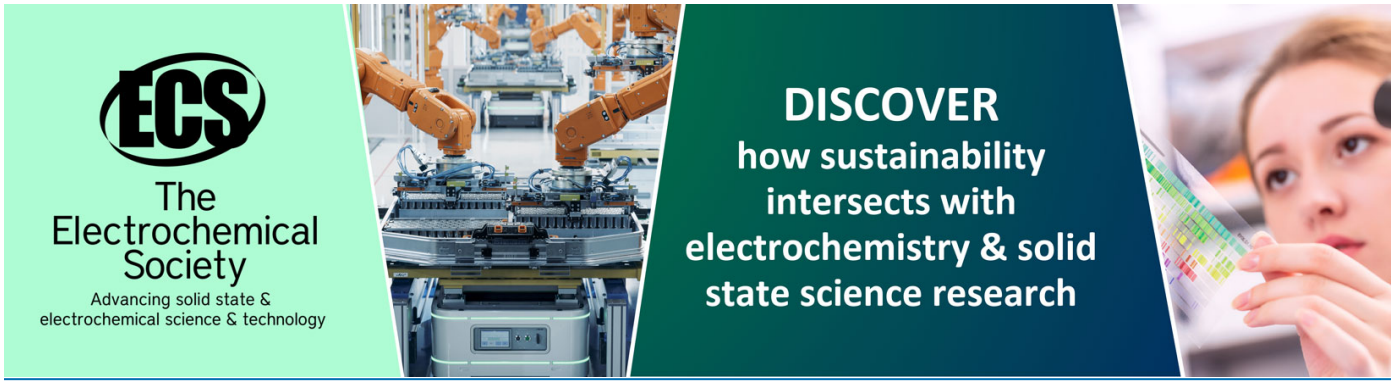
Collecting conditions usage metadata to optimize current and future ATLAS software and processing



IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 042028 doi:10.1088/1742-6596/898/4/042028

Collecting conditions usage metadata to optimize current and future ATLAS software and processing

L Rinaldi[1], D Barberis[2], A Formica[3], E J Gallas[4], S Oda [5], G Rybkin[6], M Verducci[7] on behalf of the ATLAS collaboration

[1] Dipartimento di Fisica e Astronomia, Università di Bologna, and INFN Sezione di Bologna, Via Irnerio 46, I-40126 Bologna, Italy
[2] Dipartimento di Fisica, Università di Genova, and INFN Sezione di Genova, Via Dodecaneso 33, I-16146 Genova, Italy
[3] CEA/Saclay IRFU/SEDI, 91191 Gif-sur-Yvette, France
[4] Department of Physics, Oxford University, Denys Wilkinson Building, Keble Road, Oxford OX1 3RH, United Kingdom
[5] Department of Physics, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka, 819-0395, Japan
[6] Laboratoire de l'Accélérateur Linéaire, Université Paris-Sud, CNRS/IN2P3, Orsay, France
[7] Dipartimento di Fisica, Università La Sapienza, and INFN Sezione di Roma-1, Piazzale A. Moro 2, I-00146 Roma, Italy

E-mail: Lorenzo.Rinaldi@cern.ch, gallas@cern.ch

Abstract. Conditions data (for example: alignment, calibration, data quality) are used extensively in the processing of real and simulated data in ATLAS. The volume and variety of the conditions data needed by different types of processing are quite diverse, so optimizing its access requires a careful understanding of conditions usage patterns. These patterns can be quantified by mining representative log files from each type of processing and gathering detailed information about conditions usage for that type of processing into a central repository.

1. Overview of the ATLAS conditions database

The ATLAS experiment [1] at the Large Hadron Collider (LHC) at CERN has been collecting large volumes of data from particle collisions (events) within its detector since 2009. To date, ATLAS has collected event data volumes in the hundreds of petabytes for many billions of events. In addition, large datasets of Monte Carlo (MC) simulated data have been produced which are used to compare the real data to theoretical models.

