Automatic rebalancing of data in ATLAS distributed data management

M Barisits\textsuperscript{1,2}, C Serfon\textsuperscript{3}, V Garonne\textsuperscript{3}, M Lassnig\textsuperscript{1}, T Beermann\textsuperscript{1}, T Javurek\textsuperscript{4} on behalf of the ATLAS Collaboration

\textsuperscript{1} CERN, Geneva, Switzerland
\textsuperscript{2} Vienna University of Technology, Vienna, Austria
\textsuperscript{3} University of Oslo, Oslo, Norway
\textsuperscript{4} Albert-Ludwigs-Universität at Freiburg

E-mail: martin.barisits@cern.ch

Abstract. The ATLAS Distributed Data Management system stores more than 220PB of physics data across more than 130 sites globally. Rucio, the next generation data management system of the ATLAS collaboration, has now been successfully operated for two years. However, with the increasing workload and utilization, more automated and advanced methods of managing the data are needed. In this article we present an extension to the data management system, which is in charge of detecting and foreseeing storage elements reaching and surpassing their capacity limit. The system automatically and dynamically rebalances the data to other storage elements, while respecting and guaranteeing data distribution policies and ensuring the availability of the data. This concept not only lowers the operational burden, as these cumbersome procedures had previously to be done manually, but it also enables the system to use its distributed resources more efficiently, which not only affects the data management system itself, but in consequence also the workload management and production systems. This contribution describes the concept and architecture behind those components and shows the benefits made by the system.

1. Introduction

Rucio, the ATLAS [1] collaboration’s distributed data management (DDM) system, manages more than 220 petabytes of data on more than 750 storage endpoints in the Worldwide LHC Computing Grid[2]. Rucio replaced DQ2[3] in December 2014 and was operated very successfully since then.

Rucio organizes not only the RAW data from the detector, but also handles all data inputs and outputs from the collaboration’s users. The distributed data management system also enforces the collaboration’s data placement policies and takes care of a fair and balanced usage of the resources. However, due to the heavy usage of resources, and due to imbalances between computing and storage resources, storage endpoints get to their capacity limits regularly. This usually requires strenuous, manual, data rebalancing interventions by the DDM operations team.

In this article we describe a system which takes care of these interventions automatically. We also present rebalancing mechanisms which are responsible to even prevent these imbalances from happening in the first place.

The paper is organized as follows: In Section 2 we present key concepts of Rucio which are essential to follow this article. In Section 3 we discuss storage usage issues in general and the
application of data rebalancing in ATLAS distributed data management. In Section 4 we explain the architecture and workflow of the data rebalancing system. Finally we conclude the article in Section 5.

2. Rucio key concepts and architecture

This section presents the key concepts of Rucio which are essential to follow this article. For further details please refer to [4, 5, 6].

2.1. Concepts

In Rucio, every user, group or organised production activity is represented by an account. Accounts are also the unit of assigning permissions. Every account has a data namespace identifier called scope. The scope is used to partition the data namespace, to easily separate production data from individual user data. In general, accounts can only write to their own scope, but privileged accounts (like production accounts) can be authorized to write into foreign scopes. Credentials, such as username/password, X509 certificates or Kerberos tokens are used to authenticate with Rucio. Such credentials can map to multiple accounts, for example, when a user is authorized to do operations on behalf of a group account.

Managing data is the primary function of any data management system. The ATLAS Collaboration creates and administers large amounts of data which are physically stored in files. For Rucio, these files are the smallest operational unit of data. Files, however, can be grouped into datasets and moreover, datasets can be grouped into containers. We consequently refer to files, datasets or containers as data identifiers (DID), as all three of them refer to some set of data. A data identifier is a tuple consisting of a scope and a name. In Rucio each (scope, name) tuple is unique. Datasets as well as containers may be overlapping in the sense that their constituents may be part of other datasets or containers.

To address and utilize storage systems in Rucio, the logical concept of the Rucio Storage Element (RSE) is used. An RSE is a container of physical files (replicas) and is the unit of storage space within Rucio. Each RSE has a unique name and a set of attributes describing properties such as protocols, hostnames, ports, quality of service, storage type, used and available space, etc. Additionally, RSEs can be assigned with meta-attributes to group them in many logical ways, e.g. all Tier-2 RSEs in the UK, all Tier-1 RSEs, etc.

