Nuclear Instruments and Methods in Physics Research A 741 (2014) 130–162

Contents lists available at ScienceDirect

Nuclear Instruments and Methods in Physics Research A

journal homepage: www.elsevier.com/locate/nima

The ALICE data acquisition system

F. Carena [a], W. Carena [a], S. Chapeland [a], V. Chibante Barroso [a], F. Costa [a], E. Dénes [b], R. Divià [a], U. Fuchs [a], A. Grigore [a,c], T. Kiss [d], G. Simonetti [e], C. Soós [a], A. Telesca [a], P. Vande Vyvre [a], B. von Haller [a,*]

[a] European Organization for Nuclear Research (CERN), Geneva 23, Switzerland
[b] Research Institute for Particle and Nuclear Physics, Wigner Research Center, Budapest, Hungary
[c] Politehnica Univesity of Bucharest, Bucharest, Romania
[d] Cerntech Ltd., Budapest, Hungary
[e] Dipartimento Interateneo di Fisica 'M. Merlin', Bari, Italy

For the ALICE Collaboration

ARTICLE INFO

Article history:
Received 26 November 2013
Received in revised form
4 December 2013
Accepted 4 December 2013
Available online 17 December 2013

Keywords:
Data acquisition
LHC
ALICE
CERN

ABSTRACT

In this paper we describe the design, the construction, the commissioning and the operation of the Data Acquisition (DAQ) and Experiment Control Systems (ECS) of the ALICE experiment at the CERN Large Hadron Collider (LHC).

The DAQ and the ECS are the systems used respectively for the acquisition of all physics data and for the overall control of the experiment. They are two computing systems made of hundreds of PCs and data storage units interconnected via two networks. The collection of experimental data from the detectors is performed by several hundreds of high-speed optical links.

We describe in detail the design considerations for these systems handling the extreme data throughput resulting from central lead ions collisions at LHC energy. The implementation of the resulting requirements into hardware (custom optical links and commercial computing equipment), infrastructure (racks, cooling, power distribution, control room), and software led to many innovative solutions which are described together with a presentation of all the major components of the systems, as currently realized. We also report on the performance achieved during the first period of data taking (from 2009 to 2013) often exceeding those specified in the DAQ Technical Design Report.

1. Introduction

The main goal of the ALICE [1,2] (A Large Ion Collider Experiment) experiment at the CERN Large Hadron Collider (LHC) is a precise characterization of the Quark-Gluon Plasma (QGP), the state of deconfined matter produced in high-energy heavy-ion collisions. This QGP is, in the standard Big Bang model, the state of matter which existed in the early universe from picoseconds to about 10 microseconds after the Big Bang. A precise determination of its properties would be a major achievement. The study of the QGP is performed by investigating the result of heavy ion collisions at a center-of-mass energy of 5.5 TeV per nucleon pair.

Acronyms: ACT, ALICE configuration tool; AMORE, Automatic MOnitoRing Environment; API, application programming interface; BIST, Built-In Self-Test; CASTOR, CERN Advanced STORage manager; CTP, central trigger processor; CVS, concurrent versions system; D-RORC, DAQ Read-Out Receiver Card; DA, Detector Algorithm; DAQ, data acquisition system; DATE, data acquisition and test environment; DB, database; DCS, detector control system; DDD, DCS Dedicated Daemon; DDL, detector data link; DIM, distributed information system; DIU, destination interface unit; DMA, direct memory access; DQM, data quality monitoring; DSS, DAQ services servers; ECS, experiment control system; EDD, ECS dedicated daemon; EDM, event destination manager; FEE, front-end electronics; FERO, front-end and read-out; GDC, global data collector; GUI, graphical user interface; H-RORC, HLT Read-Out Receiver Card; HI, human interface; HLT, high level trigger; HOMER, HLT Online Monitoring Environment; HW, hardware; LDC, local data collector; LHC, Large Hadron Collider; LTU, local trigger unit; NTP, network time protocol; NVRAM, non-volatile random access memory; PDS, permanent data storage; QGP, Quark-Gluon plasma; RCS interface, MISSING; RCT, run control tool; RORC, Read-Out Receiver Card; SAN, storage area network; SIU, source interface unit; SL, shift leader; SMI, state management interface; SSH, Secure SHell; SW, software; TDS, transient data storage; TDSM, transient data storage manager; TRG, trigger system; TTC, trigger timing and control

* Corresponding author. Tel.: +41 22 76 77259.
E-mail address: bvonhall@cern.ch (B. von Haller).

http://dx.doi.org/10.1016/j.nima.2013.12.015
0168-9002

Collisions of lower-mass ions will also be observed as a mean of varying the energy density and of pp or p–nucleus to provide reference data for the nucleus–nucleus collisions.

The LHC includes two adjacent parallel beamlines (or beam pipes) that intersect at four points, each containing a particle beam, which travel in opposite directions around the ring. The particle beams are made of either protons, ion nuclei, or proton and ion nuclei. A beam in the LHC is not a continuous string of particles, but is divided into hundreds of bunches, each a few tens of centimetres long. Each bunch contains more than a hundred billion protons or tens of millions lead nuclei. The LHC has been designed in order to maximize the chances of interaction between the particles in the four intersection points, where the bunches of the two beams cross each other.

Experiments have been installed at the four LHC intersection points to observe the interactions between the particles of the two beams. An interaction is often referred to in this paper as an event and, by extension, the data produced by an experiment when observing this interaction are also designated as an event.

The ALICE experiment has operated since 2008 in several different running modes with significantly different characteristics. ALICE has been primarily designed to run with heavy-ion beams (PbPb) which are characterized by a relatively low interaction rate (≤ 10 kHz), a relatively short running time (a few weeks per year) but with a very large event size produced by the large number of charged particles traversing the detectors. In proton–proton (pp) or proton–nucleus (pPb) running modes, the interaction rates are much higher (up to hundreds of kHz) whereas the event size is smaller and the running time of several months per year in pp. These running modes and the corresponding expected event rates and data throughput constitute the main requirements that have been used to design the online systems as reported in the ALICE Technical Design Report on Trigger, Data Acquisition, High-Level Trigger and Control System [3].

1.1. ALICE online systems

The ALICE experiment includes five online systems: Trigger, Data Acquisition, High-Level Trigger, Detector Control System and Experiment Control System. The functions of these systems are the following:

- The Trigger system (TRG) is combining the information from all triggering detectors and, for every bunch-crossing of the LHC, makes a decision within microseconds whether the resulting data are worth being collected. For each positive decision, it sends a sequence of trigger signals to all detectors in order to make them read out synchronously.
- The core function of the Data Acquisition (DAQ) system is to realize the data-flow from the detector up to the data storage. The DAQ system also includes software packages performing the data quality monitoring and the system performance monitoring.
- The High-Level Trigger (HLT) reduces the volume of physics data by selection and compression of the data.
- The Detector Control System (DCS) is in charge of controlling all the detector services (high and low-voltage power supplies, gas, magnet, cooling, etc.).
- The Experiment Control System (ECS) is coordinating the activities of all the online systems to fulfil their common goal.

This paper describes the DAQ and ECS systems who are in charge of the overall experiment data-flow and control, the detector software, the infrastructure and the data quality monitoring.

1.2. Data-flow

The data-flow from the detector electronics up to the data storage in the CERN computing center is organized as a sequential data-driven pipeline. Upon reception of a sequence of trigger signals requesting the data collection, the selected elements of the detectors generate data fragments that are transferred to computers via optical links. A computing farm is used to check, label, format and record the data. The data-flow has been implemented as a hardware system and a set of software packages described respectively in Sections 3 and 4.1.

1.3. Control

The ECS has several roles. It has to provide the operators with a unified view of the experiment and a central point from where to steer the experiment operations. Second, it also has to permit independent concurrent activities on parts of the experiment (at the detector level) by different operators. Finally, it has to coordinate the operation of the control systems active on each detector: the TRG, DCS, DAQ and HLT control. The ECS and the other software packages used to configure and control the experiment are presented in Section 4.2.

The ALICE Configuration Tool (ACT) serves as a configuration repository to which the different ALICE systems can access to extract their currently selected configuration. It is described in Section 4.2.3.

1.4. Monitoring, metadata, detector software and data quality monitoring

The ALICE DAQ and ECS are complex systems operated by a single person. It is therefore of paramount importance to have good services to monitor the system itself. The software packages used for the infrastructure monitoring are presented in Section 4.3: Lemon for the fabric monitoring, the infoLogger for all the operations related to the log messages generated by all the other software packages and Orthos for the alarm handling.

One package is playing a special role: the electronic logbook presented in Section 4.3.4. This package implements the bookkeeping of all the operational activities and is the only software of the DAQ and ECS systems directly used by all the members of the collaboration.

The DAQ system also includes two facilities to execute detector related algorithms such as calibrating a detector or monitoring the data quality. The Detector Algorithms package (DA) has implemented a controlled and reliable environment for the execution of detector related tasks and is presented in Section 4.4. AMORE (Automatic MOnitoRing Environment) is the environment used by all the detectors and systems for the monitoring of the data quality. It is presented in Section 4.5.

1.5. The DATE software package

The coherence of the whole DAQ and ECS systems is given by a common software framework composed of different layers of modules. The bottom layer includes the memory handling, the process synchronization and the communication layers. The application layers includes the data-flow and the control applications. This framework is called DATE (Data Acquisition and Test Environment) and has been used during all the phases of development and operation of the ALICE experiment.

2. Initial requirements

The requirements for the ALICE DAQ system have been evaluated and refined in several documents [1,3,4]. A few key parameters were essential to design the global architecture and to fix its fundamental characteristics.

The maximum event rate handled by the DAQ system was relatively modest and estimated to be of the order of 1 kHz. This did not put a big constraint on the system design.

On the contrary, the requirement for what was considered at that time the most difficult part of the system, namely the data-flow, was considered extremely challenging. It was expressed as capacities in terms of aggregated data throughput, which was evaluated by listing different scenarios corresponding to the different types of physics and estimating their event rate and event size needs. The total available storage bandwidth was the most important boundary condition of the ALICE DAQ. This limit was dictated by the cost of the media that is needed to store the data and the cost of the computing resources needed to reconstruct and analyze these data. ALICE was originally estimating that 1.25 GBytes/s would provide adequate physics statistics within the construction and exploitation budgets. The CERN management has then agreed that a Central Data Recording facility of the corresponding maximum bandwidth would be provided. This has therefore been arbitrarily set as the maximum throughput that the ALICE DAQ could produce to mass storage. However, all the efforts have been made to reduce the data throughput below this limit without jeopardizing the physics performance. This data throughput was a key element to dimension the hardware architecture.

3. Hardware architecture

3.1. Overview

The overall architecture of the ALICE DAQ (as shown in Fig. 1) was designed from the start as a data driven fully parallelized push architecture. At each stage the content of the data stream drives autonomously its routing through the processing chain from the source to the destination. The throttling is implemented by a back pressure from the destination to the source. The system is data driven because the data flow inside the system is mostly based on the data content themselves.

The first step of the data-flow concerns the detector read-out. The Central Trigger Processor (CTP) forms a decision for every bunch crossing and communicates them via several messages (L0, L1a, L2a) to the Local Trigger Units (LTU) of all detectors which need to be read out. The LTU broadcasts the positive decisions over the Trigger, Timing and Control (TTC) network. Data are pushed into the system by the detectors Front-End and Read-Out (FERO) electronics upon reception of the corresponding sequence trigger signals. The data produced by the detector electronics, the event fragments, are transported by the Detector Data Links (DDL) which are point-to-point optical links clocked at 2.1 Gb/s. Each of these DDL links has a capacity of 200 MB/s. Of the order of 500 links transport the data from the detector, or its vicinity, located in the experimental hall roughly hundred meters below the ground up to a computing room close to the surface named Counting Room 1 (CR1). The receiving end of the DDL is performed by a PCI-X or PCIe adapter called the DAQ Read-Out Receiver Card (D-RORC) which transfers the data directly into the memory of PCs using a Direct-Memory Access (DMA).

Some D-RORCs also send a copy of the data over a second DDL to the HLT farm where the HLT Read-Out Receiver Card (H-RORC) transfers the data directly into the memory of PCs: the Front-End Processors (FEP). The HLT performs a first online reconstruction, produces decisions about the data to be recorded and compresses them.

The trigger signal used to initiate the data transfer includes a time tag originated from the LHC clock and which has a precision of 25 ns corresponding to one bunch crossing. This time tag which is appended to every data fragment is used to identify it. The data

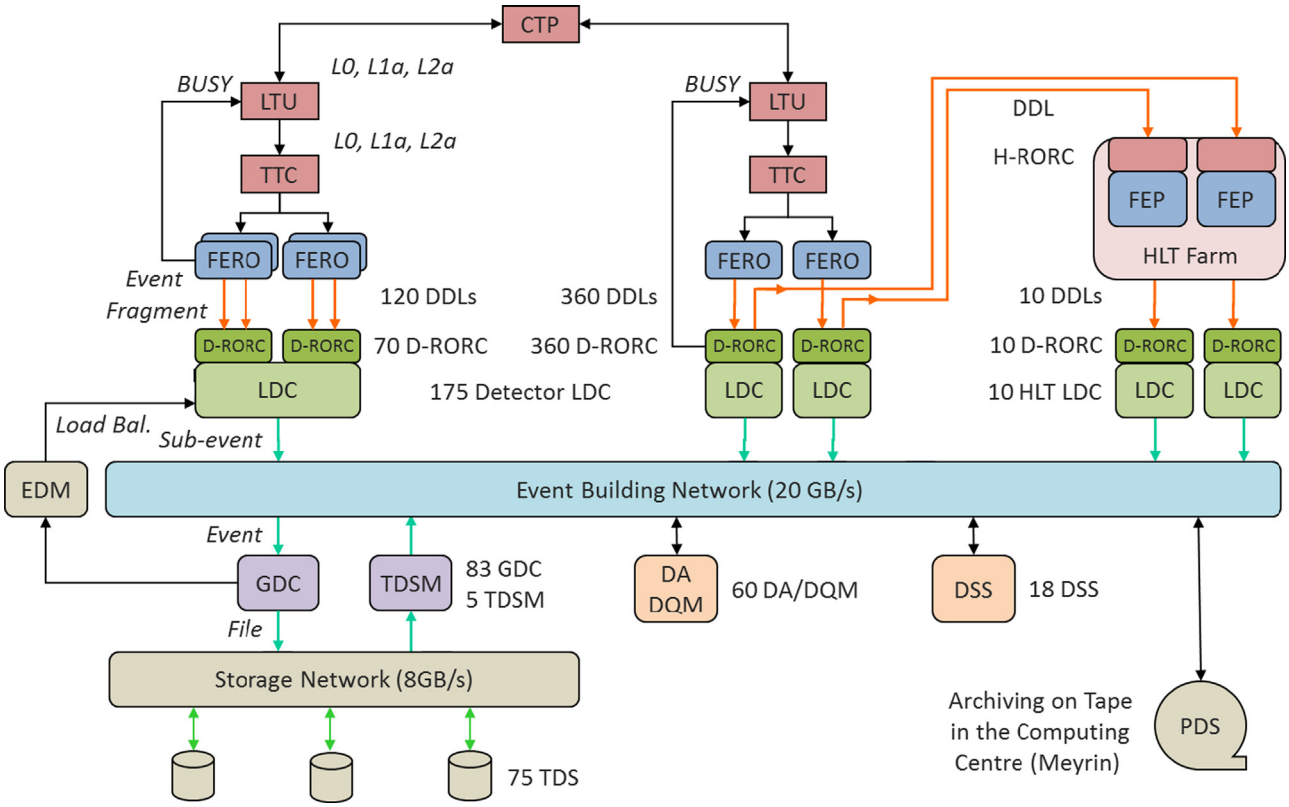


Fig. 1. Overview of the hardware architecture.

are then flowing through a farm of PCs which format, check and assemble together all the pieces of data pertaining to the same interaction.

All the DDLs of the experiment transfer the data into 175 Local Data Concentrators (LDCs). The LDCs realize the first step of a process known as event-building which consists of assembling together the data fragments belonging to the same particle collision into a sub-event. The second step is realized by the Global Data Collectors (GDCs) where complete events are assembled, using the time tag mentioned earlier. The GDCs combine the data directly transferred from the detectors with the result of the processing performed by the HLT farm. A load-balancing mechanism between the GDCs is provided by the Event Destination Manager (EDM).

The GDCs write the events into data files which are stored in the TDSs (Transient Data Storage) which is a temporary buffer located at the experimental area and which has a capacity big enough to store the data during several hours of continuous data taking. The TDSMs (TDS Manager) read back the data files and migrate them to the Permanent Data Storage (PDS) in the CERN computing centre located a few kilometers away from the experimental area.

Several dozens of servers are used for the Data Quality Monitoring (DQM) and the DAs. The DAQ Services Servers (DSS) are providing the general services for the configuration database, the ECS and the other control functions, the system monitoring and the metadata services.

3.2. DDL & RORC

Data flow from the ALICE detectors via optical link pairs (one up-link and one down-link) known as DDL (see Fig. 2).

As shown in Fig. 3, the DDL consists of a Source Interface Unit (SIU, see Fig. 4), which is attached to the front-end electronics inside the detector and a Destination Interface Unit (DIU) embedded in the Read-Out Receiver Card (RORC, see Fig. 5).

Two types of RORCs, each handling two DDL channels with embedded DIUs, are currently used in the experiment:

- The dual channel PCI-X D-RORC has a 64 bit/100 MHz PCI-X bus interface.

Fig. 2. Optical OM1 fibre for DDL with LC plugs.

Fig. 3. FEE connected to the DAQ system.

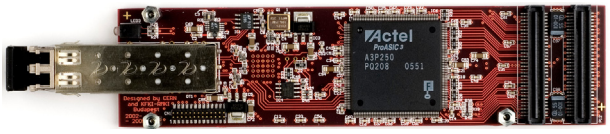


Fig. 4. SIU card.

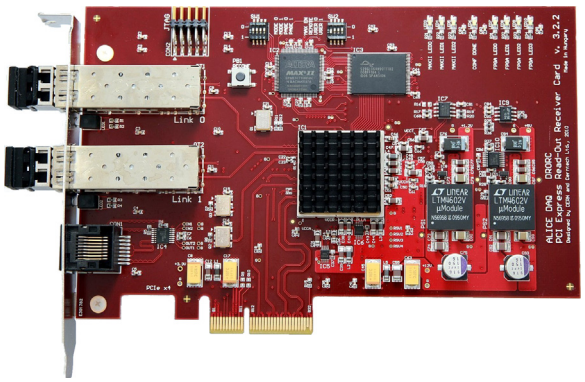


Fig. 5. PCIe RORC card.

- The dual channel PCIe D-RORC has a x4 PCIe (Gen. 1) bus interface.

In this chapter, these cards are commonly referred as RORC. The SIU and the DIU are connected through a pair of optical fibres to transmit data up to a rate of 200 MB/s with a detected bit error rate of less than 10^{-15}. Both the Front End Electronics (FEE) and the SIU are remotely controlled by the RORC through the DDL, since their placement inside the detector does not allow using any other cabling apart from the DDL medium. Therefore, commands and status information are also transmitted between the FEE and the RORC. To achieve the high reliability of the experimental apparatus, efficient tests of all the sub-systems have been provided. The DDL itself also has a powerful built-in self-test (BIST) mode that allows the verification of the full data acquisition chain.

The RORC card has a total data throughput of 400 MB/s between the RORC and the DIU. The throughput via PCI between the RORC and the PC is 800 MB/s for the PCI-X version and 1 GB/s for the PCIe RORC version. Given the requirements of the ALICE experiment a custom protocol has been developed to send data from the SIU to the RORC. For this reason the RORC and the SIU are custom hardware hosting a FPGA that allows firmware upgrade adding new features. The RORC hosts ALTERA FPGA of the following models: ALTERA STRATIX 2 or ALTERA STRATIX 2gx for the PCI-X or PCIe respectively.

The SIU operates in radiation environment and for this development a FLASH based FPGA, the ACTEL PROASIC3, has been selected. Depending on the acquisition needs and on the number of available PCI bus slots, one PC can be equipped with several RORCs (up to 6). Each RORC has a revision number and a unique serial number in its configuration EPROM. The RORC has built-in test systems used to verify the correct behavior of the board, without removing it from the host PC. By using a dedicated utility it is possible to send predefined data patterns to the RORC and receive them back looking for possible data mismatch. The RORC can be operated in three different loopback levels for testing purposes:

- RORC loopback: data is sent in through the PCI bus and sent back immediately.
- DIU loopback: data is sent up to the DIU component and sent back.

- SIU loopback: data is sent through the optical fibre to the SIU and sent back for checking.

The RORC has also an internal data generator used to test the data acquisition chain, if the SIU is not available. Activating this mode will force the card to generate data internally and to send it to the software as it happens during the normal data taking operation.

Dual channel RORCs can be switched to splitter mode: data arriving in one channel is sent out on the other channel in automatic way, this feature is used to forward an exact copy of the data to the HLT system.

The data flow from the DIU to the PC memory is driven by the DMA engine of the RORC firmware in co-operation with the RORC readout software. Data pages that belong to the same sub-event are referred as fragment, which are transferred over the DDL by one or more DDL blocks. Each block can be up to 2^{21} bytes.

The DDL system can be operated in two directions. During the data taking the SIU sends data to the RORC, but the RORC can be used to send configuration information to the detector electronics sending FEE commands to the SIU. The configuration of the detector through the DDL happens usually before starting a new run, but if needed it can be executed also during the normal data taking.

These packages handling the components described above are commonly referred as readout (see Section 4.1.2 for details on this software). The readout software initializes the RORC, DIU and SIU before starting the data taking. Once the initialization process is completed, the readout software provides memory page's addresses to the RORC that moves data to the RAM of the PC autonomously. If there are no available memory locations, the readout program enters in a loop waiting for the next free page. In this situation the data taking has to be paused to avoid data to be overwritten. A XOFF mechanism has been implemented in the DDL protocol. When the RORC cannot write data into the memory of the PC, it issues a XOFF signal, called also back-pressure, to the SIU that blocks the data flow coming from the FEE, that raises the busy to the trigger system stopping the data taking. Once free memory pages are available again the XOFF is cleared and data taking starts back normally.

3.3. The LDC and GDC

The properties of the machines used for the detector readout (LDC) and event building (GDC) are crucial to deliver the required data acquisition speed.

The LDCs are currently based on the X8DTN+ motherboard from Supermicro which provides 3 PCI-X slots and 3 PCI-e slots. With 8GB DDR3 RAM and two Xeon E5606 processors these machines are capable to move the data from the D-RORC cards to memory via DMA and to buffer it there while waiting for the HLT decision.

The GDCs are mostly focused on memory buffer and CPU for event building and data compression. They also require a fast connection to the temporary data storage system. They are currently based on HP Proliant DL160 servers with 24 GB DDR3 RAM, two Xeon E5520 CPUs and a 4 Gbps PCIe Fiber Channel adapter. This configuration provides enough memory to buffer large events and enough CPU power for event building, data compression, *rootification* (see Section 4.1.5) and storage access.

Fig. 6 shows how the event building rate, in terms of MB/s, scales with the increase of the number of concurrent streams performing event building, data compression and rootification by using the multiple cores available.

3.4. Servers

The distributed hardware and software components of the DAQ system rely on several online databases for configuration and

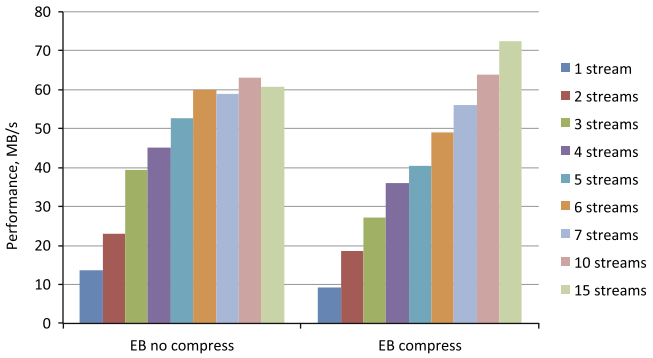


Fig. 6. Event building (EB) performance with and without compression for concurrent streams.

operation. These data are stored in MySQL, running on dedicated database (DB) servers. In this section we review them and their performance.

