Journal of Physics: Conference Series **219** (2010) 022010 doi:10.1088/1742-6596/219/2/022010

The ALICE Online-Offline Framework for the Extraction of Conditions Data

Jan Fiete Grosse-Oetringhaus[1], Chiara Zampolli[1,2], Alberto Colla[3], and Federico Carminati[1] for the ALICE Collaboration

[1] CERN, 1211 Geneva 23, Switzerland
[2] INFN-CNAF, Bologna, Italy
[3] Università "La Sapienza" and INFN, Roma

E-mail: Jan.Fiete.Grosse-Oetringhaus@cern.ch, Chiara.Zampolli@cern.ch, Alberto.Colla@roma1.infn.it, Federico.Carminati@cern.ch

Abstract. A Large Ion Collider Experiment (ALICE) is the dedicated heavy-ion experiment at the CERN LHC and will take data with a bandwidth of up to 1.25 GB/s. It consists of 18 subdetectors that interact with five online systems (CTP, DAQ, DCS, ECS, and HLT). Data recorded is read out by DAQ in a raw data stream produced by the subdetectors. In addition the subdetectors produce conditions data derived from the raw data, i.e. calibration and alignment information, which have to be available from the beginning of the reconstruction and therefore cannot be included in the raw data. The extraction of the conditions data is steered by a system called Shuttle. It provides the link between data produced by the subdetectors in the online systems and a dedicated procedure per subdetector, called preprocessor, that runs in the Shuttle system. The preprocessor performs merging, consolidation, and reformatting of the data. Finally, it stores the data in the Grid Offline Conditions DataBase (OCDB) so that they are available for the Offline reconstruction. The reconstruction of a given run is initiated automatically once the raw data is successfully exported to the Grid storage and the run has been processed in the Shuttle framework. These proceedings introduce the Shuttle system. The performance of the system during the ALICE cosmics commissioning and LHC startup is described.

1. Introduction

During data-taking the subdetectors interact with several online systems (CTP[1], DAQ[2], DCS[3], ECS[4], and HLT[5]). Data is read out by DAQ as raw data streams produced by the subdetectors. At the same time they also produce *conditions data* which is information about the detector status and environmental variables. Most of the conditions data could in principle be calculated from the raw data and extracted offline after data-taking. However, such an approach would require an additional pass over the raw data before the reconstruction which is not possible due to the limited computing resources. Therefore, conditions data is already extracted during data-taking running dedicated subdetector algorithms in the online systems.

[1] Central Trigger Processor.
[2] Data Aquisition.
[3] Detector Control System.
[4] Experiment Control System.
[5] High-Level Trigger.

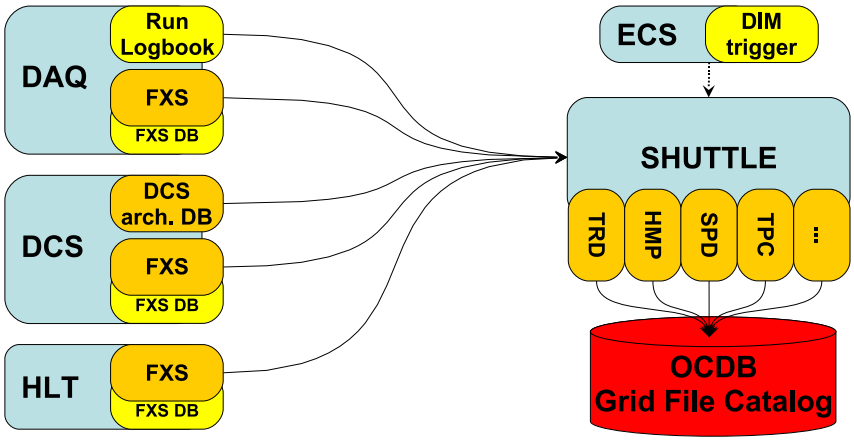

Figure 1. Schema of the Shuttle framework.

A method is required that reads the conditions data produced by the various subdetector algorithms in a coordinated way. The possibility that each subdetector publishes the produced conditions data by itself is considered too complicated and not manageable. An additional technical issue is that the machines in the online systems producing the data are protected by a firewall from the public CERN network and the internet.