The recording or generation, and then the processing and analysis of the real or simulated event-wise data require auxiliary data called "conditions data". These include a wide variety of information such as Detector Control, Trigger DAQ, Data Quality, and other information from ATLAS sub-detectors and the LHC accelerator: they characterize the states, calibrations, or alignments of these systems in specific time intervals. Conditions data are not stored with event data files, rather, their changes are chronicled by time intervals, commonly called Intervals of Validity (or IOV), and stored in the ATLAS Conditions Database grouped by logical or physical subsystem from which they originated.

Conditions data are stored in a database with a structure based on the LCG-COOL Database infrastructure [2]. The data themselves are stored in a CERN Oracle back-end and they are managed by subsystem experts via the COOL API [3]. Thus far, we have accumulated several TB of conditions data associated with the event data recorded throughout LHC data taking periods.

An overview of the Conditions Database structure is shown in figure 1.

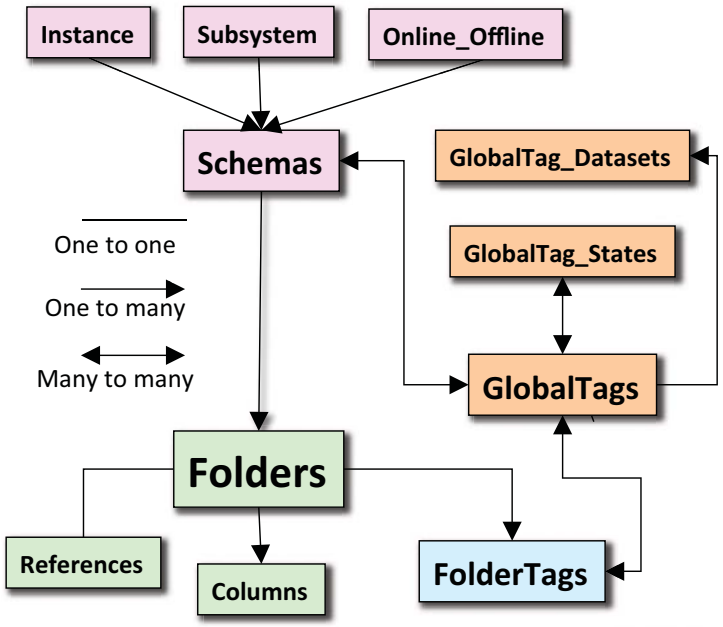


Figure 1. ATLAS Conditions Database overview.

Each subsystem stores its conditions under one or more Conditions *Schemas* which each corresponds one-to-one with a specific underlying Oracle table owner. Experts for each Schema organize and manage each of their different kinds of conditions into dedicated *Folders*, each of which corresponds to a distinct set of underlying database tables storing that specific data. Thereby, the infrastructure allows a wide range of customization in Folder (i.e. table) structure according to the nature of the conditions it is designed to contain, whether it be states, calibrations, or alignment related.

Folders which are expected to have more than one version for conditions in the same IOV are called multi-version folders. These versions (or 'tags') are stored in the *FolderTags* table. Conversely, conditions which have no versioning, such as detector configuration (i.e. settings fixed at run time), are called single-version folders and have no entries in the FolderTags table.

Event-wise processing jobs require conditions data from many subsystems to be specified at job configuration time. In preparation for that processing, each subsystem includes their preferred FolderTags for that processing in a specific *GlobalTag* which will be used as input to the event processing tasks.

We have found that the present system has many limitations. The Conditions Database implementation stores the data in over 17,000 Folders in the Conditions Database (managed under more than 30 different Oracle Schemas). This makes management of the data, at a global level, challenging to understand its content, intended use, and possible obsolescence. Furthermore, gathering exactly how specific conditions are actually used is difficult: conditions

data have quite diverse clientele from detector operations (for example, in subsystem monitoring) to a wide variety of data processing (including event processing as well as refinements in calibration or alignment, as examples). This 'usage' information, when logged, is distributed in a large number of disparate sources which are generally not collected centrally for easy access for our purposes.