The databases consists of

- The configuration DB, describing the counting room (CR1) machines, some detector-specific electronics settings and the DAQ and Experiment Control System runtime parameters (ACT, CR1, DATE and ECS on Fig. 7).
- The log DB, centrally collecting reports from running processes (LOG on Fig. 7).
- The experiment logbook, tracking the run statistics filled automatically and the operator entries (LOGBOOK on Fig. 7).
- The online archive of constantly updated data quality monitoring reports (AMORE on Fig. 7).
- The file indexing services, including the status of transient files for permanent storage, and the calibration results for offline export (FES on Fig. 7).
- The user's guide (Wiki).

The profiles (size, number of tables and rows) of the corresponding databases are shown in Fig. 7.

Hardware benchmarks have been conducted to select the appropriate platform. In addition to usual CPU, memory and disk benchmarks, we measured insertion speed of a large data set in a MySQL [5] table. We have been using this procedure since 2005, and it proved to be a good performance metric for the applications we run (even for those not doing bulk inserts).

We used several types of database servers. In 2006, we selected a machine equipped with two AMD Opteron 275, which outperformed other models in our database applications thanks to the integrated memory controller in the CPU, feature which was not available yet on other platforms. Storage was done on a separate 2TB RAID 6 disk array connected by Fiber Channel.

In 2010, the DB server type was upgraded to a dual Intel Nehalem X5677 CPU (3.46 GHz) with 24 GB RAM (6x 4 GB DDR3-1333). Tests showed that in our particular environment, it was better to look for the highest clocking in order to reduce latency of single threads. Some platforms showed better overall throughput (more cores, individually slower), but this did not match so well the database concurrency and locking models.

On this second DB platform, storage is done on two local 15 K SAS 300GB drives (plus one cold spare) configured in RAID1 by the on-board controller providing also a 512 MB Flash Backed Write Cache. It is powered by two redundant power supplies. We also added 2x2TB SATA disks for (less performant) storage space, used e.g. for backups and archiving. The databases are also saved daily to the CERN Advanced STORage manager (CASTOR), the permanent data storage used for physics data at CERN. A fresh system can be restored any time from scratch to continue data taking.

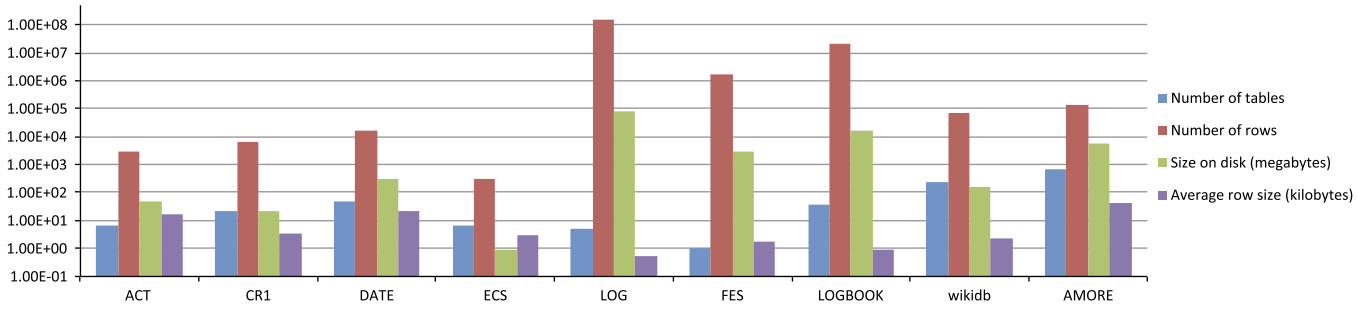


Fig. 7. Sizes of databases.

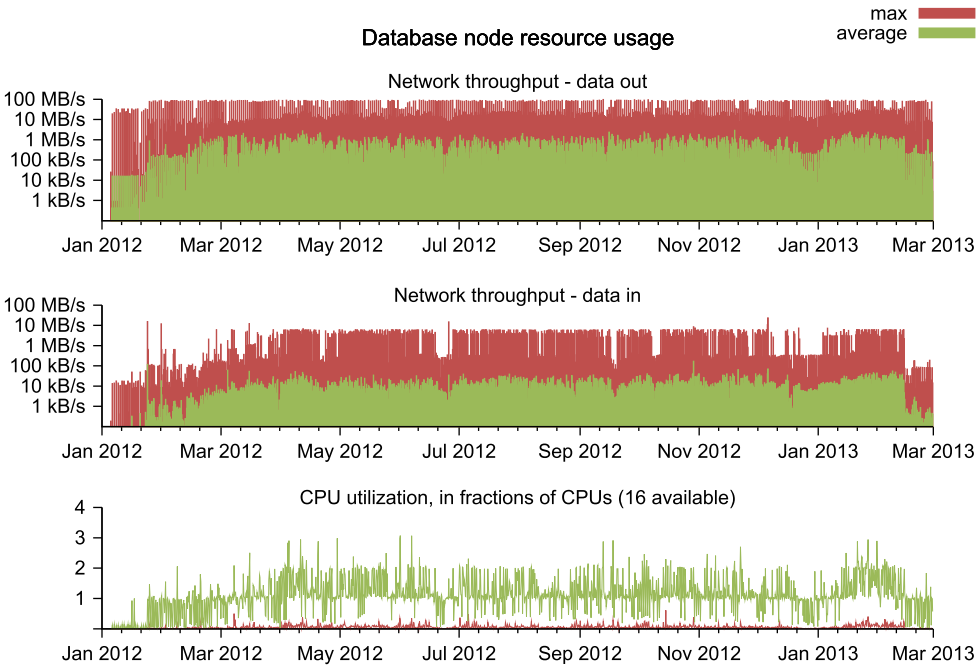


Fig. 8. DB server CPU and I/O usage.

Data updated since last backup would be recovered with a delay if hardware repair is needed, but this would not prevent data taking.

Three such servers are used: one for the main databases, one dedicated to DQM and a spare. The cold spare system can be used as replacement in case of system failure. No hardware problem was observed on the servers since in production.

The system copes well with the production load. Fig. 8 shows the I/O and CPU usage over time, in average and maximum peaks during 2012 and 2013. The server is connected with a single 1 Gb Ethernet. The network outbound traffic sometimes shortly saturates the link, but this could easily be upgraded to 10 Gb. As regards disk I/O, the relatively small data sets accessed at runtime fit well in memory, and disk read access is minimal.

There are typically between 1000 and 2000 clients connected at the same time. On average, new connections are created at the rate of a few per second. However, a large peak occurs at start of run when processes are launched. This corresponds both to transient short-lived connections to retrieve processes configuration, and persistent connections to update statistics during data taking. We observe up to 1000 new connections per second sustained during the first 10 s of a run.

A typical database query pattern with the rates of insert, delete and update statements is shown in Fig. 9. It represents how the system behaved during 10 days of Heavy Ion run in 2013. The data taking periods correspond to the time ranges with *update* queries. The *average* value is computed on 5 minutes intervals. The *maximum* value represent the highest measured sustained value (average) for any sampled 10 seconds interval.

Little tuning was done for the database server settings in order to reach production needs. The main parameters to optimize were the limit on maximum concurrent connections, and the InnoDb engine parameters to make best use of available system memory.

The DAQ system also includes the following central services:

- A machine for the DAQ and ECS control. All the processes run on a single host. There is a backup server ready to take over in case of hardware failure.
- A machine for the web services. In particular, it hosts the experiment logbook, the ACT, and the web version of the infoBrowser. The machine is connected to the CERN general purpose network, and visible from the public Internet. It is constantly kept up to date with the latest security patches. This server has a dedicated route to the DAQ databases.

3.5. Network

The ALICE DAQ consists of 175 LDCs, 83 GDCs and 5 TDSMs providing the necessary services. It was decided to use a Force10 Terascale E1200 as central router that is equipped with 7 line cards providing 48 ports of 1 Gbit each, 3 line cards providing 4 ports of 10 Gbit each and one line card providing 16 ports 10 Gbit. This port

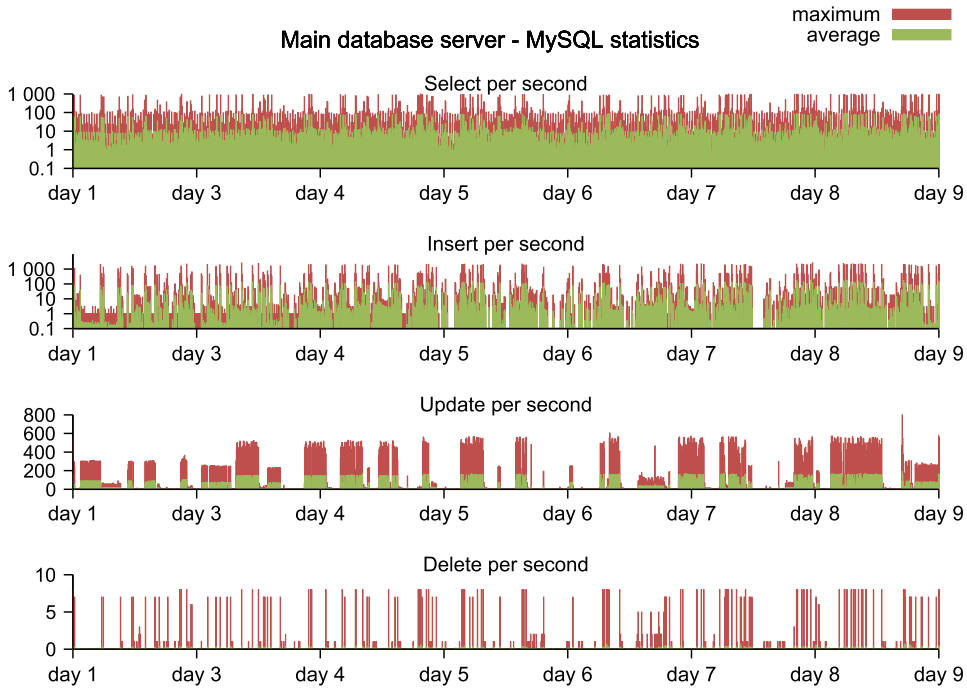


Fig. 9. Database query pattern.

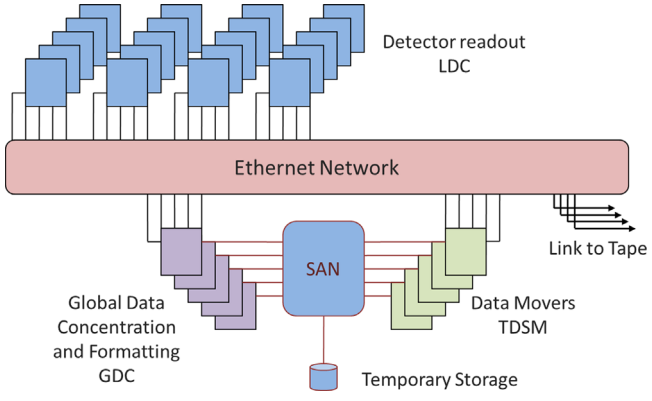


Fig. 10. Network schema.

density allowed all machines to be connected directly to the central router thus creating a fully non-blocking fabric for all nodes (see Fig. 10).

The LDCs are all connected by single UTP Gigabit links, except for some data intensive detectors that required their machines to be connected by 10 Gigabit fiber links . The 75 GDCs used in global runs are all connected by single UTP Gigabit links, while the 5 TDSM data movers were powerful machines connected by 10 Gigabit fiber, thus being able to push more than 1GB/s to tape each. The link to the IT tape facility is realized by 4 LR 10G fiber links, thus giving ALICE a (theoretical) total recording capacity of 4.5 GB/sec.

3.6. Storage

To buffer the data before transferring it to the IT tape facility, the GDCs are also connected to a Fiber Channel SAN giving them access to the temporary storage system (see Fig. 10).

This SAN consists of two Qlogic 9000 switches with 165 4G ports and 32 8G ports, as well as 75 disk-based storage arrays of the type Infortrend S16F-G1440. Each disk array exports 3 LUNs of 3 TB in RAID6 for a total of 650 TB. The GDCs and disk arrays are connected by single 4G FC links, while the data movers are connected with dual-8G FC links, matching the 10G Ethernet output and thus enabling a throughput of 1GB/sec per machine.

3.7. The DQM and DA nodes

The Data Quality Monitoring (see Section 4.5) and the Detector Algorithms (see Section 4.4) are amongst the most demanding tasks of the system in terms of CPU and memory. As a consequence, and given the limited physical space available, density was one of the main criteria to choose the machines.

Tests and benchmarks have been conducted on candidate platforms in order to assess their performance and scalability. In addition to standard memory and CPU stress tests, the final software for DQM and DA has been run on the machines with realistic as well as extreme test cases.

The final choice consisted of 64 Nehalem-based blades with 2 CPU's quad-core (Intel® Xeon®, E5450 at 3.00 GHz) ensuring both performance and density.

3.8. The ALICE Control Room stations and displays

The ALICE control room is divided into 2 sections: the main experiment control room and the working rooms used by the detector teams to work on problems and prepare configurations. Therefore the requirements for PCs and displays were also different. Due to the relatively high number of PCs in the rooms we had to choose a compromise between powerful machines and silence, which means few and low-noise fans. For the working rooms the PCs were equipped with dual-head video cards driving two 19" screens to give sufficient display space to work with all control windows. For the main control room the work places had to be equipped with 6 screens, i.e., 3 dual-head video cards per PC. Running X xinerama displays of this size required substantial amounts of memory in the PCs and PCIe video cards to be able to work smoothly. As an additional constraint we had to run RHEL5. We chose to go for an CHIEFTEC 550W chassis, ASUS Maximus IV

motherboards with i5-2500 K CPUs, 4 GB RAM with 3 x16 or x8 PCI-express slots for the handling of the three NVIDIA NVS300 graphics cards.

4. Software architecture

4.1. Dataflow

All the components of the DAQ system are based on a chain of actors, each actor taking care of one specific function performed in one of the components of the hardware system presented in Section 3.1 and shown in Fig. 1. Data are stored locally on the computing nodes in a central data buffer, shared between the actors. Processes running on the same computing node exchange descriptors for the data blocks via single-producer, single-consumer lightweight FIFOs (First-In First-Out). Access to the shared resources is done via memory references re-mapped on each process. Central routines and macros are available to effectively support the above data sharing model (buffer allocation, memory mapping, information exchange, data transfer, etc.).

Well-defined protocols and data exchange models have been defined and implemented to move data from the ALICE detectors and to/from the HLT farm.

The data blocks exchanged are all related to a single particle interaction but can have different scopes: an event fragment at the output of a DDL, a sub-event inside an LDC and an event inside a GDC. All the data exchange protocols are based on data blocks (event fragments, sub-events and events) which are labelled so that they are self-explanatory.

4.1.1. Detector data links

It is mandatory for the ALICE DAQ to be able to identify the data blocks transferred by FEE over the DDL. A common data format (see Fig. 11) has been elaborated and it is used to uniquely mark each single event. The identification of a data block and its processing requires the following information:

- The format version.
- The event identification.
- The trigger information.
- The block length.
- The event attributes.
- The ROI (Region Of Interest) data.

These fields must be present in each data block transferred. The format of the raw data themselves is detector dependent.

Data transferred from the FEE to the LDCs must be formatted as follows:

1. A header describing the associated data block(s), the trigger conditions, the error and status conditions and other information dependent on the readout electronics has to be created while the raw data is collected and must be sent first.
2. One or more data blocks described by the preceding header belonging to the same physics or software trigger, may follow the header.

While the header is mandatory and must be created for all data blocks sent over the DDL, the data blocks are optional and may be skipped in case there is no valid data associated to a given event. All the events associated to the same trigger (physics or software) must be sent over the DDL within the same block and preceded by one header. It is not possible to send two headers with the same trigger identification information.

4.1.2. Readout

The LDCs receive the raw detector data via the RORC. All the necessary software to operate RORC devices on a PC running Linux is included in the two following packages of DATE:

- Package rorc: it contains the Linux driver module, the library functions, and utility programs to configure and have an interface to a RORC device.
- Package readList: it contains the equipment software to readout data from a RORC device. The software depends on package rorc and it runs on a LDC.

The readout process is running in all the LDCs participating in the data taking. The readout process collects data from the front-end electronics by executing code that is specified in a separate software module, called readList. The readlist module consists of the following five routines:

- ArmHw, called at each start of run to perform the initialization.
- AsynchRead, called in the main event loop to perform the readout of the hardware that produces an asynchronous flow of data.
- EventArrived, called in the main event loop to discover whether a trigger has occurred.
- ReadEvent, called in the main event loop after the arrival of a trigger to perform the readout of the hardware.

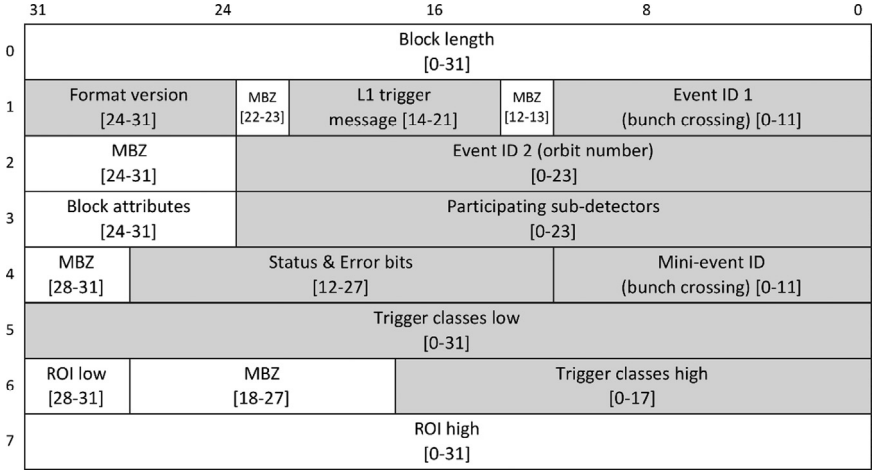


Fig. 11. Common data header format.

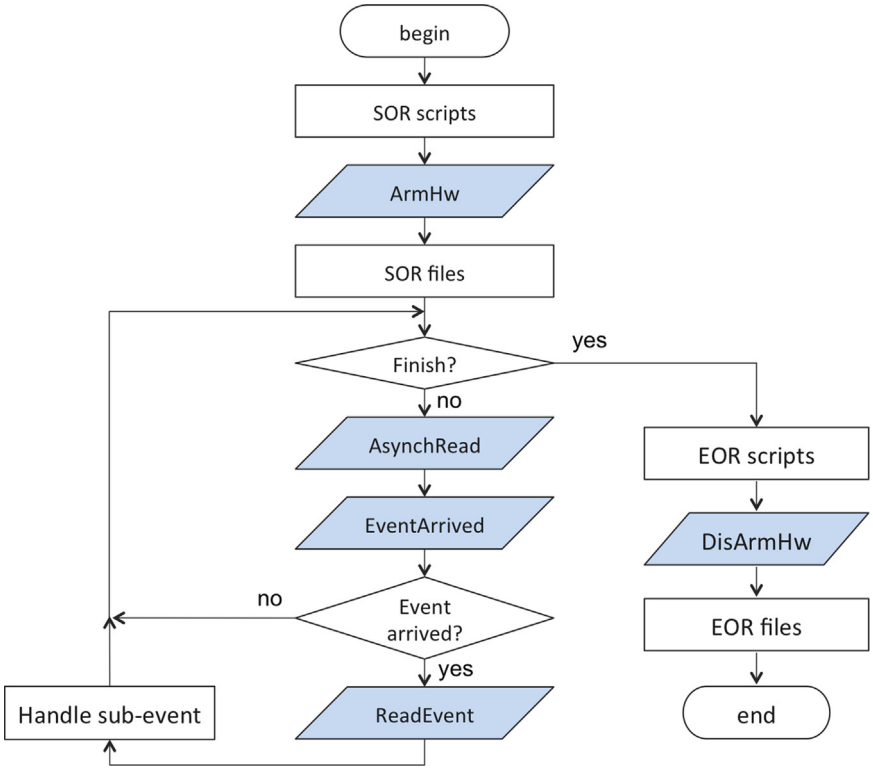


Fig. 12. Main loop executed by readout.

- DisArmHw, called at each end of run to perform the hardware rundown.

Fig. 12 shows the structure of the readout program and how these routines are called in the main event loop.

During the execution of the main loop readout allocates memory for one sub-event. The readout process calls the routine AsynchRead to activate the readout of hardware that produces an asynchronous flow of data. If no events are arriving, the innermost main event loop is executed at maximum speed as long as there is no end of run request or when the timeout to wait for events of type START OF RUN or END OF RUN has expired. Each time an event is present in the buffer, the readout process fills the base event header fields for which it is responsible (including the event time stamp to tag the sub-event), and it increments the variable number of sub-events for all event types. Then the routine ReadEvent is called, which is in charge of transferring the event data and for filling the base event and equipment header fields. Afterwards the readout process performs the following operations in the order described in Fig. 13.

The readout process exits the main event loop, if one of the following conditions is met:

- The maximum number of events to be collected has been reached.
- The maximum number of bytes to be collected has been reached.
- The arrival of an end of run request combined with the following three cases:
  - The parameter startOfData/endOfData event enabled is not set, hence there is no waiting for an event of type END OF DATA.
  - The parameter startOfData/endOfData event enabled is set and an event of type END OF DATA has been received within the timeout.
  - The parameter startOfData/endOfData event enabled is set and an event of type END OF DATA has not been received within the timeout.
- An event of type START OF DATA has not been received within the timeout when the parameter startOfData/endOfData event enabled is set.

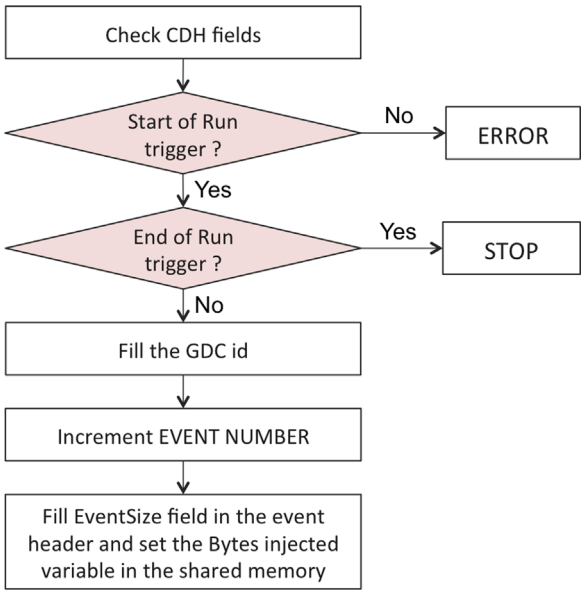


Fig. 13. Readout operations state machine.

The readout program accesses the hardware by calling the five routines of the readList module, which contains the code specific to a given electronics setup. Instead of writing several of these modules, the generic readList concept allows to group the code for

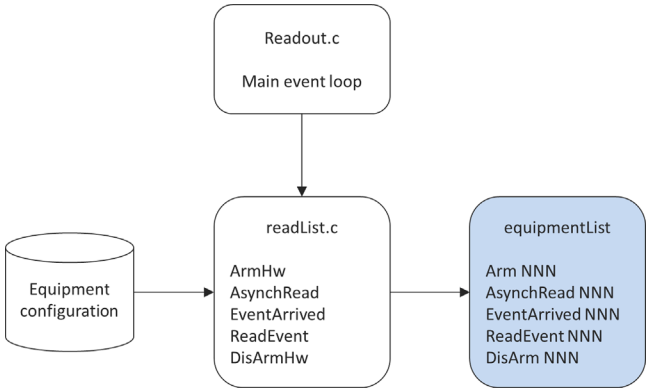


Fig. 14. The generic readList concept.

all the electronics setups in another library called equipmentList (see Fig. 14).

The readout software specific to an electronics setup can be written separately for a so-called equipment. One equipment is responsible for generating data from an electronics board or a set of electronics boards, depending on how the readout software is structured. A set of equipment-handling routines deals with one single equipment, thus the code is more modular and readable. All the five routines must be implemented for an equipment. If one of these functionalities is not required, a dummy routine should be provided. The equipment configuration defines the equipments used for each LDC. It is done with the equipment databases. An equipment may be repeated several times in a detector; each run-time call will be distinguished by a different set of equipment parameters. The configuration file specifies the selection of the active equipments and the setting of the parameters that will be passed to the readout routines. Therefore, it is possible to modify the readout program behavior without changing the readout executable code. As a result of this generic readList concept, the sub-events of an LDC are divided further into smaller parts, called equipment data or fragments. Each equipment data block begins with an equipment header, followed by the equipment raw data. Various LDCs contain different fragments from the equipments, but the LDCs have only information of the sub-events coming from a detector. To build a complete event all the LDCs have to send data to the GDCs, that collect all the sub-events and build the complete event for a specific trigger. LDCs are connected to the GDCs through a GbE network using TCP/IP protocol to exchange data. It is important that one GDC collects all the sub-events belonging to the same trigger to build properly one full event. A round-robin algorithm has been implemented in readout to share equally the data throughput from the LDCs to all the available GDCs. By using the ORBIT NUMBER of each sub-event it is possible to address a different GDC-ID for every trigger.

