC revision are kept on JLab disks. Scripts that test significant quantities (number of particles and hits on specific detectors) will run nightly as a code quality check.

3.2.18 Project Timeline

The project timeline uses the Omniplan software. Resources are allocated and projects are assigned with duration time. The timeline can be found on the GEMC webpage as a PDF gant chart or a HTML web page:

```
https://gemc.jlab.org/work/GEMC_Timeline      (HTML)

https://gemc.jlab.org/work/GEMC_Timeline.pdf  (PDF)
```

The timeline includes resources management.

4 Event Reconstruction

The event reconstruction software has been designed and developed within the ClaRA framework. The reconstruction program must reconstruct, on an event-by-event basis, the raw data coming from either simulation or the detectors to provide physics analysis output such as track parameters and particle identification.

4.1 Tracking

Charged particle tracking is separated into the reconstruction of tracks in the central ("Barrel" Silicon and Micromegas Trackers) and forward (Forward Micromegas Tracker and Drift Chambers) detectors. The forward region covers the angular range from 5° to 40°, while the central detector covers approximately 40° to 135°. In the central region a 5 T solenoidal magnetic field bends charged tracks into helices, while forward-going tracks are bent by a $\sim$ 2 T toroidal magnetic field.

For both systems, the reconstruction includes hit recognition, pattern recognition and track fitting algorithms. Hit objects, corresponding to the passage of a particle through a particular detector component, involve a combination of an electronic signal and the detector sub-system geometry. These objects are then manipulated to form the input to the pattern recognition algorithms. Hit recognition involves clustering of hits and the determination of the spatial coordinates and corresponding uncertainties for hits and clusters of hits. At the pattern recognition stage, hits that are consistent with belonging to a trajectory (i.e. track) are identified. This set of hits is then fit to the expected trajectory with their uncertainties, incorporating the knowledge of the detector material and the detailed field map.

For central tracking, hit recognition first consists of the determination of the intersection of clusters of strips in a double layer of the tracker, when projected into the bottom-most layer. This is defined as a 3-dimentional point with uncertainty estimated from the cluster error matrix. Since the layers in a double layer are not on the same plane, the location of this point is trajectory-dependent (it depends upon the crossing angle of the particle with the BST tile). Hence an iterative algorithm involving the estimate of the tangent to the track at the intersection plane is implemented in order to improve the hit position accuracy. The set of central tracker hits is the input to a Hough Transform algorithm.

The Hough Transform is a powerful pattern recognition tool to select groups of hits and is a proven technique to separate real tracks from background. Groups of points belonging to the same trajectory, such as a helix, correspond to the intersection of the continuous distributions (e.g. lines) in Hough space that the discrete points in the original space map into. The algorithm makes use of an array, called an accumulator, with dimensions equal to the number of parameters in Hough space. The bin sizes in the array are finite intervals in Hough parameters, which are called accumulator cells. The cell with the highest count corresponds to the intersection of the lines. The output of the Hough Transform algorithm

is a track seed that is the input to a fit.

A cylindrical coordinate system is a natural choice for the central tracker, with the z coordinate along the detector axis. The fit is done to state vectors in the (z, ϕ, p_x, p_y, p_z) coordinate system, where z is the coordinate in the direction of the beam axis, ϕ, is the azimuthal angle (from the origin of the laboratory frame), and p_x, p_y, p_z are the momentum components in the laboratory frame. Two track fitting algorithms have been developed to fit the set of hits that had been identified as a track candidate at the pattern recognition step. A simple helical track fit method based on a least square regression approach is employed in the current version of SOT. A Kalman Filter has also been implemented. The filter starts from the last available measurement (outermost hit) moving towards the target and stopping at the distance of closest approach to the beam axis. The algorithm was validated from a comparison of its results with estimates analytically derived from the Fast Monte Carlo program. Very good agreement was found. The fitting code was also studied in high background scenarios and the reconstruction involved a small instance of fake tracks. The inclusion of the CTOF in the reconstruction chain is aimed at further reducing fake tracks.

Hit recognition for forward tracking is detector-dependent. The Forward Micromegas strip intersection determination is similar to that used in the central region. The Drift Chamber (DC) wire hit information is given by the wire geometrical location and the drift time to the wire. Track-dependent corrections to the hit, such as left-right ambiguity and time-walk are performed.

Pattern recognition in the DC involves three consecutive steps. First contiguous hits in the same superlayer and sector are grouped into a cluster. Clusters consistent with a line are linked into segments. This allows the removal of outliers and yields a solution to the left-right ambiguity of the corresponding hits. The pruning of hits from a cluster requires an iterative procedure. At each step of the procedure a fit with a straight line is performed on the set of hits composing a cluster. The hits on the outer edges of the cluster are removed and the hits for which the left-right ambiguity is solved are kept. At this time, four such iterations are required. As for central tracking, the resulting set of hits are sent to a Kalman filter. In the forward geometry, the measurements are approximately constant in z, which makes a natural choice of parameters: x and y, the coordinates transverse to the beam axis, $tan\theta_x$ and $tan\theta_y$, the track slope in the xz and yz planes, respectively, and Q/p, the inverse particle momentum. The components of the state vector in this coordinate system are initialized to the last plane of Region 3 and from an estimate of the track angles just before and just after the toroidal field. The Kalman filter is run up to the distance of closest approach to the beam axis, as for central tracking.

In addtion to track fitting in the DC, track matching with the Forward Micromegas Tracker (FMT) is performed. Track finding in the FMT is done by selecting combinations of strips producing three almost aligned points in space pointing towards the beam axis. The track candidates from the FMT are then matched with those from the DC by extrapolating the output of the Kalman filter on DC candidates to the FMT region and looking for a

close FMT candidate. We find that requiring that the distance from the extrapolation to the FMT track be within a 1 cm circle allows for a 98% matching efficiency without background.

The last task of the tracking program will be a vertex fitting algorithm, thus providing full event reconstruction. Different outputs (not only the structure, but also the contents) will be produced, depending on the incoming request.

4.2 Time-of-Flight

The Time-of-Flight reconstruction service implemented in SOT extrapolates the DC tracks toward the Forward Time-of-Flight (FTOF) detector and attempts to associate each DC track with a hit on the FTOF.

The FTOF is designed to measure the time-of-flight of charged particles emerging in the forward direction. Its design parameters were chosen to allow pions and kaons separation up to 3 GeV, with the most energetic particles produced at small angles. The FTOF system is comprised of three sets of TOF counters referred to as panels, in each of the six sectors. Each panel is composed of an array of scintillators with a PMT on each end. Panel 1a and 1b are located at forward angles in the range 5 - 36° and panel 2 is at larger angles from 35 to 45°. A charged particle exiting the DC when passing through the scintillator ionizes the material, thereby producing scintillation light which is subsequently detected by the PMTs. The resulting electronic outputs give ADC and TDC readouts used in the reconstruction.

The current reconstruction algorithm first searches for FTOF hits associated with each DC track and calculates the distance from the FTOF to the DC. It finds the distance from the scintillator to the reference point in the tangent line to the track (assumed to be at the end of the DC) and associates FTOF hits with such distance less than 2 cm.

The timing information from the associated FTOF hits is used in conjunction with the path-length and reconstruction momentum from the forward Kalman filter to calculate particle masses for particle identification.

4.3 Cerenkov Counters

In CLAS12, two sets of threshold, non-imaging Cerenkov counters are used in the forward region to aid in charged-particle identification. The low-threshold Cerenkov Counter (LTCC), located between the drift chambers and the time-of-flight scintillators, uses the heavy gas C_4F_{10} ($n \sim 1.0015$), with a pion threshold of ~ 2.5 GeV and a kaon threshold of ~ 9 GeV, primarily to reject kaons from pion candidates for momenta above the limit for time-of-flight based π/K separation. The high-threshold Cerenkov Counter (HTCC), located between the exit of the central solenoid and the drift chambers, uses CO_2 gas ($n \sim 1.00045$) with a pion threshold of 4.9 GeV, primarily to reject negative pions from electron candidates in both the online trigger system and the offline analysis.