As a solution to the problem a system named Shuttle framework has been developed which performs the following tasks:

- copying of data in any format produced by the online systems DAQ, DCS, and HLT for each subdetector;
- preprocessing of the data, e.g. consolidation, fitting;
- reformatting to ROOT [1] format;
- storing in the Grid Offline Conditions DataBase (OCDB);
- indicating that a given run has been processed, which is a precondition to starting the reconstruction.

These proceedings describe the structure and implementation of the Shuttle framework. A more extensive description of the Shuttle framework can be found in [2, 3].

The Shuttle system is part of AliRoot [4], ALICE's offline framework for simulation, reconstruction, and analysis.

2. Structure and Program Flow

A general schema of the Shuttle framework and the connections between the main components is presented in Figure 1. Its main features are:

- the Shuttle is triggered upon end-of-run (EOR) by the ECS system; furthermore, it can be operated in a self-triggered mode;
- it accesses the ECS logbook to read the run parameters (start and end time, run type, etc.) and the list of active subdetectors (see Section 2.1);
- it accesses the online systems and retrieves the conditions data produced during the run. According to the technique implemented to retrieve them, the experimental conditions data can be divided in two subsets:

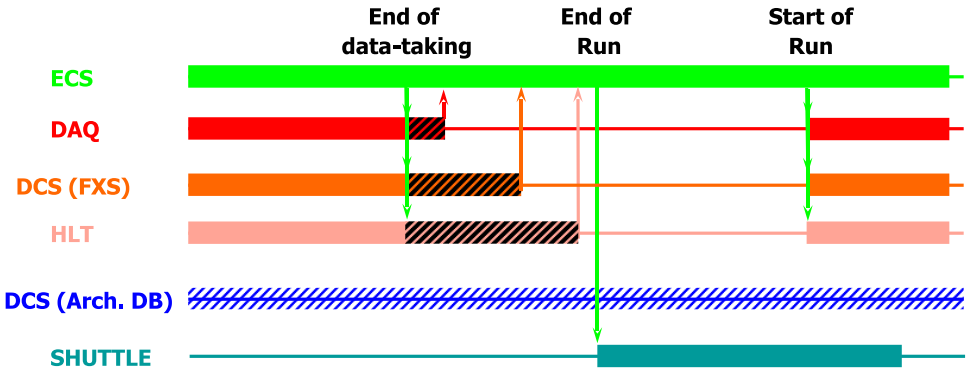

Figure 2. Sequence diagram.

  - parameters monitored continuously and archived in the DCS archive are retrieved by means of a dedicated communication protocol (see Section 2.2);
  - parameters created during data acquisition by the subdetectors' detector algorithms (DAs, see Section 2.3) are stored to files which are moved to a so-called File eXchange Server (FXS, see Section 2.4), to which the Shuttle has read access. Each online system provides its own FXS;

- it provides a base class for the implementation of the subdetector-specific code (the 'preprocessors', see Section 2.5) for the treatment of the conditions data before final storage to the OCDB (see Section 2.6);
- it accesses the OCDB to store the conditions data and, if needed by the detector preprocessors, to read previously stored conditions data.

Figure 2 shows the sequence diagram of the Shuttle system. At the end of each run the following actions are performed:

- at the end of data-taking the ECS informs the other online systems (DAQ, DCS, and HLT) that data-taking has stopped. This information is then passed by each online system to the corresponding detector algorithms (DAs);
- the DAs finalize the conditions data observed in the run, store them as files, and copy them to the corresponding FXS;
- once all the DAs of a given online system finish processing their data, the online system indicates its readiness back to ECS;
- once all the online systems are ready, ECS sends an EOR signal to the Shuttle, which performs the following tasks per subdetector:
  - querying the monitored data stored in the DCS archive;
  - running the preprocessors. The Shuttle runs them sequentially, however the different preprocessors are fully independent of each other, which in principle allows parallel processing;
  - retrieving the data from the FXSs requested by the detector preprocessors;
  - storing the conditions data produced by the preprocessors to the OCDB.

