Resource control in ATLAS distributed data management: Rucio Accounting and Quotas

Martin Barisits

On behalf of the ATLAS Collaboration

CERN PH-ADP, Geneva, Switzerland

13. April 2015

Rucio

- Rucio is the Distributed Data Management system of the ATLAS collaboration, which replaced DQ2 in 2014.
  - Manages 160 PB of data (633 Mio replicas),
  - 730 storage endpoints on more than 150 sites globally,
  - More than 5000 users/applications.
  - 240 Hz user/application frontend interaction rate.
- Architecture:
  - Scalable distributed architecture,
  - RESTful http interface to server,
  - Oauth based authentication model,
  - Client / server / daemon / resource layer.

Rucio concepts

- Accounts represent users, groups or activities.
- Namespace: Data hierarchy with meta data support:
  - Data name space is partitioned by scopes.
  - Files can be grouped into datasets.
  - Datasets can be grouped into containers.
  - $(scope, name)$ tuple identifies all data (Files, Datasets, Containers).
- Rucio Storage Element (RSE) – logical abstraction for storage space.
  - Can be assigned meta attributes.
- Replica management based on replication rules and RSE expressions.
- Synchronous accounting:
  - Counters per RSE and per $(Account, RSE)$ tuple.
- Asynchronous accounting:
  - Many different views (based on dataset metadata) calculated in Hadoop and provided as dumps.

Rucio concept: RSE expressions

- RSE expressions[1] use the RSE expression language to define a set of RSEs.
- RSE names can be directly used: `SITEA|SITEB`
- Meta-attributes on RSEs: `type=tape&country=de`
- Set operators:
  - $\cap$ (Intersect) represented by `&`
  - $\cup$ (Union) represented by `|`
  - $\setminus$ (Complement) represented by `\`
- Order of operations can be given by brackets `(,)`
- Examples:
  - `(type=tape&country=de)\SITEA`
  - `(tier=2\country=de)|CERN_EOSDISK`

[1] *Barisits et al.* – ATLAS Replica Management in Rucio, CHEP2013

Rucio concept: Replication rules

- Replica Management is based on **replication rules** set on files, datasets or containers.
- A replication rule consists of an RSE expression, defining a set of possible destination RSEs and the number of replicas it should create.

Example: A user wants to replicate a dataset to two tier-2 sites in the United Kingdom.

- Data identifier: `userA:ds1`
- copies: `2`
- RSE expression: `tier=2&country=uk`

$\rightsquigarrow$ Rucio picks two RSEs out of the set specified by the RSE expression.
Selection is based on existing replicas, quota, rule weights or just done randomly.

About quota

- **Goal:**
  - Quotas should partition the storage for different users.
  - Quotas help in protecting the storage from overload by a single user.
- Types:
  - Usage quota (or block quota): Based on volume (bytes).
  - File quota (or inode quota): Based on number of files.
  - Hard and Soft quota, allowing users to temporarily violate their quota.
- Complexity for distributed systems:
  - Performance of accounting the used space.
  - Usually organizations want have quotas for a set of storage areas and not only a single one. (Quota policy)
  - Performance of identifying the applying quotas.

Quota history

- DQ2:
  - Passive quota based on replica accounting reports.
  - Checked by a script generating automatic eMail alarms (Soft quota).
- Rucio (Quota v1):
  - Simple quota per $(account, RSE, bytes)$ tuple; ∞ $bytes$ possible.
  - Quotas are hard and instantly deny access to the resource.
  - If a quota doesn't exist access to the resource will be denied.
- Rucio (Quota v2, described in this presentation)
  - Allows quota on a group of resources (RSEs).
  - Allows overlapping quotas and handles exceptions.
  - Allows quota on # of replicas (File/inode quota).

Quota requirements

- DQ2-like passive quotas:
  - No performance requirement, as calculated passively.
  - Problem is that the system usually notifies too late, when the damage is already done and requires painful, manual, cleanup.
- Rucio v1 qutoas:
  - Architecture & workflow need to provide performance to do this actively.
  - Support storage partitioning.
  - Single-RSE quotas can only hardly be used to enforce advanced quota policy. (e.g.: A user can store 5TB of data on all Scratchdisks)
- Rucio v2 quotas:
  - Performance requirement even harder to meet due to complexity.
  - Support both storage partitioning and advanced quota policies.