4.1.3. Recording

Data recording is performed via asynchronous, single or multiple data channels. Both the LDCs and the GDCs can write the data into one or more data files hosted on locally mounted disks in raw DATE format. GDCs also support AliRoot (ALICE Offline software package) [6] format. The latter is possible only on the GDCs. Furthermore, the LDCs can send their data to one or more GDCs via TCP/IP sockets using an optimized push protocol. A special recording mode, used for debugging and profiling, sends the data to a null device where it gets disposed with the minimum possible delay. All recording modes support multiple channels served asynchronously with a starvation-safe policy.

When the LDCs send their data to the GDCs, the destination node is selected from the list of active GDCs solely as a function of the content of the events.

4.1.4. Event building

Event building takes place in the GDCs. A single-threaded process, named eventBuilder, receives the sub-events from the LDCs, stores them in local memory and decides, as a function of the event type, trigger classes, and detector pattern associated to the event if this has to be built and which contributions are expected. This procedure takes also into account a possible decision from the HLT system that may request to drop selected parts of the event (to be either discarded forever or replaced with compressed data). Whenever an event is declared as complete, it is sent to the following stage which can be either local recording or streaming. A sample of the complete events is also duplicated to the monitoring streams as requested by the setup of the DAQ System.

4.1.5. Streaming

At the output of the eventBuilder is the mStreamRecorder process which takes care of the objectification and streaming procedures (together known as *rootification*). The raw events are dispatched to a set of worker processes that create encapsulating AliRoot objects and store them into ROOT files [7], ready to be migrated to the PDS. The usage of multiple worker processes maximizes the CPU occupancy of the GDC without noticeable run-time penalties for the overall procedure.

4.1.6. Migration

The ROOT files created by the streaming processes are stored on a pool of disks (TDS) located at the ALICE experimental area. This pool ensures optimal writing recording throughput and allows a discrete autonomy, about one LHC fill, in case of temporary failures within the data migration process. Data recording has full priority over all the other operations involving the TDS and no other activity must take place on the volume(s) selected for writing. Once all their associated files are closed, the data volumes are queued for migration to the CERN Computer Center which is located several kilometers away from the experimental area. The migration takes then place between the TDS and the PDS driven by a pool of dedicated Mover nodes.

4.1.7. Monitoring

At the LDCs and GDCs level, a sample of the data flow is duplicated to various monitoring streams, to be used for the DQM and to run DAs. Monitoring clients can specify which events they have to monitor (based on parameters such as event type, trigger patterns, event attributes, etc.) and how many events they need (from few events requested for example by an event display, to most of them as for the case of rare calibration events required by specialized DAs). The selected events are then forwarded to the active monitoring clients by actors running on the LDCs and on the GDCs.

A pre-defined portion of the raw data stream is moved “as is” through the data chain without taking into account the decision coming from HLT. This allows on one side monitoring of the original, unprocessed data from the clients that need it and on the other hand provides on the output data files a significant sample of the source events and the associated HLT information (decision and replacement data) for post-run verification checks.

4.2. Control

4.2.1. Run control

The ALICE DAQ system allows parallel, independent data acquisitions: different detectors can, for example, collect calibration data at the same time.

Every data acquisition, performed for one single detector or a group of detectors defined by the ECS (see Section 4.2.2), is controlled by a runControl process that steers the data acquisition according to operator commands. Several runControl processes with different names can run at the same time and control different data acquisitions. The architecture of the runControl system is shown in Fig. 15.

Every runControl process has a runControl interface based on SMI++ Finite State Machines [8]. The interface receives all the commands sent to the runControl process and rejects those incompatible with the current status of the process. The interface also guarantees that, at any time, the source of commands is unique. It can be a runControl Human Interface or a component of the ECS.

Many runControl Human Interfaces can coexist for the same process, but at most one at a time can have the mastership and can be used to send active control commands, whereas the others can only be used to get information. When the authorized source of commands is the ECS (see next subsection), none of the runControl Human Interfaces can send active commands: this possibility is restricted to the ECS.

When the list of LDCs and GDCs to be used for a given data acquisition is defined, the runControl process spawns a Logic Engine process. The Logic Engine contains all the logic about starting and stopping the different processes on the different machines. The Logic Engine translates operator commands into sequences of commands that are then sent, in parallel, to the remote machines.

On every remote machine a process, called rcServer, starts and stops processes according to the commands it receives from the Logic Engine. The rcServer also performs some local error handling and returns various counters and information to the other DAQ processes (e.g. to the runControl Human Interfaces). A rcServer can be used, at different times, by different runControl processes and can therefore receive commands from different Logic Engines in the context of different data acquisitions.

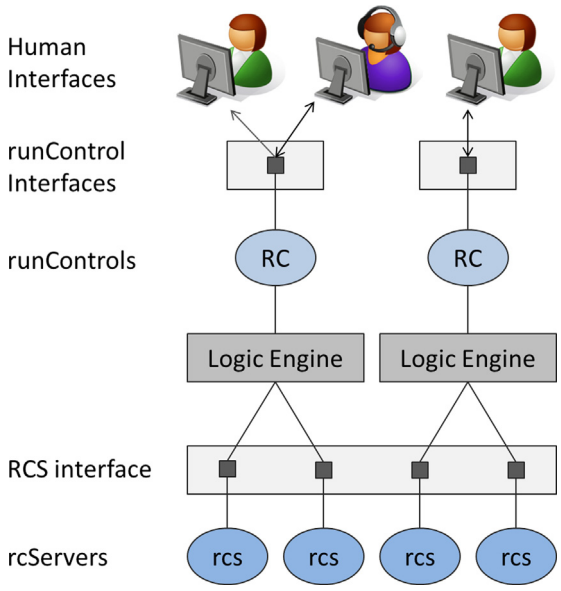


Fig. 15. Run control architecture.

An interface, common to all the rcServers, guarantees that every rcServer is used at any time by at most one runControl process and receives commands from one Logic Engine in the context of one and only one data acquisition. The interface is called RCS interface.

All the runControl processes, the runControl interfaces, the Logic Engines, and the RCS interface run on a central server. The runControl Human Interfaces are started by shifters and detector experts on different machines of the ALICE control room. The rcServers run on LDCs and GDCs.

4.2.2. ECS

The Experiment Control System is a layer of software on top of the online systems operating with the particle detectors in different domains: DCS, DAQ, TRG, and HLT.

The ECS allows parallel operations of individual detectors operated in standalone mode and of groups of detectors called partitions. In all the cases the ECS synchronizes the online system getting information from them and sending commands to them via interfaces made of SMI++ Finite State Machines [8].

The main components of the ECS are the Detector Control Agents (DCA), the Partition Control Agents (PCA), the DCA Human Interfaces (DCA HI), and the PCA Human Interfaces (PCA HI).

There is one DCA per detector to control all the standalone activities of the detector and one PCA per partition to control the Physics and Technical data taking runs performed with the partition. When the partition is not taking data, the PCA allows parallel calibrations runs of its individual detectors.

DCA HIs and PCA HIs allow to get information from and send commands to the DCAs and PCAs respectively. For a given DCA many DCA HIs can be started at the same time but at most one can send commands to the DCA. Similarly at most one PCA HI can send commands to its associated PCA.

Fig. 16 illustrates the ECS architecture with an example with three detectors, one of them being operated in standalone mode and the other two as part of a partition.

In addition to its main activity (i.e. the synchronization of the online systems to perform runs), the ECS interacts with other components of the ALICE software. In particular:

- Sends to the Alice Configuration Tool (ACT, see Section 4.2.3) requests to lock/unlock configuration items to prevent configuration changes during runs.
- Stores in the ALICE eLogbook information about all the performed runs.

4.2.3. ACT

The ALICE Configuration Tool (ACT) serves as a configuration repository to which the different ALICE systems can access to extract their currently selected configuration. As shown in Fig. 17, the ACT is operated both by the Run Coordination and by the different system experts via a Web-based Graphical User Interface (GUI). A relation database serves as a data repository and an Application Programming Interface (API), implemented in C, provides numerous functionalities to the different components.

A publish/subscribe mechanism, based on the Distributed Information Management system (DIM) [9], is also available. Two dedicated modules, running as daemon processes, use this mechanism:

- *ECS Dedicated Daemon* (*EDD*): interacts with ECS, DAQ, CTP, and HLT pushing the selected configurations for the online systems.
- *DCS Dedicated Daemon* (*DDD*): interacts with the Run Control Tool (RCT) pushing the selected configurations for the different

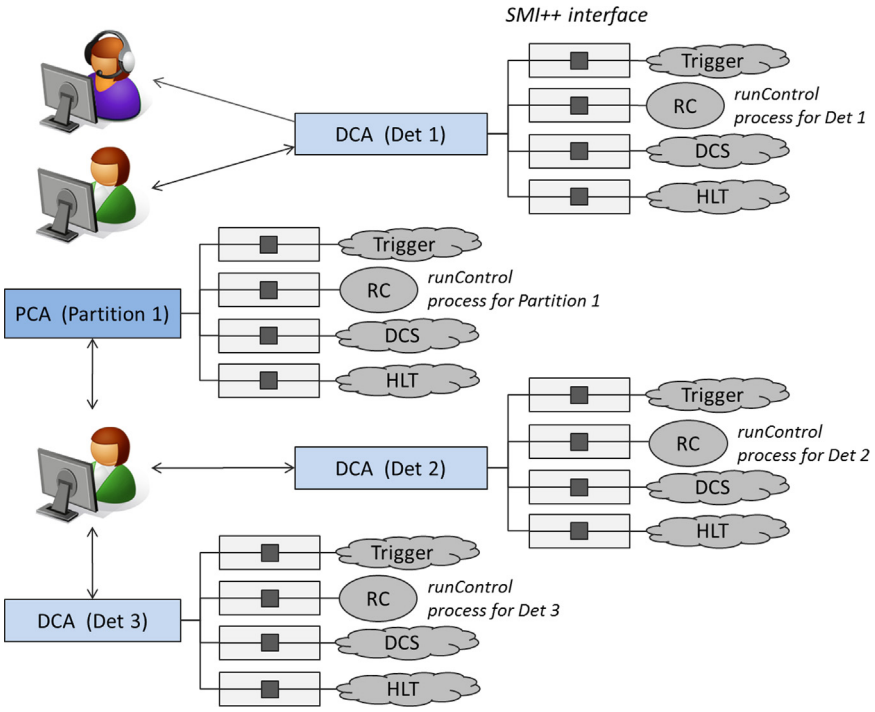


Fig. 16. ECS architecture.

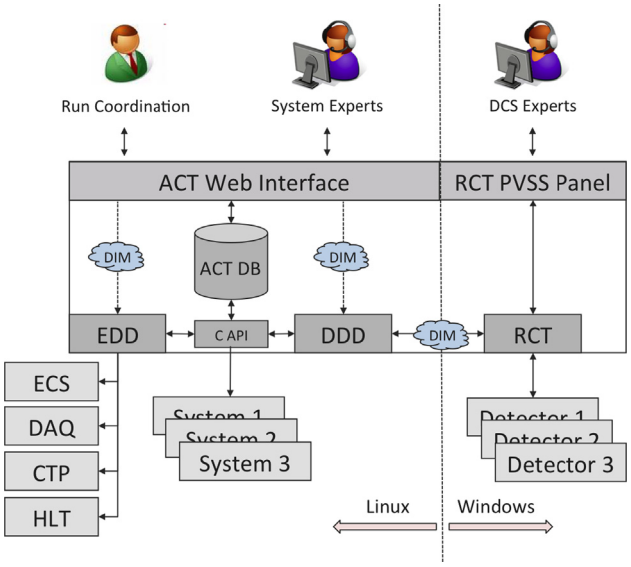


Fig. 17. ACT architecture.

ALICE detectors. The RCT then makes the configurations available to each individual DCS where the configuration is applied.

The two above daemons run on a Linux server, whereas the RCT and the DCS of all the detectors run on Windows servers.

In order to define the different systems and detector components to configure, the ACT introduces the following concept:

- *System*: an ACT system represents a physical or logical element of the ALICE experiment. Each system normally has several configurable components. Examples of system are detectors, online systems, and ECS partitions.
- *Item*: an ACT item corresponds to a configurable component of a specific ACT system. Each item normally has several possible defined configurations. Examples of item are 'partition PHYSICS_1 HLT Mode', 'TPC DCS configuration', and 'CTP L0 inputs'.
- *Instance*: an ACT instance defines a possible predefined configuration for a specific ACT item. At any time only one instance can be activated for each item.

A locking mechanism prevents the configuration of items that are being configured or used by an online system (e.g. a detector being part of a running partition). For configuration, the items are locked by the corresponding daemon (EDD or DDD). If being used by an online system, the items are locked by that system.

At a given time, each item is in a specific state, represented by its activation status. There are four possible values:

- *update requested*: a configuration has been requested for the item.
- *applying*: a configuration is being applied to the item.
- *active*: the item is configured as requested.
- *update failed*: an error occurred when configuring the item.

As seen in Fig. 18 the ACT workflow starts with the user selecting the desired configuration via the GUI. When finished, the user submits an update request which changes the activation status of the selected items to update requested. This action triggers an update of the ACT_UPDATE DIM service provided by the ACT Update Request Server daemon.

The EDD daemon reacts to the service change and handles the items related with ECS, DAQ, HLT, and CTP: it locks the items, sets their activation status to applying, interacts with the systems, sets the activation status to active or update failed and finally it unlocks the items.

The DDD handles the items related with DCS and performs the same steps. The main difference is the fact that DDD does not interact directly with the DCS of the detectors but only with the RCT that provides a standard interface for all the detectors.

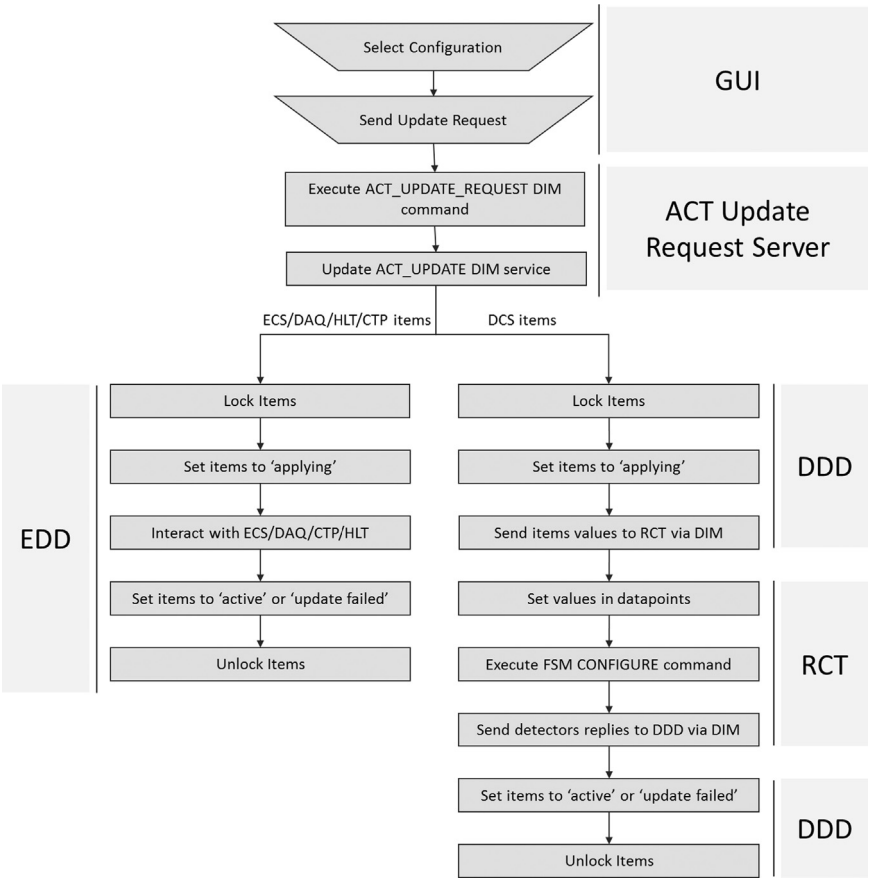


Fig. 18. ACT workflow diagram.

4.3. Monitoring and metadata

4.3.1. Lemon

The ALICE DAQ system used Lemon [10] to monitor the status of its hardware and software components, to control the behavior of the acquisition system and to provide adequate feedback to developers and run coordinators, both for global and specific subsystems. A MySQL-based scheme was installed, tailored for the ALICE-specifics requirements, and maintained throughout the whole of LHC Run 1, across evolving hardware (HW) and software (SW) components as driven by the natural evolution of the system being monitored.

Metrics: The ALICE/DAQ specific Lemon sensors could monitor very specific entities such as DDL traffic, absorbed power values, power outlet status, and ambient temperatures. One central machine was used to collect data from "dumb" units such as Power Distribution Units and Hard Disks. An example of the way these metrics were sampled and presented to the ALICE DAQ operator is shown in Fig. 19 where we can see, for a time span of 3 weeks, the sampled values for network traffic at the level of the LDCs. The difference between the data rates on the DDLs and the Ethernet output traffic (the sum of eth0 out and eth2 out) is mainly due to the data reduction achieved via the HLT: data is read from the detectors and then dropped at the level of the LDCs, as driven by the HLT runtime compression procedure.

Alarms: Several specific alarms could be handled by the DAQ Lemon sensors . Checks ranged from system mis-configurations, errors in the software installation, unresponsive daemons, hardware errors and many other abnormal conditions. Full system checks were scheduled daily (randomly distributed within a 10-minutes wide time window, in order to avoid global overloads) while checks on more critical or dynamic conditions were run

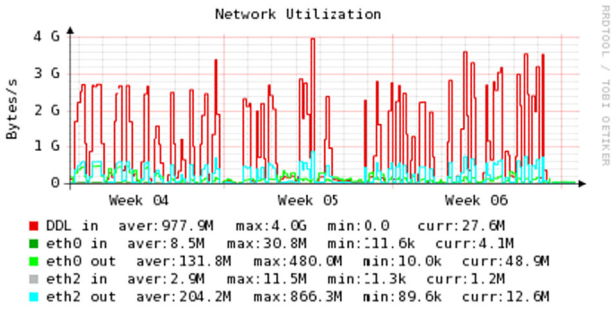


Fig. 19. Network traffic on the LDCs as shown by Lemon.

more often (up to several times an hour). Most of these alarms went directly to the DAQ developers, but few of them would also alert the ECS/DAQ operator as an assisting tool for detection and recovery of run-time abnormal conditions. One of the most typical alarms of the second type was raised whenever the DAQ and/or the HLT were found blocking (via XOFF) any of the ALICE detectors for example due to temporary peaks in the readout data rates.

Alarms were seldom removed from the DAQ lemon sensor. Once a new alarm condition was identified, a specific check was defined and added to the existing ones. At the end of its support cycle, the sensor implemented about 70 ALICE/DAQ specific alarms, many of them run in multiple instantiations (e.g. a disk full alarm would be exercised on multiple file systems).

Experience: We were quite satisfied with the capabilities of the Lemon suite. Although developed for monitoring of non-realtime computer farms, the tool could be effectively used for the online computing of an LHC experiment such as ALICE. At the end of its lifecycle, the Lemon suite active at the ALICE/DAQ site was

configured to monitor around 800 different entities (ranging from single nodes to clusters of hundred of nodes) for about 100 different metrics. More than 500 GB of historical samples were stored on local storage for the period between 2011 and 2013.

4.3.2. Infologger

The infoLogger package provides facilities to generate, transport, collect, store and consult log messages from all distributed software components running on the ALICE DAQ system.

It provides an interface to inject logs, a central repository to store the messages, and user interfaces to display and query them.

Architecture: Fig. 20 shows the overall architecture of the infoLogger system. A process calling a function of the infoLogger library sends the message to the local infoLoggerReader daemon. This process collects all the messages of the node where it runs, and sends them to a central infoLoggerServer daemon, which stores the received messages in a database. The infoBrowser user interface allows to read messages, either stored in the database or received online by the central server.

Log message structure: Each log message handled by the infoLogger system is structured by an extensive set of attributes. Some of them are provided by the client creating the log message, others are set automatically by the infoLogger API when it is called. Each log message consists of the following fields:

- Severity: the information level of each message. This can be one of *Information* (messages concerning normal running conditions), *Warning* (to report a condition which could be the source of problems), *Error* (when an abnormal situation has been encountered), and *Fatal* (when an unrecoverable situation has been detected, usually causing the end of the current run).
- Level: an integer used to prioritize messages, essentially based on their target audience. Typically, it allows to tag messages (from higher level to lower level) for operators, DAQ support team, DAQ developer, and debugging information. A range has been defined for each of these categories, so that a finer granularity is possible for further ordering within a category.
- Timestamp: time of the message creation, with microsecond resolution, as provided by the local operating system where the message is created.
- Host name: host where the message was created (physical name of the machine).
- Role name: DAQ role where the message was created (logical name of the DAQ entity running the process, e.g. the name of a LDC or GDC).
- Process ID: operating system identifier of the process creating the message.
- User name: user running the process creating the message.

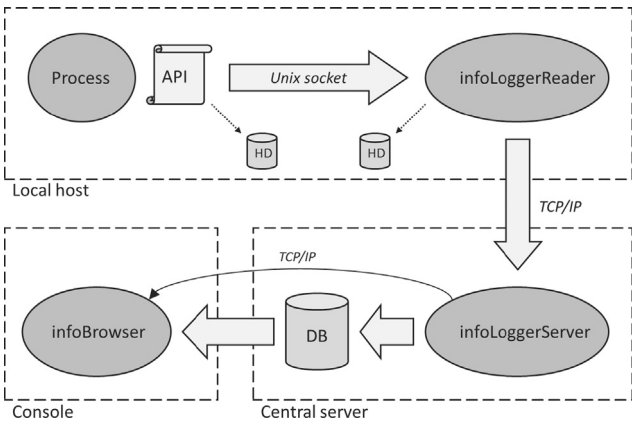


Fig. 20. Infologger architecture.

- System: system originating the message. For all DATE processes, this is set to DAQ. As the infoLogger facilities are shared with other systems, it can be also set to ECS, TRG, HLT.
- Facility: the activity family, usually the DATE package name creating the message. This can be for example readout, recorder, runControl, or operator in case of a message coming from the command line.
- Detector: the name of the detector associated to the running process.
- Partition: the name of the global partition associated to the running process.
- Run number: the run number associated to the message, if any.
- Error code: an error code to tag the message. This is used to point to appropriate documentation and procedures for the operator. This also allows to make statistics on error occurrences, even if the message content is different (for example when variables are printed in the message).
- Source line: line number where the message is issued in the source file.
- Source file: the name of the source file issuing the message. It allows (with previous field) to easily trace which part of the code issues the message.
- Message: the log information. It is a text string.

The client process calling the API typically provides the severity, level, error code, and message. All other fields are usually inferred automatically from the compiling and runtime environments. Although some of the fields are associated with specific ALICE DAQ semantics, the infoLogger is a general purpose logging system and can be used in other environments (with or without modifying the list of fields). It has for example been adopted for the HLT as well. The infoLogger is already available as a standalone package, and it is also planned to fully decouple the infoLogger from DATE bindings in order to ease its dissemination.

Implementation: The infoLogger is written in C, and the native client API is provided in C. Extensions for other languages (e.g. Tcl/Tk) are generated using SWIG [11]. The client API provides a set of functions to send messages. They are eventually all used for the same purpose, but differ in the calling arguments depending on the fields being set by the client. Some functions also allow to globally define once for all some of the fields for a running process, in order to simplify further infoLogger calls. Finally, a few control functions allow to define infoLogger parameters for the process, like timeouts and message filters settings (e.g. to drop messages if their level detail is too high). The API has a built-in protection to detect messages flood, when a client sends too many messages per second (which can for example occur if there is a bug in a loop). Corresponding thresholds are adjustable, and default values are to max 500 messages in a second, or 1000 messages in a minute. Logs are afterwards redirected to a local file (with a maximum size, after which they are dropped). Normal operation resumes if the flow goes back to acceptable levels. The API provides also the possibility to log to a file (instead of the standard server) for verbose processes which logs do not need to be centrally accessible. It includes auto log rotation based on file size or number of messages, with different file formats (showing different subsets of infoLogger fields). The client package also provides a command line tool used to pipe to infoLogger the output of processes (mainly shell scripts) which are not using the infoLogger API.