One might think that GlobalTags, which are collections of FolderTags for a specific type of processing, would offer a coherent picture, but they also are not exhaustive enough:

- GlobalTags include conditions only from multi-version folders. They do not include conditions from single-version folders (folders which have no FolderTags).
- Specific processing tasks may add the use of additional FolderTags in their task configuration which are outside of the GlobalTag.
- GlobalTags may be used for a type of processing which is some variation from the specific type of processing for which the GlobalTag was built, so the associated jobs may access conditions from the database which the job never actually utilizes.

The specific goals of the project described herein are to improve our understanding of conditions data usage by event-wise processing and to develop improved tools and resources for future processing. In this paper we describe the system being developed to collect the conditions usage metadata per job type and describe a few specific (but very different) ways in which it can be used. Specifically:

1. It can be used to cull specific conditions data into a much more compact package to be used by jobs doing similar types of processing: these customized collections can then be shipped with jobs to be executed on isolated worker nodes (such as HPC farms) that have no network access to conditions.
2. It can be used to understand the current usage of conditions data by specific types of processing. This information can be fed back into the creation of more appropriate GlobalTags for special forms of processing.
3. Another usage is in the design of future ATLAS software, to provide Run-3 software developers essential information about the nature of current conditions accessed by software. This helps optimizing internal handling of conditions data in order to minimize its memory footprint while facilitating access to these data by the sub-processes that need them.

2. The ATLAS database release

ATLAS has two modes of providing database-resident data (from ATLAS Conditions, Geometry and Trigger Databases) to processes using those data anywhere on the global computing grid or processes 'off the grid':

- **ATLAS Frontier [4]**: is accessible from any grid site. Processes direct their queries to Frontier Squid nodes, which deliver the database response either from their caches (if the data are available and up to date) or otherwise via Frontier queries to the Oracle database at CERN or one of the Oracle replicas at selected Tier-1 centers.
- **ATLAS DB Release [5]**: is a file-based distribution. DB Releases are composed of file-based 'static' SQLite replicas containing relational data from the above databases plus the POOL ROOT [3] Payload data files pointed to by reference in some Conditions Database Folders. DB Release tarballs are distributed on the grid and on shared file systems.

These two modes of database distribution each have advantages and disadvantages depending on the nature of the process requiring the data.

For real data processing, the Frontier mechanism works well: the breadth of needed conditions varies widely so the case for the creation of specific DB Releases for these purposes is untenable.

For MC data processing, the extent of involved conditions is not so wide *but* MC processing is executed on a wider variety of computing environments including on isolated environments such as HPC farms where network access to conditions data are simply not available. So for 'off-grid' MC processing, the DB Release mechanism is the *only* way to provide these jobs with the database data they need. Therefore, the DB Release mechanism has been maintained over the years principally for use by specific cases of MC processing.

The system for the provision of the DB Release, however, is in great need of refinement: The default MC DB Release has grown exponentially in size in the last $\sim$ 2 years, beyond the limitations of usability. The reason it has grown is because of the way new conditions are added to the DB Release for MC. A new MC DB Release version is built when a new GlobalTag with the latest MC Conditions is created [6]. By default, the system simply adds the new GlobalTag and any new conditions to the content of the SQLite file of the existing DB Release (which contains all previous GlobalTags). As shown in figure 2, the data volume of the DB Releases is steadily increasing up to a size of 9 GB (in the latest version) because new conditions are simply appended to the existing file. The average composition of a DB Release is shown in figure 3.

