Journal of Physics: Conference Series **513** (2014) 042021 doi:10.1088/1742-6596/513/4/042021

Rucio - The next generation of large scale distributed system for ATLAS Data Management

V. Garonne[1], R. Vigne[1], G. Stewart[1], M. Barisits[1], T. Beermann[1], M. Lassnig[1], C. Serfon[1], L. Goossens[1] and A. Nairz[1] on behalf of the ATLAS Collaboration

[1] CERN, Geneva, Switzerland

E-mail: vincent.garonne@cern.ch

Abstract. Rucio is the next-generation Distributed Data Management (DDM) system benefiting from recent advances in cloud and "Big Data" computing to address HEP experiments scaling requirements. Rucio is an evolution of the ATLAS DDM system Don Quijote 2 (DQ2), which has demonstrated very large scale data management capabilities with more than 140 petabytes spread worldwide across 130 sites, and accesses from 1,000 active users. However, DQ2 is reaching its limits in terms of scalability, requiring a large number of support staff to operate and being hard to extend with new technologies. Rucio will deal with these issues by relying on a conceptual data model and new technology to ensure system scalability, address new user requirements and employ new automation framework to reduce operational overheads. We present the key concepts of Rucio, including its data organization/representation and a model of how to manage central group and user activities. The Rucio design, and the technology it employs, is described, specifically looking at its RESTful architecture and the various software components it uses. We show also the performance of the system.

1. Introduction

The ATLAS experiment [1] at the LHC is a general purpose particle physics detector designed to investigate physics at the energy frontier. The Distributed Data Management (DDM) project manages ATLAS data on the grid and provides functionalities for data placement, deletion and data access. The current implementation is called Don Quijote 2 (DQ2) [2] and has demonstrated very large scale data management capabilities with more than 150 Petabytes spread worldwide across 130 sites, and accesses by 1,000 active users. However, DQ2 is reaching its limits in terms of scalability, requiring a large number of support staff to operate and being hard to extend with new technologies. The Rucio project will deliver the new version of DDM system services which will address these issues, support new use cases and cope with the requirements of LHC Run2 in terms of scalability and the expected huge amount of data.

In this paper we give an overview of the ATLAS Rucio system, covering core concepts (Section 2) and its architecture and technologies (Section 3). We will then discuss the scalability requirements of the DDM system and show the performance results obtained during scale test exercises (Section 4). We will then conclude and give our future plans (Section 5).

2. Concepts

In this section, we describe the core concepts Rucio uses to manage accounts, files, datasets and storage systems.

2.1. Accounts

The entry point of the system is a Rucio account which can represent individual users, a group of users or a service like the organised production activity for the whole ATLAS collaboration. Actions in Rucio are always conducted by a Rucio account. A Rucio user is identified by their credentials, such as X509 certificate or Kerberos token. Credentials can map to one or more accounts. Rucio checks if the credentials are authorized to use the supplied Rucio account.

2.2. Data identifiers, scope and metadata attributes

For Rucio, files are the smallest operational units of data. Physicists need to be able to identify and operate on any arbitrary set of files. Files can be grouped into datasets (a named set of files) and datasets can be grouped into containers (a named set of datasets or, recursively, containers).

Files, datasets and containers follow an identical naming scheme which is composed of the scope and a name, called a data identifier. The scope string partitions the namespace in several sub spaces. The primary use case for this is to have separate scopes for production and individual users. The figure below gives an example of an aggregation hierarchy.

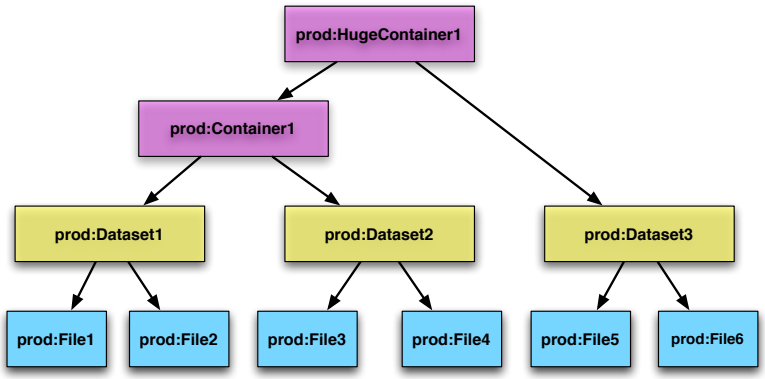

Figure 1: Aggregation hierarchy example.