Quota concept

- A quota can be defined as $(account, RSE_{expression}, bytes, files, level)$ tuple.
  - The $account$ describes the account the quota is for, e.g. `jdoe`.
  - The $RSE_{expression}$ describes the RSEs this quota is enforced for, e.g. `country=de&disk=1`.
  - $bytes$ gives the number of bytes, $files$ the number of files to enforce.
  - $level$ is used in case of overlapping quotas to allow exceptions.
- Accounting for used resources, per $(account, RSE)$ is done synchronously in the system.
- $level$ attribute is used to manage overlapping quotas:
  - Higher level quota overrides setting of lower level quota.
  - Quotas with same level: The higher $bytes$/$files$ value is taken.

Quota concept - examples

- Users have a global quota of 20TB on all Scratchdisks:
  - $(account = jdoe,$
    $RSE_{expression} =$ "scratchdisk=1",
    $bytes = 20TB,$
    $files = \infty,$
    $level = 0)$
- But, users should not have more than 2TB on a single Scratchdisk:
  - $(jdoe,$ "CERN_SCRATCHDISK", $2TB, \infty, 1)$
- But user jdoe is a power-user of BNL-OSG2_SCRATCHDISK:
  - $(jdoe,$ "BNL-OSG2_SCRATCHDISK", $\infty, \infty, 2)$

Architecture I/III

- Quotas are checked actively by the servers or daemons when creating rules.
- Current usage counters are managed actively, and lookups by primary key for the usage of $(bytes, files)$ for an $(account, RSE)$.
  Table size: $O(\#accounts \cdot \#RSEs)$
  Lookup: **Fast**
- Quotas are stored in a table indexed on the $account$ column, to quickly retrieve all applicable quotas for an account. Table size: $O(\#accounts \cdot n) \mid n \ll \#RSEs$
  Lookup: **Fast**
- For privileged accounts (root, ddmadmin, etc.) quotas are disabled and no lookups are made in the workflow.

Architecture II/III

- **Bottleneck** is the identification of applicable quotas if a user wants to write to an RSE_1.
- This is due to the case that, while quotas for an account can be retrieved fast, they are in the form $(account, RSE_{expression}, ...)$ tuples.
- This means, the system has to evaluate the $RSE_{expression}$ of each quota to decide if RSE_1 is part of it. This is potentially **slow**.

Architecture III/III

- The solution to solve this bottleneck is caching as the results of RSE Expression evaluations only change very rarely.
- A daemon reads all unique RSE Expressions from the quota and evaluates them regularily. It stores the $(RSE_{expression}, [RSE])$ tuple in a cache.
- Lookups of RSE Expressions in the cache are **very fast**.
- The selection of quotas and resolving of overlapping quotas is purely done on the server/daemon side without the database.
- Changes in quotas are enforced immediately.
- Changes in RSE Expressions (e.g. a new scratchdisk is added) take one cache lifetime to be enforced.