The infoLoggerReader is started at boot time on all machines. If the infoLoggerReader process is not reachable by a client process (i.e. the initial issuer of the log message), the client process tries to start a new instance of the infoLoggerReader. If this fails, messages are written to a local file on disk to avoid loosing them. Connexion between local client and reader is done through a named pipe.

Messages collected by infoLoggerReader processes are stored in a persistent disk FIFO, which ensures their (delayed) delivery to the central infoLoggerServer in case it is unavailable at a given point in time. The infologgerReader ships data to the central server by a TCP/IP link with custom protocol. The list of fields has increased with time and needs. Protocol has been adapted to be easily extended, and includes versioning to allow decoding of different versions by the same server (e.g. when some clients still run an older version of the infoLogger library).

The infoLoggerServer runs a thread waiting for new infoLoggerReader connexions and reading out the existing sockets. Messages collected are then inserted in a common message FIFO, for insertion in a relational database table by one or several (number can be configured) independent threads. It also forwards the incoming messages to a dispatch thread, which duplicates them to online subscribers (as described later). The database is implemented with MySQL, and the table structure follows the message structure (i.e. one column per message field). Insertion is done with MySQL C API and prepared statements. All fields are indexed to optimize query performance, although it considerably increases the database size on disk. The infoLogger message table is partitioned by hash on the timestamp column (with a division factor, so that there is one partition per day or so, i.e. keep less than 1 million messages per partition) in order to speed-up query performance: most queries being based on time, MySQL effectively implements partition pruning and does not scan partitions outside the time range. Otherwise, query performance may degrade as table size grows over few millions of rows. The messages table is manually archived every few months to keep it at a reasonable size. The table is simply renamed, and a fresh blank 'current' messages table created. Archived tables are still accessible for queries.

Because of the buffers and many-to-one links in the chain, the order of message insertion is not guaranteed, only the message timestamp (set at earliest point in the chain by the client API) is reliable to order messages coming from a given machine. The accuracy of clock synchronization is critical when correlating events from different nodes, and is ensured by a classical NTP (Network Time Protocol) setup for all the DAQ machines.

The display of the messages is tackled by the infoBrowser human interface. It works both in offline and online mode. In offline mode, it queries the messages available in the database. In online mode, the infoBrowser connects by TCP/IP directly to the infoLoggerServer, which dispatches a copy of all incoming messages to the subscribers. In both modes, the user can define filters based on the message fields content, in order to select or exclude what messages should be displayed. In online mode, the filtering is done in the browser, i.e. the server sends the full log data stream. The infoBrowser is available as a Tcl/Tk application used in the runtime environment, and as a web interface (PHP/HTML) to ease remote (and secure) access to the ALICE DAQ logs. The web interface only allows offline queries to the database. The infoBrowser has also a direct connection with the DAQ Wiki for the resolution of problems: when an error is reported, the corresponding documentation is reachable from a single click in the user interface. The assignment of error codes is maintained with the infoLogger source code in CVS (Concurrent Versions System). Each DAQ package is assigned an error code range, and documents new error codes in a central file (together with the procedures in the Wiki, in structured pages providing for each error code: component, description, urgency, operator and expert actions to be taken, support procedures to be followed, who should be contacted, link to extra documentation, etc).

Operation: In production, the infoLoggerServer receives logs from 350 infoLoggerReader processes, and dispatches them to around 40 online infoBrowsers. During the 2012–2013 data taking

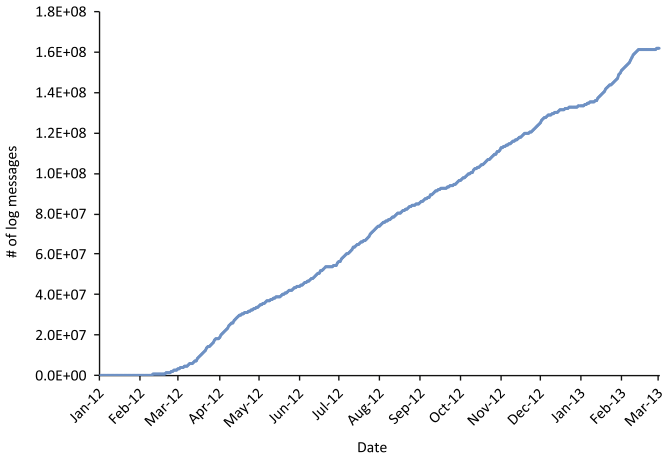


Fig. 21. Number of log messages collected over time by infoLogger.

period, it collected in average 400,000 messages per day in average (800,000 per day during most active periods like the heavy ion run, with daily peaks up to 1.6 million messages in the same day). In total, more than 160 millions of messages have been recorded, as seen in Fig. 21. This amounts to 83GB of MySQL data files, including indexes which account for 53 percent of the log database size.

Given the huge number of log messages received (several tens of thousands per run), a review was done to propose solutions in order to reduce the number of messages, at least for the shifter. These recommendations were implemented in 2012, in particular the improved tagging of messages in all DAQ components. This ensures that the DAQ operator receives an acceptable number of messages (about a hundred in a normal run) by proper filtering. In order to allow developers to keep their debugging messages, they are by default filtered out for standard operations.

The log API overhead has been measured around 4 microseconds to inject a log message from client to reader, on a standard desktop machine. The latency for a single message log issued from the command line to display in the infoBrowser is around 1 s, and is the same for a burst of 1000 message (all 1000 messages are visible in the infoBrowser within the same second).

Insertion speed in the database reaches 6000 messages per second, with a partitioned InnoDB table and all indexes. For comparison, the insertion speed is 9000 messages per second without indexes, and reaches 12000 messages per second with MyISAM storage engine and no index.

Good query performance is ensured by the multiple indexes, at the cost of (affordable) increased disk space (and slight insertion time overhead). Taking a random day (by defining timestamp upper and lower limits in the SQL 'where' clause) with around 800,000 messages, the following performance is observed (for uncached queries): scanning (e.g. count all rows) the full logs takes less than 0.5 s; more complex SQL operations, like grouping by facility or host, take less than 2 s; retrieving the full data (about 200MB) is done in 8 s (or a read speed of 100,000 rows per second). Complex queries to extract yearly statistics on the full log data set take more time, but results are available within minutes.

Therefore, the system evolved since data taking started and scales according to the needs.

4.3.3. Orthos

The ALICE DAQ system uses hundreds of hardware and software components susceptible to fail. Abnormal situations have to be detected, advertised, and fixed rapidly in order to maximize the

experiment data taking time. Initially (until 2012), information about such events was available from various sources, and handled mostly manually. Therefore, a common tool was needed to collect and process them. Orthos is the alarm system developed to detect, log, report, and follow-up abnormal situations on the DAQ machines at the experimental area.

Orthos provides a unified interface to raise alarms from the various DAQ components, and means for the DAQ actors (operators and experts) to handle them. It brings together information and action flows. Orthos usually does not make measurements itself, but is rather fed from existing monitoring metrics and status reports. Orthos is a layer on top of all sources of monitoring information to make sure that appropriate actions are taken when needed, and following uniform procedures independently of the source reporting an abnormal situation.

It improves the status visibility of the DAQ background components, in comparison with those actively handled by the DAQ control flow and for which failure feedback was already extensively available through the control and log human interfaces.

Orthos integrates alarm detection and notification mechanisms with a full-featured issues tracker, in order to prioritize, assign, and fix system failures optimally.

We define as alarm an event to be detected and followed-up by Orthos. It corresponds to any abnormal situation, harmless or critical. It may (or not) require a preventive or corrective action, immediately or at a later point in time. It might also be a simple event notification, to be aware of if something else happens.

In Orthos, all alarms of the same kind are globally defined by an alarm class, which describes a general behavior (alarm handling and follow-up actions). At run time, multiple occurrences of the same alarm class can be raised, typically by different components. Each occurrence of the same alarm class is an alarm instance, and is uniquely identified by its alarm class name, a source (where it comes from, i.e. node or device name), and an optional key (which refines the context in which it happens). For example, the alarm class 'disk space low' would have as key the 'disk partition name', and an alarm instance could be the tuple (disk space low, mypc123, /tmp).

Fig. 22 represents the alarm workflow.

Components of the DAQ system report to Orthos metrics. A metric is a Boolean value defining the condition of an alarm instance at a given point in time (1=active, 0=inactive). It is the result of an external measurement of some environmental conditions which summarizes if there is an abnormal situation or not. Metrics are inserted periodically or upon status change to Orthos. The logic engine processes each new metric measurement, archives it, and updates the corresponding alarm state machine accordingly in a persistent repository. Successive measurements (in time) of the same tuple (class, source, key) trigger a status

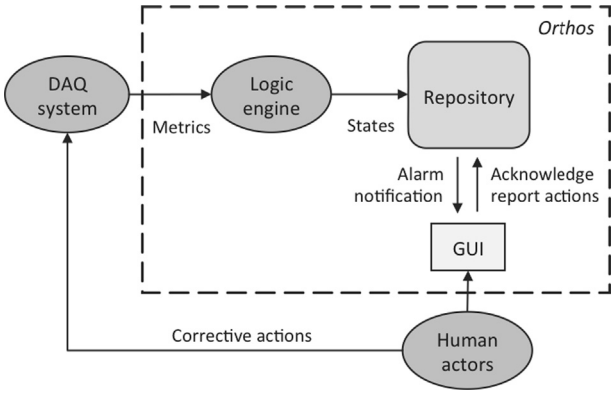


Fig. 22. Orthos alarm workflow.

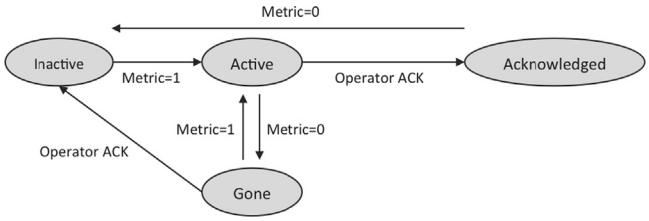


Fig. 23. State machine describing the life-cycle of an alarm in Orthos.

update of the same alarm instance state machine. This state machine describes the alarm follow-up, as shown in Fig. 23).

An alarm instance is initially in the inactive state. When a positive alarm condition is met, it goes to active state. It goes back to inactive only after the operator has acknowledged it in the GUI, and the alarm condition disappeared (in any order, hence the intermediate 'acknowledged' and 'gone' states). This ensures that no error is left unnoticed, even if it was transient.

The operator being notified by Orthos can then take corrective actions if necessary (which eventually will clear the 'alarm active' when the metric is next sampled). He will also report actions taken or open a ticket in the tracking system, which issues are cross-referenced in the Orthos repository for each alarm instance.

Also, Orthos can be configured to notify by e-mail or SMS alarms which are left unacknowledged too long, in order to make sure actions are taken if nobody is checking the status display.

The core of the repository is based on a MySQL database server. Measurements are inserted in a table. The logic engine runs as a trigger procedure upon reception of new metrics, and updates the corresponding alarm states in a separate table. No additional process is needed: the logic runs directly in the MySQL server. The GUI interacts with the repository by means of SQL select queries and procedure calls to update alarm states. The tracker feature is implemented in JIRA [12], this tool being used already for ALICE DAQ development and operation reports. The GUI interface is written in PHP [13] and generates HTML code with CSS for the layout, and published by an Apache web server [14]. PHP interface to JIRA is done using Representational State Transfer (REST), i.e. HTTPS/JSON POST/GET methods, to gain full read/write access to the tickets. Authentication and corresponding access rights are granted using Shibboleth [15] and the CERN central single-sign-on service [16]. Fine-grain permissions are easily defined by creating and populating corresponding E-groups [17], which are list of users created on the CERN central services, and for which properties (e.g. check if a user belongs to a given group) are available through SOAP Web Services. A distributed sensor is installed on all DAQ nodes for some basic measurements. It is indeed sometimes more convenient to re-implement some trivial system measurements (e.g. "disk full") to avoid being dependent on the existing monitoring tools in use, which were in the process of being changed at the time Orthos was initially deployed. We prefered to have a stable source of measurements during this transition phase. A C++ class provides easy sensor implementation for custom measurements, with built-in configuration and monitoring loop. A C API provides simple metric injection function to insert health reports from existing running processes (e.g. DQM). The corresponding Orthos binaries are distributed as separate RPM [18] packages (devel, shared, www, repository, and sensor). Documentation on alarm classes and corresponding issue-fixing procedures are available in the DAQ wiki (implemented with TikiWiki [19]), and information directly linked from the human interface for easier access.

Effort has been put on simplicity of the implementation in order to minimize development time, and validate the initial concept. We tried to apply a maximum of existing tools and

technologies already used in our group. We had no down time in production in 2012 for the 350 nodes monitored, and 10^8 measurements received without significant system load. More details as well as example metrics can be found in a dedicated article about Orthos [20].

4.3.4. ALICE electronic logbook

Introduction: In large scientific experiments, it is essential to have a bookkeeping facility that keeps a record of the experiments operational activities. As shifters come and go, a central information repository is needed to store reports of incidents, configuration changes, achievements or planned operations. Furthermore, an historical record of data-taking conditions and statistics is needed not only to allow the selection of good run candidates for prioritized offline processing but also to detect trends and correlations, create aggregated reports and assist the run coordination in fulfilling the scientific goals. In ALICE, these requirements are implemented through the ALICE Electronic Logbook (eLogbook), a custom-made application developed by the ALICE Data Acquisition team and in production since August 2007.

Architecture: The eLogbook architecture is based on a LAMP (Linux, Apache, MySQL and PHP) software stack, with the relational database (DB) serving as a data repository and the web-based Human Interface (HI) providing interactive access to members of the ALICE collaboration. As shown in Fig. 24, the different ALICE online subsystems access the DB via a C API and the shifters and members of the collaboration via a web based graphical user interface. A daemon process collects information published by the DCS and the LHC and stores it in the DB. A REST API is also provided to allow non-interactive access to the eLogbook repository.

Database: The DB, running on a MySQL 5.5 Community Server, is used to store heterogeneous data related with the experiments activities. InnoDB is used as a storage engine for its support of both transactions and foreign keys constraints. The tables that compose this DB can be grouped into four different categories:

- Run centric: related to a specific run.
- Fill centric: related to a specific LHC fill.
- Log entry centric: related to a specific human or automatic text report with optional file attachment.
- User centric: related to the users of the HI.

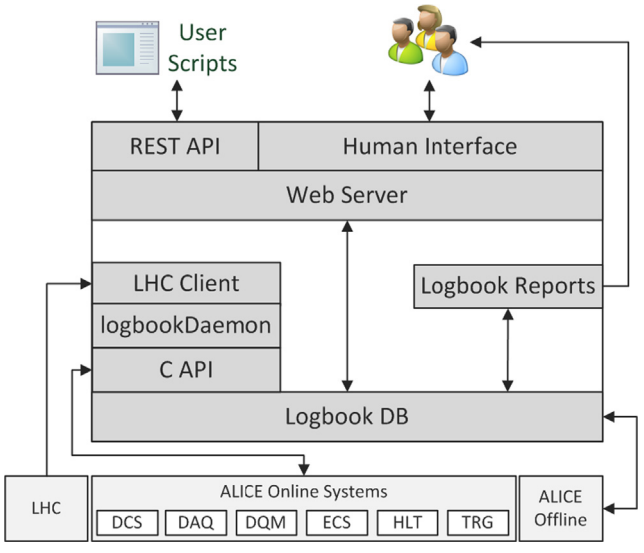


Fig. 24. eLogbook architecture.

Stored procedures are executed periodically to update the different global counters in the eLogbook tables whose value depends on partial counters spread throughout several tables. Daily backups are performed to both a RAID 6 disk array and the CERN Advanced STORage manager (CASTOR).

CAPI : The eLogbook C API allows the different ALICE subsystems to access the DB in order to either store or read data. It is ported to other languages (like Tcl) using the SWIG tool and distributed both with the DATE software and as a standalone package.

REST API : The eLogbook REST API is an HTTP based RESTful API that interacts with the eLogbook via the HTTP protocol. This makes it platform independent, allowing access to the eLogbook repository from any platform capable of performing HTTP requests and receiving the subsequent responses. Implemented in PHP and based on the Zend Framework [21], it allows a limited set of operations:

- POST fill-luminosity-scan: insert an LHC Fill Luminosity Scan image. Since it is used to insert a binary file, the request content type must be multipart/form-data.
- POST log-entry: insert a Log Entry (including file attachments). Since it can be used to insert one or more binary files (the Log Entry attachments), the request content type must be multipart/form-data.

Authentication and authorization are operation and HTTP method dependent. Some operations are only allowed from predefined IP addresses. Others require HTTP Basic Authentication over SSL.

Logbook Daemon: The Logbook Daemon is a daemon program that runs constantly on the background and, at start of run, extracts data concerning the ALICE magnets and the LHC configuration published by the DCS using the DIM system and inserts it in the DB. Additionally, it also provides a publish mechanism (via DIM) to notify run events (Start Of Run/End Of Run for partitions and sub-detectors), thus avoiding the need from the different online subsystems processes to constantly poll the DB.

LHC client: The LHC publishes multiple operational parameters that are extremely relevant for the data taking conditions bookkeeping. Therefore, a dedicated program called LHC Client was developed to collect these parameters and publish them in the DCS DIM Name Server, from where they are extracted and inserted in the DB by the Logbook Daemon process. Additionally, certain parameters (especially for the LHC fills) are inserted directly by the LHC Client. More details can be found in Section 4.3.6.

Logbook reports: To support the daily operations of the Run Coordination and the follow-up of the EOR Reasons stopping runs, a reporting tool was introduced that not only provides an overview of the data taking activities but also improves the dissemination of the information through email notifications and presentations in daily meetings. Implemented in PHP and based on the Zend Framework, this tool runs daily, collecting the necessary information from the eLogbook database, generating the report and sending it via email to a predefined mailing list. An example of such a report is shown in Fig. 25.

4.3.5. ALICE electronic logbook human interface

Introduction: The eLogbook web-based HI was developed using modern web technologies, including PHP5, Javascript and Cascading Style Sheets (CSS). It is hosted on an Apache web server and it can be accessed from the Internet. Charts are generated using the PHP JpGraph library and several components of the dhtmlxSuite javascript framework [22] are used for improved usability.

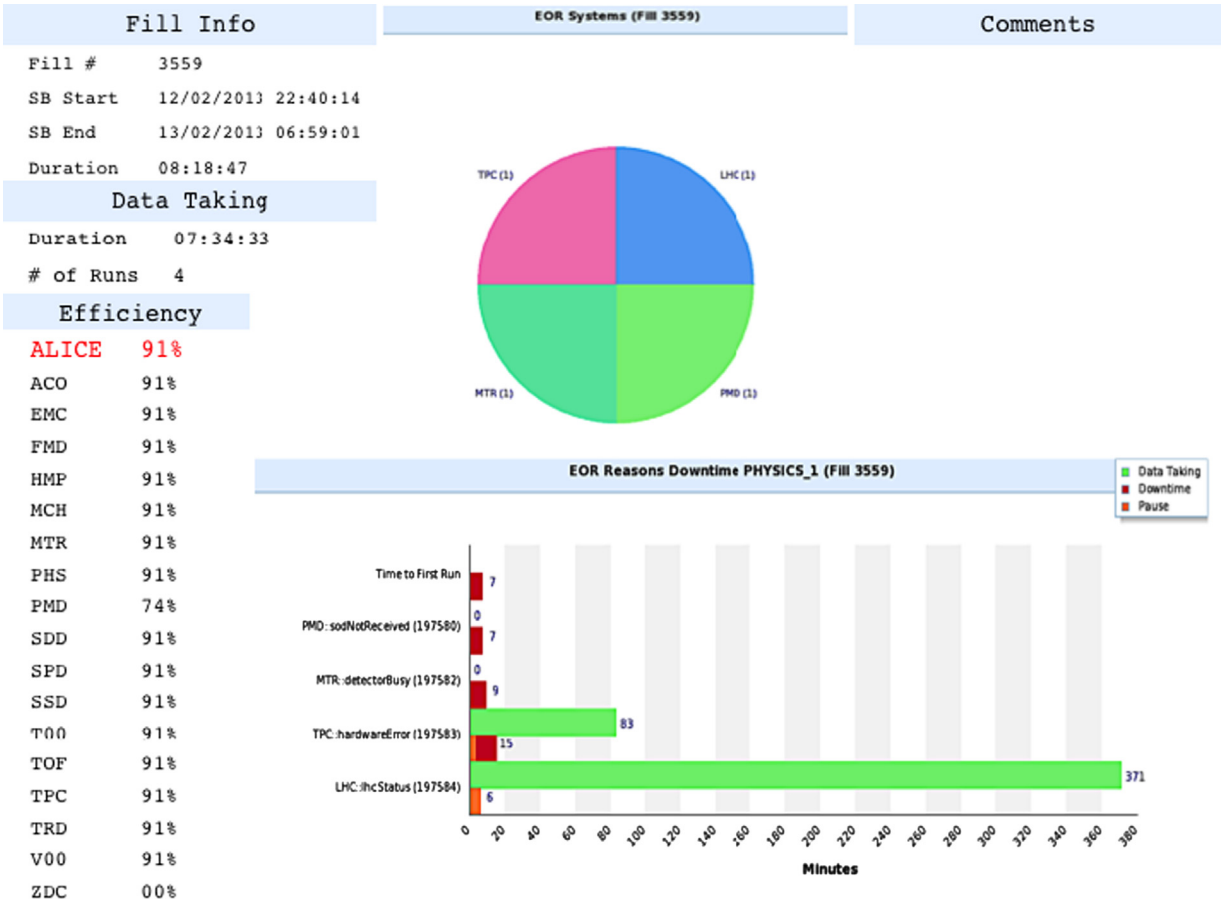


Fig. 25. Example of an LHC Fill summary report.

Authentication and authorization: Authentication is implemented via the CERN Authentication central service, providing Single Sign On (SSO) and removing the effort of authenticating the users from the eLogbook software. This way, when a user tries to access the HI, he is redirected to the CERN Login page where it has to provide his credentials. If successful, he is then redirected back to the HI. Authorization is implemented in the HI with five different levels of privileges:

1. NONE: no access to the HI.
2. READ: read-only access.
3. WRITE: read/write access (e.g. can write reports).
4. ADMIN: same as WRITE + can change privileges of users (except ADMIN and SUPER).
5. SUPER: same as WRITE + can change privileges of users (except SUPER).

Run Statistics: The Run Statistics pages give users access to several parameters and statistics of each individual run in a tabular format. Multiple ALICE and LHC parameters and statistics fields are available, providing users with a complete view of both the conditions during which data taking occurred and how the ALICE online system performed. Examples of such fields are magnetic field currents, LHC beam energy and data and event rates. Extensive information is also available regarding both the Trigger and HLT systems, including Trigger Clusters definition, Trigger Classes rates and counters and HLT compression factors. Introduced in 2011 and extensively used ever since, the reason (s) responsible for the end of the runs (EOR Reasons) provide users with a clear identification of what caused each run to stop. Additional information concerning the quality of data taking and the Offline pre-processing status is also available.