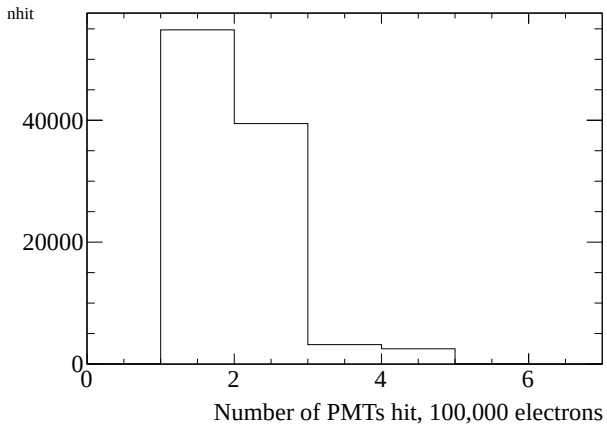


Figure 19: Number of HTCC hits per event in GEANT4 for 100,000 electrons with uniformly distributed momentum ($1 \leq p$ (GeV) ≤ 11), polar angle ($5 \leq \theta(^\circ) \leq 35$) and azimuthal angle ϕ ($-\pi \leq \phi \leq \pi$).

4.3.1 HTCC

In the HTCC, the Cerenkov emission angle for electrons with $\beta \approx 1$ is 1.7°. The HTCC optics design consists of 48 individual mirror segments, arranged in four rings subtending approximately equal θ-intervals for $5^\circ \leq \theta \leq 35^\circ$, and 12 half-sectors, each subtending a 30° interval in ϕ. The ellipsoidal mirror geometry focuses all the light collected onto a single PMT, resulting in a one-to-one mapping between a given PMT and the range of θ and ϕ covered by its associated mirror segment. The electron path length in CO_2 ranges from 1.3-1.8 m depending on θ.

Owing to the small size of the Cerenkov emission cone in the HTCC, resulting from the low index of refraction of CO_2, the most probable electron signal in the HTCC is a single PMT hit. Figure 19 shows the distribution of the number of PMT hits in the HTCC for 100,000 generated electrons with flat distributions in momentum from 1-11 GeV, θ from 5-35°, and ϕ in 2π. Approximately 55% of electrons cause a single hit in the HTCC, while another $\sim 40\%$ of events in which electrons pass near the boundary between two mirrors cause two PMTs to fire. The remaining $\sim 5\%$ of events near the intersection point between four mirrors cause 3 or 4 PMTs to fire.

Reconstruction of electron hits in the HTCC involves grouping hits from adjacent mirrors into clusters to be associated with a single particle track. The simple clustering algorithm consists of the following procedure:

1. Populate a list of hits in the event that are not yet part of a cluster.

2. Within this "unused hits" list, find the hit with the largest amplitude; i.e., the greatest

number of photoelectrons, and remove it from the "unused hits" list.

3. Search for nearest-neighbor hits of the maximum in the "unused hits" list. For each nearest-neighbor hit found, add it to the cluster and remove it from the "unused hits" list.

4. Repeat step 3 for all hits in the current cluster, allowing the cluster to grow in any direction, until no more unused nearest-neighbor hits exist to be added to the cluster. At this point, all hits directly connected to the maximum will have been added to the cluster seeded by the maximum.

5. Repeat steps 1-4, always starting with the maximum-amplitude unused hit, until no more hits can be found with which to seed clusters.

The behavior of the clustering algorithm is controlled by a series of three thresholds and a timing cut. The thresholds specify the minimum number of photoelectrons in a cluster, the minimum number of photoelectrons in a hit to be considered a valid maximum around which to build a cluster, and the minimum number of photoelectrons in a hit to be considered a valid hit to add to the cluster. The timing cut specifies the maximum time difference between hits in a cluster. Time offsets for each of the four θ segments are used to correct for optical photon path length differences between the different mirrors. The central θ and ϕ values for each ring/half-sector combination are also specified for rough scattering angle reconstruction, neglecting the effect of the solenoid field on the forward electron trajectories.

Figure 20 shows the results of the HTCC clustering algorithm for the same 100,000 simulated electrons shown in Fig. 19. Exactly one cluster is found in virtually all events, and the number of hits in the cluster equals the number of hits in the event in the overwhelming majority of events. This indicates that the HTCC reconstruction code works as intended. Figure 21 shows the number of photoelectrons per cluster in the GEANT4 simulation of HTCC and the efficiency for electrons as a function of threshold assuming single-photoelectron resolution, showing an efficiency exceeding 99% for thresholds up to 16 photoelectrons.

4.3.2 LTCC

Each of the six LTCC sectors is spanned in θ by 18 segments, and in ϕ by 2 modules. The optics of each θ module was designed to focus the light onto the PMT associated with that module and located in the region obscured by the torus coils. The trajectory of the light produced by a typical electron passing through the Cerenkov detector is illustrated in Fig. 22. Since the distance between coils increases approximately linearly with θ, each of the 18 modules has unique optical design parameters.

In the LTCC reconstruction routines clusters of hits are formed. A LTCC cluster is defined as a set of contiguous hits that are time-consistent and contain a threshold

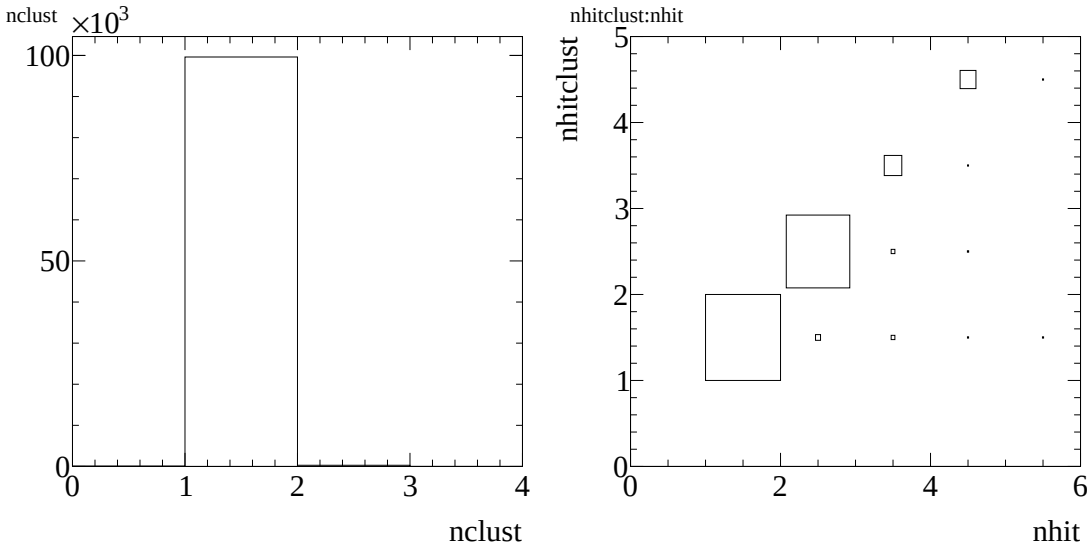


Figure 20: Left: number of clusters found per event for the same simulated electron sample shown in Fig. 19. Right: correlation between the number of hits per cluster and the number of hits per event. A small fraction of events have more hits per event than hits per cluster, reflecting the small additional backgrounds from δ-rays and optical photons reflected by the PMT windows ($\sim$ 4% probability) onto the mirrors and back into a different PMT (these "echo" hits are less common and separated in time from the primary electron hits by an interval much larger than the timing resolution of the PMTs).

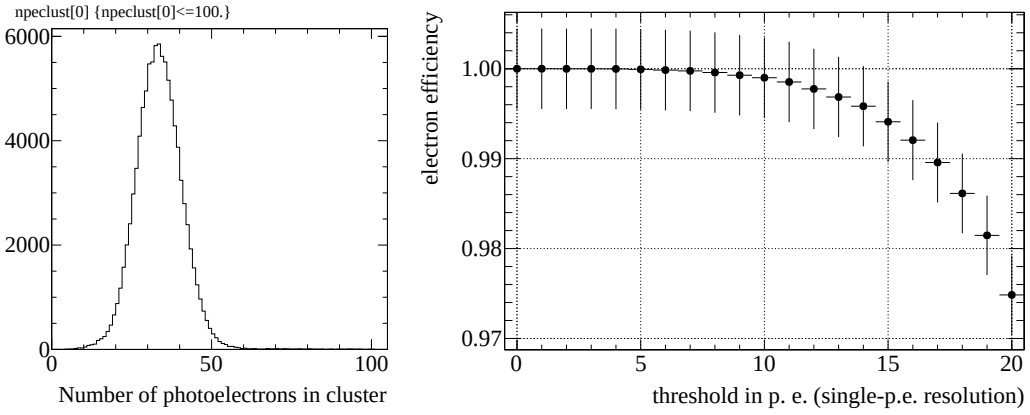


Figure 21: Left: Number of photoelectrons per cluster in the HTCC GEANT4 simulation. Right: Efficiency for electrons as a function of threshold in photoelectrons, assuming single-photoelectron resolution, averaged over all p, θ and ϕ.

number of photo-electrons (to be determined based on the PMT quantum efficiency and simulation studies). Once clusters are determined, a matching algorithm will couple them with existing tracks, if within certain distance and time parameters. An example is given in Fig. 22 (right), where the signals in the PMTs are associated with the (electron) track that generated the Cerenkov light.

