The ATLAS Distributed Data Management project: Past and Future

Vincent Garonne\textsuperscript{1}, Graeme A Stewart\textsuperscript{1}, Mario Lassnig\textsuperscript{1}, Angelos Molfetas\textsuperscript{1}, Martin Barisits\textsuperscript{1}, Thomas Beermann\textsuperscript{1}, Armin Nairz\textsuperscript{1}, Luc Goossens\textsuperscript{1}, Fernando Barreiro Megino\textsuperscript{1}, Cedric Serfon\textsuperscript{2}, Danila Oleynik\textsuperscript{3}, Artem Petrosyan\textsuperscript{3} on behalf of the ATLAS Collaboration

\textsuperscript{1}CERN, CH-1211, Genève 23 Switzerland
\textsuperscript{2}Ludwig-Maximilians-Universität München, Munich, Germany
\textsuperscript{3}Joint Institute for Nuclear Research, Dubna, Russia

E-mail: vincent.garonne@cern.ch

Abstract. ATLAS has recorded more than 8PB of RAW data since the LHC started running at the end of 2009. Many more derived data products and complimentary simulation data have also been produced by the collaboration and, in total, 90PB is currently stored in the Worldwide LHC Computing Grid by ATLAS. All this data is managed by the ATLAS Distributed Data Management system, called Don Quijote 2 (DQ2). DQ2 has evolved rapidly to help ATLAS Computing operations manage these large quantities of data across the many grid sites at which ATLAS runs, and to help ATLAS physicists get access to this data.

In this paper, we describe new and improved DQ2 services, and the experience of data management operation in ATLAS computing, showing how these services enable the management of petabyte scale computing operations. We also present the concepts of the new version of the ATLAS Distributed Data Management (DDM) system, Rucio.

1. Introduction

The ATLAS experiment\cite{1} at the Large Hadron Collider (LHC) is a general purpose particle physics detector designed to investigate physics at the energy frontier. The ATLAS detector is capable of operating at the 20MHz collision rate of the LHC, however, online trigger systems reduce the rate of data taking to offline to 200-400Hz. Even with this drastic reduction ATLAS records a huge amount of data – more than 8PB of raw collision data have been taken since the LHC started running. This data is first processed by the ATLAS Tier-0\cite{2}, which runs first pass reconstruction of events and writes the data to tape at CERN. It then registers this data in the ATLAS Distributed Data Management system (DDM)\cite{3}. DDM organises, transfers and manages ATLAS data across more than a hundred individual grid sites that are part of the Worldwide LHC Grid\cite{4} in accordance with the policies established in the ATLAS Computing Model\cite{5}. This not only covers raw data, but the entire lifecycle of derived data products for the collaboration physics groups and individual physicists.

In this paper we give an overview of ATLAS Distributed Data Management, covering core concepts (section 2), performance with LHC data (section 3) and then, in some detail, new services and features (section 4). We also describe the concepts of the new version of ATLAS Distributed Data Management system Rucio (section 5) and its replica management model (6).
2. ATLAS distributed data management

2.1. Data model

ATLAS naturally has a large amount of data, which is physically stored in files. The actual distribution of the data over files is mostly incidental. The data consists usually, but not exclusively, of persistent C++ objects associated with physics events. DDM considers files to be its elemental unit. However, files themselves are usually defined by operational considerations and rarely correspond to a complete set of data of interest to the user of the system.

For this reason DDM allows the aggregation of files into *datasets*. Datasets are the operational unit of replication for DDM – they may be transferred to grid sites, whereas single files may not. Datasets in DDM may contain files that are in other datasets, i.e., datasets may overlap. This is illustrated in figure 1.

Datasets may be *open* (able to have new files added to them) or *closed* (no new files may be added). Closed datasets may have new versions created, which might have a different file content; however, datasets which are *frozen* cannot have files added or new versions created. These datasets are immutable (excepting that files which have been permanently lost can be removed).

![Figure 1. ATLAS Data Management core concepts. Individual files are collected into datasets (coloured outlines). Datasets may overlap and contain the same files (e.g., green and blue). Datasets themselves may be aggregated into containers, e.g., the red, blue and green datasets form the grey container.](image)

There is a further level of aggregation provided by DDM. This is the concept of a *container*, which is a collection of datasets (see figure 1). Containers are not units of replication, but allow large blocks of data, which could not be replicated to single site, to be described in the system.