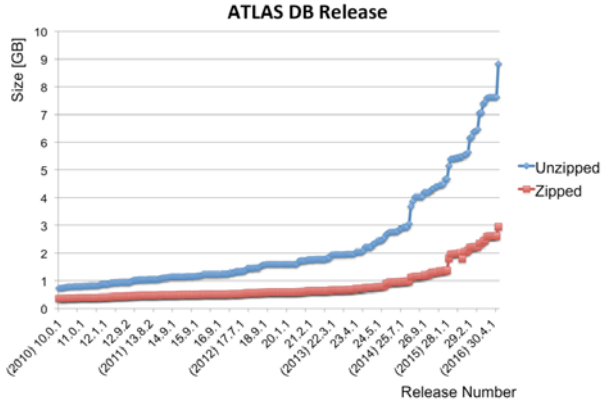


Figure 2. Size of the DB Releases as a function of time and release version. Compressed and uncompressed sizes are shown.

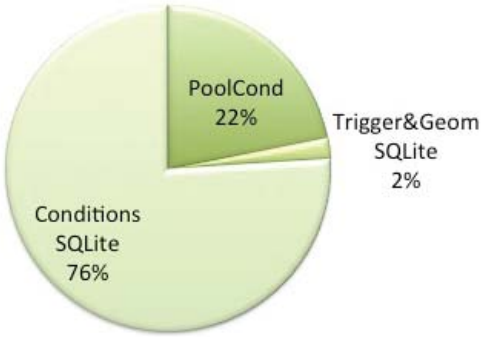


Figure 3. Composition of the DB Release volume (percentage).

The historical motivations behind the above implementation were a combination of simplicity and completeness: MC DB Releases did not grow appreciably in the early years and the fact that the DB Release contained all MC GlobalTags meant that any GlobalTag, even the oldest, could be used at any time using the latest DB Release.

But times have changed ! Reference [6] describes how GlobalTag management in ATLAS has evolved in recent years, moving to a model where the latest GlobalTags (containing the latest stable conditions) should be used. This model makes much older MC GlobalTags obsolete for current processing and thus no longer needed in the MC DB Release.

Investigations of recent MC job log files confirms that not all the information stored in DB Releases is needed. In fact, many types of recent MC processing use the latest GlobalTag. Our initial goal is to produce a customized and lightweight DB Release that contains only the information needed for specific types of MC processing. We found that collecting this information from job logs manually is time consuming, tedious, and error prone: but the exercise in doing so demonstrated the "proof of concept" that the generated customized DB Release did successfully provide the needed data to test jobs. So the next step is to develop a more automated system.

We describe in the following sections a system which collects and stores the needed information into a repository which can be used to more automatically create streamlined DB

Releases containing customized snapshots of conditions to be used for MC processing. The same information helps us meet our other goals: to gain a better understanding of current conditions usage as well as to provide software developers with information they need to improve conditions handling in future software.

3. Extending ATLAS conditions DB metadata

We build upon an existing component of the COMA system described at CHEP in 2013 [7] which is a repository storing ATLAS Conditions Database descriptive and structural metadata including its Schemas, Folders, FolderTags and GlobalTags. This enhancement builds upon an existing stable data infrastructure which is already accessible via a number of methods (in addition to the web-based COMA Reports): via a python-based API (pyAMI [8]) and with a RESTful service using direct PL/SQL [9].

An overview of the schema changes to existing COMA Database tables is shown in figure 4 with new columns shown in a red font and new tables shown in yellow.