  The Shuttle framework monitors the resource consumption of the preprocessors and aborts them if they exceed critical values or if they time out. In case a preprocessor fails, it is restarted at a later stage until a certain number of retries is exceeded. In such a case the conditions data usually produced by this preprocessor will be missing. Reconstruction may

still be performed using previously collected conditions data; however, the impact on the quality of the reconstructed data has be evaluated manually.

It is important to note that the Shuttle does not interfere with data-taking: as shown in the sequence diagram (Figure 2), a new run can be started before the processing of the Shuttle for the previous run finishes. Therefore, the Shuttle does not delay data-taking under any circumstances.

In the following paragraphs the components that interact with the Shuttle are shortly described.

2.1. ECS Logbook

The ECS framework writes the relevant information about data acquisition in a database called 'logbook'. The Shuttle uses the logbook to determine which runs have to be processed. Three tables of this database are accessed by the Shuttle:

- the **run logbook** holds general information about the run;
- the **shuttle logbook** contains the 'global' processing status of each run as well as the processing status of each of the subdetectors that participated in the run. The ECS fills the table at the EOR, and the Shuttle updates it during processing;
- the **trigger configuration logbook** contains the CTP configuration.

2.2. DCS Archive and the AMANDA Protocol

Certain conditions parameters (e.g. device temperatures and gas pressures) are monitored and archived continuously and asynchronously with respect to data acquisition by the DCS system. This data is stored by every detector in an Oracle database using the PVSS framework [5]. Each monitored value is identified by a datapoint (DP) name (optionally also an alias name). Each value is associated with a timestamp that contains the exact moment of time when it was stored.

A server-client communication protocol was developed in collaboration with the DCS group with the goal to make these parameters available to the Shuttle. The protocol is described in [2, Appendix A]. It describes the communication between a service called AMANDA (Alice MANager for Dcs Archives, which is running in DCS) and the Shuttle. The AMANDA service retrieves conditions data from the archive database and passes it to the Shuttle.

2.3. Detector Algorithms

A detector algorithm (DA) is a program that runs in one of the online systems and produces conditions data. It publishes the collected data in the FXSs of the online systems which are accessed by the Shuttle. There are no other means of communication between DAs and the Shuttle.

2.4. File Exchange Servers

A file exchange server (FXS) is used as a temporary storage for data produced in a run that is to be picked up by the Shuttle. It is the 'data link' between the DAs and the Shuttle. Each of the three online systems DAQ, DCS, and HLT provides a FXS. A FXS consists of a database that contains information about the available data and a storage solution. Data is stored in files.

The online systems provide framework functions for DAs running in their sphere to write to the FXS. The Shuttle communicates with the FXS, retrieves the available files, and provides them to the preprocessors.

Each online system implements its own FXS and associated database as well as the software for creation, handling, and transfer of the conditions parameter files (produced by detector-specific code).

2.5. Preprocessor
The preprocessor contains the specific code of the subdetector that handles the processing of the conditions data related to it. It allows the subdetector experts to query the conditions data, reformat it into ROOT format if needed, and store it in the OCDB.

Examples of already implemented functionality in preprocessors are:

- fitting of temperature and pressure evolution (Time-Projection Chamber);
- analysis of spectra to obtain channel delays (Time-of-Flight Detector);
- finding of dead and noise maps (Silicon Pixel Detector, Time-of-Flight Detector).

To aid the development of the preprocessor, a test suite is provided that allows to debug it with artificial inputs without the need of the online systems. Additionally, an automatized nightly test is performed on the most recent code changes. It uses test instances of the Shuttle and the online systems with simulated data.

2.6. OCDB Framework
The Offline Conditions Database (OCDB) is the location where the experimental conditions data is stored. It is not a relational database but a set of entries in the AliEn file catalog that point to physical entities (ROOT files stored in various storage elements of the Grid) containing the conditions data.

The organization of the database is handled by the OCDB access framework, a package included in AliRoot. The OCDB design follows the following principles:

- conditions data is stored in ROOT `TObjects` that are stored in ROOT files;
- calibration and alignment objects are run-dependent objects;
- the database is of write-once-read-many (WORM) type. Once an object is stored, it is never removed. However, an object with higher version number can be added (automatic version control of the stored objects);

Each OCDB object is valid in a specific run range and has an associated version number. This version number is automatically increased every time a new object is inserted in the OCDB when another object with overlapping run range is already present. If no version is specified upon retrieval of a certain object for a specific run, the object with the highest version valid for that run is returned.