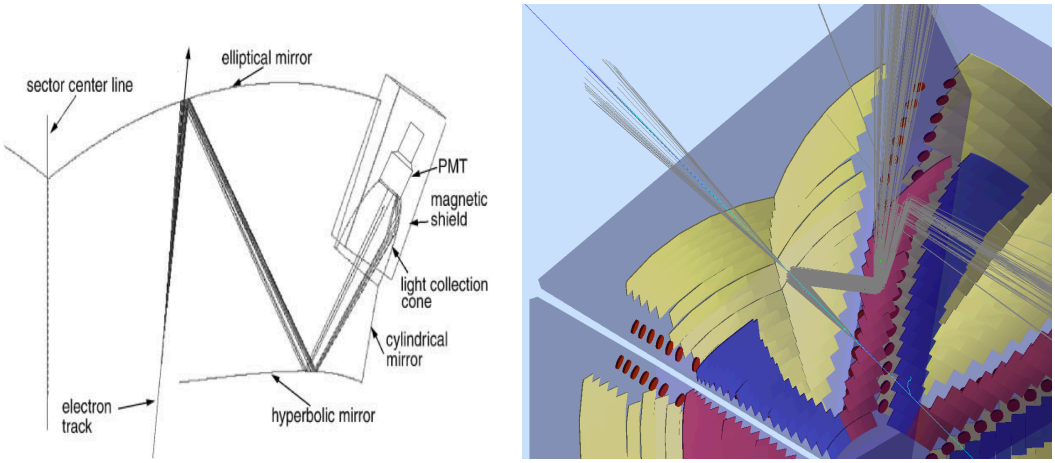


Figure 22: Left: Optical arrangement of one of the 216 optical modules of the CLAS Cerenkov detector showing the optical and light collection components. Note that the Cerenkov PMTs lie in the region obscured by the magnet coils. Right: Simulation of electron Cerenkov light and its reflections on the LTCC mirrors.

4.4 Calorimeters

The CLAS12 calorimeters will consist of a pre-shower (PCAL) detector which will be built and installed in front of the current electromagnetic calorimeter (EC). Simulations indicated that the EC alone would not be able to absorb the full energy of the electromagnetic showers produced by electrons and photons with momenta above 5 GeV/c. The PCAL was designed with a fine enough granularity in order to allow the reconstruction of high-energy showering particles and to separate high-energy π^0's from photons. Both the EC and PCAL will have similar geometries with three stereo read-out planes composed of scintillator layers segmented into strips and sandwiched between lead sheets.

The calorimeter reconstruction package contains classes to represent the structure of the calorimeter detectors with sector, layer, view and strip components. At this point the EC reconstruction is fully functional and integrated into the ClaRA framework as a service.

In the EC detector, each sector has four layers, each layer has three views, and each view has a list with the strips that were stored in the input file. The EC reconstruction contains several packages. The event package contains the classes to store the information generated by the reconstruction algorithm for the event being reconstructed. The reconstruction

package contains the algorithms to find peaks in each view, then hits in each layer, to correct the timing and energy of the peaks, and finally to find matches of hits between layers.

The EC reconstruction algorithm is split into three services. The first service fills the information of strips using the TDC and ADC data obtained from the input file and the calibration data. The second service is the main one. It iterates over the layers of the sector to get the peaks from the strips and then the hits from the peaks. The third service matches the hits between the layers in the sector. Finally, when all the six sectors have been evaluated, the orchestrator fill the output file with the calculated values.

In addtion, the EC software contains calibration services which at present get the calibration constants from the CLAS database.

4.5 PID for CLAS12

Particle identification (PID) is the procedure of selecting signal events while rejecting background events, and is a key step in the data analysis for essentially all particle physics experiments. HEP experiments include various approaches from the Bayesian statistical and log-likelihood technique [8, 9] to more sophisticated Boosted Decision Trees (BDT) [10] and Artificial Neural Networks (ANN) [11].

The identification of the scattered lepton is crucial in electroproduction experiments. The comprehensive research program of CLAS12 comprises both physics with semi-inclusive and hard exclusive processes, where identification of hadronic particles is also required. The identification of electrons and hadrons at CLAS12 is based on the detector responses of the forward (EC) and preshower (PCAL) calorimeters, and the High Threshold Cherenkov Counter (HTCC). Hadron identification includes the Low Threshold Cerenkov Counter (LTCC), central (CTOF) and forward (FTOF) time-of-flight counters. Particle identification at CLAS12 will be accomplished combining the reconstructed vertex, momentum and angles from tracking detector (FST, BST, DC) information with responses of the relevant detector components (HTCC, LTCC, TOF, PCAL, EC). One possible PID scheme for CLAS12 could be a probability-based procedure, where for each particle type a probability that the measured detector signal was caused by that particle is calculated by comparing the signal with parent distributions which are the typical responses of the different particle types in the detector.

4.5.1 Probability Analysis

The probability analysis is based on the extraction of probabilities that a measured detector response E was caused by particles of a certain type, A_i. This conditional probability $P(A_i|E)$ is related through Bayes's theorem to the conditional probability $P(E|A_i)$ that a particle of a given type A_i causes a detector response E:

$$P(A_i|E) = \frac{P(A_i) \cdot P(E|A_i)}{\sum_{k=1}^{n} P(A_k) \cdot P(E|A_k)}, \tag{1}$$

where the sum runs over all particle types A_k and $P(A_i)$ is the probability that the particle of type A_i is present in the detector.

The conditional probability $P(E|A_i)$ for a given detector D is equivalent to the parent distribution $\mathcal{L}_D^i$ that depends on the particle type, the detector response E, and the momentum p of the particle:

$$\mathcal{L}_D^i \equiv \mathcal{L}_D^i(E, p, \theta). \tag{2}$$

The parent distributions are an intrinsic property of a PID detector that can be calculated through normalizing the appropriate particle counts as a function of the detector response. Eventually the parent distributions will be extracted from the data collected

during the normal operation of the experiment, as during the time the detector response may change. In the extraction of the parent distributions, clean samples of certain type of particles are required, which can be achieved by introducing very tight cuts on the output given by the PID detectors. Compared to other DIS experiments employing this technique, CLAS12 benefits from the observation of exclusive processes with well-defined PID for the decay products. In the design stage we use the parent distributions obtained from the GEANT4 simulation of the relevant detectors. The parent distributions are normalized to 1 and give the probability for a certain type of particle to give a response E. Some typical parent distributions for pions and electrons in certain bins of momentum ($p = 4$ GeV) and polar angle ($\theta = 15^\circ$) obtained from the GEANT simulation of CLAS12 [12, 13] are shown in Fig. 23.

The probability $P(A_i)$ is given by the normalized flux ϕ^i which depends on the momentum p and the scattering angle θ of the particle:

$$P(A_i) = \phi^i \equiv \phi^i(p, \theta). \tag{3}$$

The conditional probability $\mathcal{P}^i \equiv P(A_i|E)$ that a certain particle type has been observed can be determined from the detector response using Bayes's theorem if the parent distributions and the particle fluxes are known. The parent distributions can be extracted from collected data or theoretical models.

The probability for a certain particle (e.g. electron) in a detector D is given by:

$$\mathcal{P}^e_D = \frac{\mathcal{L}^e_D}{\Phi\mathcal{L}^h_D + \mathcal{L}^e_D}, \tag{4}$$

where $\Phi \equiv \phi^h/\phi^e$ is the flux ratio of hadrons and electrons. In order not to rely on the determination of individual fluxes, one can use the logarithm of the ratio of the electron and hadron probabilities for a given detector:

$$log_{10}\frac{\mathcal{P}^e_D}{\mathcal{P}^h_D} = log_{10}\frac{\mathcal{L}^e_D}{\Phi\mathcal{L}^h_D} = log_{10}\frac{\mathcal{L}^e_D}{\mathcal{L}^h_D} - log_{10}\Phi. \tag{5}$$

The final discrimination between the particle types will be accomplished by combining the probabilities from all relevant PID detectors:

$$PID = log_{10}(\prod_D \frac{\mathcal{L}^e_D}{\mathcal{L}^h_D}) = \sum_D (log_{10}\frac{\mathcal{L}^e_D}{\mathcal{L}^h_D}). \tag{6}$$

To improve the efficiency of the detection system while retaining good particle separation, different combinations of PID detectors could be combined in a set of PID observables. In addition to one-dimensional PID cuts, this will allow two and more dimensional scatterplots for the PID variables.