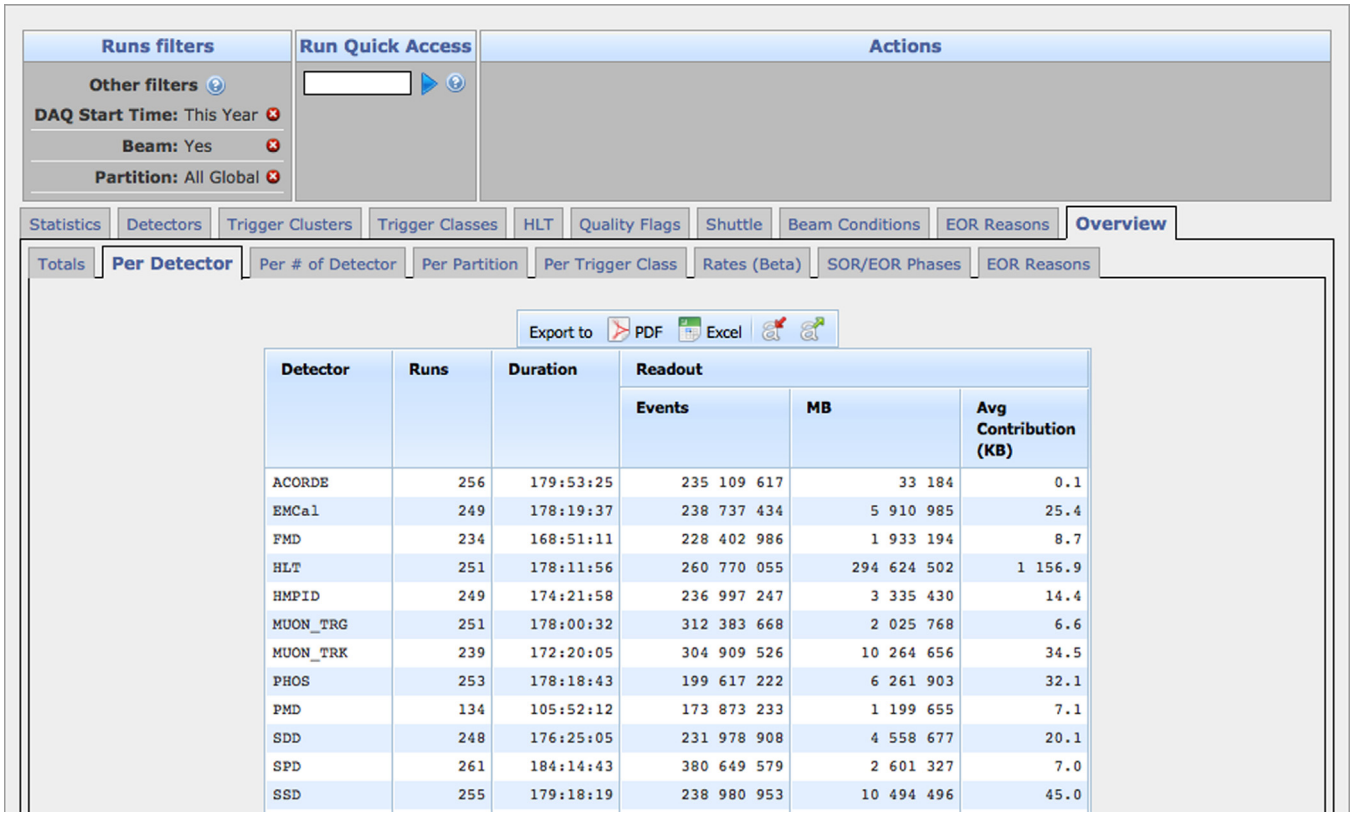
In total, there are 152 available fields grouped into 10 different tabs. Users can order the runs, select the list of fields to be displayed or export the results in XML, ASCII or Microsoft Excel™format. Some fields have extra information made available through onMouseOver boxes.

Run Overview: The Run Overview page, accessible via the Run Statistics page, displays aggregated counters in a tabular and/or graphical format. This allows users to have a global view of the experiments data-taking statistics over multiple runs (see Fig. 26), serving as a source for management reports, conference papers and presentations.

Users can access multiple counters at different stages of the data flow aggregated per detector, per number of detectors and per partition. Aggregation of Trigger Classes counters are also available, with users being able to choose which Trigger Classes to display (given the high number of available Trigger Classes).

Although not related to data taking counters, two more aggregated views exist providing users with an overview of operational values. The first one displays the duration of the different SOR and EOR phases in a summary table with minimum, maximum and average duration for each phase. Users can also plot the duration of a specific phase as a function of time. The second one aggregates the EOR Reasons, displaying a summary table and several pie charts grouping the reasons by type, subsystem and most common individual reasons.

In total, there are 7 summary tables and 21 charts grouped into 8 different tabs. All tables are displayed using the dhtmlxGrid component of the dhtmlxSuite javascript framework, providing features such as inline ordering, automatic totals footer and export



Detector	Runs	Duration	Readout		
			Events	MB	Avg Contribution (KB)
ACORDE	256	179:53:25	235 109 617	33 184	0.1
EMCal	249	178:19:37	238 737 434	5 910 985	25.4
FMD	234	168:51:11	228 402 986	1 933 194	8.7
HLT	251	178:11:56	260 770 055	294 624 502	1 156.9
HMPID	249	174:21:58	236 997 247	3 335 430	14.4
MUON_TRG	251	178:00:32	312 383 668	2 025 768	6.6
MUON_TRK	239	172:20:05	304 909 526	10 264 656	34.5
PHOS	253	178:18:43	199 617 222	6 261 903	32.1
PMD	134	105:52:12	173 873 233	1 199 655	7.1
SDD	248	176:25:05	231 978 908	4 558 677	20.1
SPD	261	184:14:43	380 649 579	2 601 327	7.0
SSD	255	179:18:19	238 980 953	10 494 496	45.0

Fig. 26. Run Overview page snapshot.

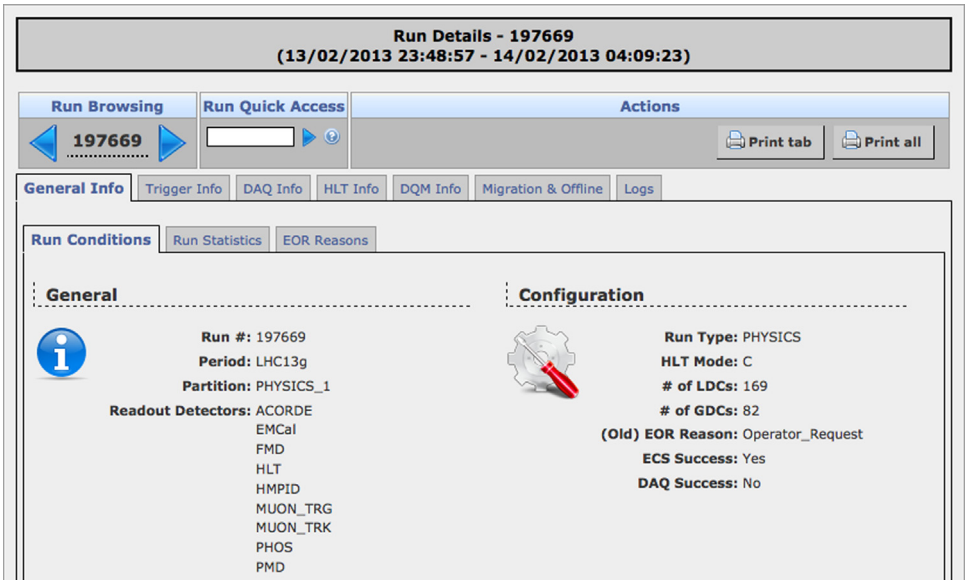


Fig. 27. Run Details page snapshot.

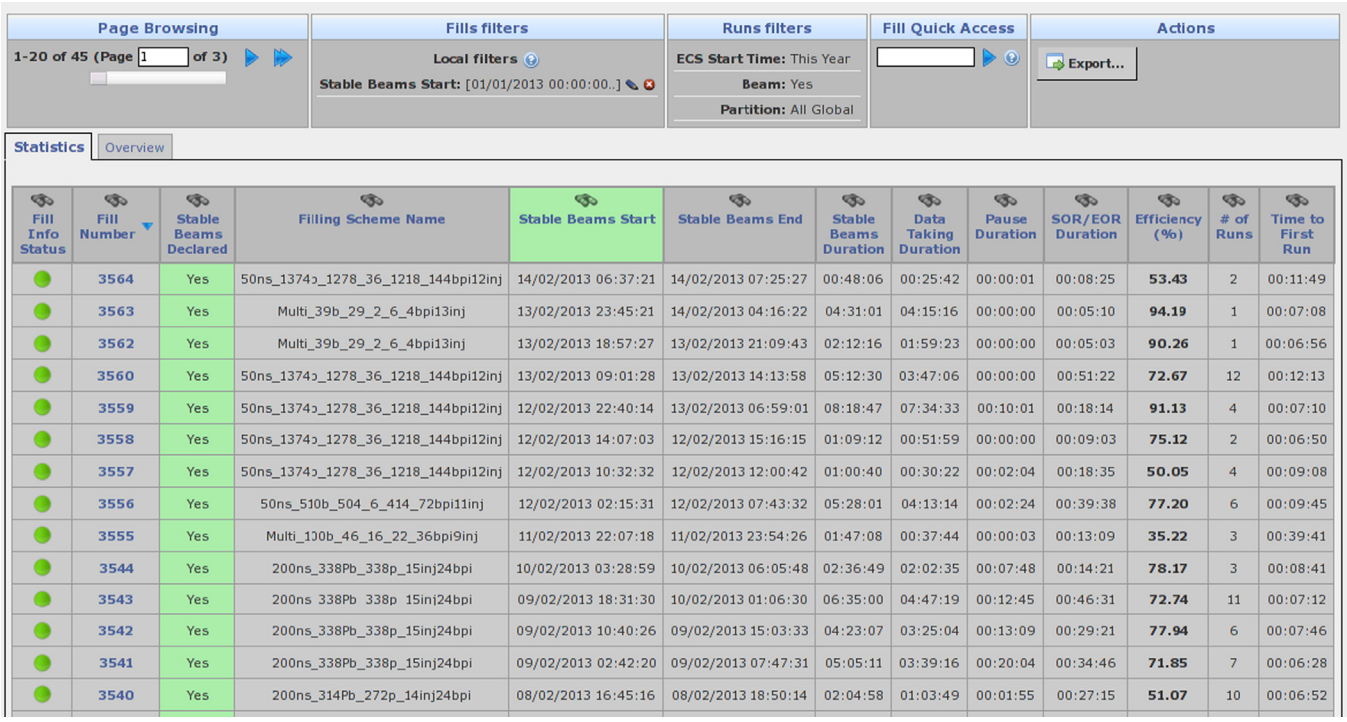
to PDF or Microsoft Excel™. All charts are implemented using the JpGraph library and allow users to select between linear and logarithmic scale and bar or time-series chart. For this last option, users also have the possibility to select which series to display and stack or integrate series.

Run Details: The Run Details page, presented in Fig. 27, provides a detailed view of the available information concerning a specific run.

In the General Info section users have access to the conditions during which data taking took place, the high level configuration of the ECS and multiple timestamps, counters and rates. Once the run is finished, the respective EOR Reasons also become available.

In the Trigger Info section, extensive information concerning Trigger Clusters, Trigger Classes, Trigger Aliases and Trigger Inputs is available. Additionally, a graphical representation of the ALICE Collision Schedule allows users to see in which slots the LHC is colliding particles. A dedicated view for Trigger experts allows for remote access to the Trigger system configuration used in the run.

The DAQ Info section displays the duration of the different SOR and EOR phases, multiple counters for each of the DAQ system



Fill Info Status	Fill Number	Stable Beams Declared	Filling Scheme Name	Stable Beams Start	Stable Beams End	Stable Beams Duration	Data Taking Duration	Pause Duration	SOR/EOR Duration	Efficiency (%)	# of Runs	Time to First Run
	3564	Yes	50ns_1374b_1278_36_1218_144bpi12inj	14/02/2013 06:37:21	14/02/2013 07:25:27	00:48:06	00:25:42	00:00:01	00:08:25	53.43	2	00:11:49
	3563	Yes	Multi_39b_29_2_6_4bpi13inj	13/02/2013 23:45:21	14/02/2013 04:16:22	04:31:01	04:15:16	00:00:00	00:05:10	94.19	1	00:07:08
	3562	Yes	Multi_39b_29_2_6_4bpi13inj	13/02/2013 18:57:27	13/02/2013 21:09:43	02:12:16	01:59:23	00:00:00	00:05:03	90.26	1	00:06:56
	3560	Yes	50ns_1374b_1278_36_1218_144bpi12inj	13/02/2013 09:01:28	13/02/2013 14:13:58	05:12:30	03:47:06	00:00:00	00:51:22	72.67	12	00:12:13
	3559	Yes	50ns_1374b_1278_36_1218_144bpi12inj	12/02/2013 22:40:14	13/02/2013 06:59:01	08:18:47	07:34:33	00:10:01	00:18:14	91.13	4	00:07:10
	3558	Yes	50ns_1374b_1278_36_1218_144bpi12inj	12/02/2013 14:07:03	12/02/2013 15:16:15	01:09:12	00:51:59	00:00:00	00:09:03	75.12	2	00:06:50
	3557	Yes	50ns_1374b_1278_36_1218_144bpi12inj	12/02/2013 10:32:32	12/02/2013 12:00:42	01:00:40	00:30:22	00:02:04	00:18:35	50.05	4	00:09:08
	3556	Yes	50ns_510b_504_6_414_72bpi11inj	12/02/2013 02:15:31	12/02/2013 07:43:32	05:28:01	04:13:14	00:02:24	00:39:38	77.20	6	00:09:45
	3555	Yes	Multi_100b_46_16_22_36bpi9inj	11/02/2013 22:07:18	11/02/2013 23:54:26	01:47:08	00:37:44	00:00:03	00:13:09	35.22	3	00:39:41
	3544	Yes	200ns_338Pb_338p_15inj24bpi	10/02/2013 03:28:59	10/02/2013 06:05:48	02:36:49	02:02:35	00:07:48	00:14:21	78.17	3	00:08:41
	3543	Yes	200ns_338Pb_338p_15inj24bpi	09/02/2013 18:31:30	10/02/2013 01:06:30	06:35:00	04:47:19	00:12:45	00:46:31	72.74	11	00:07:12
	3542	Yes	200ns_338Pb_338p_15inj24bpi	09/02/2013 10:40:26	09/02/2013 15:03:33	04:23:07	03:25:04	00:13:09	00:29:21	77.94	6	00:07:46
	3541	Yes	200ns_338Pb_338p_15inj24bpi	09/02/2013 02:42:20	09/02/2013 07:47:31	05:05:11	03:39:16	00:20:04	00:34:46	71.85	7	00:06:28
	3540	Yes	200ns_314Pb_272p_14inj24bpi	08/02/2013 16:45:16	08/02/2013 18:50:14	02:04:58	01:03:49	00:01:55	00:27:15	51.07	10	00:06:52

Fig. 28. LHC Fill Statistics page snapshot.

nodes, a list of which readout links were active and the status and log messages of the Detector Algorithms executed during the run.

The HLT Info section provides different configuration parameters and counters related with the High Level Trigger system.

In the DQM Info section, users have access to Data Quality Monitoring and run quality information. It includes, for each active DQM agent, some overview (including a summary image) and a list of Monitoring Objects either temporarily stored via a sampling mechanism or permanently stored via an explicit action from a shifter or an expert.

The Migration and Offline section provides details concerning the status of the migration of the data from the experimental area to the permanent data storage location in the CERN Computer Center.

Finally, in the Logs section, users have access to the Log Entries inserted for this run by automatic processes, shifters and experts. Additionally, users also have access to a subset of the log messages generated by the online systems.

Fill Statistics: The Fill Statistics page provides a view per LHC fill, providing users with information such as the duration of stable beams, the number of runs and most importantly, the data taking efficiency of each fill (as seen in Fig. 28).

Fill Overview: The Fill Overview page provides an overview of the LHC Fill statistics, including charts of the data taking efficiency distribution and of each fill.

Fill Details: The Fill Details page provides a detailed view of the available data concerning a specific LHC Fill. In this page, users have access to the general configuration of the fill, the global and per-detector data taking efficiency and an overview of the runs executed during the fill.

The Collision Rates section displays a plot with the variation of the collision rates as seen by ALICE during the fill (see Fig. 29).

Extensive information concerning the EOR Reasons that occurred during the fill is also available, including several charts grouping the reasons by type, subsystem and most common individual reasons.

Log Entries: The Log Entries pages allow users to either read existing reports or (if they have the necessary privileges) write new reports. Any type of file can be attached to a report, with thumbnails being created for images. Users can choose between Compact, Extended or Detailed view mode for different detail levels. Dedicated sections display all file attachments and aggregated summaries.

Each report can be assigned to one or more subsystem (ranging from individual sub-detectors to services such as Safety or Run Coordination). Each of these subsystems can be configured to allow users to receive automatic email notifications each time a new Log Entry is inserted.

Search Filters: To allow the possibility to search for runs or log entries that match a given set of criteria, column based filters were implemented in the Run Statistics, Fill Statistics and Log Entries pages. Using either predefined or user-defined values, users can combine several filters to restrict the displayed set (see Fig. 30). These filters are also active in the Overview pages, allowing users to only aggregate runs, fills or log entries that match the specified criteria.

Operations: In production since 2007, the eLogbook repository has been growing steadily both in number of runs and of Log Entries. As shown in Fig. 31, the number of runs stored reached 100,000 in the end of 2009 and is currently very close to 200,000. The small slowdown since 2010 is justified by the ALICE move from commissioning to operations.

The number of Log Entries, as shown in Fig. 32, reached 50,000 in the end of 2010 and is currently more than 100 000, evenly distributed between human and process generated. The high increase in Log Entries seen in October 2009 was due to a bulk fix in the eLogbook repository to properly close runs which did not have an end timestamp.

The eLogbook has been extremely well received by the ALICE collaboration and is now the main entry point to follow the activities in the experimental cavern. As shown in Fig. 33, the number of unique visitors (based on IP address) has stabilized

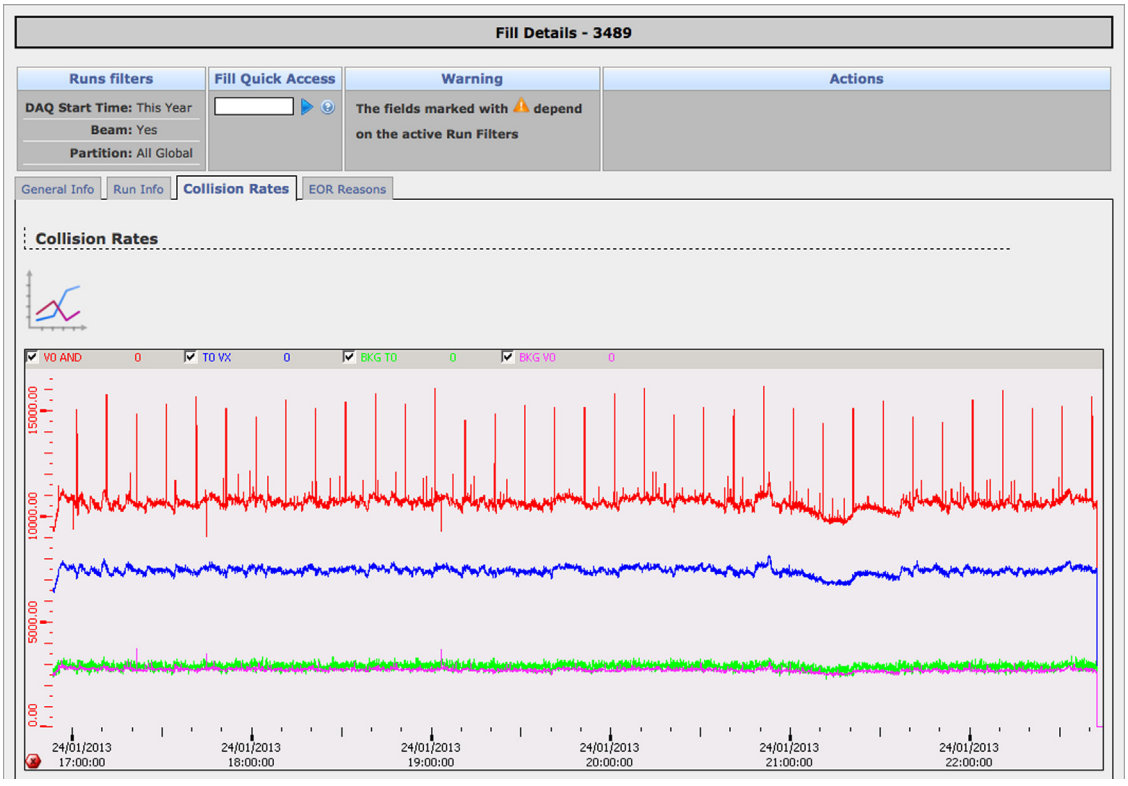


Fig. 29. LHC Fill Details page snapshot.

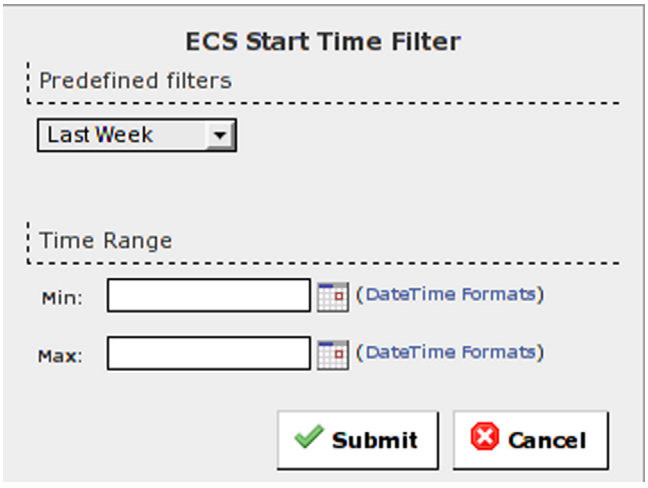


Fig. 30. Example of a Search Filter window.

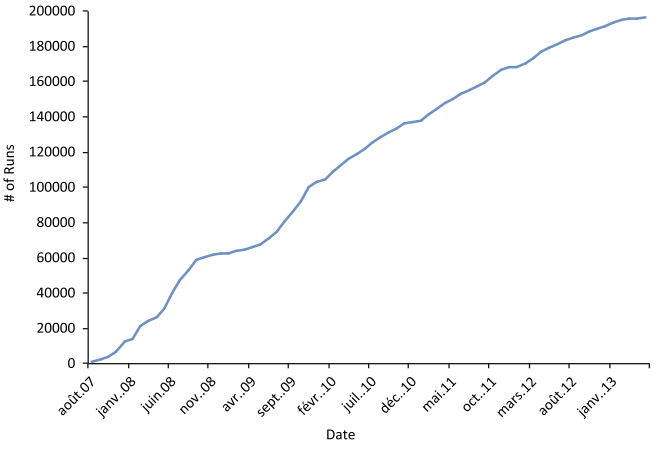


Fig. 31. Temporal evolution of the number of runs stored in the eLogbook.

around 1500 per month and the number of visits follows a well-defined pattern of a peak during the Heavy Ion run followed by a sharp drop during Christmas and a ramp-up to a plateau once the proton–proton run starts.

4.3.6. LHC interface

Many kinds of information about run conditions are stored in the ALICE electronic logbook during data taking. Among these, the information on the configuration of the two beams (energy, collision scheme, intensities, etc.) delivered by the LHC plays an especially important role. Two tools have been developed to collect and check this data: the online DIP client and the offline cross-check application.

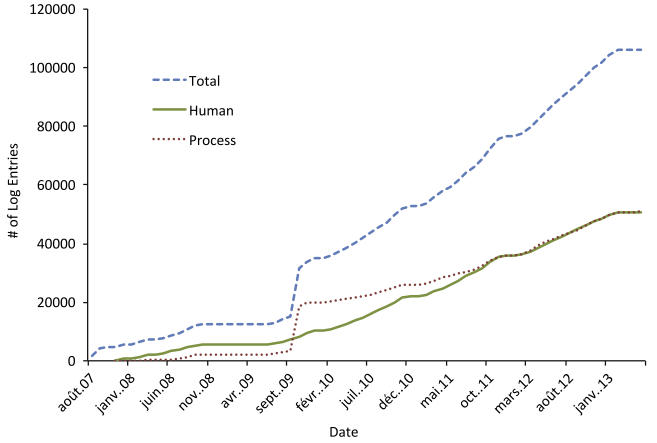


Fig. 32. Temporal evolution of the number of Log Entries stored in the eLogbook.

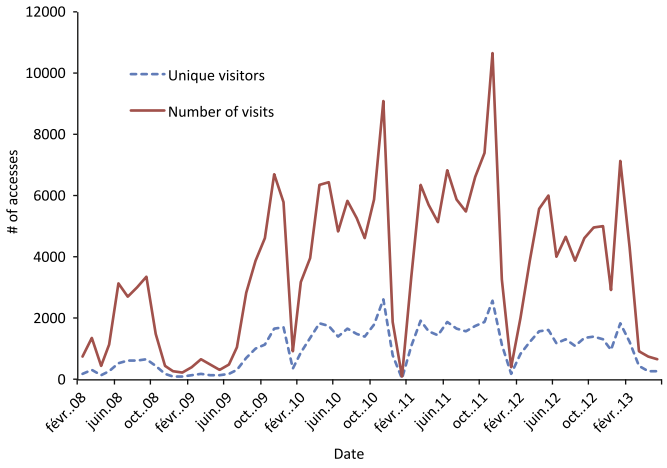


Fig. 33. Temporal evolution of the eLogbook HI usage.