The data identifier status is reflected by a set of attributes. For instance, files have a status related to their availability, and datasets/containers have the "open" status which means that content can be added to them.

Metadata associated with a data identifier is represented using attributes, which are key-value pairs. The set of available attributes is restricted. Some metadata attributes are user settable, e.g. physics attributes (number of events, run number, run period) or production attributes (job identifier). Metadata which is not user settable includes system attributes, such as size, checksum, creation time. For datasets and containers, 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.

2.3. Rucio Storage Element

A Rucio Storage Element (RSE) is a repository for physical files. It is the smallest unit of storage space addressable within Rucio. It has an unique identifier and a set of properties such as:

- supported protocols (e.g. file, https, srm)
- quality of service; storage type (e.g. disk, tape)
- physical space properties (e.g. used, available, non-pledged)
- a weight value, used for data distribution (see §2.4)
- a threshold for deletion (see §2.4)

A set of RSEs can be identified directly by enumeration of their names, or indirectly by a boolean expression over their attributes. Attributes are key-value pairs, e.g. CLOUD=UK, Tier=1, T2D=True, MoUShare=15. RSE attributes are used to manage data with replication rules (see §2.4).

Physical files stored on RSEs are identified by their Physical File Name (PFN). The PFN is a fully qualified path identifying a replica of a file. PFNs may take the form of file names, URIs, or any other identifier meaningful to a RSE. The mapping between the file identifier and the PFN is a deterministic function of the identifier, RSE and protocol.

2.4. Replica management: Replication rule, subscription and data deletion

Replica management in Rucio [3] is based on replication rules defined on data identifier sets. A replication rule is owned by an account and defines the minimum number of replicas to be available on a list of RSEs. Rules may specify a grouping policy that controls which files are grouped together at the same RSEs. The list of RSEs can be specified with RSE attributes (see § 2.3).

Rucio processes each replication rule and creates replica locks that satisfy the rule. A replica lock is defined for a replica on a specific RSE and is owned by the issuer account. A replication rule can trigger a data transfer prior to the replica lock being satisfied. As transferring a file to a site can fail, a replication rule has a state to reflect its status. In case a replica on a particular RSE has no associated replica locks anymore, it can be deleted. Locks disappear when accounts decrease the number of desired replicas in their replica rules, or the whole rule altogether.

For storage accounting, Rucio accounts will only be charged for the files on which they have set replication rules. The accounting is based on the number of replicas an account requested, not on the number of physical replicas in the system. Accounting and quota calculations use the replica locks generated from replication rules. Quotas are constraints on the replica locks, based on limits set per account.

If the file content of a dataset is widely distributed across many sites, this will maximize the CPU resources that can be used to process this data. If the files are concentrated at one site, this will be optimal for processing that requires the use of many files at once (e.g. merging and archiving to tape). Rucio uses the dataset/container hierarchy to control how files are grouped together at sites and offers different levels of grouping, which will affect how the file replicas in a hierarchy are grouped. Clients should ensure that their dataset/container hierarchies, in particular the definition of the datasets, are created in such a way as to provide suitable file groupings for further data processing.

Subscription is another mechanism in Rucio to control data replication. Subscriptions are used by accounts to generate replication rules based on matching particular metadata at registration time. A subscription will therefore create replication rules for every new file matching the criteria.

3. Architecture

The Rucio system is based on a distributed architecture and can be decomposed into four main components: Server, Daemons, Resources, and User Interface (see in Figure 2.