A similar procedure could be used for hadron identification involving the low threshold Cherenkov counter (LTCC) and forward TOF for forward tracks and central TOF for large angle track identification.

Figure 24 shows (on the left-hand side) the distribution of β as a function of the momentum in the laboratory frame for reconstructed pions, kaons and protons candidates. These events were generated using the GEMC simulation program and reconstructed with SOT with the EC and PID services included in the service chain. Clear bands allowing for particle identification are visible. On the right-hand side of the figure the invariant mass distributions for the three types of particles reconstructed are shown. The upper plot indicates the clear separation between e^- and π^- candidates obtained from the EC.

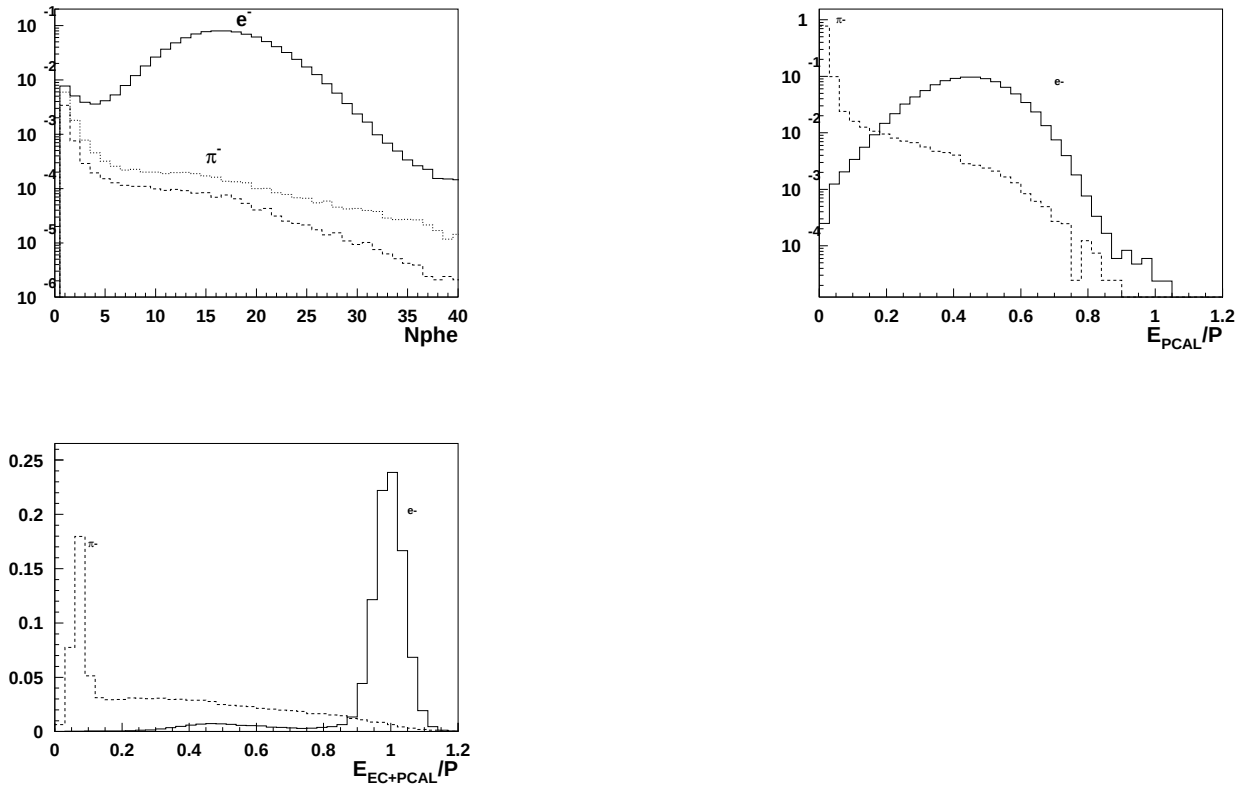


Figure 23: Parent distributions for π^- and electrons for the high threshold Cherenkov counter (HTCC), the preshower (PCAL) and the sum of the responses of the preshower and the forward calorimeter (PCAL+EC). The distributions for electrons (pions) correspond to histograms with solid (dashed) lines. In the upper left plot, the dotted-line histogram corresponds to the distribution of 2 GeV pions.

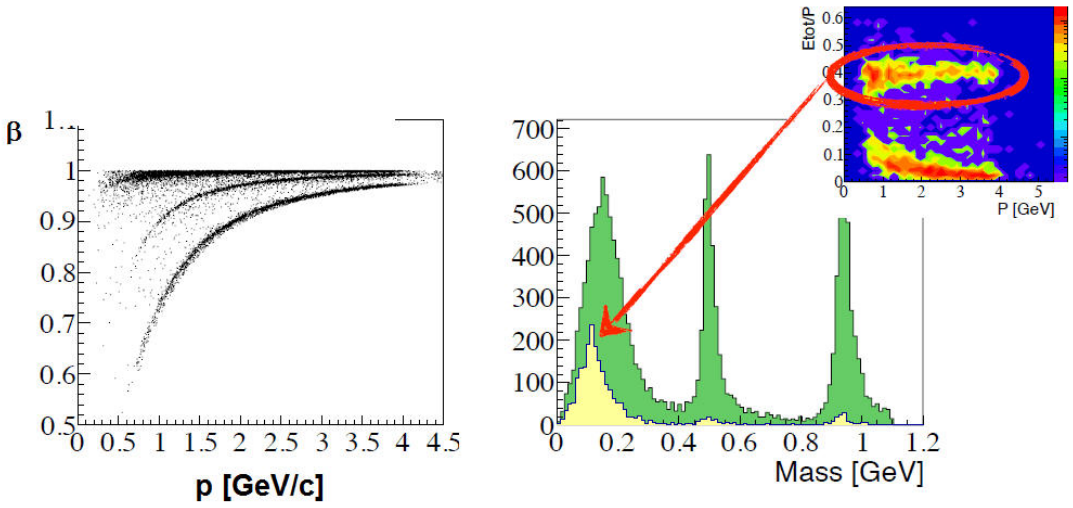


Figure 24: [left] The distribution of β as a function of the momentum in the laboratory frame for reconstructed pions, kaons and protons candidates. [right] The invariant mass distributions for the three types of particles reconstructed.

5 Software Tools

Various object-oriented programs and libraries developed for particle physics data analysis are currently in existence and freely available. ROOT, the C++ program library developed by CERN includes packages for histogramming, curve fitting, matrix algebra, multivariate analysis, four-vector operation and various statistical tools used in data analysis. For Python developement, SciPy [16] is the scientific data analysis system based on NumPy [17] classes. Numpy contains a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code, as well as useful linear algebra, Fourier transform, and random number capabilities. The SciPy library is built to work with NumPy objects, and provides efficient numerical routines such as numerical integration and optimization. Together, they run on all popular operating systems, are easy to install, and are free of charge.

Java-based data analysis systems include JAS (Java Analysis Studio) [18], which is compliant with AIDA (Abstract Interface for Data Analysis) [19], and jHepWork, whose framework is based on FreeHep. FreeHEP is an open source Java library designed to make programming applications easier and to encourage the sharing and reuse of Java code in High Energy Physics. FreeHEP provides utilities for event display (WIRED), fast detector simulation (Lelaps) and HEP software. It also provides an implementation for AIDA. The Colt project is another Open Source set of libraries in Java. This is a partial Java port of CLHEP. CLHEP (Class Library for High Energy Physics) [20] is a C++ library that provides utility classes for general numerical programming, vector arithmetic, geometry, pseudorandom number generation, and linear algebra, specifically targeted for high energy physics simulation and analysis software.

The toolkits listed above are recognized in the High Energy Physics community as excellent standards which the CLAS12 software group will support. The CLAS12 software group also maintains a set of general software tools both in Java and C++, and it is anticipated that a set of specific tools will also be developed in the future in Python. The Java package is the so-called jMath package, and includes a wide variety of software utilities, ranging from graphic and animation tools, relativistic kinematics, matrix algebra, partial wave analysis, numerical methods, and others. In some cases, similar tools are also available in C++. In the future, it is anticipated that useful packages in Maple [21] and/or Mathematica [22] will become a standard part of the CLAS12 software suite.