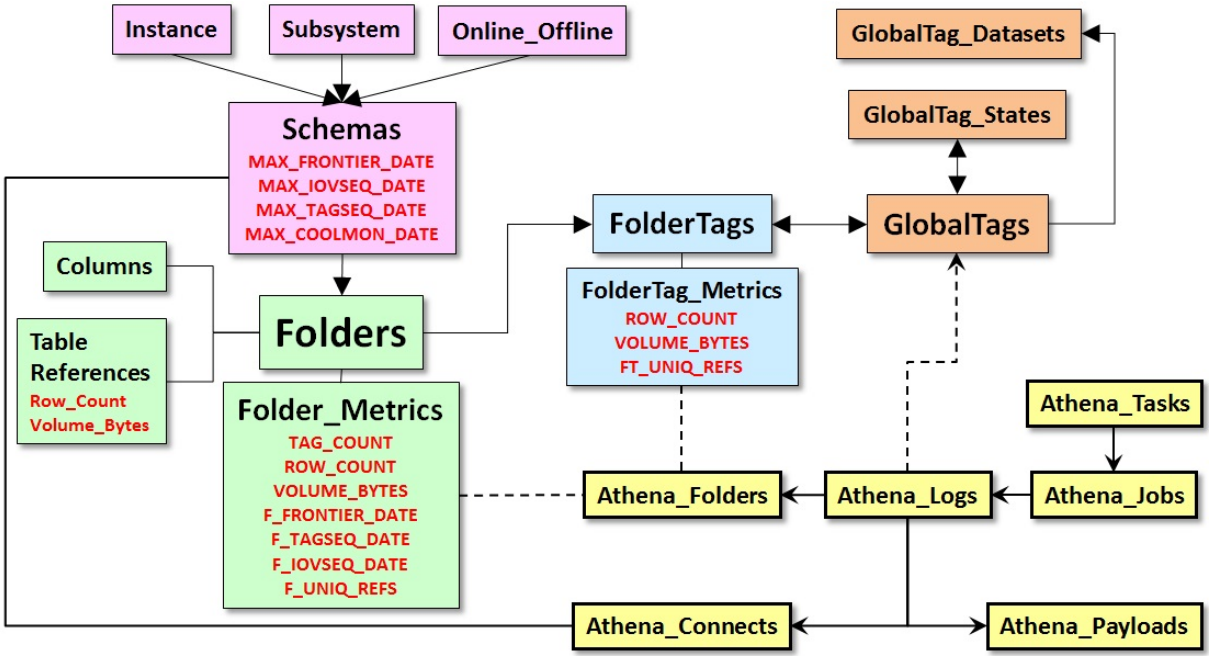


Figure 4. COMA Conditions DB Metadata Schema: Adding columns to include more metrics and dates (shown in red) and adding new tables to contain conditions usage from Athena jobs (shown in yellow).

We extend the schema in two ways:

1. Include more information about the content of existing Folders and Tags (generically called 'metrics' below) and
2. extend the COMA schema to include conditions usage collected from Athena [10] job logs and associate those with the other information collected about Folders and Tags.

The details and motivations for such supplementation are described in the subsections below.

3.1. Adding conditions database metrics

The existing COMA database schema reflects the structure of the ATLAS Conditions Database in terms of the relationships between the Schemas, Folders and FolderTags. To further our

mission to better understand the conditions database content and improve tools for extracting subsets of conditions for specific purposes, we have added additional columns to existing tables as shown in the above figure in red font.

The new information is collected from a variety of sources:

- **Oracle stats** are gathered by Oracle scheduler jobs executed daily on the ATLAS Conditions Database. These include row counts and average row length per Oracle table and the associated timestamp of this information. Using this information, we can compute an approximate average row length per Folder.
- **Oracle Sequence tables** are incremented each time a new row is added to a Folder (new conditions data have been inserted) or FolderTag (a new tag has been inserted). The associated timestamp of that insert is stored in the sequence table.
- **Frontier cache consistency tables** are used by the Frontier system to check when a Folder was last modified, which is critical to provide clients with the latest conditions. These data reflect the most up-to-date information about recent activity, but do not include accurate timestamps for Folders which have not seen new data recently (beyond the scope of what is needed by Frontier).
- **ATLAS Conditions monitoring** stores Oracle stats metrics as described above on a daily basis, creating a daily history of the growth of ATLAS Conditions Folders for monitoring purposes.
- **Conditions data external references** are utilized in some folders by a number of subsystems: These are GUID-like pointers stored in the Conditions Database which refer to external files stored in the ATLAS data file management system Rucio [11]. We do not collect the references themselves but count the number of unique external references by Folder and FolderTag.