The Rucio clients layer offers a command line client for users as well as application programmer interfaces in Python which can be directly integrated into user programs. All

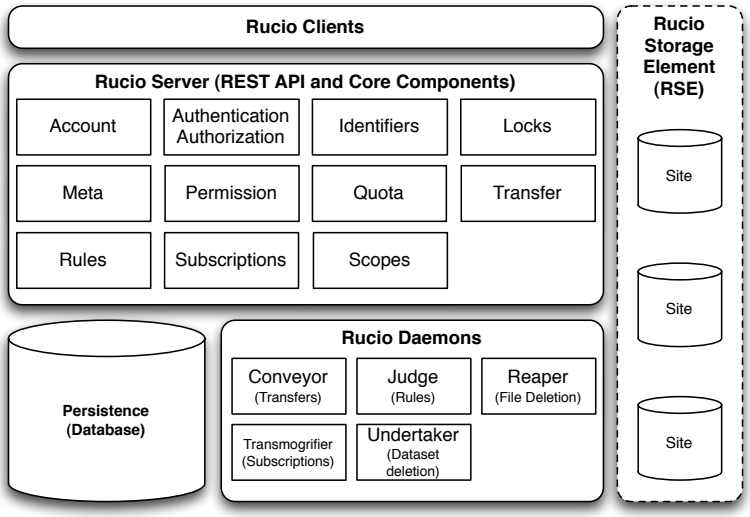

Figure 2: Overview of the Rucio architecture.

interactions are transformed by the client into https requests which are sent to the REST [4] interface of the Rucio server. Consequently, external programs can also choose to directly interact with the REST API of the server.

The Rucio server is a passive component listening to incoming queries. It connects several Rucio core components together and offers a common REST interface for external interaction. The users with their credentials interact first with the Authentication&Authorization component to be mapped to an account. This protocol follows the Oauth model [5]. The authentication component checks the used credentials and, if they are valid, it returns a short lifetime token. Once authenticated, the users provide these tokens for the next interactions with the system. The permissions of the account to execute the given request are then checked by the Authorization component.

The RSE abstraction layer is responsible for the interactions with different grid middleware tools and storage systems. It effectively hides the complexity of these tools and combines them into one interface used by Rucio.

The Rucio daemons are active components that orchestrate the collaborative work of the whole system. The following daemons are part of the Rucio system:

Conveyor is in charge of file transfers.

Reaper deals with file replica deletion.

Undertaker is responsible of obsoleting data identifiers with expired lifetime.

Transmogrifier applies subscriptions on newly created/existing data to generate replication rules.

Judge is the replication rule engine (see §2.4).

The persistence layer keeps all the logical data and the application states. Rucio uses SQLAlchemy [6] as an object relational mapper and can therefore support several relational database management systems (RDBMS) like Oracle, MySQL or PostgreSQL.

4. Scaling and performance

4.1. Requirements and emulation

To evaluate the scalability of the system, the Rucio emulation framework [7] has been developed to continuously operate performance tests. To emulate Rucio at very high frequencies (multiple kHz), this framework was designed to be very scalable and modular. For instance the emulation dispatches correlated use cases, derived from DQ2, in real-time to workers through a distributed task queue. This technique guarantees the horizontal scalability of the emulation framework.

To define the workload, DQ2 has been instrumented to provide more logging information about the system usage like the method called or the application identifier. As DQ2 can produce up to 100GB of ASCII log data per day, the powerful data mining framework, Hadoop [8], has been used to map-reduce the log data corresponding to long periods.

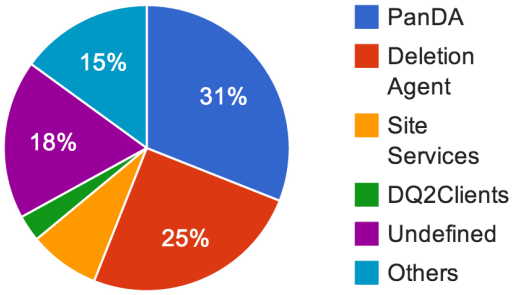

Figure 3: Workload per application.

Operations	Rates
Transfer	20 Hz
Deletion	25 Hz
Upload	0.04Hz
Download	7Hz

Table 1: Requirements at nominal load.

Figure 3 illustrates how the observed workload is distributed over different applications. PanDA [9] is the workload management system of the ATLAS experiment and is accountable for 31% of the load. PanDA is in charge of executing production and analysis jobs on the grid. It interacts with DDM for the management of the input and output data for job scheduling. Site-services are the DQ2 components responsible for file transfers. The deletion agents are dealing with file deletion. The undefined part corresponds to old clients which have not been updated and did not send the relevant information. We observed about 7% noise and assumed that this is related to retries of failed calls.