The CLAS12 simulation package, GEMC, is based on GEANT4, a toolkit for the simulation of the passage of particles through matter. Geant4 is now a standard not only in the High Energy community, but wider scientific and industral applications as well.

The CLAS12 collaboration has adopted the Calibration and Conditions database originally developed for the GlueX collaboration at JLab (see section 2.6). The database needs of both CLAS12 and GlueX are so similar, it did not seem prudent to develop an entirely separate and independent system. Excellent collaboration between the two experiments has guaranteed a good product meeting the specific needs of both groups.

While a final 'post-processed' data format has not yet been adopted, there are several candidates. *EVIO* is a data format designed by and maintained by the Jefferson Laboratory Data Acquisition Group, and is the data format of the raw data. It may also be utilized as the post-processed data format. In addition, we are evaluating HDF5, originally developed by the HDF Group at the National Center for Supercomputing Applications at the University of Illinois, which has developed this data format over the past twenty years. The HDF5 serial and parallel I/O library is the result of the collaboration of NCSA with three DOE laboratories: Lawrence Livermore National Laboratory (LLNL), Sandia National Laboratory (SNL) and Los Alamos National Laboratory (LANL). HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data. HDF5 is portable and is extensible, allowing the evolution of the data format. The HDF5 suite includes a versatile data model that can represent very complex data objects and a wide variety of metadata. It is a portable file format with no limit on the number or size of data objects in the collection, and also includes a software library that runs on a wide range of computational platforms, and implements a high-level API with C, C++, Fortran 90, and JAVA interfaces. Integrated performance features allow for access time and storage space optimizations.

6 Post-Reconstruction Data Access

Due to the distributed nature of the CLAS12 software, a new approach (format) was needed to store physics data. The data formats used by CLAS collaboration lack some key features which we need for distributed multi-process cloud computing.

A new data format based on $HDF5$ was developed, where the data is stored in an indexed tree for random access, and also the structure of the data is stored in a form of a dictionary.

Each data segment that is read from the file comes with a dictionary defining the data structure. This flexibility allows the services (that process user data) to have no predefined data structures, but rather to receive the data structure through run-time dictionaries. With this approach several users can simultaneously use services with data sets that have different structures. This format was developed by the Data Mining group at Old Dominion University[2]. The Data Mining group created a framework around $HDF5$ library that allows flexible dictionary driven data structures to be effectively exchanged between the services and the storage.

The Data Mining project makes use of the ClaRA framework. The distributed cloud computing environement of ClaRa is well-suited for data mining as well as for CLAS12 post-reconstruction data access goals.

The data from the CLAS experiment on nuclear targets is stored on ODU NAS disks. The data is converted from the standard CLAS data format into $HDF5$ format, and it is indexed by experiment and run conditions. The user can access specific sets of data specifying what beam energy, target and beam intensity are required. The ODU computer cluster is running several services that provide information on what data is available and which services are available to analyze the data. The newly developed framework allows multiple processors to cooperate to handle each job, making the data transfer from the disk the ultimate bottleneck. In the future we plan to distribute the data to several servers at participating universities, which will significantly increase the data analysis.

[2]The Data Mining project aims to collect the data from multi-hadron runs from the CLAS detector and to develop tools for easy access and analysis for participating universities. This framework could also be extended to include other data sets from Jefferson Lab, which will allow users to analyze the existing data while the JLab accelerator is shut down for the upgrade.

7 Online Software

During the actual data-taking periods of CLAS12, it is of course expected that there will be full reconstruction of a significant fraction of the acquired data. Online event reconstruction allows data quality monitoring and assessment. It also allows the detailed monitoring of the individual detectors beyond the information given by hardware readouts by providing various reconstruction outputs such as wire profiles, ADC spectra, etc. The capability to study the performance of the CLAS12 detector by examining specific events and extracting their physics parameters will require *full* event reconstruction. In addition, online reconstruction can also act as a *Level 3* trigger to filter unwanted events from the data stream, thus minimizing storage, bandwidth, and other precious resources.

The entire suite of services will be available to the online and data acquisition systems, either directly as network available resources, or as shared code. It is anticipated that a modest cluster of multi-processor, multi-core nodes, of the order of 20 nodes, will be able to keep up with the data rate, based on current timing studies of SOT (SOT was used to obtain a conservative estimate as the track-reconstruction time dominates the processing time of an event).

8 Code Development and Distribution

8.1 Code Management

For CLAS12, we have elected to use the widely adopted and free (Open Source) "subversion" revision control system. Subversion is the Open Source software community's replacement for cvs. It has many of the same features and employs the same no-lockout paradigm[3] In addition, subversion plugins are available for the popular integrated development environments, such as the widely used Eclipse [23]. This allows one to check in, check out, track changes, and merge differences with mouse-clicks in a development environment rather than through a command line.

In CLAS12, we have decided to implement a three-tiered code distribution system. The first level will give access to the subversion repository. Only developers will access code in this manner. The second level will be for code releases, in the form of archives, and intelligent build scripts that do not rely on environment variables. In the CLAS12 environment the user will go to a web page and download a *specific, tested* release. A third tier of release for limited systems (JLab-supported Linux systems) will be for the distribution of binaries.

8.2 Code Release

The CLAS12 software release process will be based on the "agile programming " approach. Part of agile programming is a rapid release schedule. The exact frequency has not yet been determined, but the canonical duration is of the order of one month: two weeks of development and two weeks of testing and bug fixing. So about every month a new version of all software will be released, typically with modest changes from the previous release. Functionality is advanced incrementally as opposed to infrequent but massively different updates.

8.3 Software Tracking

Complicated software development is aided by requirements, tasks, and bug-tracking. Mantis [7] is been utilized for bug-tracking. Project management is implemented via Gantt charts. No release schedule is currently available as there isn't yet an official release of a reconstruction suite.

Ultimately, time development will match the code release cycles. For each cycle, the new tasks and necessary bug fixes will be entered into project management charts. Developers, in communication with the Project Managers, will enter estimates regarding the time it will take to complete the tasks and fix the bugs. Project managers will evaluate if the estimated time fits within the cycle duration and adjust the schedule accordingly by postponing or

[3]Conflicts are resolved through merging rather than avoided through code locks–the latter is generally found to be too draconian and a hindrance to productivity.

adding tasks. Developers will keep a log of the time they spend on a specific task or bug fix, which will help them fine-tune their task-specific time estimates. The subversion revision control system can be set up to require that code checked-in have a comment tying it by ID to a task or to a bug.

9 Quality Assurance

The Service Oriented Architecture is composed of many integrated services that are loosely connected. The usual interaction between the consumer of a service and the supplier of the service is not direct: generally there will be several processes and a network in between. As a result, during an extended development process, many errors can be introduced in the code, rendering it either unusable or incorrect. Quality assurance of developing projects then becomes a major concern.

The standard CLAS12 reconstruction suite of services will be built daily and the reconstruction package tested using a set of standard datasets. A reconstruction validation package will output a series of indicators of the reconstruction performance and result consistency.

The software validation studies will be performed on simulated and real data. Monte Carlo simulations of events produced in fixed target reactions in the CLAS12 detector (see Section 7) will be run through the full reconstruction suite. Detailed checks for discrepancies between reconstructed and generated values exceeding the range expected from resolution effects will be performed using these simulated data. In addition, benchmark datasets recorded by the CLAS12 detector will be used to test the reconstruction software and track its development.

The reconstruction outputs will be stored in a database and tracked over time to identify changes in the program performance. In addition, the Subversion code repository (see Section 8) will allow immediate identification of unanticipated code changes. Subversion also allows individual code developers to check *any* version against the standard suite.

10 Computing Requirements

In this section we present an estimate of the computing resources required by the CLAS collaboration to acquire, reconstruct, simulate and analyze the CLAS12 data in a timely fashion. We assume that tasks like reconstruction and simulation will keep pace with the data acquisition after the start of data taking for CLAS12 in 2015. The computing enterprise for CLAS12 is divided into stages: data acquisition, calibration, reconstruction, simulation, reconstruction studies and physics analysis. The first four stages represent the process of taking the raw data and turning it into 4-momenta and identified particles. The reconstruction studies are needed to optimize the reconstruction of the data and the simulation, while the physics analysis stage represents a broad range of activities using the results of the final event sample (measured and simulated).