Data stored in the OCDB is replicated to the various Grid sites where reconstruction takes place. In this way the condition files are easily accessible when the reconstruction is performed. Besides in the OCDB, the Shuttle can store data in a second OCDB-like database, called *Reference DB*. The difference with respect to the OCDB is that reference data is not used for reconstruction: the Reference DB is the place to store anything that may be useful for 'manual' debugging of the physics data. Such debugging is expected to cause less frequent and less bandwidth-consuming access than reconstruction. For these reasons, contrarily to the OCDB, the Reference DB is not replicated.

3. Shuttle Status and Error Recovery
The Shuttle stores the processing status for each run and each subdetector's preprocessor. This information is used for error recovering. Figure 3 shows the different statuses together with the possible transitions between them. The following statuses and transitions exist for the processing of one preprocessor for one run:

(i) the Shuttle starts processing the subdetector (status: `Started`). First, it is determined if the preprocessor requires the processing of this run. This is done by verifying if the run

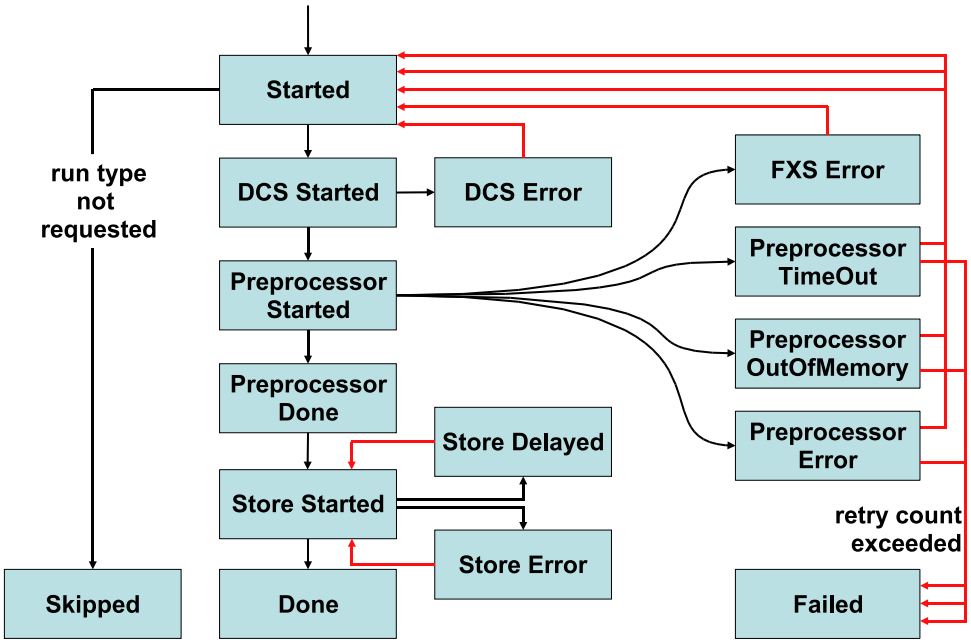

Figure 3. Preprocessor statuses during the processing.

type of the current run is part of the list of run types defined in the preprocessor. If the processing for this run is not required the status `Skipped` is set for the current subdetector and the processing is finished;

(ii) the program forks. The preprocessor is run in the child process while the parent monitors the child's progress. This method assures that the parent Shuttle process cannot crash by malfunctioning preprocessor code. Furthermore, possible memory leaks in the child preprocessor do not affect the parent process;

(iii) if the preprocessor requires data from the DCS archive, the child process retrieves it (status: `DCS Started`). In the event of a failure to retrieve the DCS archive data, the Shuttle sets the `DCS Error` status for the current subdetector and the processing is finished;

(iv) the Shuttle calls the subdetector's preprocessor (status: `Preprocessor Started`). The possible exit statuses of the preprocessor are:

- `FXS Error`: during the processing a connection to one of the FXSs failed. Therefore, the run is to be reprocessed at a later stage;
- `Preprocessor Error`: the preprocessor failed to process the data for the current run and returned an error code;
- `Preprocessor TimeOut`: the preprocessor exceeded the allowed processing time;
- `Preprocessor OutOfMemory`: the preprocessor exceeded the allowed memory usage;
- `Preprocessor Done`: the processing ended successfully. Conditions data has been stored on the local disc and is ready to be transferred to the Grid.