In practice, most dataset overlaps and aggregation into containers are hierarchical: e.g., monte-carlo production will use small datasets, referring to a few jobs processed at a site; these are then aggregated into the main dataset, which refers to a particular task in the ATLAS production system (all of the small datasets overlap with this main dataset). Then these main datasets may be added to a container, where the output of several similar simulation tasks is gathered.
2.2. Data management responsibilities

The ATLAS Distributed Data Management system is charged with managing and organising ATLAS data for the collaboration. In particular this means:

- Registering and cataloging ATLAS data
  - Registration of datasets and containers
  - Registration of files into datasets
  - Registration of datasets into containers
- Transferring data between sites
  - Registering the data transfer request
  - Allowing requests to be queried and cancelled
- Delete replicas from sites
- Ensure dataset consistency on sites
  - In particular manage file loses on sites
- Enforce ATLAS Computing Model policies
- Monitor ATLAS data activities

The current implementation of DDM is called DQ2 (*Don Quijote 2*).

2.3. DQ2 architecture

The architecture of DQ2 is based on a service stack – clients are at the top and use a well defined API to interact with the system. Internally, the system core utilises an Oracle database for all state. The access points for the central services are stateless Apache web servers, which then talk to the database, which allows horizontal scaling of this component. To interact with the grid, the *Site Services* components shield the central services from dependencies on any particular grid implementation or technology. However, there are certain basic common libraries which are shared between DQ2 components, as illustrated in the architecture overview (figure 2).

![Figure 2. Schematic illustration of DQ2 architecture.](image-url)
2.4. Scope and users

As ATLAS is a collaboration of several thousand physicists, managing data across hundreds of sites and running hundreds of thousands of jobs a day, DDM has a wide base of clients. The majority of calls to DDM (see section 3) are from other ATLAS systems, e.g., the ATLAS PanDA workload management system[6]. Thus, a few clients which produce the majority of the DDM load. However, DDM also supports a user base of thousands of individuals: about 500 unique users each day, 1000 each week and 1500 every month.

3. Scaling and performance

Since the start of LHC data taking the total amount of data managed by DQ2 has grown steadily, from about 10PB in 2008 to more than 70PB in summer 2011 (figure 3).

![Figure 3. Total size of data managed by DDM from LHC startup. Note the sharp increase in data taking rate once the LHC centre of mass energy reached 7TeV in spring 2010.](image)

![Figure 4. Total number of files managed by DDM from LHC startup. Note the sharp increase in data taking rate once the LHC centre of mass energy reached 7TeV in spring 2010.](image)

Likewise, in figure 4, the evolution of the total number of managed files is shown.

Over the last twelve months, the ATLAS DDM system (DQ2) has seen an overall load increase by about a factor two, as measured by the tracer service (Figure 5).

The figures show that the current implementation is scaling well with data volume. However, this scaling is achieved at the cost of considerable tuning efforts, especially in the area of interactions with Oracle. Indeed, to achieve sufficient performance with Oracle, a de-normalised schema had to be adopted, where columns are replicated using triggers in order to reduce the number of table joins needed for popular queries. The queries also require significant numbers of Oracle hints to ensure that the database executes them in an efficient manner. Currently DQ2 has a load of about 25M reads and 1M writes per day, as shown in Figure 6.

As the amount of data managed by DQ2 has increased, so the transfer rates of data around the grid. In figures 7 and 8 it can be seen that transfer rates average 2GB/s continuously for ATLAS. Peaks of up to 10GB/s have been observed after reprocessing campaigns, when large amounts of data need to be replicated across the grid.
Increased load on the DQ2 system and thus on the database backend. Improvements and tuning done on the database and client code to address the higher load. The DB is coping so far, but always working close to the limits. A machinery for logging and analysis of the DQ2 Apache’s logs has to be put in place. Improved DQ2 deletion service in interactions with the DB. Several heavy objects removed and changes in the client code as well. Optimization in the PanDA DB schema to reduce the polling rate from the JOBSACTIVE4 table. What is pending is creation of view objects that would comprise the tables from the PANDA and PANDAARCH schemes for simplifying the PanDA monitor client code.

4. New services and features
As DQ2 has evolved to become a more complete system for managing ATLAS data throughout its life, a number of new services have been introduced.

4.1. Tracer
One of the key problems facing a large collaboration like ATLAS in an inhomogeneous resource scarce environment is to determine which data should be widely replicated because it is, or it is likely to be, popular and which data should be replicated less.