We now discuss the computing requirements for the data acquisition and focus on the number of computing cores, disk space, and tape storage. Table 8 shows the assumptions that go into our calculations. Using the data in Table 8 we calculate the data rate, the

Event rate	10 kHz	Weeks running	35
Event size	10 kBytes	24 hour duty factor	60%

Table 1: Data Acquisition parameters.

number of events collected in a year of running, and the volume of that data.

$$\text{Data Rate} = \text{Event Rate} \times \text{Event Size} = 100 \text{ MByte/s} \tag{7}$$

$$\text{Average 24-hour rate} = \text{Data Rate} \times \text{24 hour duty factor} = 60 \text{ MByte/s} \tag{8}$$

$$\begin{aligned} \text{Events/year} &= \text{Event Rate} \times \text{Weeks Running} \times \text{24 hour duty factor} \\ &= 1.3 \times 10^{11} \text{ Events/yr} \end{aligned} \tag{9}$$

$$\text{Data Volume/year} = \text{Events/year} \times \text{Event size} = 1270 \text{ TByte/yr} \tag{10}$$

These results will be used in the calculations below.

We next consider the resources we will need to calibrate our data and keep pace with the incoming data in CLAS12. We have determined the CPU-time to reconstruct an event from previous work with the CLAS12 physics-based simulation GEMC. That result and other assumptions are in Table 9. Calculations of the necessary CPU time in seconds

CPU-data-time/event	180 ms	Data fraction	5%
Data passes	5		

Table 2: Calibration parameters.

(CPU-s) for calibration for one year and the number of cores required to keep pace with the data flow follow.

$$\begin{aligned} \text{CPU time/year} &= \text{Events/year} \times \text{CPU-data-time/event} \times \\ &\quad \text{Data fraction used} \times \text{Data passes} \\ &= 5.7 \times 10^9 \text{CPU-s/year} \end{aligned} \tag{11}$$

$$\text{Cores} = \frac{\text{CPU time/year}}{T_{yr}} = 201 \text{ cores}$$

The quantity T_{yr} is the number of seconds in one year multiplied by the efficiency of an individual core in the JLab computing farm which is close to 90%.

Reconstruction of the CLAS12 data (known as cooking) will be the second-most computer intensive task behind simulation (see below). We have estimated the time required to reconstruct an event using simulated data from the CLAS12 physics-based simulation GEMC and the CLAS12 reconstruction code SOCRAT. We have found that 180 ms is long enough to reconstruct most of the events that CLAS12 will collect. The other parameters have been estimated based on experience with the CLAS detector and are listed in Table 10. We first estimate the CPU time (in CPU-s) required for the reconstruction to keep

CPU-data-time/event	180 ms	Output size/input size	2
Data passes	2	Output fraction on work disk	10%

Table 3: Reconstruction parameters.

pace with the incoming data and use this to determine the number of cores need. We also estimate the disk and tape storage and the average bandwidth needed to move these data since these tasks will require considerable resources. The calculation of the bandwidth includes time for reading in each event and writing out the reconstruction results.

$$\begin{aligned} \text{CPU time per year} &= \text{Events/year} \times \text{CPU-data-time/event} \\ &\quad \times \text{Data passes} \\ &= 4.6 \times 10^{10} \text{ CPU-s/year} \end{aligned} \tag{12}$$

$$\text{Dedicated farm cores} = \frac{\text{CPU time per year}}{T_{yr}} = 1611 \text{ cores} \tag{13}$$

$$\begin{aligned}\text{Cooked data to tape} &= \text{Data Volume/year} \times \text{Data passes} \\ &\quad \times \text{Output size/input size} \\ &= 5080 \text{ TByte/yr}\end{aligned} \tag{14}$$

$$\text{Disk storage} = \frac{\text{Cooked data to tape}}{10} = 508 \text{ TByte}$$

$$\begin{aligned}\text{Average bandwidth} &= \text{Event size} \times (1 + \text{Output size/input size}) \times \\ &\quad \frac{\text{Dedicated farm cores}}{\text{CPU-data-time/event}} \\ &= 268 \text{ MBytes/s}\end{aligned} \tag{15}$$

Simulation of the CLAS12 response will be an essential part of the reconstruction and analysis because the precision of many experiments will not be limited by statistical uncertainties, but by systematic ones. Understanding the detector is necessary to distinguish physics effects from possible experimental artifacts. Table 11 shows the parameters used in the calculations that follow. The CPU-sim-time/event is the time required to simulate an event using the CLAS12, physics-based simulation GEMC and to reconstruct it with SOCRAT (the CLAS12 reconstruction package) on a single core. To estimate the number

CPU-sim-time/event	510 ms	Fraction to disk	2%
Sim-events/year	3.2×10^{11}	Fraction to tape	10%
Output event size	50 kBytes	Multiplicity	1.5

Table 4: Simulation parameters.

of simulated events we need in a year we have studied the properties of the planned trigger in CLAS12 and the backgrounds associated with those events. We find that of the expected 1.3×10^{11} events we expect to collect in one year (see Equation 27) about half will have a good electron that will be reconstructed. Of that sample we expect about half will be background, leaving us with about one-fourth of the event rate as good physics events. In order to adequately simulate the properties of this sample we need about ten times as many simulated events so the statistical uncertainty on the simulated events will be much less (about one-third) than the statistical accuracy of the data. This number, Sim-events/year, is the number of events for a single, high-statistics simulation of the final physics sample

and is listed in Table 11. The multiplicity factor in Table 11 is included to account for computer time to optimize the simulation and to study systematic effects, *e.g.* comparing high-statistics simulations using different event generators. The results of the calculations follow.

$$\begin{aligned}\text{CPU-time/year} &= \text{CPU-sim-time/event} \times \text{Sim-events/year} \times \text{Multiplicity} \\ &= 2.4 \times 10^{11}\ \text{CPU-s/year}\end{aligned} \tag{16}$$

$$\text{Dedicated farm cores} = \frac{\text{CPU-time/year}}{T_{yr}} = 8,558\ \text{cores} \tag{17}$$

The simulation of the CLAS12 is resource intensive so we also considered the volume of disk and tape storage required and the bandwidth necessary for transporting the data.

$$\begin{aligned}\text{Work disk} &= \text{Sim-events/year} \times \text{Output event size} \times \\ &\quad \text{Fraction to disk} \\ &= 318\ \text{TBytes}\end{aligned} \tag{18}$$

$$\begin{aligned}\text{Tape storage} &= \text{Events/year} \times \text{Output event size} \times \\ &\quad \text{Fraction to tape} \\ &= 1588\ \text{TBytes/year}\end{aligned} \tag{19}$$

$$\begin{aligned}\text{Average bandwidth} &= \frac{\text{Output event size} \times \text{Dedicated farm cores}}{\text{CPU-sim-time/event}} \\ &= 839\ \text{MByte/s}\end{aligned} \tag{20}$$

We also expect that in addition to cooking considerable computing resources will be devoting to optimizing the reconstruction of the physics data and the simulated events for particular analysis projects. This task may require studying the reconstruction for a subset of the data (*i.e.*, skim files). The assumptions we make for this part of the CLAS12 computing are shown in Table 12. The calculations of the number of cores necessary to keep

CPU-data-time/event	180 ms	Fraction of desired events	5%
Data passes	10		

Table 5: Reconstruction studies parameters.

pace with the CLAS12 data acquisition follow. The disk storage and average bandwidth

were calculated in the same manner as the previous ones and we found requirements of 508 TBytes for disk and 78 MByte/s for bandwidth.

$$\begin{aligned}\text{CPU time per year} &= \text{Fraction desired} \times (\text{Events/year} + \text{Sim-events/year}) \times \\ &\quad \text{Data passes} \times \text{CPU-data-time/event} \\ &= 4.0 \times 10^{10}\ \text{CPU-s/year}\end{aligned} \tag{21}$$

$$\text{Dedicated farm cores} = \frac{\text{CPU time per year}}{T_{yr}} = 1410\ \text{cores} \tag{22}$$

Once the reconstruction has been optimized for a particular analysis project, we expect there will be considerable computing resources devoted to analyzing the results. These data will not require a full reconstruction so the compute time per event will drop considerably, but the full data set will usually be studied. The assumptions we make for this part of the CLAS12 computing enterprise are shown in Table 13. The calculations of the number of

CPU-analysis-time/event	9 ms	Fraction of desired events	50%
Data passes	10		

Table 6: Physics Analysis parameters.

cores necessary to keep pace with the CLAS12 data acquisition follow. The disk storage and average bandwidth were calculated in the same manner as the previous ones and we found requirements of 889 TBytes for disk and 279 MByte/s for bandwidth.

$$\begin{aligned}\text{CPU time per year} &= \text{Fraction desired} \times (\text{Events/year} + \text{Sim-events/year}) \times \text{Data passes} \times \\ &\quad \text{Data passes} \times \text{CPU-analysis-time/event} \\ &= 2.0 \times 10^{10}\ \text{CPU-s/year}\end{aligned} \tag{23}$$

$$\text{Dedicated farm cores} = \frac{\text{CPU time per year}}{T_{yr}} = 705\ \text{cores} \tag{24}$$

To summarize our estimates we present Table 14. It lists the number of cores, disk, and tape storage required for the different stages of the CLAS12 computing enterprise plus totals for each item. This is our estimate of the computing resources necessary for the CLAS collaboration to analyze the data from CLAS12 in a timely and productive manner. Note the number of cores required for simulation is more than the number for reconstruction, post-reconstruction analysis, and physics analysis combined.