To select a set of RSEs, RSE Expressions were introduced. RSE Expressions are strings based on the RSE Expression language defined in [4]. The expressions are interpreted by Rucio and result in a set of RSEs. For example, to specify an expression considering all German and French Tier-2 sites, the suitable expression rule would be "tier=2&(country=FR|country=DE)", which is equivalent to the set of all Tier-2s intersected with the set of all French and German RSEs.

Replica management in Rucio is based on replication rules. The general idea of replication rules is that instead of defining a specific destination for data to be replicated to, the user expresses the intention behind the replication request. Consequently, the system is able to interpret those requests and choose the appropriate destinations while preserving system resources, like storage space and network bandwidth. Replication rules can explicitly address a specific RSE, or be more generic such that they result in a list of RSEs, e.g. a user wants to replicate a dataset to two Tier-2 RSEs in the United Kingdom. The user can create a replication rule with 2 copies and the RSE expression 'tier=2&country=uk'. The string 'tier=2' represents the set of all Tier-2 RSEs and the string 'country=uk' the set of all RSEs in the United Kingdom. The set-union operator '&' is used to create the set-union of both sets. Thereupon Rucio picks two ideal destinations based on existing and queued replicas.

A replication rule can be created for any data identifier in Rucio, independently of the scope or creator of the data identifier. When specified on a dataset or container, the rule will affect
all contained datasets or containers. Subsequent changes to these datasets or containers will be considered by the replication rule.

Internally, Rucio processes replication rules and creates a replica lock for each replica created or covered by the rule. Replicas with at least one replica lock are exempt from the deletion procedure. Once a replication rule is removed, the associated replica locks are removed as well. Replicas without any replica lock are flagged to be picked up by the deletion service.

2.2. Architecture
The Rucio software stack is separated into three horizontal layers and one orthogonal vertical layer. It is implemented in Python 2.6[7].

The Rucio clients layer offers a command line client for users as well as application programming interfaces which can be directly integrated into user programs. All Rucio interactions are transformed by the client into https requests which are sent to the REST[8] interface of the Rucio server. Consequently, external programs can also choose to directly interact with the REST API of the server (e.g. by libcurl).

The Rucio server layer connects several Rucio core components together and offers a common, https-based, REST API for external interaction. After a request is received by the REST layer, the authorization component checks the used credentials. If permitted, the permissions of the account to execute the given request are checked by the permission component. If allowed, the request is passed to the responsible core component for execution. Rucio core components are allowed to communicate with each other, as well as with the Rucio storage element abstraction.

The Rucio Storage Element (RSE) abstraction layer is responsible for all interactions with different Grid middleware tools which interact with the Grid storage systems. It effectively hides the complexity of these tools and combines them into one interface used by Rucio. The abstraction layer is used by the clients, the server as well as the Rucio daemons.

The Rucio daemons are used to asynchronously operate on requests made by users or by the Rucio core. These can be transfer requests, executed by the Conveyor, expired replicas or datasets deleted by the Reaper or Undertaker as well as rule re-evaluations and subscriptions performed by the Judge and Transmogrifier.

The Database is used to persist all the logical data as well as for transactional support. Only the Rucio server or daemons directly communicate with the database. Rucio uses SQLAlchemy[9] as an object relational mapper for performance as well as for development reasons.

3. Data rebalancing
There are several issues why storage resources reach their limit of capacity and thus require data rebalancing. Data deletion in Rucio is based on replication rules. If a replication rule is set on a data identifier, such as a dataset, the data is protected from deletion on that RSE. The data is considered primary data. If no replication rule is set for data on a storage element, the data is not immediately deleted but kept until a specific storage threshold (watermark) is passed. This data is considered secondary. Once the data threshold is met the data deletion service deletes the least used secondary data until the threshold is met again.

