Rucio overview

rucio-dev@cern.ch

Rucio in a nutshell

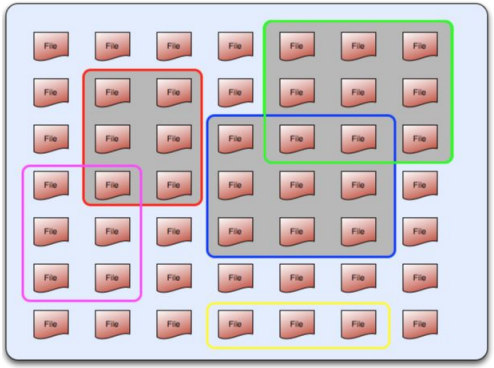
- Initially developed by the ATLAS experiment
- Provides services and libraries for scientific collaborations/experiments/communities
  - Designed with more than 10 years of operational experience in data management
  - Full, complete and generic data management service
  - The number of data intensive instruments generating unprecedented data volume is growing

- Store, manage, and process data in a heterogeneous distributed environment
  - Data can be scientific observations, measurements, objects, events, images saved in files
  - Manage transfers, deletions, and storage
  - Connects with workflow management systems
  - Supports both low-level and high-level policies and enforces them
  - A rich set of advanced features and use cases supported
  - Facilities can be distributed at various locations belonging to different administrative domain

Core concepts

Namespace handling

- Data Identifier (DID) is the primary addressable unit
  - DIDs can be either files, collections (*datasets*), or collections of collections (*containers*)
  - Datasets only hold files, containers only hold datasets
- DIDs are standalone
  - Files do not need to be in a dataset
  - Datasets do not need to be in a collection
- DIDs are globally unique
  - Files cannot have the same name as collections, and vice versa
  - Cannot reuse names of deleted DIDs
    - Why? Prevents reuse of modified files for consistently repeatable science results
- Collections can be organised freely
  - Files can be in multiple datasets, datasets can be in multiple containers
  - Crude analogy: emails with multiple labels in GMail



Namespace handling

- The global namespace containing all DIDs can be partitioned (into *scopes*)
  - At least a single partition must exist (i.e., fallback global)
  - Distinguish different communities, users, groups, or activities (*user.jdoe, group.phys-higgs, …)*
  - Also helps with namespace scalability
- DIDs are thus always tuples *<scope>:<name>*
  - Cannot have DIDs with *<name>* alone
  - Corollary: Names must be unique inside a scope only, whereas DIDs are globally unique
- Example
  - FILE *user.jdoe:my-analysis-data-123.tar.gz*
    user.jdoe:susy-analysis-script.py
  - DATASET *user.jdoe:run-123* [contains: *user.jdoe:my-analysis-data-123.tar.gz, …*]
  - CONTAINER *user.jdoe:all-my-runs* [contains: *user.jdoe:run-123, …*]
  - CONTAINER *group.phys-higgs:all-user-analy* [contains: *user.jdoe:run-123, …*]

Namespace handling

- DIDs always belong to at least one account
  - Delegated via federated identities: X509, X509 proxies, Kerberos/GSS, SSH Pubkey, UserPass
  - Under evaluation: SciTokens, Macaroons
  - Accounts can be mapped to users/groups/service activities
  - Multiple DID ownership across accounts is possible
    - Prevents deletion of data (*pulling-the-carpet* scenarios)
- Large set of available metadata, e.g.,
  - Data management: size, checksum, creation time, access time, …
  - Physics: run identification, derivation, number of events, …

Storage abstraction

- Rucio Storage Elements (RSEs) are a logical entity of space
  - No software needed to run at the site
  - RSE names are arbitrary (e.g., "CERN-PROD_DATADISK", "AWS_REGION_USEAST", …)
- RSEs collect all necessary metadata for a storage
  - protocols, hostnames, ports, prefixes, paths, implementations, …
  - data access priorities can be set (e.g., to prefer a protocol for LAN access)
- RSEs can be deterministic or non-deterministic
  - Deterministic: Function (e.g., one-way hash) takes care of storage namespace
  - Non-deterministic: Client provides explicit storage path
- RSEs can be tagged
  - Key/Value pairs (e.g., *country=UK*, *type=TAPE*, *support=brian@unl.edu*)
  - Leads to implicit grouping as necessary (e.g, all tapes in Australia)
- Existing data on storage can be registered into RSEs

Declarative data management with rules

- Express what you want with rules
  - *"Three copies of this dataset, distributed evenly across three institutes on different continents, with two copies on DISK and one on TAPE"*
  - Support for different data replication policies, e.g.
    - Archive: difficult/expensive to recreate data
    - Primary cache: data that should be readily available, job inputs/outputs, ...
    - Secondary cache: extra replicas created and deleted based on system usage for performance
- Rules allow a fully dynamic and automated data distribution
  - Rules can be dynamically added and removed by all accounts, some pending authorisation
  - Rucio constantly evaluates all rules and tries to satisfy them
    - Ensuring a minimum viable set via transfers and deletions
  - Rules enforce data lifecycles with lifetimes (e.g., automatically delete temporary data after a week)
  - Rules enforce user and group quotas (e.g, 50 PB globally for a physics group, 10 extra PB at a site)

Replicas

- Eventually, rules on DIDs lead to replicas
  - Physical representation of the file, i.e., bytes on storage
  - Collection replicas exist as a convenience
- DID *<scope>:<name>* can be different than path and filename in storage namespace
  - user.jdoe:my-analysis-data-123.tar.gz