	Cores	Disk (TByte)	Tape (TByte/yr)
DAQ	-	-	1,270
Calibration	201	-	-
Reconstruction	1,611	508	5,080
Simulation	8,558	318	1,558
Reconstruction Studies	1,410	508	-
Physics Analysis	705	889	-
Sum	12,485	2,223	7,938

Table 7: Requirements summary.

11 Computing Requirements

In this section we present an estimate of the computing resources required by the CLAS Collaboration to acquire, reconstruct, simulate and analyze the CLAS12 data in a timely fashion. We assume that tasks like reconstruction and simulation will keep pace with the data acquisition after the start of data taking for CLAS12 in 2015. The computing enterprise for CLAS12 is divided into stages: data acquisition, calibration, reconstruction, simulation, reconstruction studies and physics analysis. The first four stages represent the process of taking the raw data and turning it into 4-momenta and identified particles. The reconstruction studies are needed to optimize the reconstruction of the data and the simulation while the physics analysis stage represents a broad range of activities using the results of the final event sample (measured and simulated).

We now discuss the computing requirements for the data acquisition and focus on the number of computing cores, disk space, and tape storage. Table 8 shows the assumptions that go into our calculations. Using the data in Table 8 we calculate the data rate, the

Event rate	10 kHz	Weeks running	35
Event size	10 kBytes	24 hour duty factor	60%

Table 8: Data Acquisition parameters.

number of events collected in a year of running, and the volume of that data.

$$\text{Data Rate} = \text{Event Rate} \times \text{Event Size} = 100 \text{ MByte/s} \tag{25}$$

$$\text{Average 24-hour rate} = \text{Data Rate} \times \text{24 hour duty factor} = 60 \text{ MByte/s} \tag{26}$$

$$\begin{aligned} \text{Events/year} &= \text{Event Rate} \times \text{Weeks Running} \times \text{24 hour duty factor} \\ &= 1.3 \times 10^{11} \text{ Events/yr} \end{aligned} \tag{27}$$

$$\text{Data Volume/year} = \text{Events/year} \times \text{Event size} = 1270 \text{ TByte/yr} \tag{28}$$

These results will be used in the calculations below.

We next consider the resources we will need to calibrate our data and keep pace with the incoming data in CLAS12. We have determined the CPU-time to reconstruct an event from previous work with the CLAS12 physics-based simulation GEMC and the CLAS12 reconstruction code *SOT*. We found that 155 ms is long enough to reconstruct most of the events that CLAS12 will collect. That result and other assumptions are in Table 9. Calculations of the necessary CPU time in seconds (CPU-s) for calibration for one year

CPU-data-time/event	155 ms	Data fraction	5%
Data passes	5		

Table 9: Calibration parameters.

and the number of cores required to keep pace with the data flow follow.

$$\begin{aligned} \text{CPU time/year} &= \text{Events/year} \times \text{CPU-data-time/event} \times \\ &\quad \text{Data fraction used} \times \text{Data passes} \\ &= 4.9 \times 10^9 \text{CPU-s/year} \end{aligned} \tag{29}$$

$$\text{Cores} = \frac{\text{CPU time/year}}{T_{yr}} = 173 \text{ cores}$$

The quantity T_{yr} is the number of seconds in one year multiplied by the efficiency of an individual core in the JLab computing farm which is close to 90%.

Reconstruction of the CLAS12 data (known as cooking) will be the second-most computer intensive task behind simulation (see below). The time required to reconstruct an event using the CLAS12 reconstruction code *SOT* is mentioned above. The other parameters have been estimated based on experience with the CLAS detector and are listed in Table 10. We first estimate the CPU time (in CPU-s) required for the reconstruction to

CPU-data-time/event	155 ms	Output size/input size	2
Data passes	2	Output fraction on work disk	10%

Table 10: Reconstruction parameters.

keep pace with the incoming data and use this to determine the number of cores needed. We also estimate the disk and tape storage and the average bandwidth needed to move these data since these tasks will require considerable resources. The calculation of the bandwidth includes time for reading in each event and writing out the reconstruction results.

$$\begin{aligned} \text{CPU time per year} &= \text{Events/year} \times \text{CPU-data-time/event} \\ &\quad \times \text{Data passes} \\ &= 3.9 \times 10^{10} \text{ CPU-s/year} \end{aligned} \tag{30}$$

$$\text{Dedicated farm cores} = \frac{\text{CPU time per year}}{T_{yr}} = 1387 \text{ cores} \tag{31}$$

$$\begin{aligned}\text{Cooked data to tape} &= \text{Data Volume/year} \times \text{Data passes} \\ &\quad \times \text{Output size/input size} \\ &= 5080 \text{ TByte/yr}\end{aligned} \tag{32}$$

$$\text{Disk storage} = \frac{\text{Cooked data to tape}}{10} = 508 \text{ TByte}$$

$$\begin{aligned}\text{Average bandwidth} &= \text{Event size} \times (1 + \text{Output size/input size}) \times \\ &\quad \frac{\text{Dedicated farm cores}}{\text{CPU-data-time/event}} \\ &= 268 \text{ MBytes/s}\end{aligned} \tag{33}$$

Simulation of the CLAS12 response will be an essential part of the reconstruction and analysis because the precision of many experiments will not be limited by statistical uncertainties, but by systematic ones. Understanding the detector is necessary to distinguish physics effects from possible experimental artifacts. Table 11 shows the parameters used in the calculations that follow. The CPU-sim-time/event is the time required to simulate an event using the CLAS12, physics-based simulation GEMC and to reconstruct it with the CLAS12 reconstruction package *SOT* on a single core. To estimate the number of

CPU-sim-time/event	485 ms	Fraction to disk	2%
Sim-events/year	3.2×10^{11}	Fraction to tape	10%
Output event size	50 kBytes	Multiplicity	1.5

Table 11: Simulation parameters.

simulated events we need in a year we have studied the properties of the planned trigger in CLAS12 and the backgrounds associated with those events. We find that of the 1.3×10^{11} events we expect to collect in one year with CLAS12 (see Equation 27) about half will have a good electron that will be reconstructed. Of that sample we expect about half will be background leaving us with about one-fourth of the event rate as good physics events. In order to adequately simulate the properties of this sample we need about ten times as many simulated events so the statistical uncertainty on the simulated events will be much less (about one-third) than the statistical uncertainty of the data. This number, Sim-events/year, is the number of events for a single, high-statistics simulation of the final physics sample and is listed in Table 11. The multiplicity factor in Table 11 is included to account for computer time to optimize the simulation and to study systematic effects, *e.g.*

comparing high-statistics simulations using different event generators. The results of the calculations follow.

$$\begin{aligned}\text{CPU-time/year} &= \text{CPU-sim-time/event} \times \text{Sim-events/year} \times \text{Multiplicity} \\ &= 2.3 \times 10^{11}\ \text{CPU-s/year}\end{aligned} \tag{34}$$

$$\text{Dedicated farm cores} = \frac{\text{CPU-time/year}}{T_{yr}} = 8,139\ \text{cores} \tag{35}$$

The simulation of the CLAS12 is resource intensive so we also considered the volume of disk and tape storage required and the bandwidth necessary for transporting the data.