Problematic are storage elements which are at their capacity limit but do not have any secondary data to delete. These are the storage resources which need data rebalanced to other RSEs in order to be functioning again. There are three reasons why these situations arise:

- Imbalances between computation resources and storage resources at a site. Thus, due to the higher computing capacity at a site more data is transferred and generated at the site than the storage element can hold. This is usually not a very erratic behavior, as it takes
several cycles for a site to fill up and delete secondary data until there is no secondary data left anymore, but these situations do arise regularly. Mostly Tier-1 sites are affected by this phenomenon.

- Non optimal placement of data. This can either happen due to policies or users requesting replication. The data distribution policies are rather static, thus they do not make the most optimal decision at every point in time. Due to changes in the environment, such as downtimes, storage elements can get more data than they can handle. This also includes non optimal definitions of lifetime for data. In general data is often given too much lifetime which prevents it from deletion until expiration.

- In some cases computation jobs also request too large transfers or the job output is too large for a site to handle. This brings a storage element also into a situation of crisis.

The general idea of data rebalancing in ATLAS distributed data management is to move data away from the problematic storage endpoint while still adhering to the original data replication policy as much as possible. Data integrity is a key requirement in this process, thus, the original data is not deleted until the newly created data has been transferred and validated. The rebalancing is thus conceptualized in three different mechanisms: manual (emergency) rebalancing, automatic emergency rebalancing and automatic background rebalancing.

### 3.1. Manual (emergency) rebalancing

The manual rebalancing mechanism gives DDM operators a tool to easily rebalance a given amount of data away from a storage element. This can be either done in emergency situations, such as a RSE being full without anything to delete, or also due to other data distribution needs. A rebalancing operation can be issued via the command line client interface by just specifying the RSE from which to rebalance data, and the amount of bytes to rebalance. Besides these two mandatory parameters also a set of optional parameters can be specified, such as priority, maximum number of files to rebalance, and blacklisted sites. This mechanism gives operators a very comfortable way to rebalance data, as they do not have to select, transfer and delete each single dataset, but just request a certain amount of bytes to be rebalanced.

### 3.2. Automatic emergency rebalancing

The automatic emergency rebalancing mode relieves DDM operators by autonomously detecting RSEs in crisis and rebalancing data away from them to make room for new data. Figure 1 shows a problematic data utilization workflow of an RSE. The primary data occupancy rises, eventually replacing all secondary data and reaching the volume limit of the storage element. The concept of the automatic emergency rebalancing mode is to look for RSEs which are over the watermark and which do not have any secondary data to delete. In these cases, the mechanisms rebalance a fixed amount of data away from the RSE.

### 3.3. Automatic background rebalancing

Both the manual as well as the automatic rebalancing mode are only used in situations which are already considered an emergency. The idea of the automatic background rebalancing is to prevent situations like this from even occurring. To this effect the automatic background rebalancing daemon tries to balance the amount of primary to secondary replicas at a set of RSEs, such as all Tier-1 storage elements. The idea behind this is, if the ratio is in balance, all RSEs have relatively the same amount of secondary replicas available for deletion. The optimization objective is to rebalance data from RSEs above the ratio to RSEs below the ratio. Thus data rebalanced at the source will switch from primary to secondary, thus decreasing the
ratio and data at the destination will increase the primary volume, thus increasing the ratio. The ratio is calculated as follows:

\[ \text{ratio} = \frac{\sum_{i=1}^{n} \text{primary\_volume}_i}{\sum_{i=1}^{n} \text{secondary\_volume}_i} \]  

All RSEs are ordered in decreasing ratio order and data is rebalanced from the RSE with the highest ratio to the RSE with the lowest ratio.

4. Architecture and Workflows

The implementation of the rebalancing features touches all layers of the data management system. The clients were extended with the necessary parsers and parameters to process rebalancing operations. The server was extended to digest rebalancing operations and a new daemon was developed which monitors and executes the rebalancing operations.