In order to achieve an objective measure of this the DDM team introduced the tracer system[7, 8]. The tracer has an API library that is used by DQ2 clients to send a callback when data on the grid is accessed. Data pertaining to the dataset, file, site (both local and remote), user, activity and timestamps are included in the tracer message, which is sent via http. These messages are collected via the tracer service, which then inserts them into the DQ2 database.

The tracer service is optimised to accept a high rate of data insertion (more than 5M per day) and recent work has investigated the use of the HBase NoSQL database to buffer insertions.
Figure 7. ATLAS grid data transfer rates for different activities since LHC startup. An increase in data rate is observed once 7TeV data taking began.

Figure 8. Data transfer rates per activity for May 2011. The system has a continuous substantial workload.

before doing a bulk insert of data into Oracle. This helps the system to scale.

4.2. Popularity
The DQ2 tracer keeps a very fine granularity for data access – each single access results in multiple database row entries. This makes it unsuitable for querying directly, as data aggregation would be very slow. For this reason the DQ2 popularity service[7] was introduced. This service analyses the tracer data each day and provides summaries of data access per local or remote site, user, dataset, or other attributes.

The popularity service can be queried by clients via a web interface or an API to discover
Figure 9. Results of a typical query to the popularity service, giving the grid-wide popularity of AOD datasets over a 30 day period, ranked by number of accesses.

4.3. Deletion

While deletion might seem like a fairly straight-forward activity on the surface, in a complex distributed environment, such as that managed by DDM, it is far from trivial. Dataset deletion requests on a particular site need to be done with care to ensure that:

- The dataset replica entry is deleted from the DDM central catalog.
- Corresponding files are physically deleted from storage.
- All file replica locations are removed properly from the local file catalog.

Each of these steps might fail, so it is necessary for the deletion service to have an internal state engine that records the state of deletion for any dataset at a particular site. It is also necessary to throttle both catalog and physical deletion requests for files in order to prevent...
external services from being overwhelmed. In addition, there is also the additional complexity of overlapping datasets. If two datasets share files on a site and only one of them is deleted then the shared files should not be deleted, otherwise the remaining dataset would become incomplete. This requires some care in mapping dataset deletion requests onto file deletions.

Deletion rates in ATLAS can reach significant levels, with millions of file deletions per day and terabytes of data being cleaned. Figure 10 shows a typical example.

Figure 10. Deletion across the ATLAS clouds over a 24 hour period. Deletion rates of more than 300k files per hour are frequent.

Analysis of this level of deletion shows that it primarily arises from the use of intermediate files in processing chains, e.g., simulation can only produce small detector hits files as jobs are limited by cpu/runtime considerations to only process about 50 events; these small hits files are then merged to larger files and afterwards the original hits are deleted. A considerable load also comes from the deletion of user outputs from transient disk areas.

4.4. Consistency

Grid operations across the large number of sites used by an experiment such as ATLAS are far from trivial (see, e.g., [9]). Data loss at sites occurs frequently and recovery from this state must be an automated process to be efficient. In addition consistency problems can occur between the different catalog layers in the system, which need to be rectified.

The consistency service[10] has been developed in order to achieve this goal. This service operates by allowing storage URLs which have been lost to the site to be declared via a dedicated API. The consistency service will then set about re-establishing the consistency of the data on the site by:

- Declaring affected datasets on a site to be incomplete, so that they will not be used for data processing
- Cleaning the storage and file catalog namespaces of lost files
- Re-subscribing files which have other replicas elsewhere back to the site
- Deleting files which have no other replicas from the definition of the affected datasets (thus permanently lost)
- Declaring datasets to be complete again at a site after consistency is re-established

The consistency service also keeps records of which files have been lost and which recovery actions have been taken. This allows a historical view of data loss on the grid and for sites with particular problems to be identified.
4.5. Accounting
Knowledge of how storage space is used on the grid is vital for the effective management of computing operations. This problem is complicated by the number of different dimensions along which ownership can be measured. e.g., data type (RAW, ESD, AOD) is orthogonal to project (data11, T7TeV, mc10), which are both orthogonal to accounting by the data owner.

For this reason the accounting system, which was previously based on pattern matching, has been replaced with a system based on key-value pairs. Thus a wildcard query which previously might have been ‘data10.*.ESD.*’ + ‘CERN’ can be expressed as {‘project’: ‘data10’, ‘type’: ‘ESD’, ‘location’: ‘CERN’}. The new system allows arbitrary combinations of key-value pairs to be specified, which offers considerably more flexibility than the old pattern based approach. Once a set of key-values has been established the system will query these periodically. In this way historical data will be built up and trends can be analysed.