In case of one of the mentioned error statuses (`FXS Error`, `Preprocessor Error`, `Preprocessor TimeOut` and `Preprocessor OutOfMemory`) or if the child process terminates abnormally (e.g. segmentation violation) the processing is finished;

(v) the Shuttle stores the produced data in the OCDB (status: `Store Started`). The following error statuses may occur:

- if conditions data is supposed to be stored in the OCDB with infinite validity it

Run#	Run type	First seen	Last seen	SHUTTLE	ACO	EMC	FMD	GRP	HLT	HMP
			Last day							
92582	PHYSICS	today 09:00	today 09:24	Done h	Done (1) h	Done (1) h		Done (1) h		Done (1) h
92581	STANDALONE	today 00:41	today 00:44	Done h				Done (1) h		
92580	PHYSICS	today 00:28	today 01:18	Done h	Done (1) h	Done (1) h		Failed (2) h		Done (1) h
92579	STANDALONE	today 00:07	today 00:07	Skipped						
92578	STANDALONE	today 00:03	today 00:04	Done h				Done (1) h		

Figure 4. The main Shuttle monitoring page. See the text for a description of the different columns.

is required that all previous runs (i.e. runs with a smaller run number) have been processed for this detector already. If this is not the case the storing of these objects is delayed, the status `Store Delayed` is set and the processing is finished;
- in case of a failure transferring the conditions data to the OCDB, the status `Store Error` is set and the processing is finished;

(vi) if storage to the OCDB is successful, the Shuttle sets the `Done` status for the current subdetector and updates the ECS Shuttle logbook.

Upon failure of a preprocessor, its status in the ECS Shuttle logbook remains 'UNPROCESSED'. In the next iteration over the same run (occurring after a processing attempt is done on all other open runs) the program reads the previous processing exit status. The subsequent action depends on the previous exit status:

- if the status is `DCS Error` or `FXS Error`, the Shuttle restarts the processing without increasing the 'retry number'. In other words, the Shuttle tries to query the data from the DCS archive and the FXSs until the retrieval is successful;
- if the status is one of the possible preprocessor error statuses (`Preprocessor Error`, `Preprocessor TimeOut`, `Preprocessor OutOfMemory`) or `Preprocessor Started`, which means that the preprocessor crashed in the previous iteration, the Shuttle checks the retry number, which is saved together with the exit status. If the number of the allowed retries is exceeded the program declares the processing failed and updates the status and the Shuttle logbook. If the limit is not yet reached, it increases the retry count and restarts the processing;
- if the status is `Store Delayed` or `Store Error`, the Shuttle retries the transfer of the conditions data to OCDB, without increasing the retry count.

4. MonALISA Monitoring

The processing status of the Shuttle is monitored using MonALISA [6]. All status changes are sent to a MonALISA service and are visualized by the ALICE MonALISA repository, accessible on a dedicated web page. From this page, the processing status and the history of statuses can be seen for each run and subdetector. Furthermore, the output of the processing (log file) can be accessed, separately by subdetector and run.

Figure 4 shows an excerpt from the monitoring page. For each run that has been processed by the Shuttle the following information is available: the run type, the period in which the Shuttle has processed the run, and the overall Shuttle processing status. All subsequent columns (labeled ACO for the ACORDE subdetector and EMC for the EMCAL subdetector etc.), show the processing status per subdetector[6] which participated in the run (e.g. 'Done (1)'). The

[6] For visibility not all subdetector columns are shown in Figure 4.