Adding more content to COMA adds to the challenge of synchronizing the metadata to reflect the up-to-date Conditions DB content mainly due to the sheer volume of the conditions data. While, in principle, changes can occur in any of the Schemas at any time, in reality, only the Run 2 folders are generally active: The new timestamps being introduced into the system inform the loading program which Schemas (and Folders) have had new content added recently so it can skip elements where the timestamp indicates inactivity in that area, speeding up the synchronization.

Furthermore, a relaxed synchronization, updating every few hours, is sufficient for a number of reasons:

- Some of the new data sources (noted above) are updated only once per day: There is no gain in synchronizing with those sources more frequently.
- The use cases do not call for up-to-the-minute synchronization: The objectives outlined in this presentation call for data which are in the system for at least a day.
- Approximate values for some metrics are all that is really needed for the use cases.

One may note that there is some redundancy in the above information which provides some cross checks of the input data sources. Some timestamps are used only by the loading program while others are stored to relay the date of latest changes, per Schema, per Folder and per FolderTag which is useful to identify which folders are 'active' (have recent new data) and which are 'dormant'.

Row counts and volumes are useful to understand total data volumes by Folder and by FolderTag. The count of unique external references per Folder and per FolderTag is useful in understanding the extent of their use. And because we store sufficient granularity of these quantities, we can use them to gauge the size of customized DB Releases in advance of them

being produced: to know the volume of the selected in-line conditions and to know exactly the number of POOL files which will be included (if the customized DB Release is based on a single GlobalTag).

In addition, they can be used to understand the relative fraction of data accessed by processing jobs (as discussed in the next section), a common question raised by future software developers.

3.2. Collecting conditions database usage patterns

Conditions data access from event data processing is by far the most intensive use of the conditions data. We seek to understand this usage for many reasons, as previously mentioned.

The only source of this information per job is found by mining the data in log files of the jobs produced within Athena [10], the ATLAS offline processing framework. The Athena framework contains a module called the *IOVDbSvc* which manages all database connections and queries to the conditions database centrally, and in the process, collects metrics of those connections and interactions. While Athena log files are unwieldy beasts, the IOVDbSvc writes its metrics to the log file in a consistent and coherent manner. This makes mining this information from the logs relatively straightforward.

The log file itself, however does not contain all the information needed. In particular, it does not contain general information about the ATLAS task which deployed the job or even the names of the input and output files. Collecting this information is critical to understand the purpose of the job. After all, we need to be able to associate the information collected from the log with the specific use case of interest to experts: They would use information at the task or job level for data discovery in this system. Also, one of our general goals is to build up a repository of the conditions usage of many jobs, so that we can compare and contrast their usage. So we want to design our system to include a broader view of conditions usage as well as provide that information about one specific instance.

Therefore, information collection starts at the task level. Then we identify a representative job deployed by that task which executed successfully, collecting information about that specific job such as its input and output data files. The Athena log from that job is then mined for the IOVDbSvc metrics.

The new tables, shown in yellow in figure 4 reflect the above workflow. An Athena task deploys some number of jobs and the schema allows information from one or more of those jobs to be loaded into the database. Because of multiprocessing, each job may deploy multiple sub-jobs, so the schema also includes this potential one-to-many relationship between Athena Jobs and Athena Logs. This layout also allows for the case when jobs include many steps in processing as is the case for simulation, where steps include event generation, digitization, followed by reconstruction, each of which has its own pattern of conditions access. The Athena Folders table stores the Folder access of each process in the Logs table. Additional information from the IOVDbSvc such as a summary of the number of database connections (per Schema) and other payload metrics are included in the two tables at the bottom of the figure. The new tables have relationships to the existing tables as shown.