An initial implementation is available based on an Oracle backend, but an assessment of NoSQL backends, which are inherently more suitable to such key-value stores, is completed with the Hadoop map-reduce framework.

5. Rucio: the new version of ATLAS Distributed Data Management system
It turned out difficult to extend the current DDM system (DQ2) for satisfying new high-level use cases and for integrating new technologies and middleware. All these necessary, evolutionary changes partly were incompatible with the original DQ2 vision, partly the old design itself showed limitations and scaling problems, e.g. due to unused concepts like versioning. Although the current implementation works, the operational burden is heavy. We therefore started to implement the next generation of the ATLAS DDM system, called Rucio.

In this section, we describe the core concepts Rucio uses to manage accounts, files, datasets and storage systems. Figure 11 shows the logical Rucio architecture.
5.1. Rucio account
A Rucio account is the unit of assigning privileges in Rucio. It can represent individual users, a group of users or an organised production activity for the whole ATLAS collaboration.

Rucio actions are always conducted by a Rucio account. Each account has a namespace identifier called scope that is included in every name assigned to a collection of data created by that account (see section 5.3). By default, Rucio accounts can only create identifiers in their own scope and not in any other.

Rucio users are identified by their credentials, like X509 certificate, username/password, or token. Credentials can map to one or more accounts (N:M mapping). The Rucio authentication system checks if the used credentials are authorised to use the supplied Rucio account.

Rucio assigns permissions to accounts. Permissions are boolean flags designating whether an account may perform a certain action (read, write, delete) on a resource (dataset, account, replica, etc.).

5.2. Datasets and files
a dataset might be a single file or multiple files. Datasets may be overlapping in the sense that a subset of data, i.e., a single file or more files, can be part of multiple datasets.

New datasets can be defined based on the contents of existing datasets. In particular, it is possible to aggregate the contents of two or more datasets into a new one by taking the set union of their respective contents. While successive aggregations will implicitly create an aggregation hierarchy, this is not reflected in the naming of the datasets. Instead Rucio will merely record in the dataset metadata that it was created by performing such a union. Otherwise the dataset will be identical to one created by simple enumeration of the resulting contents, i.e., the corresponding files.

The most common aggregation scheme in ATLAS is hierarchical, e.g., run datasets aggregated into sub-periods, sub-periods into periods, periods into a year of data. However, Rucio also will support horizontal aggregation of files between any datasets.

5.3. Dataset/file identifiers and scope
To be able to unambiguously refer to a logical file it needs to have an identifier. For a logical file this is the Logical File Name (LFN) which is composed of two strings: the scope identifier and the file name. A single file is a dataset in itself and as such dataset identifiers follow the same scheme: Each dataset is identified by the Dataset Name (DSN) which is composed of the scope identifier and the dataset label.

The scope identifier partitions the dataset name space into several sub-spaces. The primary use case for this is to have separate scopes for production and individual users. There is a one to one relationship between account and scope. Datasets/files are uniquely identified over all time. A DSN/LFN once used to refer to a dataset/file can never be reused to refer to another dataset/file, not even if the former has become obsolete or has been deleted from the system.

The following status flags are supported for files:

- **Obsolete**: The obsolete flag indicates a file which, for one reason or another, should not be used by the collaboration. A file marked as obsolete will have all replicas removed from the system. Obsolete LFNs are remembered by the system to prevent accidental reuse in the future.

- **Lost**: The lost flag indicates that although the logical definition of the file still exists in Rucio, no physical replicas of the file currently exist. If the file is recovered from outside Rucio this flag can be changed from True to False.

The dataset status is reflected by a set of attributes. Datasets in Rucio can have the following attributes:
Open: A dataset might be a result of more than one computational process, therefore the definition of a dataset is not an atomic operation and can even extend over a large amount of time. For this purpose, a dataset has an open status to publish its availability, i.e., to reflect that its content is (not) complete. Open datasets cannot be used in aggregations. When the filling of the dataset is done, its state changes to closed and cannot thereafter be reopened.

Monotonic: If the monotonic mode is enabled files cannot be removed from an open dataset. Once the monotonic flag is set to True it cannot be unset.

Hidden: Datasets can be hidden so they do not show up in normal listing operations.