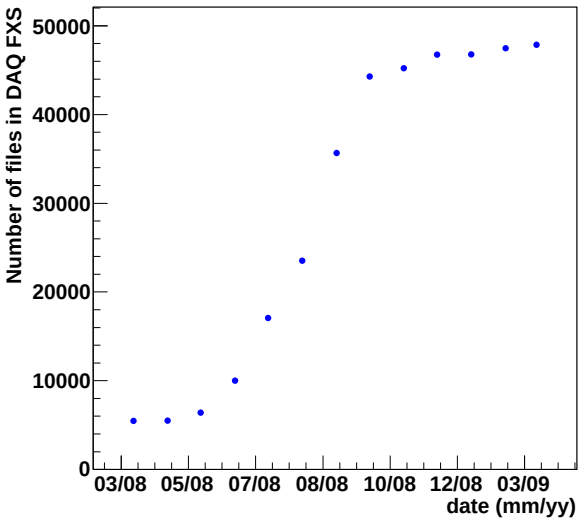

Figure 5. Cumulative number of files stored in the DAQ FXS in the period March 2008 to March 2009.

number next to the status of the subdetector indicates the number of retries that have been performed.

5. Usage Statistics

The Shuttle framework is in use at the ALICE experimental area (P2) since December 2007. Since then (up to March 2009), it has successfully processed about 45 000 runs. These were either cosmics runs, calibration runs, or runs taken with the LHC first circulating beams. During this period, the FXSs have been extensively used to store files for the transfer to the Shuttle. As an example, Figure 5 shows the cumulative number of files stored in the DAQ FXS in the period from March 2008 to March 2009. The total size of these files is about 90 GB.

During the same period, about 37 000 files have been stored in the OCDB, with a total size of 5.2 GB. As a result of the consolidation done in the preprocessors, this is about 17 times smaller than the size of the files on the DAQ FXS. Meanwhile, about 9 500 Reference Data files have been stored in the Reference Data DB, for a total size of 3.2 GB.

The core Shuttle system turned out to be very robust and restarts were needed less than two times per month, mainly due to software updates. Retries were needed in case one of the input online systems was temporarily unavailable. The preprocessors themselves were less stable, especially at the beginning. However, because of the previously described strategy of running the preprocessors in a subprocess the core system was not affected. Providing the necessary fixes for the subdetectors can be cumbersome because careful reproduction of a problem is needed outside of the production system.

6. Summary

The ALICE online-offline framework for the extraction of conditions data called Shuttle has been running smoothly at the ALICE experimental area since December 2007. It serves as the only possible link between the data produced in the online systems DAQ, DCS, ECS, HLT, and Trigger, and the experiment subdetectors. It is in charge of handling the detector procedures called preprocessors necessary for the gathering of the conditions data needed for reconstruction.

The preprocessors are run sequentially and independently by the Shuttle, under constant monitoring of the status and the resources used. The Shuttle has been running continuously during cosmics runs, calibration runs, and during the first LHC data taking. Upgrades, fixes, and new features have been introduced in the code following the requests of both the online systems and the detectors.

References

[1] Brun R and Rademakers F 1997 ROOT: An object oriented data analysis framework *Nucl. Instrum. Meth.* A **389** 81.

[2] Colla A and Grosse-Oetringhaus J F 2008 The Shuttle Framework - a System for Automatic Readout and Processing of Conditions Data *ALICE internal note* **ALICE-INT-2008-011** *https://edms.cern.ch/document/924807/1*

[3] Grosse-Oetringhaus J F 2009 Measurement of the Charged-Particle Multiplicity in Proton–Proton Collisions with the ALICE Detector (Appendix D) *Ph.D. thesis, University of Münster* **CERN-THESIS-2009-033**

[4] *http://aliceinfo.cern.ch/Offline*

[5] ALICE collaboration 2004 ALICE Technical Design Report of the Trigger, Data Acquisition, High-Level Trigger, Control System **CERN/LHCC 2003/062** *https://edms.cern.ch/document/456354/2*

[6] Legrand I C, Newman H B, Voicu R, Cirstoiu C, Grigoras C, Toarta M and Dobre C 2004 MonALISA: an Agent Based, Dynamic Service System to Monitor, Control and Optimize Grid Based Applications, *Proc. Conf. for Computing in High-Energy and Nuclear Physics (CHEP), Interlaken, Switzerland 27 Sep - 1 Oct 2004*