One of the key requirements of data rebalancing is not to jeopardize data integrity. When a replication rule gets rebalanced to a new RSE, the source data cannot be removed until the destination data is fully available, even if other replicas would be available in the system, as this would temporarily decrease the global replication factor and thus data availability. To ensure this, the Rucio rule system is extended by a parent/child relationship between rules. Each rule has a new attribute called child rule and if set, the rule cannot be deleted unless the child rule is in state OK and thus fully replicated. This mechanism allows to protect source replication rules from deletion until they are fully rebalanced.

Once a manual or automatic rebalancing operation is issued, the data rebalancing daemon works in two phases:

(i) **Data selection**: The data selection phase is responsible for selecting the datasets to be rebalanced to a new RSE. In contrast to data pre-placement algorithms, where only highly popular data is replicated, the rebalancing daemon wants to achieve the inverse effect. It
replicates data which has to be kept, due to policy reasons, but which is not used, thus it does not have any impact on concurrent analysis on the site. Hence data is used which has a long lifetime and the sorting of the list is done inverse to the last access time.

(ii) Destination selection: Once the data is selected a destination has to be picked. The general idea behind the destination selection is to replicate the original placement decision as closely as possible. There are three different cases to consider:

(a) The source rule was created by a subscription. In this case the subscription is fetched and a different destination is picked from the RSE expression of the subscription.
(b) The source rule was created with an RSE expression, such as ”tier=1&type=DATADISK”. In this case the destination is picked from any of the RSEs covered by the expression.
(c) The source rule is a conventional rule. In these cases tier consistency is enforced. Thus, if the original rule was for a Tier-1 site, the destination is also a Tier-1 site.

In any case, no RSE which already has a primary copy of the dataset is allowed to be a destination, as this would also decrease global availability of the data.

The workflow then loops and rebalances the datasets filtered and ordered by the data selection until the requested volume is reached. All three workflows presented in Section 3 use exactly the same workflow.

5. Conclusion
Rucio, the data management system of the ATLAS experiment, manages 220 PB of collaboration data globally. Due to several reasons, storage resources regularly reach their limit of capacity. As simply deleting data is often not an option, rebalancing the data to another storage element is the only alternative. Until now, this time consuming and complex task had to be done manually by operators. In this article we introduced a data rebalancing system for Rucio. The system takes care of manual and fully automatic data rebalancing, to prevent critical situations from even happening. Figure 2 shows that since May 2016, already 2.5 PB of data were rebalanced, both by manual and automatic workflows.
References

[1] ATLAS Collaboration 2008 Journal of Instrumentation 3 S08003 URL http://dx.doi.org/10.1088/1748-0221/3/08/S08003

[2] Bird I, Bos K, Brook N, Duellmann D, Eck C, Fisk I, Foster D, Gibbard B, Girone M and Grandi C 2008 EGEE, Technical design report CERN-LHCC-2005-024

[3] Branco M, Zaluska E, De Roure D, Lassnig M and Garonne V 2010 Concurrency and Computation: Practice and Experience 22 1338–1364 URL http://onlinelibrary.wiley.com/doi/10.1002/cpe.1489/full

[4] Barisits M, Serfon C, Garonne V, Lassnig M, Stewart G, Beermann T, Vigne R, Goossens L, Nairz A, Molfetas A and on behalf of the ATLAS Collaboration 2014 Journal of Physics: Conference Series 513 042003 ISSN 1742-6588

[5] Garonne V, Vigne R, Stewart G, Barisits M, Eermann T B, Lassnig M, Serfon C, Goossens L, Nairz A and on behalf of the ATLAS Collaboration 2014 Journal of Physics: Conference Series 513 042021 ISSN 1742-6588

[6] Barisits M, Serfon C, Garonne V, Lassnig M, Beermann T and Vigne R 2015 Journal of Physics: Conference Series 664 062002 ISSN 1742-6588 URL http://stacks.iop.org/1742-6596/664/i=6/a=062002?key=crossref.fb68f4788ce19b8e88060807982f1ff3

[7] Python Software Foundation 2015 URL https://docs.python.org/2.6/ Accessed on the 23 of April 2015

[8] Fielding R T and Taylor R N 2002 ACM Transactions on Internet Technology (TOIT) 2 115——150

[9] Bayer M 2015 URL http://www.sqlalchemy.org/. Accessed on the 22 of April 2015