With the API calls executed by the various applications and users, we have been able to summarize, identify and map the load to distinct use cases, e.g. data discovery for job scheduling. This results in information like how often an use case is executed, by which application and for which purposes. With this knowledge, we deduced target numbers for each use case in order to validate the workload generated by the emulator. We identified various performance indicators to verify against defined measures and metrics. Table 1 summarizes the requirements in term of file transfer, deletion, download and upload rates for the nominal load. These numbers are derived from the following use cases:

Data export from T0. Detector data is stored at Tier 0 (CERN) and then exported to Tier-1 data centers.

Data distribution of production data. Production data is replicated over the grid.

PanDA user analysis. It represents the user analysis activity. The load is based on the peak period of February 2013. During this period, we measured in average 400,000 user jobs per day.

PanDA production. Two different production workflows have been considered: production at Tier-1 sites without the movement of input/output data and production at Tier-2 sites including data movement. The reference peak period is April 2013. We observed an equal

repartition in term of number of jobs and data created for the two workflows. The total amount of jobs per day was 480,000.

Data download. It is mostly downloads done by end-users to retrieve job outputs.

Wildcard/Metadata searches. It represents the searches to discover data.

4.2. Results

The following figures show the results obtained with the emulation framework running the use cases previously described . For this exercise, we ran the system at the nominal load and then increased the load by a factor two every few hours (at least 24 hours). The deployment model for the Rucio server was one node for the server (Intel Xeon CPU L5640, 24*2.27GHz, 48 G), another node for the daemons (Intel Xeon CPU L5520, 8*2.27GHz, 24 G) and a dedicated database Oracle cluster. The database has been populated according to the distribution of data extracted from DQ2. The data volume reflects the same order of magnitude observed in production and exposes the same characteristics to provide realistic experimentations. Table 2 gives a summary of the data volume pre-filled in the database before the tests. The information about core's internal workings is exposed in the highly scalable and real-time graphing system Graphite [10]. In addition to the storage backend, Graphite provides a web-based visualization frontend that gives immediate feedback about where Rucio spends most of its time.

Entities	Volume
Data identifiers	0.5 billion
Content	0.6 billion
Subscriptions	100
Replication rules	25 millions
Locks	1.7 billion
File replicas	1 billion

Table 2: Volume of data pre-filled in the database before the tests.

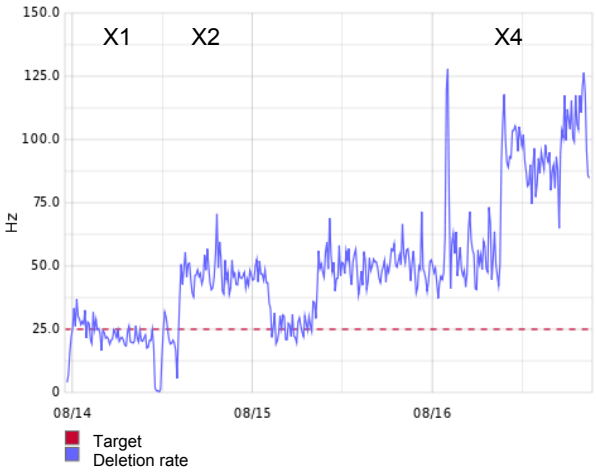

Figure 4: Deletion rate of file replicas with an increase of the load.

Rucio scales at nominal load and has operated smoothly with the desired level of transfers and deletion. Figure 4 shows the evolution of the deletion rate with an increase of the load. The system has been able to sustain the target value rate for deletion (25 Hz) during hours and gives some good response with higher load. We registered a peak deletion rate at 125 Hz without the creation of any backlogs. To focus on the Rucio performance, the interactions with the storage have been emulated by using a mock implementation of the RSE component (see § 2.3). Figure 5 shows the draining of the transfer backlog. The conveyor daemon has been interfaced with the WLCG (Worldwide LHC Computing Grid) File Transfer Service 3 (FTS3). It has been able to submit transfers to FTS3 at a rate of 240 Hz. The conveyor polled the status of each individual file transfer to know when they are complete (green curve on the plot). In the future, this status check will be based on notifications: this is more efficient and scalable. Figure 6 represents the metadata searches done by end-users. We can see that independently from the number of results returned, the performance stayed stable (on average 0.15 ms per result).