Obsolete: The obsolete status means that a dataset and its definition should not be used anymore. Obsoleting a dataset does not obsolete its contents (however, one can instruct Rucio to obsolete all files which are in a particular dataset).

Complete: File loss is reflected in the lost status of the file and by the complete/incomplete status of all aggregate datasets containing them. The file content can be recovered and re-injected in the system causing the corresponding lost and complete/incomplete statuses to be updated. Note that Rucio has no concept of dataset versioning. The loss of files is simply recorded as described above with a single flag, hence not recording in what order they were lost. Adding further files requires the definition of a new dataset with a new identifier. The latter dataset might reflect the relation with the former, but this is not required.

5.4. Metadata attributes
Metadata associated with a dataset/file is represented using key/value pairs. The set of available attributes is restricted. Metadata attributes are classified into four categories:

(i) System-defined attributes: e.g., size, checksum, creationtime, modificationtime, status
(ii) Physics attributes: e.g., number of events
(iii) Production attributes: e.g., task and job ID that produced the file, processing campaign ID
(iv) Data management attributes: necessary for the organisation of data on the grid (see §6)

For datasets, it is possible that the value of a metadata attribute is a function of the metadata of its constituents, e.g., the total size is the sum of the sizes of the constituents. In this case it is obviously not possible to assign a value to it.

When appropriate and requested, Rucio will check metadata values for validity, rejecting the attempt to set invalid values. This can be used to ensure that, e.g., the Job ID is a positive integer.

Rucio supports searching for files and datasets based on metadata values. Wildcard queries or range operations will only be supported on certain metadata fields. e.g., processing tag can be searched using wildcards, task ID can be searched using range operators, but scope can only be an exact match.

5.5. Rucio Storage Element
A Rucio Storage Element (RSE) is a container for physical files. It is the smallest unit of storage space addressable within Rucio. It has an unique identifier and a set of meta attributes describing properties such as supported protocols, e.g., file, https, srm; host/port address; quality of service; storage type, e.g., disk, tape, ...; physical space properties, e.g., used, available, non-pledged; and geographical zone.
Rucio Storage Elements can be grouped in many logical ways, e.g., the UK RSEs, the Tier-1 RSEs, or the ‘good’ RSEs. One can reference groups of RSEs by metadata attributes or by explicit enumeration of RSEs.

RSE tags are expanded at transfer time to enumerate target sites. Post-facto changes to the sites in an RSE tag list will not affect currently replicated files.

The Physical File Name (PFN) is a fully qualified name identifying a replica of a file. PFNs may take the form of file names, URIs, or any other identifier meaningful to a Rucio Storage Element. The mapping between the LFN and the PFN is a deterministic function which also takes RSE and protocol into account. It may also consider the filename, allowing a separation of PFN based on scope, project or other criteria.

6. Rucio replica management model
Replica management is based on replication rules defined on datasets. A replication rule is owned by an account and defines the minimum number of replicas to be available on a list of RSEs. Accounts are allowed to set multiple rules. The system may reject rules if these violate other policies, e.g., a normal ATLAS user would not be allowed to set a rule which committed the system to generate 5PB of new replicas or to request replicas on an RSE tape system. Rules may optionally have a limited lifetime and can be added, removed or modified at any time. Rules can also be locked, which will prevent their modification until an explicit unlock operation is performed.

An example listing of replication rules is given below:

(i) prod: 1x replica @ CERN, no lifetime
(ii) prod: 1x replica @ T1TAPE, no lifetime, locked
(iii) higgs: 2x replica @ GROUPDISK, no lifetime
(iv) barisits: 1x replica @ UST2, until 2012-01-01 00:00
(v) vgaronne: 2x replica @ T1, no lifetime
(vi) graemes: 1x replica @ GLASGOW, until 2011-12-25 00:00

The Rucio rule engine validates the rules and creates transfer primitives to fulfil all rules, e.g., transfer some files from RSE A to RSE B. The rule engine is triggered when a file is created in the system, when a new rule is added to a dataset or when one explicitly requests for the rule to be applied on existing data. The rule engine will only create the minimum set of necessary transfer primitives to satisfy all rules.

Files inherit all rules from all datasets they are members of. This allows users of the system to interact at the dataset level and Rucio will propagate these rules to the appropriate files.

6.1. Subscription Policies
Subscription Policies are system entities which generate rules or transfer requests based on matching particular dataset metadata at registration time. Policies are owned by an account and can only generate rules for that account. Policies may have a lifetime, after which they will expire.