$$\begin{aligned}\text{Work disk} &= \text{Sim-events/year} \times \text{Output event size} \times \\ &\quad \text{Fraction to disk} \\ &= 318\ \text{TBytes}\end{aligned} \tag{36}$$

$$\begin{aligned}\text{Tape storage} &= \text{Events/year} \times \text{Output event size} \times \\ &\quad \text{Fraction to tape} \\ &= 1,588\ \text{TBytes/year}\end{aligned} \tag{37}$$

$$\begin{aligned}\text{Average bandwidth} &= \frac{\text{Output event size} \times \text{Dedicated farm cores}}{\text{CPU-sim-time/event}} \\ &= 839\ \text{MByte/s}\end{aligned} \tag{38}$$

We also expect that in addition to reconstruction and simulation considerable computing resources will be devoted to optimizing the reconstruction of the physics data and the simulated events for particular analysis projects. This task may require studying the reconstruction for a subset of the data (*i.e.*, skim files). The assumptions we make for this part of the CLAS12 computing are shown in Table 12. The calculations of the number of

CPU-data-time/event	155 ms	Fraction of desired events	5%
Data passes	10		

Table 12: Reconstruction studies parameters.

cores necessary to keep pace with the CLAS12 data acquisition follow. The disk storage and average bandwidth were calculated in the same manner as the previous ones and we

found requirements of 508 TBytes for disk and 78 MByte/s for bandwidth. The amount of data archived to tape we expect to be small compared to our other requirements.

$$\begin{aligned}\text{CPU time per year} &= \text{Fraction desired} \times (\text{Events/year} + \text{Sim-events/year}) \times \\ &\quad \text{Data passes} \times \text{CPU-data-time/event} \\ &= 3.4 \times 10^{10}\ \text{CPU-s/year}\end{aligned} \tag{39}$$

$$\text{Dedicated farm cores} = \frac{\text{CPU time per year}}{T_{yr}} = 1214\ \text{cores} \tag{40}$$

Once the reconstruction has been optimized for a particular analysis project, we expect there will be considerable computing resources devoted to analyzing the results. These data will not require a full reconstruction so the compute time per event will drop considerably, but most of the data set will usually be studied. The assumptions we make for this part of the CLAS12 computing enterprise are shown in Table 13. The calculations of the number

CPU-analysis-time/event	8 ms	Fraction of desired events	50%
Data passes	10		

Table 13: Physics Analysis parameters.

of cores necessary to keep pace with the CLAS12 data acquisition follow. The disk storage and average bandwidth were calculated in the same manner as the previous ones and we found requirements of 889 TBytes for disk and 279 MByte/s for bandwidth. We expect the amount of data archived to tape to be small.

$$\begin{aligned}\text{CPU time per year} &= \text{Fraction desired} \times (\text{Events/year} + \text{Sim-events/year}) \times \text{Data passes} \times \\ &\quad \text{Data passes} \times \text{CPU-analysis-time/event} \\ &= 1.7 \times 10^{10}\ \text{CPU-s/year}\end{aligned} \tag{41}$$

$$\text{Dedicated farm cores} = \frac{\text{CPU time per year}}{T_{yr}} = 607\ \text{cores} \tag{42}$$

To summarize our estimates we present Table 14. It lists the number of cores, disk, and tape storage required for the different stages of the CLAS12 computing enterprise plus totals for each item. This is our estimate of the computing resources necessary for the CLAS Collaboration to analyze the data from CLAS12 in a timely and productive manner. Note the number of cores required for simulation is more than the number for reconstruction, reconstruction studies, and physics analysis combined.

	Cores	Disk (TByte)	Tape (TByte/yr)
DAQ	-	-	1,270
Calibration	173	-	-
Reconstruction	1,387	508	5,080
Simulation	8,139	318	1,558
Reconstruction Studies	1,214	508	-
Physics Analysis	607	889	-
Sum	11,520	2,223	7,938

Table 14: Requirements summary.

11.1 JLab Responsibilities and Institution Responsibilities

The CLAS12 hardware project is part of the DOE-funded 12 GeV Upgrade project at JLab. The development of analysis software is, however, not included in the CLAS12 upgrade project and no CLAS12-related staff is assigned towards the development of specific CLAS12 related software. The development of the software architecture and software tools has to make use of the resources assigned to the operation and analysis of ongoing experimental programs in Hall B and the involvement of users from Universities and foreign institutions. Figure 25 shows the JLab/Hall B staff currently working part time on the software development for CLAS12.

Jefferson Lab Staff involved in CLAS12 Software development 2012-2016			
Personnel	Software Function	FTE * yrs	Specific work assignments
D. Weygand (senior staff)	CLAS12 Software architect	2.5	Oversee and direct Software Development
V. Gyurjyan (senior staff)	CLARA Developer	1.5	Service oriented architecture
M. Mestayer (senior staff)	Advisor	0.5	Advise junior personnel on tracking code
M. Ungaro (staff)	GEMC Developer	1.25	Geometry/material DB Digitization of signals
V. Ziegler (staff)	Event reconstruction	1.25	Develop version 3 of tracking code
A. Puckett (staff)	LTCC & HTCC reconstruction	1.25	Pattern recognition/hit reconstruction

University and Research Institutions involved in CLAS12 software development 2012-2015					
Group Leader	Software Function	MOU	Sr. Res. Faculty (FTE*yrs)	Postdocs & Grad. Stud.	Specific work assignments
K. Hicks, Ohio U,	Project management	✓	1.25	2.5	Software management CLAS12 Data base
G. Gilfoyle, U. Rich	CLARA services	✓	1.75	4.7	Software development team
D. Heddle, CNU	CLARA services	in process	1.25		clas12 event display (ced)
M. Wood, Canisius College	CLARA services	✓	1.25	2.5	PCAL reconstruction software
D. Ireland, Glasgow Univ.	Slow Control	in process	3.5	3.65	Head of Hall B EPICS, simulation, reconstruction
J. Ball, Saclay/IRFU	Tracking	in process	tbd	tbd	Charged particle tracking
M. Battaglieri, R. De Vita, INFN/Genova	Head of CLAS12 Commissioning team	in process	3.0	4.0	Forward tagger calibration & commissioning software
W. Brooks, USM, Chile	Calorimeter analysis	in preparation	tbd	tbd	Pattern recognition, reconstruction in PCAL/EC

Figure 25: Human resources available for the CLAS12 software development, showing both JLab staff personnel (top) and resources from non-JLab institutions through MOUs.

References

[1] CLAS12 Technical Design Report Version 5.1, Jul 2008, http://www.jlab.org/Hall-B/clas12_tdr.pdf.

[2] B.A. Mecking *et al.*, Nucl. Instrum. and Meth. **A503**, 513 (2003).

[3] http://clasweb.jlab.org/wiki/index.php/CLAS12_Calibration_and_Commissioning (password-protected site)

[4] G. Barrand *et al.*, GAUDI: A Software Architecture and Framework for Building HEP Data Processing Applications, Comp. Phys. Commun. **140**, 44 (2001).

[5] V. Gyurjyan *et al.*, CLARA: A Comtemporary Approach to Physics Data Processing, Journal of Physics Conference Series **331**, 032013 (2011).

[6] C. Timmer *et al.*, A General Purpose, Publish-Subscribe, Inter-process Communication Implementation and Framework. Proceedings of the International Conference in Computing in High Energy and Nuclear Physics, Victoria, BC, Canada, 2007.

[7] http://www.mantisbt.org/

[8] B. Casey *et al.*, Phys. Rev. **D66**, 092002 (2002).

[9] A. Airapetian *et al.*, Nucl. Instrum. and Meth. **A417**, 230 (1998).

[10] B.P. Roe *et al.*, Nucl. Instrum. and Meth. **A543**, 577 (2005).

[11] B. Denby Nucl. Instrum. and Meth. **A534**, 343 (2004).

[12] S. Stepanyan and N. Dashyan, CLAS note 2007-001.

[13] A. Vlassov, CLAS note 2006-009.

[14] M. Mestayer, SLAC Eng. Note 72,1977.

[15] J. Wendland, http://www-hermes.desy.de/notes/pub/03-LIB/wendland.03-032.thesis.ps.gz

[16] http://docs.scipy.org/doc/scipy/reference/

[17] http://docs.scipy.org/doc/numpy/reference/

[18] http://www-sldnt.slac.stanford.edu/jas/documentation/UsersGuide/java.htm

[19] http://www.aidaweb.si/class-reference

[20] http://proj-clhep.web.cern.ch/proj-clhep/manual/RefGuide/index.html

[21] http://www.maplesoft.com/

[22] http://www.wolfram.com/

[23] http://www.eclipse.org/documentation/