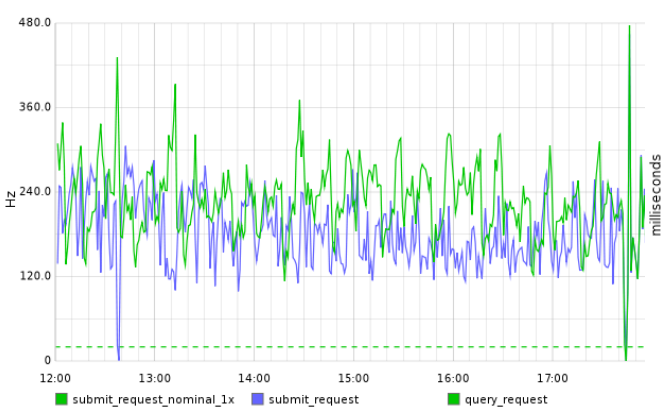

Figure 5: Transfer rate when draining a backlog.

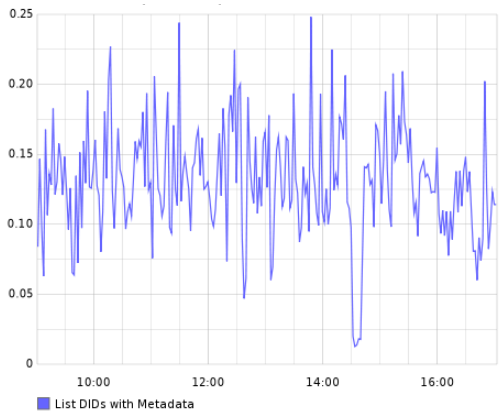

Figure 6: Evolution of the response time for metadata searches (normalized by returned results).

The Rucio load on the database is I/O bound. At nominal load, we measured a throughput of 20 MB per second with a total number of 900 I/O operations per second. With four times the nominal load, we observed after few hours an increase of the database latency, some backlogs of requests and degradation of the response times. The CPU used on the database stayed relatively constant while the user and commit I/O went up to 40 MB per second conjointly with the load increase. It was hard to understand the behavior in detail as we learned that at the same time the database got hardware problems with impact on the commit latency. But the Rucio nodes behaved well and we did not see any issues on neither the server nor the daemon node.

5. Conclusion

In this paper, we have shown that we have a complete framework for continuous scale exercises and that Rucio with its concepts and technologies scales well at nominal load. The nominal load defined is based on peak periods in 2013. Our tests have shown the critical importance of the database and our plans are to continue the stress-test of the system with the new database production hardware scheduled for end of 2013. We are now also involved on how Rucio will be integrated by external ATLAS applications like the PanDA system. For this purpose, we have provided an integration testbed to gradually commission Rucio functionalities and the integration with external applications and services like the new WLCG File Transfer Service. The new version of DDM, Rucio, is scheduled for beginning of 2014 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] V Garonne et al 2012 The ATLAS Distributed Data Management project: Past and Future *Journal of Physics: Conference Series*
[3] M Barisits et al 2013 Replica management in Rucio: Replication rules and subscriptions *Journal of Physics: Conference Series*
[4] Fielding R T and Taylor R N 2002 *ACM Transactions on Internet Technology (TOIT)* **2** 115–150
[5] Leiba B 2012 Oauth web authorization protocol vol 16 (Los Alamitos, CA, USA: IEEE Computer Society)
[6] Copeland R 2008 *Essential sqlalchemy* (O'Reilly)
[7] R Vigne et al 2013 DDM workload emulation *Journal of Physics: Conference Series*
[8] White T 2009 *Hadoop: The Definitive Guide* (O'Reilly Media, Inc.)
[9] T Wenaus et al 2012 PanDA: Next generation workload management and analysis system for big data *SC Companion* (IEEE Computer Society) pp 1521–1522
[10] Graphite - Scalable Realtime Graphing 2013 http://graphite.wikidot.com/