We store considerable detail in the Athena Folder table including the number of data rows retrieved and their associated data volume with the time required to retrieve those data. The IOVs may also be stored if they are available in the log (a special job configuration makes that information available). For multi-version folder access, the FolderTag is also stored, while for single version folders, there would be no relation to tags. The schema is provisioned to store the number of times a job requests conditions data per folder and the number of POOL files accessed. The IOVDbSvc also manages the internal memory of conditions in the job and relays to the log when data are retrieved but never used by the job. This reveals potential inefficiencies in the system which then could be investigated and addressed.

The schema can accommodate information from both real data as well as MC processing, so the system can readily reveal the differences in conditions usage workflow between them.

4. Results: creation of a slimmed custom DB release

Several tools are already available to use the metadata information available in the COOL Conditions Database in order to extract the conditions data for a GlobalTag. In this case only versioned folders in COOL are easy to select by means of the GlobalTag association. Python tools that can access this information either directly from COOL (or from COMA) have been developed by using the REST API.

We were able to perform several tests by running typical simulation transformations, accessing the initial default configuration. Then, at the end of the job, a post-exec command has been used to dump all the Folders and FolderTags loaded by the job, by analyzing the Athena log files. As a result we observed that only a fraction of relations between Folder and FolderTag were used (ranging from 10% to 20%). In this way we were able to reduce the volume of the SQlite file in the DB Release to a value between 10% and 20% of the original size. Further tests with this slimmed custom DB Release in a typical job work-flow were successfully performed.

5. Conclusions and outlook

Information contained in the ATLAS Conditions Database is spread over a large number of sources and collecting such metadata is challenging in some very specific tasks. We have studied the ATLAS Database Release use case and we managed to produce a prototype of a slimmed custom DB Release by collecting metadata information with new tools based on the analysis of the Conditions Database access patterns. In the near future our results may be improved by using information collected by the Conditions DB metrics to set up an automated procedure for DB Release slimming, based on both the analysis of the Conditions DB metrics and access patterns.

References

[1] ATLAS Collaboration 2008 JINST **3** S08003.
[2] Valassi A et al. 2004 LCG conditions database project overview CHEP 2014, Interlaken, Switzerland. .
[3] Trentadue R et al. 2012 LCG persistency framework (CORAL,COOL, POOL): status and outlook in 2012 *J. Phys.: Conf. Series* **396** 053067.
[4] Barberis D et al. 2012 Evolution of grid-wide access to database resident information in ATLAS using Frontier *J. Phys.: Conf. Series* **396** 052025.
[5] Borodin M et al. 2001 Scaling up ATLAS database release technology for the LHC long run *J. Phys.: Conf. Series* **331** 042004.
[6] Böhler M et al. 2015 Evolution of ATLAS conditions data and its management for LHC Run-2 *J. Phys.: Conf. Series* **664** 042005.
[7] Gallas EJ, Albrand S, Borodin M and Formica A (2014) Utility of collecting metadata to manage a large scale conditions database in ATLAS *J. Phys.: Conf. Series* **513** 042020.
[8] Odier J, Aidel O, Albrand S, Fulachier J and Lambert F 2015 Evolution of the architecture of the ATLAS metadata interface (AMI) *J. Phys.: Conf. Series* **664** 042040.
[9] Formica A and Gallas E J 2015 A JEE RESTful service to access conditions data in ATLAS *J. Phys.: Conf. Series* **664** 042016.
[10] Calafiura P et al. 2015 Running ATLAS workloads within massively parallel distributed applications using Athena multi-process framework (AthenaMP)" *J. Phys.: Conf. Series* **664** 072050. .
[11] Garonne V, Vigne R, Stewart G, Barisits M, Beermann T, Lassnig M, Serfon C, Goossens L and Nairz A 2014 Rucio The next generation of large scale distributed system for ATLAS data management *J. Phys. Conf. Ser.* **513** 042021.