ATLAS Tile Calorimeter Conditions Database architecture and operations in Run-2

Yuri Smirnov, Dhiman Chakraborty
Northern Illinois University, USA
Alexandre Solodkov, Institute for High Energy Physics, Russia
Siarhei Harkusha, The National Academy of Sciences, Belarus

1. Introduction

An overview of the Conditions Database (DB) structure for the hadronic Tile Calorimeter (TileCal), one of the ATLAS Detector sub-systems, is presented. ATLAS Conditions DB stores the data on the ORACLE backend, and the design and implementation has been developed using COOL (Conditions Objects for LCG) software package as a common persistency solution for the storage and management of the conditions data. TileCal Conditions and calibration data are stored in 4 separate Databases (DB) also known as schemas: TileCal Online and Offline DBs for data, DB for Monte Carlo (MC) simulation and Detector Control System (DCS) DB. In order to support the smooth TileCal operations during data taking period, experts perform the necessary calibrations, add the changes of detector status and other conditions data, prepare new conditions for data reprocessing and MC production campaigns, and upload the new up-to-date information into DB using the custom-made software tools. The procedure of TileCal conditions preparation, validation and uploading to DB is described, and some DB-related statistics collected in RUN-2 is provided.

2. Architecture

- In RUN-2 two ORACLE DB instances have been used: CONDBR2, keeping conditions for data (online data taking, offline data processing), and OFLP200, keeping conditions for MC simulation data production.
- Every ATLAS sub-system, including TileCal, uses several COOL DB schemas. Two of them are shown in Figure 1.
- Each schema in COOL is organized into a hierarchical folder structure, similar to a file system, and every leaf folder in the Offline DB keeps multiple versions of tags linked to ATLAS Global tags. New leaf tags (and hence new ATLAS Global tag) with a new set of conditions data are usually created for every round of reprocessing. In particular, two different Global tags always exist: tag for the prompt reconstruction of express stream (10% of data), so-called UPD1 tag; and tag for the bulk reconstruction, so-called UPD4 tag. In the Online DB the single (head) version tags are used.
- Every tag contains a set of conditions which are split into Intervals of Validity (IOVs) to allow for conditions to vary between runs and/or lumiblocks within the same run.
- To store the conditions in COOL DB we use Coral Blobs (coral::blob). A Binary Large Object (Blob) is a collection of binary data stored as a single entity in a DB management system.
- TileCal conditions data is stored with modules granularity: for 4x64=256 TileCal modules we use 256 COOL channels containing one Blob each as a conditions payload.
- 64K Blobs are sufficient for the majority of conditions, but there are a few special cases when we use the larger 16M Blobs: for Cell Noise RMS the only COOL channel 48 used, adopted as CALO Blob (shared by TileCal and LAr), and for OFC the only COOL channel 0 is used to store the constants for all modules. An extra COOL channel 1000 keeps text comments on a given DB update.
- We apply offline module-hash [0-275] for indexing, using indices [0-19] to store partition depending defaults.This reduces significantly (~20%) the DB volume and avoids duplication.

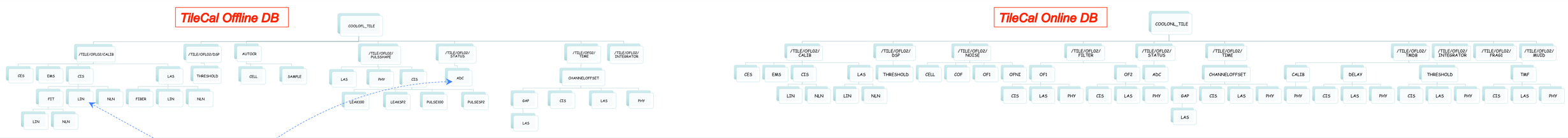


Figure 1: Schema of folders in TileCal conditions DB, CONDB2 instance. Each leaf folder in the Offline DB keeps multiple tags linked to ATLAS Global tags, in the Online DB the single (head) version tags are used.

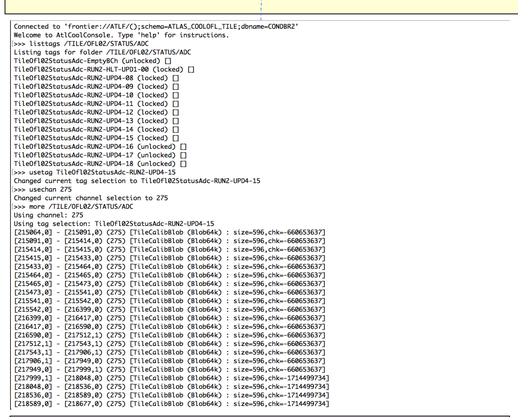
3. TileCal conditions DB software

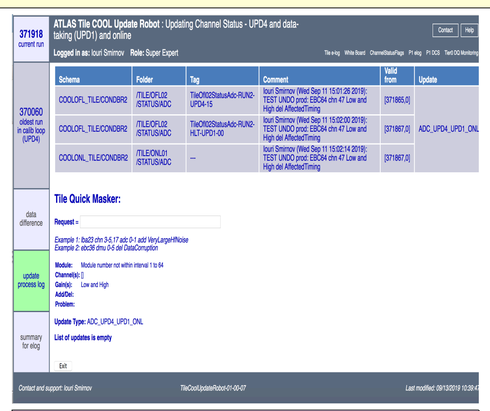
To communicate with TileCal Conditions DB (read, write, update) both ATLAS-made DB tools such as *AtlCoolConsole, AtlCoolCopy, AtlCoolMerge, COMA*, etc. and the following TileCal-made specific software can be used:

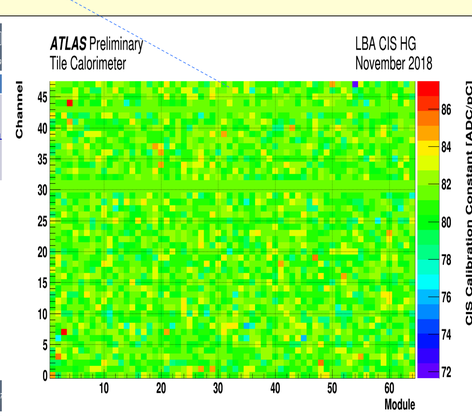
- *TileCalibBlobObjs* − definitions of TileCal data types in COOL
- *TileCalBlobPython* − python methods to work with COOL
- *TileConditions* − package to work with IOVs
- *TileCalibAlgs, TUCS* − client tolls for calibrations
- *TileMonitoring, Tile-in-One* − tools for monitoring
- *TileCalibWeb* − application for COOL DB updates
- *TileDQ* − software for TileCal data quality verification

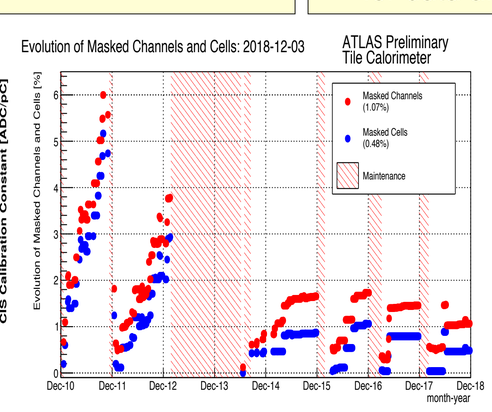
4. RUN-2 DB operations

TileCal Conditions database operations include both the preparation, validation and performing frequent COOL DB updates, supporting the Tile DB infrastructure (client tools and environment) and troubleshooting. Frequency of TileCal DB updates during the entire RUN-2 period (2015-2018) is shown on Figure 6. This statistics includes TileCal Online and Offline Conditions DBs both for data and MC simulations.









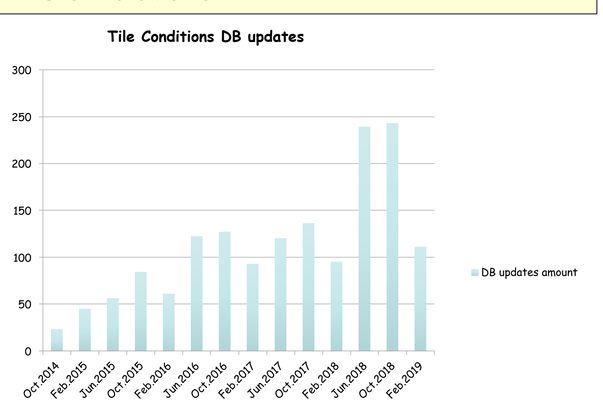


Figure 2: Structure of Tile DB folder, tag and IOV. AtlCoolConsole view

Figure 3: TileCalibWeb (Robot) tool for Conditions DB updates

Figure 4: CIS conditions for LBA modules UPD4-18 tag

Figure 5: Evolution of Masked Channels and Cells

Figure 6: TileCal Conditions DB updates summary

To see the structure and content of any TileCal Conditions DB the common ATLAS-written command line interface AtlCoolConsole.py can be used. On Figure 2 an example of TleCal Offline RUN-2 Conditions DB in the AtlCoolConsole view for /TILE/OFL02/STATUS/ADC folder is presented. It is easy to see all UPD1 and UPD4 tags for that folder, as well as a list of IOVs and payload Blob types used for various tags and 277 COOL channels. The COOL channel 275 corresponding to TileCal EBC64 module is presented. We use two types of TileCal Conditions DB updates: a large DB update usually requires creation of the new tag needed for data reprocessing or for MC production campaigns, and can be performed by uploading the corresponding SQLite file with help of AtlCoolMerge command executed from the command line. The second type of update is a minor correction of few COOL channels (for instance masking or unmasking the bad channels or cells) in TileCal. Such an operation can be done by adding a new IOV to the existing tag directly from the custom TileCalibWeb interface presented on Figure 3. To read the conditions back from COOL and plot them we use python scripts (ReadCalibFromCool.py, PlotCalibFromCool.py, etc.) from TileCalibBlobPython package. An example of the plot showing the Charge Injection System (CIS) calibration constants (ADC counts / pC) for all the high gain channels in TileCal LBA modules for a single run 367000 taken in November 2018 is presented on Figure 4. It is created by PlotCalibFromCool.py script reading CIS constants back from TileOfl02CalibCisLin-RUN2-UPD14-18 tag of COOLOFL_TILE Database and represents a map of the individual CIS calibration constants, for each channel and module in one partition. Figure 5 shows the percentage of all cells and channels in the detector that are masked as a function of time starting from December 2010. The detector status of the 3-th December, 2018 is noted. The hatched area represents the maintenance period of the detector. The legend includes the amount of masked cells (0.48%) and masked channels (1.07%) at that time. The total number of cells, including gap, crack, and minimum bias trigger scintillators, is 5198. Amount of the TileCal Conditions DB transactions, in particular update operations, increased significantly during Run-2 every year up to the corresponding end-of-year shutdown, and decreased during the maintenance periods. TileCal DB software allows us to optimize and automatize the procedure of preparation conditions data, validation and uploading to ORACLE DB.