An example of a policy is given below:

| Attribute | Value |
|-----------|-------|
| Owner     | tzero |
| match     | project=data11_7TeV, dataType=RAW, stream=physics_* |
| rule      | 1@CERNTAPE, 1@T1TAPE |
| lifetime  | 2012-01-01 00:00 |
Policies can also create transfer primitives, so generate extra copies of data as it is produced:

| Attribute | Value |
|-----------|-------|
| Owner     | prod  |
| match     | project=mc11_7TeV, dataType=merge.AOD, tag=*(p795|p796|p805)*, ReplicationPolicy=RPValue |
| rule      | 1@T1DISK, 1@T2DISK |
| transfer  | 1@T1DISK, 2@T2DISK |
| lifetime  | 2011-12-01 00:00 |

In this case the transfer request is for extra copies, in addition to those set by rules. (This is different behaviour to that for rules themselves, which are always independent.)

6.2. Data deletion
Deletion is triggered per RSE when storage policy dictates that space must be freed. A reaper service will look for replicas on that RSE that can be deleted without violating any replication rules. The reaper will use a Least Recently Used (LRU) algorithm to select replicas for deletion. The reaper service will also immediately delete all replicas of any file which is declared obsolete.

Moreover, accounts can inject transfer primitives directly, e.g., transient replicas required for production operations. Notifications can be provided for the transfer request. All transfer requests are transient.

The injection of transfer primitives requires a specific privileges to prevent abuse of this feature.

6.3. Accounting and quota
Accounting is the measure of how much resource, e.g., storage, an account has used as a consequence of its actions. Quota is a policy limit which the system applies to an account. Quotas will be available which apply to an account at a specific RSE tag or globally.

For storage accounting, Rucio accounts will only be accounted for the files they set replication rules on. Accounting is based on the replicas an account requested, not on the actual amount of physical replicas in the system. Consequently, it is expected that over-booking of physical space will be employed as a particular physical replica may satisfy several replication rules.

7. Conclusions and future directions
The ATLAS Distributed Data Management project has provided ATLAS with a well functioning system in which to organise and manage data during LHC running. New services have been introduced to help automate the data lifecycle and to improve the robustness of the system to real operating conditions. Introducing and managing these new features, while keeping the system stable have taken considerable and continual efforts of developers and database experts, but the multi-petabytes of data moved and stored shows that the system has scaled well to date.

However, it is also true that some conceptual and design limitations have arisen in the current system, which now hamper future development. For example, dataset versions (section 2.1), while supported, are largely unused and have not proved to be really necessary; however, the inclusion of versioning in the current schema introduces an indirection between a dataset and its constituent files that requires a table join with a consequent performance impact in the database layer. Consequently the ATLAS DDM group are undertaking a review and re-implementation of DQ2, looking forward to the long LHC shutdown of 2013-2014. A new version of DDM, Rucio, is planned for 2013 to take ATLAS forward into the next years of high luminosity LHC running.
References

[1] The ATLAS Collaboration 2008 *Journal of Instrumentation* **3** S08003

[2] Elsing M, Goossens L, Nairz A and Negri G 2010 *Journal of Physics: Conference Series* **219** 072011

[3] Branco M, Cameron D, Gaidioz B, Garonne V, Koblitz B, Lassnig M, Rocha R, Salgado P and Wenaus T 2008 *Journal of Physics: Conference Series* **119** 062017

[4] Knobloch J et al 2005 *LHC Computing Grid Technical Design Report* Tech. Rep. CERN-LHCC-2005-024 CERN

[5] Jones R and Barberis D 2008 *Journal of Physics: Conference Series* **119** 072020

[6] Maeno T, De K, Wenaus T, Nilsson P, Stewart G A, Walker R, Stradling A, Caballero J, Potekhin M, Smith D and Collaboration T A 2011 *Journal of Physics: Conference Series* **331** 072024

[7] Molfetas A, Megino F B, Tykhonov A, Lassnig M, Garonne V, Barisits M, Campagna S, Dimitrov G, Jezequel S, Ueda I and Viegas F T A 2011 *Journal of Physics: Conference Series* **331** 062018

[8] Lassnig M, Garonne V, Branco M and Molfetas A 2010 *Journal of Physics: Conference Series* **219** 062054

[9] I Ueda and the ATLAS collaboration 2011 *Journal of Physics: Conference Series* **331** 072034

[10] Serfon C, Garonne V and The ATLAS Collaboration 2011 *Journal of Physics: Conference Series* **331** 072065