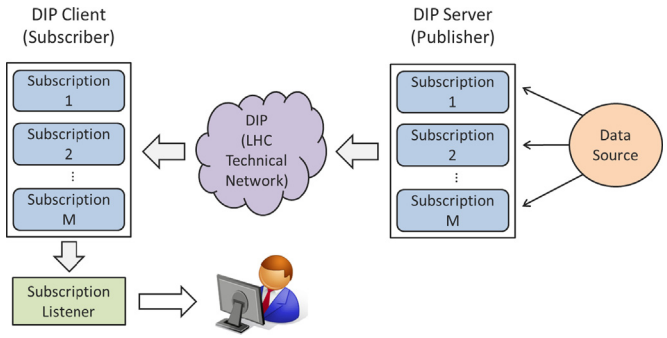


Fig. 34. DIP architecture.

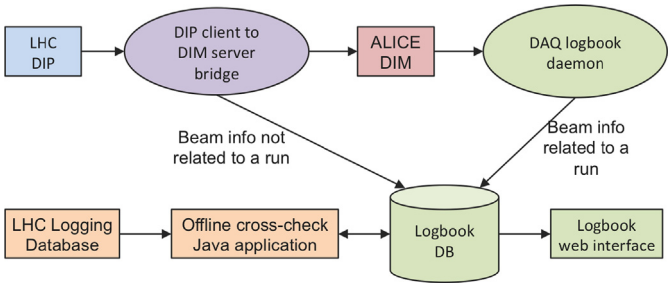


Fig. 35. Full scheme of the ALICE LHC Interface to store and cross-check the beam information in the eLogbook.

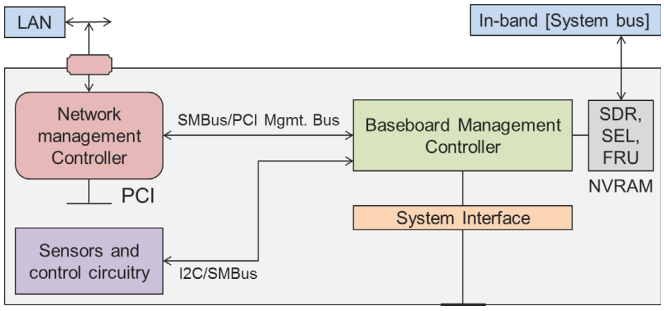


Fig. 36. IPMI schematics.

The Java application runs every 8 hours. It connects to both the LHC Logging Database and the ALICE logbook in order to extract the information stored since the last execution. If discrepancies are found, an update of the ALICE logbook values is made with the LHC Logging ones.

Online DIP client: The LHC values are published via DIP (Data Interchange Protocol) [23], which allows real-time data to be exchanged between very loosely heterogeneous systems. It is an information distribution service, which may contain event based data updated as and when the event(s) occur. A general architecture is presented in Fig. 34.

A DIP client has been developed to subscribe to the LHC information of interest and, after a processing step, to publish the corresponding information in the ALICE DIM [24] server. It can then be read and stored by DAQ logbook daemon in the Electronic Logbook at start/end of each run and fill (Fig. 35). The logbook daemon runs constantly on the background and, at start of run, stores run-related values. Values that are related to a fill, such as the *STABLE BEAM* time and duration, are stored by the DIP client directly in the logbook database using the logbook C API. Finally, all the values without distinction are continuously sent to the infoLogger.

Offline cross-check application: The beam values stored during data taking period are sometimes corrected afterward. Therefore it is necessary to perform an offline cross-check of these values with the ones stored in the LHC Logging Database [25]. This database is part of the LHC Logging Service at CERN to permanently store and manage the measured values of the most important parameters, configurations, and working characteristics of the accelerator parts and experiments.

An ALICE custom application has been developed to perform the offline cross-check between the beam information provided online by the DIP client and the values stored by the LHC Logging Service. Amongst the three available methods to extract data from the LHC Logging Database [26,27] we chose the Java API.

4.3.7. IPMI

IPMI introduction: IPMI is an acronym that stands for Intelligent Platform Management Interface, promoted by Intel, HP, NEC, Dell and having 179 hardware manufacturers adopters in 2005 [28]. The schematics in Fig. 36 is intentionally simplified, as to the usage in our environment.

Connections to the IPMI card are done via RMCP+ protocol and in our setup we use the out-of-band communication to the Baseboard Management Controller that accesses the various sensors on the system and stores Sensor Data Records (SDR), System Event Logs (SEL) and Field Replaceable Unit (FRU) information in the local NVRAM. The IPMI provides Chassis Power Control, Sensors information on temperature, voltage, fans speed, alerting abilities via Platform Event Trap (PET) Alert and Platform Event Filtering (PEF) Filter, System Event Log (SEL) messages, watchdog timer that could be configured to reset the system if communication to Operating System stops, Serial over LAN, possibility to send non-maskable interrupts to OS kernel.

There are specific vendor implementations for example DELL implemented Dell Remote Access Card (DRAC) and HP has the Integrated Lights-Out (iLO).

Rationals and use: We have several hundreds PCs in the DAQ system therefore the possibility of the OS being stuck and the incidence of non-responsive HW is high. The usual intervention recipe in case of non-responsive machines was to go to the servers site and do a complete power cycle then put the machine back in production. While being very effective, the downside of this recipe is that, in the best case scenario, the onsite intervention would take about 40 min which represents an important loss of data taking time.

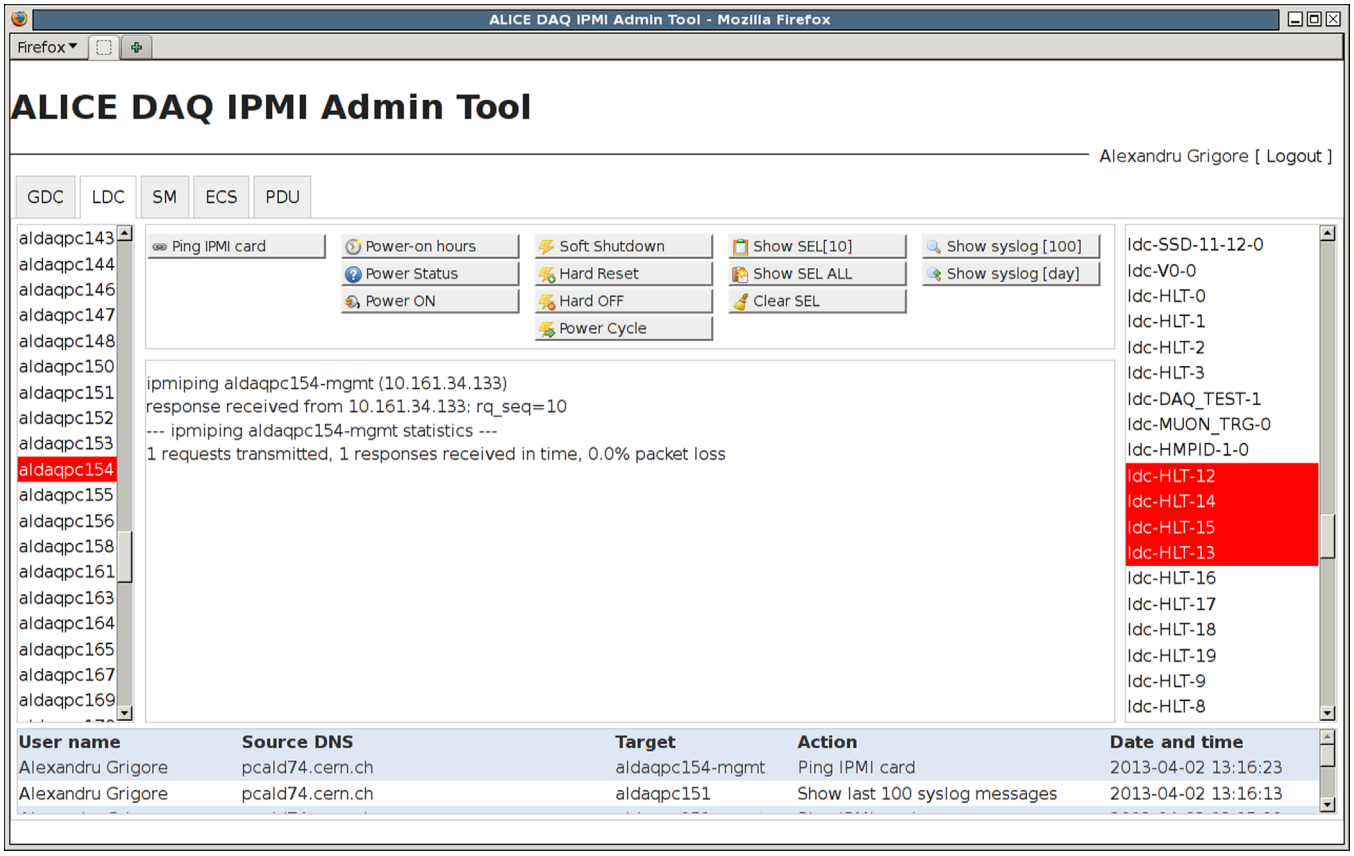


Fig. 37. The IPMI web interface for system administration.

In order to reduce the intervention and investigation time, we have decided to configure the IPMI cards and make use of the possibility of out-of-band power control, system event log and sensors reading. The network interface associated with the IPMI card and the authentication have been configured via shell commands based on the ipmitool provided by net-snmp-utils package.

Command line interface. To allow for command line access and host range execution a command line tool has been developed based on Python and net-snmp-utils. The execution is threaded, additional switches for debug and logging are available.

Here follow the main categories of the commands implemented by the command line interface:

1. Show information on the IPMI card configuration.
2. Identify a PC in a rack by flashing the identification led.
3. Show/clear the events registered in the NVRAM of the IPMI card.
4. Show temperature, voltage, and fan sensors information.
5. Control power/show power status/set boot-up parameters.

Web interface for system administration: As using the above IPMI Command Line Interface would imply connecting through several gateways via SSH, to further increase the reachability of the commands, a web interface has been developed that integrates Linux syslogd messages, IPMI functionalities and PDU ports control. The language and libraries used in the development process are PHP, javascript and JQuery for the frontend, MySQL for storing the syslog messages, syslog-ng for centralizing the messages forwarded from every Linux host via syslogd, python for the backend.

Once authorized, the user has to ping the IPMI card and if the card is reachable, subsequent IPMI functions will be enabled in the web interface as shown in Fig. 37.

To improve and complete the investigation process, syslogd messages are one-click away. PDU ports control is also available, being useful for remote control of a machine whose IPMI card stopped responding.

4.4. Detector software

The 18 ALICE sub-detectors need to be calibrated regularly in order to deliver accurate physics measurements. This involves analyzing events in different experimental conditions, which produce results used at different stages of the data flow, from electronics configuration (e.g. zero-suppression from pedestal values) to online data quality monitoring, and offline events processing. It is important to generate calibration data in the shortest delays in order to have minimal latency in the data-taking/monitoring/analysis loop, and hence detect problems as early as possible.

To achieve these goals, a framework was developed to run calibration tasks online in the DAQ farm. It consists of two types of operations:

- Running a dedicated standalone run with a detector configured specifically for this calibration task, record data locally, and generate calibration results by analysis of the recorded data. This is typically the scenario for a pedestal run.
- Running calibration task in the background of physics data taking. Events are sampled from the main data stream without interfering with the normal DAQ flow, and analyzed on the fly on dedicated machines. Results are available at end of run, and

partial results may be exported during the run to DQM for visualization and checks. This is the scenario for example for dead pixel or noisy channels detection.

The Detector Algorithm (DA) framework provides means for the detector experts to execute their code online. The framework takes care of the I/O interfaces (retrieve configuration, sample events, export results), ensures runtime control (launch, monitor and terminate processes) and checks that all follow common packaging guidelines. Procedures are defined for a unified testing, validation, and deployment in production.

In practice, writing a DA mainly consists of adding event-processing code to a common empty-loop skeleton. Once the code is committed in the Offline software repository, the corresponding binary executable can be built on a reference system and validated with simulation or experience data. When performance, interfaces and stability are fine, the DA is deployed in production and controlled at runtime by the DAQ system. It receives as input data recorded locally or sampled from the main stream (depending on the scenarios described previously). Results produced are exported Offline (where they are further processed by a system called the Shuttle [29]) and/or used locally at the experimental area.

Detailed description of the DA framework architecture and features can be found in Refs. [30,31].

We can raise the following points from our experience in the past years:

- Static binary building was an effective way to decouple detector code from mainstream Offline software releases. It provided stability, at the cost of more work on the rare occasions when dependencies needed to be updated. The static building allowed easy cohabitation of different versions on the same hosts.
- Control, logging and accounting are key features to be able to run the code, check errors, and follow them up with the corresponding developers. Runtime encapsulation is done by a specific launcher tracing and reporting the behavior of each DA. It redirect logs, detect crashes or exit errors, and gather statistics on performance and resource usage. It also ensures limits are respected (system settings were sometime not enough, or would not allow a fine granularity), and if necessary terminates processes taking too much time or memory. Statistical analysis of the collected runtime information was very helpful to detect, reproduce, and fix problems in the detector code.

4.5. Data quality monitoring

4.5.1. Introduction

Data quality monitoring is an important aspect of every High-Energy Physics experiment, especially in the era of LHC where the detectors are extremely sophisticated devices. In order to use the data taking time and the precious bandwidth in an optimal way, one needs an online feedback on the quality of the data being recorded. DQM software provides this feedback and helps shifters and experts to identify potential issues early. DQM involves the online gathering of data, their analysis by user-defined algorithm and the storage and visualization of the produced monitoring information.

4.5.2. Design and architecture

AMORE (Automatic MOnitoRing Environment) is the DQM framework developed and used in ALICE [32,33]. It is a flexible and modular software framework which is used to analyze data samples and produce and visualize monitoring results. It is mainly written in C++ and founded on the widely used data analysis framework ROOT (see Fig. 38) ; it uses the DATE monitoring library (see Section 4.3) to collect data from files, LDCs or GDCs. In case the same analysis is needed online and offline, the use of the ALICE Offline framework for simulation, AliRoot [6], is encouraged.

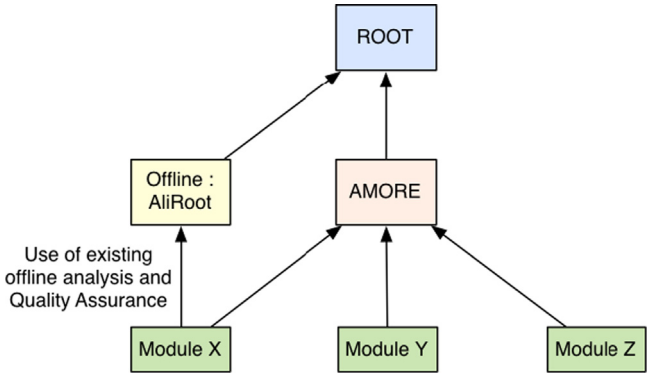


Fig. 38. General schema of AMORE.

AMORE is based on a publisher-subscriber paradigm (see Fig. 39) where a large number of processes, called agents, execute detector-specific decoding and analysis on raw data samples and publish their results in a pool. Clients can then connect to the pool and visualize the monitoring results through a dedicated user interface. The serialization of the published objects, which occurs on the publisher side before the actual storage in the database, is handled by the facilities provided by ROOT. The only direct communication between publishers and clients consists of notifications by means of DIM. The notifications coming from the outside world, especially from the ECS (see Section 4.2.2) and the DCS, use the same technology (see Section 4.5.7 for details on integration with other systems).

The data samples feeding the agents might come from the DAQ nodes, from other agents or from files. The resulting quantities, often histograms, although there is no restriction on their type, are published encapsulated in *MonitorObjects* that essentially contain metadata allowing a proper and coherent handling by the framework. In addition they include extra details about the objects such as their quality or their target "audience". The target audience gives the possibility to distinguish between objects that can be easily interpreted and others which need a more thorough knowledge and experience to be of use.

4.5.3. Pluggable architecture

AMORE uses a plug-in architecture to avoid any framework's dependency on users' code. The plug-in mechanism is implemented through the ROOT reflection feature. Users, usually detectors teams, develop specific code that is built into dynamic libraries called *modules* that are loaded at runtime by the framework if, and when, it is needed and following the "Strategy" design pattern.

Modules are typically split into two parts corresponding to the publishing and the subscribing sides of the framework (see Fig. 40). There are typically 4 libraries produced (stacked boxes on the left on the figure), one for each package (the 4 items at the bottom): Common, Publisher, Subscriber and UI. The module's publisher can be instantiated several times, to collect more statistics per instance, each instance corresponding to an *agent*. The same is true for the UI part of the module; we call these instances clients or GUI. Note that a module can contain several publisher and subscriber classes (not shown on the figure).

An AMORE agent is a finite state machine, executing custom user code in a well defined time sequence depending on the DAQ environment status. For example, a tight analysis loop is executed during data taking while special functions are called at start and

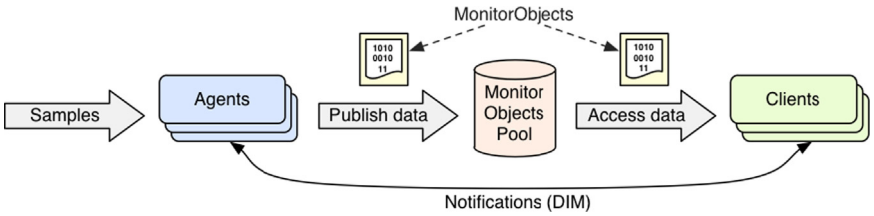


Fig. 39. The publisher–subscriber paradigm in AMORE.

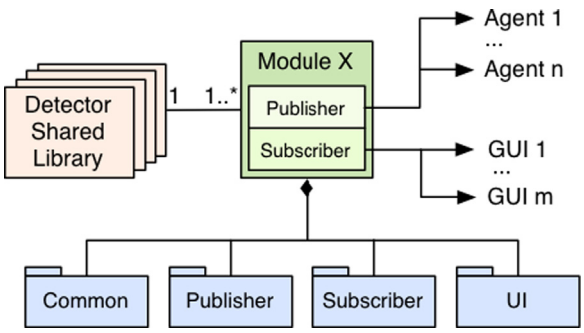


Fig. 40. Description of a module.

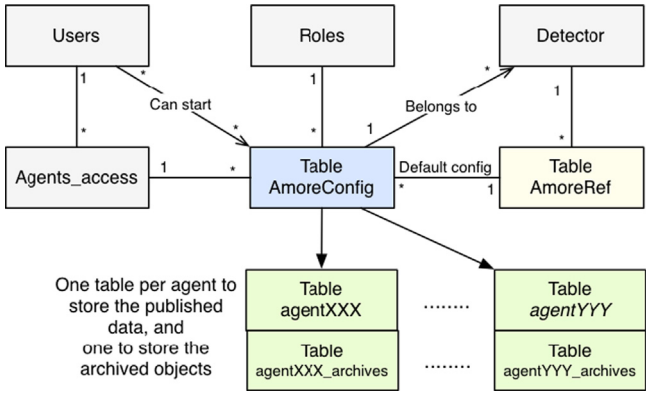


Fig. 41. Database schema.

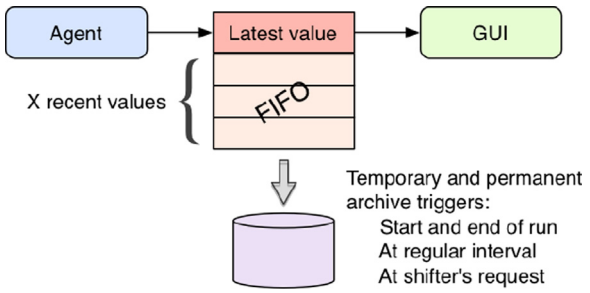


Fig. 42. The archiving system in AMORE.

end of run effectively implementing the "template method" design pattern.

4.5.4. Database

The data pool is implemented as a database. The MySQL system was chosen because it is light-weight and freely distributable and our experience proved it to be reliable and performant. Fig. 41 shows a schema of the database main tables.

The database is used not only to keep the data published by the agents (in the table named after the agent) but also to store configuration of AMORE as a system. The table *amoreconfig* lists the agents and their details (main class to load, detector they belong to and default running options). Nodes where the agents can run are listed in the table *Roles* which maps the real name with a role name. This indirection allows for a quick and transparent move of agents from one physical machine to another. The tables *Users* and *Agents_access* provides basic security: only the owner of an agent and the DQM operator can start and stop it. Finally *amoreref* contains the optional configuration files for the agents.

4.5.5. Archives

Users often need to check and study an object even though the run is over, sometimes since days, or to compare the evolution of metrics. Therefore, snapshots of the MonitorObjects must be saved, if not permanently at least for some time.

The archives are stored for a couple of weeks in a table whose name ends with "_archives" which is very similar to the agent's data table (see Fig. 41). The objects can be marked as *permanent* in order to avoid their deletion.

A special process called "archiver" is meant to give a way to create archives and recover interesting MonitorObjects for further study. The archiver must always be running and available to receive requests. It also performs a clean up every night to erase archives older than the age limit and to optimize the database. The archiver uses plug-ins as the publishers and subscribers do. It uses DIM to receive users' requests and notifications at start and end of run. Archives can also be created by calling stored procedures in the database.

In addition to these mid- and long-term archives, it might also be interesting to keep a very detailed short-term history to discover when a problem occurred or started. This is done through the so-called FIFO, which is directly implemented within the database. It consists in keeping former versions of the objects in a FIFO queue (see Fig. 42) in the data table of an agent. The maximum size of the table is set in the AmoreConfig table and versions of the objects are kept as long as the table does not exceed it.

4.5.6. Generic GUI

The subscriber part of the users' modules consists mainly in GUIs capable of handling the objects produced by the publishing part.

As the basic needs and requirements of most of the detector teams are very similar, a generic GUI has been developed to avoid code duplication and to minimize users development. It allows to browse and visualize any object of any running agent. The thousands of objects published by the detector agents are displayed as a tree (see left panel in Fig. 43). The right panel of the window displays the MonitorObjects selected by the user, automatically fitting them on the available space by splitting the tab in a grid. Finally, the layout can be saved as XML files in the database or in the local file system for future reuse. Different buttons permit, for instance, to filter agents by name or status (idle vs running), or to show only certain objects.

Custom user interfaces have also been developed by the detector teams. They help experts to commission, test and

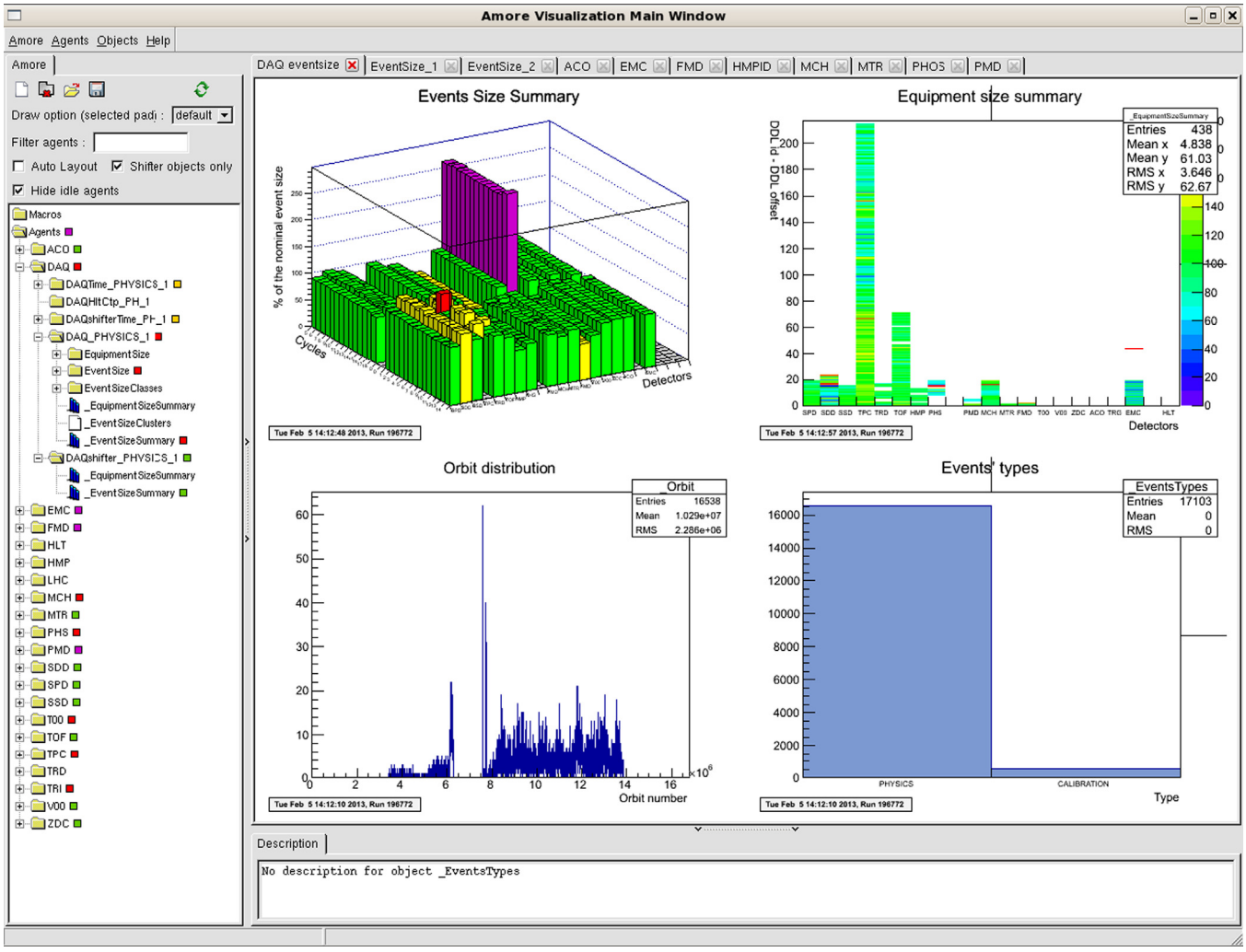


Fig. 43. The AMORE Generic GUI.