Workflow

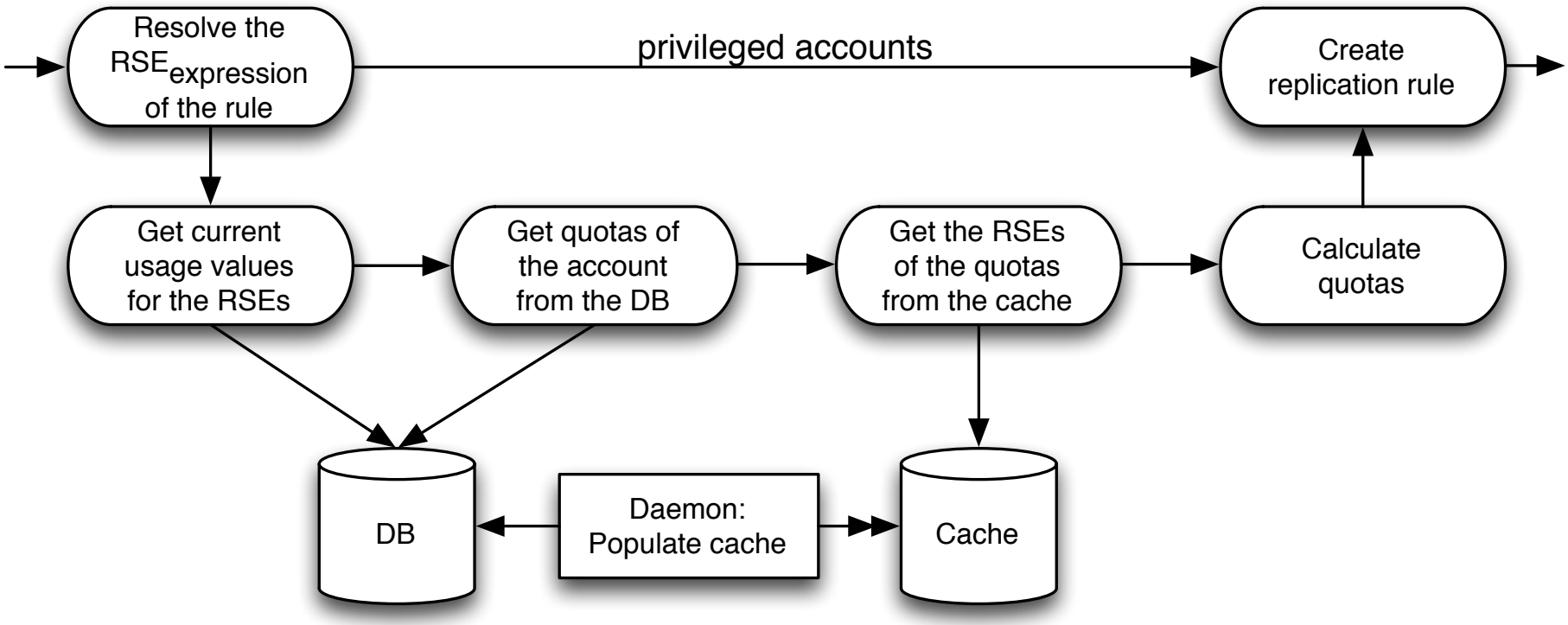

Figure: Rule creation

Evaluation

- Evaluation is difficult, as Rucio quota (v2) allows a very different (and more complex) policy.
- Compared are the quota timings of random rule creations in production with quota v1 vs. random rule creation in the test framework with quota (v2).

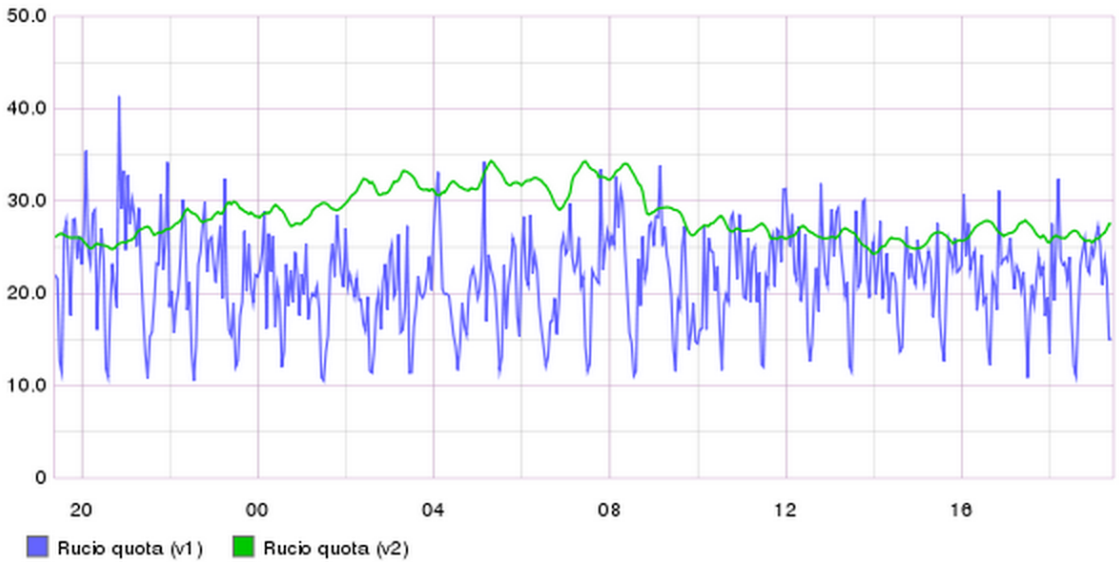

Figure: Quota checks [ms] during a period of 24h

Conclusion

- Presented a quota framework for Rucio which offers a wide toolset to implement the data policies of the collaboration, allowing complex overlapping quotas with exceptions.
- The architecture solving the performance critical parts of the system was described,
- and the workflow of the system was shown.
- Future work includes quotas on the number of concurrent transfers and volume of concurrent transfers.

Resource control in ATLAS distributed data management: Rucio Accounting and Quotas

Martin Barisits

On behalf of the ATLAS Collaboration

CERN PH-ADP, Geneva, Switzerland

13. April 2015