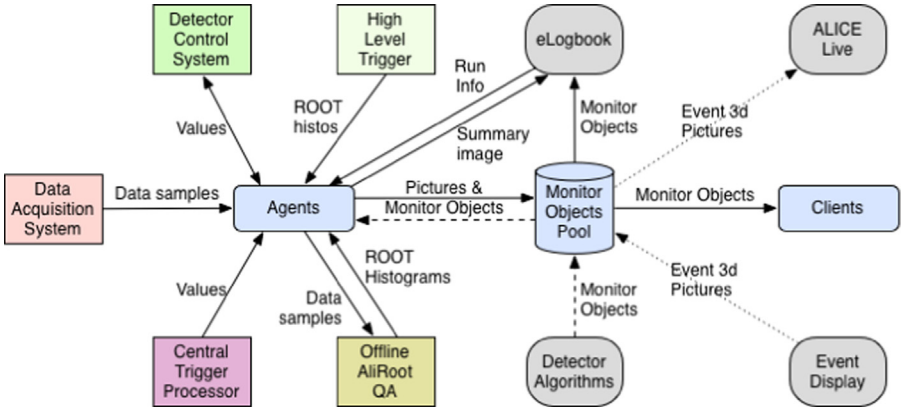


Fig. 44. Data providers feeding the AMORE agents or directly the AMORE data pool.

investigate their detectors in greater details. Moreover custom user interface gives the possibility to use custom objects types that could not be handled in a meaningful way by the generic GUI.

4.5.7. *Interaction and integration with other systems*

Publishers as bridges to other systems and libraries: The publishers are meant to analyze the raw data they receive and to publish their results for future visualization or post-processing. However, not all publishers directly do the number crunching. Indeed, one should avoid, if possible, to duplicate code but rather choose to delegate the processing to existing frameworks and libraries. As a consequence some modules use external data providers and either transmit directly their data to the data pool or post-process it first.

Fig. 44 shows the interactions of AMORE with other systems. The Quality Assurance (QA) module, for instance, delegates the processing to the AliRoot QA framework. The HLT module retrieves the objects from a private network (HOMER, see [34]) and

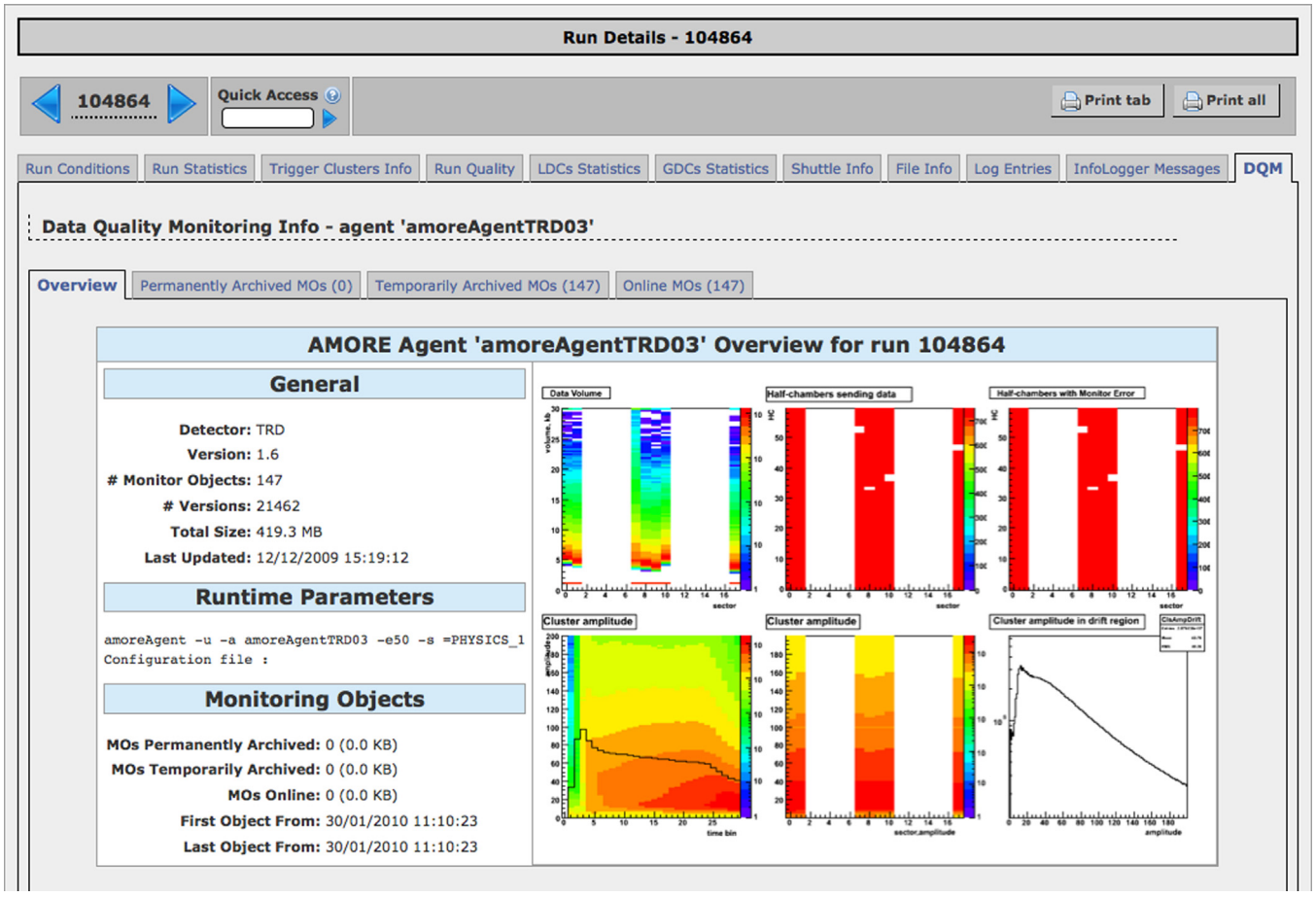


Fig. 45. ALICE electronic logbook interface to the DQM data.

publishes them in the pool. This is particularly interesting in terms of data quality evaluation because the HLT is able to reconstruct data on the fly and thus provides complex results. The CTP and DCS exchange values with the DQM. DAs (see Section 4.4) publish in the AMORE data pool histograms and results of their computation. Even the images of the Event Display 3D reconstruction are made available to the ALICE Live website via the AMORE data pool.

Web access via eLogbook: The ALICE Electronic Logbook (see Section 4.3.4) gives access to the monitoring data as a web client. Besides to information about data acquisition runs, the Electronic Logbook provides information about the DQM agents running during every data acquisition run. Fig. 45 shows the information available in the logbook when selecting a particular agent. The DQM tab contains configuration parameters, information about number of archived objects and gives the possibility to save the object image as well as to download the objects and manipulate them afterwards.

4.5.8. Tools

A number of tools have been designed and created to ease the configuration of AMORE as well as the operation of the DQM subsystem. They have been written using the Tcl scripting language in combination with the Tk GUI toolkit in order to get easy access to the AMORE database and provide user-friendly interfaces. In particular, there are two main categories of tools available to the AMORE users: the amoreAgentsManager, which is meant to be used by both shifters and experts; the amoreConfigFileBrowser and amoreEditDb which are exclusive to expert usage.

The amoreAgentsManager GUI (see Fig. 46) provides an overview of all the available agents as well as an easy way to check their status and launch their execution. It is mainly composed by two areas that respectively list the running and the available (non running) agents. The interface allows the user to start, restart and stop the agents and to check the parameters with which the agents are running. From the bottom part of the interface it is possible to start/restart/stop the archiver and to access the log messages that it generates.

The amoreConfigFileBrowser and the amoreEditDb are very similar to each other and they mainly operate on the AMORE configuration database. The amoreConfigFileBrowser allows the user to browse the configuration files associated with each detector, to modify them and/or to add new ones. The amoreEditDb provides an interface to change the execution parameters of the agents, the machines where the agents run as well as global parameters.

4.5.9. Benchmarks and optimizations

Database: As the data pool is the backbone of the system where all data transit, it is important to guarantee good performance and reliability even at peak time. The database benchmark consists of several tests; some, hereafter called *stress tests*, check extreme conditions by running agents and clients continuously publishing and retrieving data; other tests called *validation tests* correspond to more standard and typical usages and consist of agents running on predefined data, the same way we do for continuous integration. Our benchmarks proved that the pool can sustain high loads and thus that the database within the current design scale well

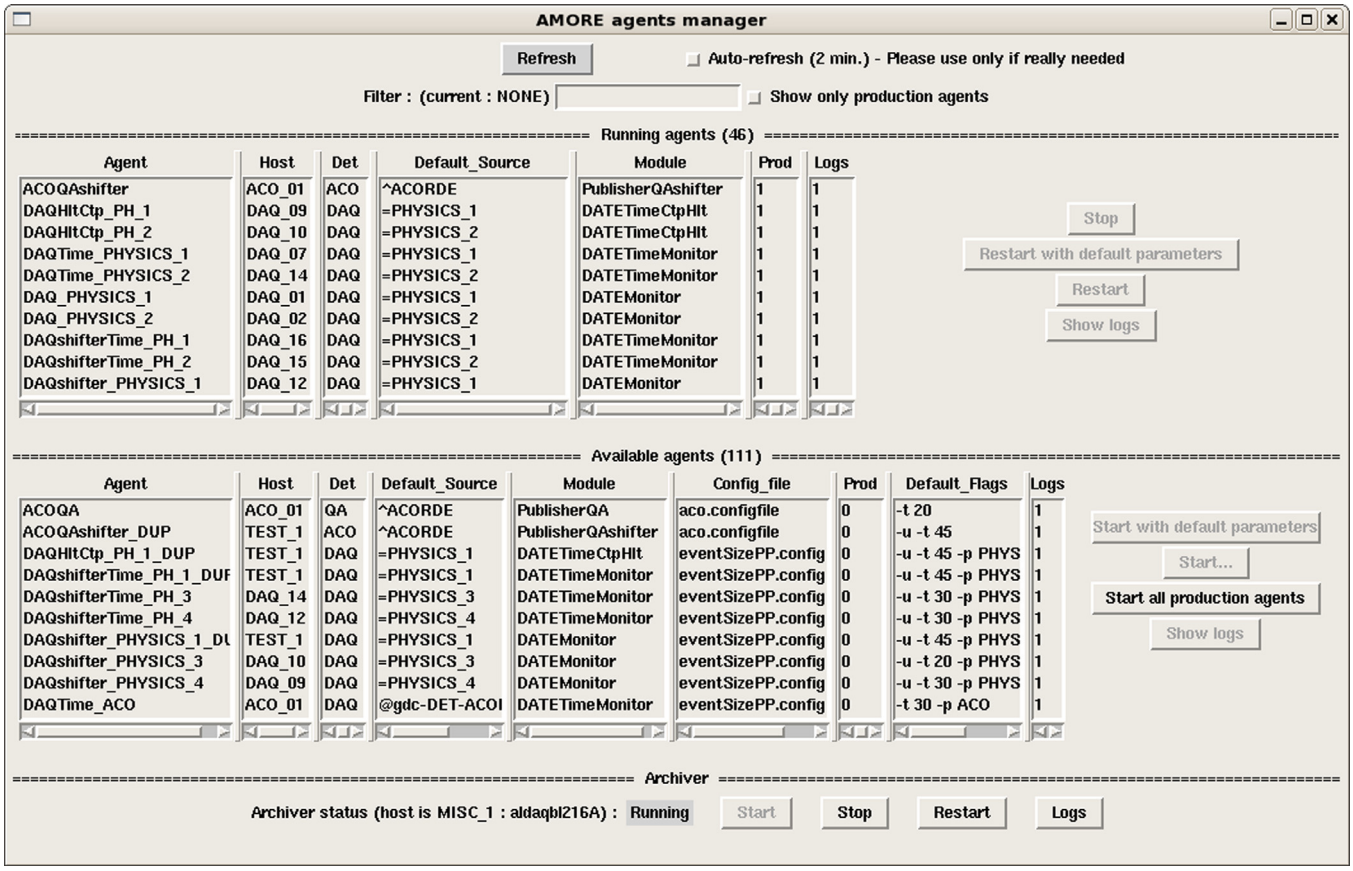


Fig. 46. amoreAgentsManager GUI.

and is capable of handling the increasing number of agents and clients.

The benchmarks helped improving the way the database is accessed. For example, stress tests showed that the results were far better when small queries were concatenated into very large queries of 2 MB as shown in Fig. 47. Findings also concerned the configuration of the database itself. The difference between MySQL storage engines proved to be very important. InnoDB engine supports ACID-compliant transaction features and row-level locks whereas MyISAM is much simpler and only have table-level locks and no transactions. Even in simple cases with a single running agent, InnoDB is twice slower than MyISAM. When running 10 agents concurrently, the execution time with InnoDB becomes 10 times larger than the one with MyISAM (see Fig. 47). The improvement comes at the cost of possible loss of data integrity but it was decided that being able to publish more often data counter-balanced this drawback. It also required a new archiver module (see Section 4.5.5) to optimize the data and archives table. It reorganizes the physical storage of MyISAM tables and index data to reduce storage space and improve I/O efficiency.

As discussed in Section 4.5.5, many versions of a MonitorObject are stored in the data tables. Adding an index to it improved the time of retrieval of objects by the clients but increased the time of inserting data. The write being more frequent than read, the index was discarded. However, the same has been achieved by using an in-memory table storing the list of the last version of each objects. These objects are the ones retrieved by most of the clients rather than the past version. By using this table when querying for objects, and by populating the in-memory table with triggers on insertions, the performance of the clients improved without pejorating the agents.

Compilers and computer architectures: Validation tests were carried out to test new servers, new platforms and new computer architectures. They showed an important improvement in performance when moving from 32 to 64 bits architecture and when upgrading the hardware. They also showed important differences when using different compilers (icc, gcc, clang) and different compilation options.

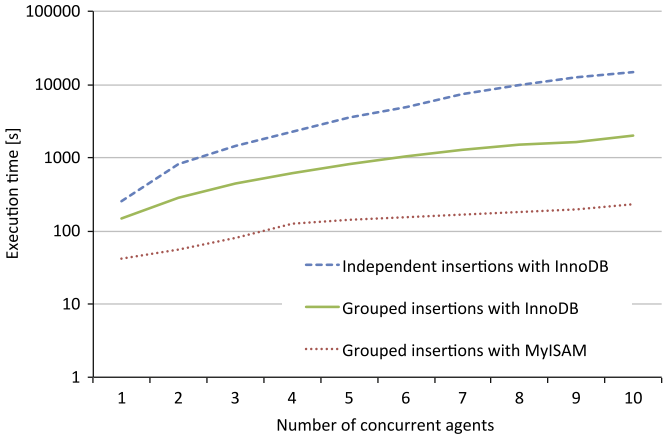


Fig. 47. Execution time of insertions queries vs number of concurrent agents. When queries are grouped, the blocks are of size 2 MB. In production we use grouped queries with MyISAM storage engine.

Multithreaded image generation: Nowadays parallel computing has become the dominant solution in performance improvement and, at a computer architecture level, it has been translated in the use of multicore technologies. As a first step for a complete parallelization of AMORE, a dual thread logic has been implemented by delegating what was a single process operation to two independent threads, one for the analysis and one for the objects

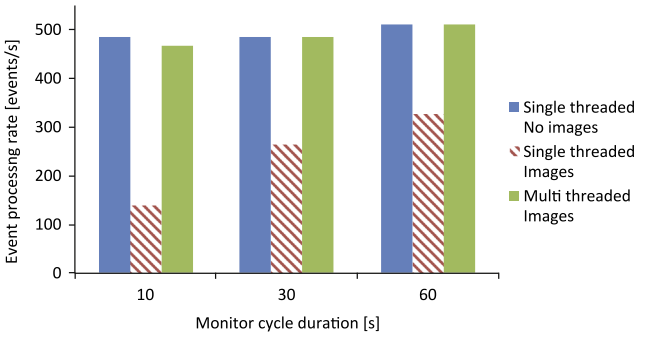


Fig. 48. Comparison between single-threaded and multi-threaded AMORE image creation. The bar on the left corresponds to the base case when no image is created. The middle bar represents the performance when images are generated. Finally the bar on the right shows the performance once we introduce parallelism.

image generation. This solution provides the performance shown in Fig. 48 for different monitor cycle duration that is the period during which events are processed in order to obtain the results and at the end of which the updated objects are published into the database. The *y*-axis represents the processing rate calculated in events per second. As we can see, a multithread solution with images creation provides performance very similar to the one shown by a single thread approach in which the images creation is disabled. The heaviness of the image creation process is confirmed by the bar representing the single thread approach with images creation. The results clearly show that the images generation dramatically affects the event processing rate whereas having a dedicated thread produces performance similar to the case when no images are created. To be noted that this approach implies that images might not be generated for all objects at every cycle.

4.5.10. Experience and results in production

During the past 3 years of data taking with LHC beam, AMORE has been intensively used in a real and challenging environment. More than 40 agents ran continuously, publishing an average of more than 10,000 objects updated every minute making a total of 10 MB/s of data published during a typical run.[1] In the past years, the system experienced peaks with more than 3000 objects per second factually saturating the link at 1 Gb/s. A tuning of the parameters of the agents and a better understanding of the detectors needs made it possible to avoid such peaks in 2012. In general, the architecture proved to be extremely stable and capable of scaling seamlessly with the growing number of agents and clients.

5. System operations and global performance

The history of the ALICE DAQ system operations started with its first installation at test beams in several CERN sites, then moved on to R&D, validation, and commissioning setups for the ALICE teams (detectors and facilities) to finally settle with its current configuration at the LHC Interaction Point 2, where it has been in continuous operation since 2008.

The ALICE experiment needs multiple and different running scenarios as required by the work-plan of the experiment. We can group these scenarios into four separate categories:

1. Technical: used for testing and validation, this activity can include a highly variable number of detectors and may adopt varying profiles (trigger, compression, recording) as required by the specific target of the requester(s).
2. Cosmic: used mainly for calibration and validation purposes, these runs have a specific trigger setup and can include a variable number of detectors. Trigger and data rates are extremely low (of the order of a few Hz and MB/s), events are very small (no pileup), and it is not unusual to sustain non-stop running periods of 6 hours or more.
3. pp Physics: the most frequent ALICE activity, its characteristics are high trigger rate (few KHz input to the DAQ) for data rates of the order of several hundreds of MB/s.
4. Heavy ion physics: the main target of the ALICE experiment, it is the most challenging of all the running modes as it produces, due to the high multiplicity of the events, the peak sustained recording data rates – up to several GB/s – during relatively short time periods (about 30 days per year). It is in this running mode that all the actors of ALICE (hardware, software, and human) have to ensure their top efficiency and therefore withstand the highest stress. The DAQ of the ALICE experiment has been designed with the main target of exploiting at the best this running mode, maximizing the so-called running efficiency (the percentage of the LHC stable beam time that is effectively used for Data Acquisition purposes) while recording data having the best possible quality from the physics point of view.

The ALICE DAQ must be able to cope with all of the above scenarios, even when operated by personnel with little or no knowledge of data acquisition techniques and of the insight of the DAQ system itself. A particular emphasis has therefore been imposed on the following features:

- Easy and safe re-configuration of all components as required by specific activities in function of the ALICE program of work.
- Simple and modular operation of the ALICE experiment with a minimum quorum of on-site operators.
- Easy to follow error detection and recovery procedures, to be executed by the operators on shift either in person or with the assistance of experts always available for On-Call and/or On-Site support.

From the day of its first runs, the ALICE DAQ system has considerably evolved to satisfy the above requirements, using direct experience from the developers, the evolution of the other components (Detectors, Detector Control System, High Level Trigger System, and Central Trigger Processor System), and the valuable feedback provided by the ALICE Run Coordination and by the On-Site operators.

In its final configuration for LHC Run 1 – which ended in February 2013 – the ALICE Control Room was attended by a quorum of 4 persons:

1. The Shift Leader, who acted as the interface between the ALICE Run Coordination, the LHC Controls, and the various ALICE Systems (detectors, systems, and services).
2. The ECS/DAQ + CTP + HLT operator, handling all issues related to the Data Taking procedures.
3. The DCS operator, in charge of controlling the services for the ALICE Experiment (power, cooling, gasses, access, etc.).
4. The DQM operator, who took care of live quality assurance for the data acquired by the detectors.

These 4 persons alone were enough to take care of the ordinary operation of the ALICE Experiment and of all its detectors and services 24 hours on 24 during several consecutive months.

5.1. Operating procedures

The operation of the ALICE DAQ System was ensured by a dedicated role in the ALICE Control Room (ACR). This person took

[1] 2012 global runs with beam and with at least 16 detectors.

care of the ECS, the DAQ, the CTP and the HLT. This role also assisted the ALICE Shift Leader (SL) to respect and fulfil the ALICE plan of work (for example by ensuring an adequate status of all the active ALICE Detectors at key phases of the LHC fill). The preparation for this role was ensured by a strict procedure where a theoretical training had to be taken by all the candidates (either on-site or remotely), followed by an on-site hands-on training period of minimum 3 days, and completed by an online questionnaire. The preparation of each candidate was evaluated via the outcome of the questionnaire and by personal supervision at the end of the training period. As operators often came at CERN for just few weeks a year, a refresh training period was proposed after long absences in order to get them re-acquainted with the working environment and to familiarize with all the new features deployed at the ALICE controls.

ALICE is a very complex experiment; up to 18 detectors must be synchronized and configured to work in unison as defined by the experiment's program. This job, performed by the SL, can be rather challenging and error-prone. The ACT has therefore been designed and adopted to control and verify the configuration of all the active ALICE partitions. The SL selects via the ACT the configurations that has to be used by the various ALICE components (e.g. detectors settings and triggers classes) and then hands over the operation to the DAQ shifter.

The operation of the ALICE Experiment is organized into partitions, groups of detectors running all together using a common trigger configuration. It is allowed to have multiple partitions running simultaneously, as long as all the detectors belong at most to one partition at a time. It is possible, for example, to have two parallel runs having different Physics targets and using their own set of trigger classes. The ALICE SL and DAQ operator have been given the means to easily handle multiple partitions simultaneously without explicit synchronization and with a clear separation of commands and statuses for each partition.

It is possible to operate any of the ALICE detectors standalone, for example to perform a specific operation such as a calibration procedure. The ALICE DAQ operator has the capability to either perform these standalone runs directly from the main control panel – for example to prepare the detectors at the beginning of an LHC fill – or by handing over the detectors to the Detector experts. Standalone runs include one and only one detector and do not use the Central Trigger Processor: the trigger signals are delivered to the detector front-end electronics via an emulator which is local to the detector itself and which follows a fixed pre-programmed sequence. A standalone run for a given target can be prepared, tested, and validated in advance by the Detector experts and then be used "as is" by the DAQ operator. Each ALICE Detector provides its own set of dedicated standalone run types, to be executed at well-defined moments outside, before, during, and after every LHC fill. Multiple parallel runs can be performed simultaneously: this feature is used, for example, during ramp-up of the LHC Beam where many detectors must all get ready for the approaching stable beam condition. Adequate tools are given to the shifter in order to easily check in one central point the outcome of multiple standalone runs.

One of the central facilities for the ALICE operations is undoubtedly the ALICE Electronic Logbook. Operational events are systematically reported in the ALICE logbook. The full operational history of the ALICE experiment can be retrieved at any time thanks to this key facility.

5.2. HW and SW operational experience

The ALICE DAQ as installed at the LHC Interaction Point 2 is the result of several years of on-the-field evaluation and testing exercises. Both HW and SW components proved to fulfil the requirements and to behave as expected.

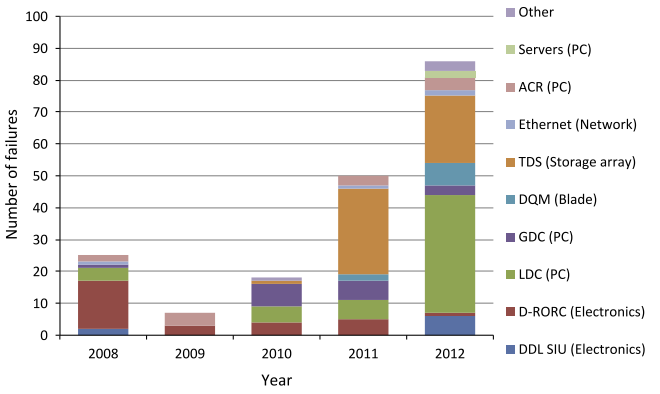


Fig. 49. Number of hardware failures.

On the HW side, certain components proved to be surprisingly resistant; for example, we did not experience a single disk failure during the whole operation period. On the other hand, three major problems did arise during this period (see Fig. 49 for numbers):

- *HW aging*: Observed on equipments with more than 5 years of intensive usage, aging has triggered the unforecasted upgrade of components such as the Disk Array supporting the Transient Data Storage system.
- *Hostile environment*: The running conditions at the experimental area were not easy to cope with and imposed a heavy burden on the HW installed at Point 2. Power cuts, cooling system failures, and the heat generated by the densely packed hardware in the Counting Room required multiple on-site interventions to resuscitate unresponsive components such as computers and Power Distribution Units.
- *Wrongly dimensioned power supplies*: In the second year of data taking, power supplies started breaking down one after the other. It was traced down to the fact that they had been underdimensioned by the PC manufacturer, not matching the mother board needs when all PCI slots were loaded.

The original ALICE DAQ HW infrastructure, as used during LHC Run 1, was based upon the infrastructure installed by the previous users of LHC Point 2 (the L3 experience at LEP). This infrastructure proved to be rather inadequate for the material in use by the ALICE DAQ (not enough air flow, non-standard dimensions, and insufficient cabling space). For this reason, the ALICE DAQ Project decided to replace all the HW infrastructure during LHC Long Shutdown 1 (LS1, in the course of 2013). The ALICE DAQ Counting Room will be ready again in early 2014, to be used to support the recommissioning of the ALICE experiment before LHC Run 2.

For what concerns the SW, the LHC Run 1 period saw the upgrade of the ALICE DAQ SW from CERN Scientific Linux 4 32 bit to CERN Scientific Linux 5 64 bit. The transition, which took place during the 2011 Winter shutdown, was eventful and brought considerable improvements in the efficiency of the overall system. Extra buffer memory, better scheduling, more efficient usage of the system resources: all these improvements gave the expected results during the last 2 years of the LHC Run 1 operations. The adoption by the ALICE DAQ of commercial off-the-shelf (COTS) solutions for most of its SW components (networking, memory management, inter-process communication libraries, etc.) substantially contributed to a smooth and easy transition. All the detector-specific software could be pre-tested in validation sites and their migration to the new environment was practically eventless.

Overall, both HW and SW behaved very well. Runs were stable, performances were sustained without particular effort and the requirements could easily be satisfied. The DAQ system proved to be able to sustain 13 GB/s from the detectors (without event building) and 7 GB/s to the TDS. These figures gave enough headroom to the ALICE DAQ in order to absorb transient peaks in the data flow, for example whenever the trigger setup would create short overloads in the event rates.

5.3. Evolution of software components

The DAQ software in operation at ALICE saw very few changes during the LHC Run 1 period.

The major event during the first 5 years of operation was the usage in production of the HLT as a source of highly compressed data. This operational choice, which took place in 2011, required few important modifications to the flow of monitoring data used for some of the DAs and DQM modules (in order to keep a predefined portion of the original, uncompressed data available for a detailed analysis of the original payloads). All the other data streams worked un-modified as expected and could easily sustain the modified data flow.

Another main improvement, made in the course of 2012, was the implementation of an in-run detector recovery capability, the so-called Pause-And-Configure (PAC) procedure. The target of this new feature was the reset of the FEE triggered by the detection of an otherwise non-recoverable failure in the readout equipments (e.g. a Single Event Upset – SEU – in a FPGA). This procedure was eventually implemented and validated for the Time Projection Chamber (TPC), the Muon Chamber (MCH), and the Photon Multiplicity Detector (PMD). It proved its efficiency during the 2013 p–Pb run period, where the ALICE detectors had to confront a very hostile environment due to the high level of background radiation. The PAC procedure will be further enhanced and adopted by other ALICE detectors in the course of the LHC LS1.

As the ALICE data flow follows a 100% data-push architecture, the requirement emerged for a validation checkpoint in the course of a run. This checkpoint would be used to validate the status of all detectors and to probe their status after a major event such as a reconfiguration request. This checkpoint was implemented using a special trigger, the so-called SYNC event. Proposed in late 2011, the SYNC event was quickly adopted by all systems (Detectors, CTP, ECS, DAQ, and HLT) and was activated in 2012, when it became integral part of the PAC procedure.

From the Detectors' side, a continuous evolution took place throughout the whole of the LHC Run 1 period. This caused frequent updates in the detector-specific software installed on the LDCs, of the DAs, and of the DQM modules. Strict pre-validation and run-time checking procedures were put in place to ease up the updates. The DQM and DA frameworks saw several improvements to provide a better work environment and to perform run-time checks on the module themselves, in order to avoid system failures due, for example, to memory leaks in the Detectors' code.

The ECS also followed the evolution of the ALICE systems (CTP, DCS, and HLT) and detectors. New procedures were introduced to allow, for example, the recovery of tripped TPC chambers on the fly without having to stop the run. Contrary to the procedures followed for DAs and DQMs, the ECS could not easily pre-validate its updates and most of the changes had to be tested on the real ALICE environment, usually during Technical stops of the LHC. Valuable input was provided by the ALICE Run Coordination, by the ALICE operators, and by the ALICE Detector Teams to the ECS project to improve efficiency and functionality of the ECS system.

Seen the capabilities of the ALICE trigger system (50 fine-configurable trigger classes which evolved as required by the ALICE work plan), a new scheme was proposed and adopted in order to simplify the Offline processing of ALICE events. The concept of trigger aliases was introduced to allow the reference to sets of trigger classes using abstract entities. A set of predefined trigger aliases has been associated to each new trigger setup, allowing the reference of equivalent trigger conditions using a standard abstract layer.

5.4. Periodic validations

The ALICE DAQ project made intensive use of periodic practical exercises for evaluation and validation purposes.

Already in 1998, a collaboration between ALICE Online, ALICE Offline, and CERN/IT was started within the LCG project Fabric area, which had direct responsibility for prototyping the Tier0/1 center at CERN. Milestones were set and plans drawn for a series of periodic exercises having as main objective the evaluation of the proposed technologies for the implementation of the ALICE DAQ system. This activity led to the so-called ALICE Data Challenges (ADCs) that took place periodically between 1999 and 2007. Each of the ADCs culminated in non-stop simulated run periods of 1 or 2 weeks duration, each targeting an increasingly rate of sustained data-transfer. The last of these challenges [35] used the hardware and the software installed at Point 2 for the LHC start-up and gave results equivalent to the targets that had been defined for the final ALICE DAQ system.

At LHC Point 2, the installation of the ALICE Detectors were followed by the ALICE DAQ by means of dedicated “Feuille de Routes” (FDRs), a series of validation exercises having as target the proof of the capability for each system to properly integrate within the ALICE environment. These FDRs assisted the ALICE collaboration to follow and validate the evolution of all the ALICE Detectors during the installation and development stages up to the day of the first LHC Beam.

During the LHC Run 1, series of mid-run technical runs were performed, usually during the end-of-year break and the LHC machine development periods, to exercise all components and re-validate all the data paths. An example of these tests was a simulation of Heavy-Ion data traffic made right before the PbPb and pPb beams, when real events were loaded in the Front End of the TPC Detector, to be injected as live data in the actual data acquisition chain. All the data streams of the ALICE DAQ could therefore be thoroughly tested in order to ensure a maximum efficiency right before each LHC critical fill.

5.5. Increasing requirements

The original ALICE DAQ Technical Proposal (TP) [1] quoted a peak throughput of 2.5 GB/s before compression and 1.25 GB/s sustained after compression. The first installation of the ALICE DAQ at LHC Point 2 was designed around these requirements.

In 2011 it became obvious that the setup of the ALICE experiment would create a data flow by far exceeding the above requirements. At the same time, the data compression achievable online via the HLT system on the data coming from the TPC detector – the main contributor for the DAQ input links – reached an unprecedented ratio of 1-to-4, meaning that 75% of the data could be effectively discarded at the output of the LDCs. Following these new features, the DAQ and the HLT were upgraded to allow a sustained data flow of 3.8 GB/s at the output of the HLT farm towards the DAQ system. More powerful LDCs equipped with higher quantities of memory and with 10 GB Ethernet output links were installed together with an equivalent increase in the number of data links between DAQ and HLT. The new configuration proved to be able to sustain an output stream of 14 GB/s coming from the detectors for an equivalent compressed flow of about 4 GB/s into

the GDCs. In practice, the overall throughput sustained during the Pb–Pb production period in 2011 gave a peak sustained rate of 2.2 GB/s after compression while achieving the forecasted reduction factor of 1-to-4.

The p–Pb production period in 2013 created an equivalent challenge for the DAQ and HLT system, only this time the emphasis was on the event rates rather than on the data flow. A new upgrade was performed on both systems: the DAQ was equipped with much more buffer memory at the LDCs/GDCs and the HLT online algorithms were optimized to reduce the latencies as much possible regardless of the pileup in the ALICE TPC. The result was a system that proved to be able to handle data at 2 kHz (the double of what was described in the ALICE Technical Proposal) for a data rate of about 2 GB/s after the HLT compression. In production, the peak typical sustained rate observed in ALICE was of 1.2 kHz which corresponded to a data flow of 1.4 GB/s.

5.6. Statistics

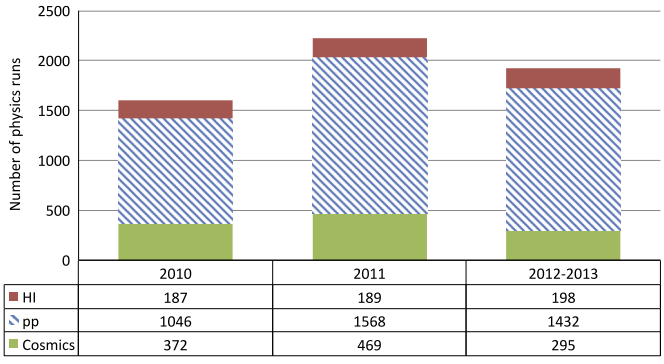
What follows are a few statistics extracted from the ALICE logbook relative to the last three years of operations.

Fig. 50 shows the trend of the number of physics runs split by run type (cosmic runs, pp physics run, and Heavy Ion physics runs).

In Fig. 51 we can see the number of bytes read into the LDCs (before the HLT compression) split into pp physics runs and Heavy Ion (PbPb and pPb) runs.

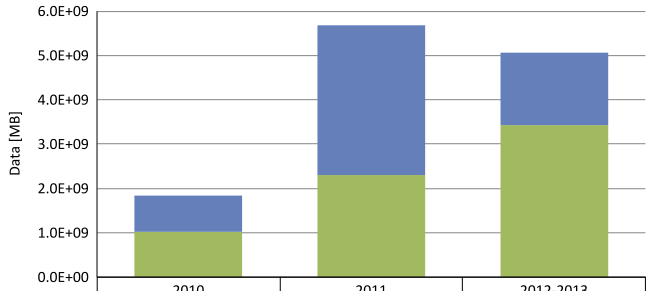
The bytes recorded (after HLT compression) are shown in Fig. 52, split into pp physics runs and Heavy Ion (PbPb and pPb) runs. Note how in 2012–2013, despite having 1 month more of pp physics compared to 2011 as well as a higher pp luminosity, the quantity of recorded data was less than the previous year. This effect is mainly due to the HLT compression.

Fig. 53 shows the summaries by physics type (pp physics, Heavy Ion physics and cosmics). The reduction factor achieved by the HLT online compression is very well visible for the 2011 and 2012–2013 run periods. Note that in 2011 the compression was enabled only near the end of the pp production (which was therefore mainly performed without this key feature). Furthermore, the pp physics program in 2012–2013 lasted 1 month more and saw higher luminosity compared to the previous year.



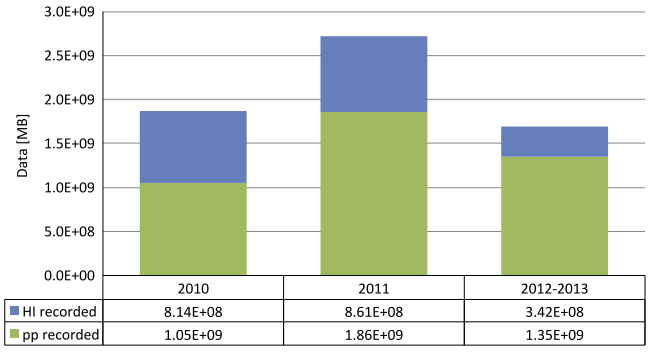
	2010	2011	2012-2013
HI	187	189	198
pp	1046	1568	1432
Cosmics	372	469	295

Fig. 50. Number of physics runs.



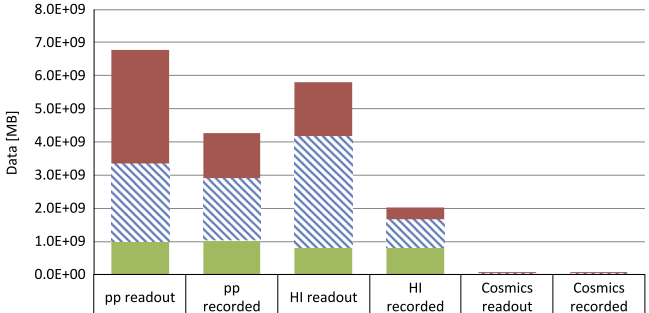
	2010	2011	2012-2013
HI readout	8.13E+08	3.37E+09	1.62E+09
pp readout	1.03E+09	2.31E+09	3.44E+09

Fig. 51. Bytes readout (in MB).



	2010	2011	2012-2013
HI recorded	8.14E+08	8.61E+08	3.42E+08
pp recorded	1.05E+09	1.86E+09	1.35E+09

Fig. 52. Bytes recorded (in MB).



	pp readout	pp recorded	HI readout	HI recorded	Cosmics readout	Cosmics recorded
2012-2013	3.44E+09	1.35E+09	1.62E+09	3.42E+08	3.81E+07	1.57E+07
2011	2.31E+09	1.86E+09	3.37E+09	8.61E+08	3.00E+07	2.33E+07
2010	1.03E+09	1.05E+09	8.13E+08	8.14E+08	8.47E+06	8.78E+06

Fig. 53. Bytes readout and recorded (in MB).

6. Conclusion

The R&D work on the ALICE DAQ system has started in 1995. The system has been designed and implemented in the years 1996 till 2005. The system has been successfully used during the phase of development, tests and commissioning of all the ALICE detectors. Two of these tests were particularly important: in 2004, there was the first test of the 3 detectors of the Inner Tracking System (ITS) and in 2006 the tests of the TPC made on the surface before its transport in the underground cavern.

The first elements of the DAQ system have been installed and commissioned at the experimental area in 2005. In September 2008, the ALICE DAQ was ready for the first pp collisions and is in full operation mode since then. During its operation with LHC beams, the system performance has been increased by a factor 4 above the original requirements and has collected more than 6 PB of physics data during the LHC Run 1 (2010–2013) [36].

During the LHC LS1 (LS1) in 2013–2014, the system is presently prepared for the Run 2 (2015-2017) with an adaptation of the software and a replacement of the obsolete computing hardware.

Since 1997, the system has also been used by several experiments at CERN (Compass [37], NA57 [38]) or in other labs such as the MICE experiment at the Rutherford Appleton Laboratory (RAL) in the UK [39] or the developments for the detection and imaging of High-Z Materials with a Muon Tomography [40].

References

[1] ALICE Collaboration, Technical Proposal for a Large Ion Collider Experiment at the CERN LHC, Technical Report LHCC 95-71, CERN, 1995.
[2] ALICE Collaboration, The ALICE experiment at the CERN LHC, Journal of Instrumentation 3(08) (2008) S08002. http://dx.doi.org/10.1088/1748-0221/3/08/S08002.
[3] ALICE Collaboration, ALICE Technical Design Report on Trigger, Data Acquisition, High-Level Trigger and Control System, Technical Report LHCC 2003-062, CERN, 2004.
[4] P. Vande Vyvre, Physics Requirements for the ALICE DAQ System—ALICE Internal Note 2000-030, Technical Report, CERN, 2000.
[5] Mysql, ⟨http://www.mysql.com⟩.
[6] AliRoot: ALICE Off-Line Framework for Simulation, Reconstruction and Analysis, ⟨http://aliceinfo.cern.ch/Offline⟩.
[7] R. Brun, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 389 (1–2) (1997) 81, http://dx.doi.org/10.1016/S0168-9002(97)00048-X.
[8] B. Franek, C. Gaspar, IEEE Transactions on Nuclear Science NS-045 (1998) 1946.
[9] C. Gaspar, A distributed information management system for the Delphi experiment at CERN, in: Proceedings of the IEEE Real Time Conference, Vancouver, Canada, 1993.
[10] I. Fedorko, V. Lefebure, D. Lenkes, M.O. Pera, Journal of Physics: Conference Series 396 (4) (2012) 042019 ⟨http://stacks.iop.org/1742-6596/396/i=4/a=042019⟩.
[11] Swig, Simplified Wrapper and Interface Generator, ⟨http://www.swig.org⟩.
[12] Jira, ⟨http://www.atlassian.com/jira⟩.
[13] Php, ⟨http://www.php.net/⟩.
[14] Apache web server, ⟨http://httpd.apache.org/⟩.
[15] Shibboleth, ⟨http://shibboleth.net/⟩.
[16] E. Ormancey, Journal of Physics: Conference Series 119 (8) (2008) 082008 ⟨http://iopscience.iop.org/1742-6596/119/8/082008/⟩.
[17] Cern e-groups, ⟨https://espace.cern.ch/e-groups-help⟩.
[18] Rpm, ⟨http://www.rpm.org/⟩.
[19] Tiki wiki cms groupware, ⟨http://info.tiki.org⟩.
[20] S. Chapeland, F. Carena, W. Carena, V.C. Barroso, F. Costa, E. Dénes, R. Divià, U. Fuchs, A. Grigore, G. Simonetti, C. Soós, A. Telesca, P.V. Vyvre, B. von Haller, Journal of Physics: Conference Series 396 (1) (2012) 012013 ⟨http://stacks.iop.org/1742-6596/396/i=1/a=012013⟩.
[21] dhtmlxsuite javascript framework, ⟨http://dhtmlx.com/⟩.
[22] Zend framework, ⟨http://framework.zend.com⟩.
[23] Dip and dim, ⟨http://j2eeps.cern.ch/wikis/display/EN/DIP+and+DIM⟩, (accessed 1-November-2013), 2013.
[24] C. Gaspar, M. Dönszelmann, P. Charpentier, Computer Physics Communications 140 (1–2) (2001) 102, http://dx.doi.org/10.1016/S0010-4655(01)00260-0, cHEP2000.
[25] What is the CERN Accelerator Logging Service? ⟨https://espace.cern.ch/be-dep/CO/DA/Services/CERN%20Accelerator%20Logging%20Service.aspx⟩.
[26] Measurement Service, ⟨https://espace.cern.ch/be-dep/OP/LHC/projects/measurement_database/index.aspx⟩.
[27] Lhc Logging Project, ⟨http://lhc-logging.web.cern.ch/lhc-logging/software/⟩.
[28] Ipmi, ⟨http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html⟩.
[29] C. Zampolli, F. Carminati, A. Colla, The shuttle: the ALICE framework for the extraction of the conditions data, in: PoS ACAT2010, 2010, p. 66.
[30] S. Chapeland, V. Altini, F. Carena, W. Carena, V.C. Barroso, F. Costa, R. Divià, U. Fuchs, I. Makhlyueva, F. Roukoutakis, K. Schossmaier, C. Soós, P.V. Vyvre, B. von Haller, The Alice collaboration, Journal of Physics: Conference Series 219 (2) (2010) 022004 ⟨http://stacks.iop.org/1742-6596/219/i=2/a=022004⟩.
[31] S. Chapeland, F. Carena, W. Carena, V.C. Barroso, F. Costa, E. Dénes, R. Divià, U. Fuchs, A. Grigore, G. Simonetti, C. Soós, A. Telesca, P.V. Vyvre, B. von Haller, Journal of Physics: Conference Series 396 (1) (2012) 012012 ⟨http://stacks.iop.org/1742-6596/396/i=1/a=012012⟩.
[32] A. Telesca, B. von Haller, S. Chapeland, F. Carena, W. Carena, V. Barroso, F. Costa, R. Divià, E. Dénes, U. Fuchs, G. Simonetti, P. Vande Vyvre, The ALICE data quality monitoring system, in: Real Time Conference (RT), 2010 17th IEEE-NPSS, 2010, pp. 1–6. http://dx.doi.org/10.1109/RTC.2010.5750364.
[33] B. von Haller, A. Telesca, S. Chapeland, F. Carena, W. Carena, V. Chibante Barroso, F. Costa, E. Dénes, R. Divià, U. Fuchs, G. Simonetti, C. Sos, P. Vande Vyvre, The ALICE collaboration, Journal of Physics: Conference Series 331 (2) (2011) 022030 ⟨http://stacks.iop.org/1742-6596/331/i=2/a=022030⟩.
[34] M. Richter, K. Aamodt, T. Alt, S. Bablok, C. Cheshkov, P. Hille, V. Lindenstruth, G. Ovrebekk, M. Ploskon, S. Popescu, D. Rohrich, T. Steinbeck, J. Thader, IEEE Transactions on Nuclear Science NS-55 (1) (2008) 133, http://dx.doi.org/10.1109/TNS.2007.913469.
[35] Alice Tests Its Digital Chain. ALICE Teste sa Chane Numrique (BUL-NA-2007-012. 08/2007. 09/2007. 08/2007), 2007, p. 3.
[36] F. Carena, W. Carena, S. Chapeland, V.C. Barroso, F. Costa, E. Dénes, R. Divià, U. Fuchs, G. Simonetti, C. Sos, A. Telesca, P.V. Vyvre, B. von Haller, The ALICE collaboration, Journal of Physics: Conference Series 331 (2) (2011) 022028, URL ⟨http://stacks.iop.org/1742-6596/331/i=2/a=022028⟩.
[37] Compass Experiment at CERN, ⟨http://wwwcompass.cern.ch/⟩.
[38] Na57 Experiment at CERN, ⟨http://wa97.web.cern.ch/WA97/⟩.
[39] M. Bonesini (On Behalf of the MICE Collaboration), Progress of the Mice Experiment at ral, arxiv:1303.7363.
[40] K. Gnanvo, B. Benson, W. Bittner, F. Costa, L. Grasso, M. Hohlmann, J.B. Locke, S. Martoiu, H. Muller, M. Staib, A. Tarazona, J. Toledo, Detection and imaging of high-Z materials with a muon tomography station using GEM detectors, in: Proceedings of 2010 IEEE, Nuclear Science Symposium Conference Record (NSS/MIC), October 30 2010–November 6 2010, p. 552, 559. ⟨http://dx.doi.org/10.1109/NSSMIC.2010.5873822⟩.