    RSE A: /pfns/ex/users/jdoe/13465161461

    RSE B: /stor/user.jdoe/my-analysis-data-123.tar.gz
- Rucio will automatically resolve user requests for DIDs to appropriate replicas
  - Which protocol to use
  - Which storage frontend/hostname to use
  - Distance to RSE
  - …
- Monitoring and accounting is provided at the replica level

Rucio operations

Data management operations model

- Large-scale and repetitive operational tasks can be automated
  - Bulk migrating/deleting/rebalancing data across facilities at multiple institutions
  - Popularity driven replication
  - Management of disk spaces and data lifetime
  - Identification of lost data and automatic consistency recovery
- People at the sites are not operating any local Rucio service
  - Sites only operate their storage
- Rucio services run centrally, are scalable and can be easily installed
  - Strong use of open and standard technologies
  - E.g., HA, RESTful APIs, Token-based authentication
  - Lightweight, thread-safe and horizontally scalable
  - Support for RDBMS: Oracle, PostgreSQL, MySQL, MariaDB, SQLite

Third party copy & Information services

- Rucio provides a generic transfer tool API for third party copy
  - Independent of underlying transfer service
  - Asynchronous interface to any potential third-party tool
- Currently available implementation of transfer tool API is FTS3
- GlobusOnline can be integrated if requested/needed
- Rucio can be interfaced with external information or federated identity services
  - E.g., hostnames, protocols, ports, paths, data access protocols, network distances, etc.
  - E.g., users, groups, roles, identities, contact information, etc.

Operations

Monitoring & Analytics

- RucioUI
  - Provides several views for different types of users
  - Normal users: Data discovery and details, transfer requests and monitoring
  - Site admins: Quota management and transfer approvals
  - Central administration: Account / Identity / Site management
- Monitoring
  - Internal system health monitoring with Graphite / Grafana
  - Transfer / Deletion / … monitoring built on HDFS, ElasticSearch, and Spark
- Analytics and accounting
  - E..g, Show which the data is used, where and how space is used
  - Data reports for long-term views
  - Built on Hadoop and Spark

Rucio development & commissioning

- Well-established collaborative open source project
- Support community (experts, developers, user)
- 5-6 FTEs
- In Pypi: Bi-weekly patch releases, monthly feature releases
- Long initial process with gradual migration from predecessor system
  - Design phase ~1 year
  - Initial development ~2 years
  - Commissioning ~1 year

In a Nutshell, Rucio...

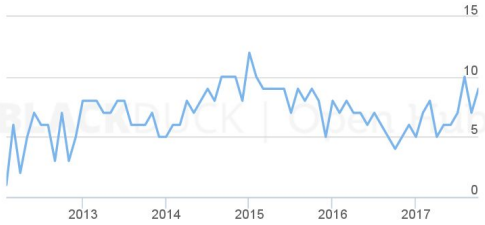
... has had 5,506 commits made by 40 contributors representing 101,729 lines of code

... is mostly written in Python with an average number of source code comments

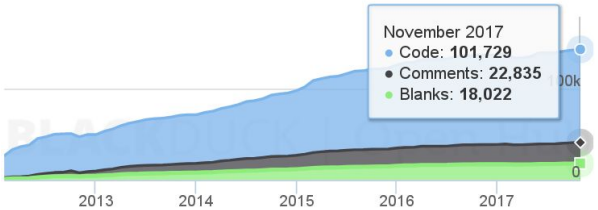
... has a well established, mature codebase maintained by a large development team with decreasing Y-O-Y commits

... took an estimated 25 years of effort (COCOMO model) starting with its first commit in February, 2012 ending with its most recent commit about 23 hours ago

Contributors per Month



Lines of Code



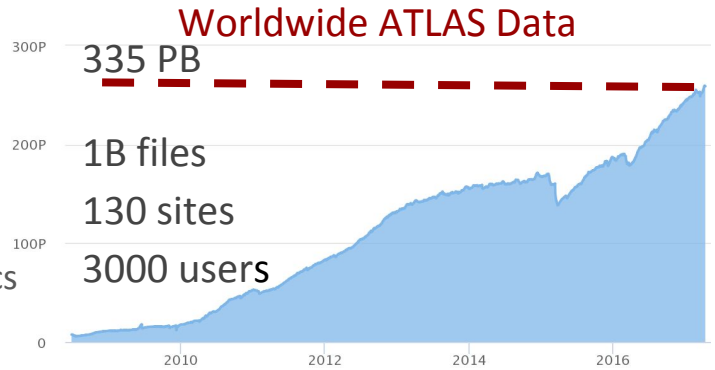
Commits per Month

Instances

Rucio & ATLAS

- Charged with managing all data products
  - C++ objects representing tracks, parts of detector etc, saved in files
  - Data is reconstructed and reduced through various formats: Detector, Simulation, Analysis (GB to MB)
- Main functionalities
  - Discovery, Location, Transfer, Deletion
  - Quota, Permission, Consistency, Monitoring, Analytics
  - Can enforce computing models
- Easy integration with workload management
  - 1M ATLAS Jobs/day
- Enables heterogeneous data management
  - No storage vendor/product lock-in to follow the market

- Rucio has demonstrated very large scale 24/7 data management service



Transfers:
40M files/Month
40 PB/Month

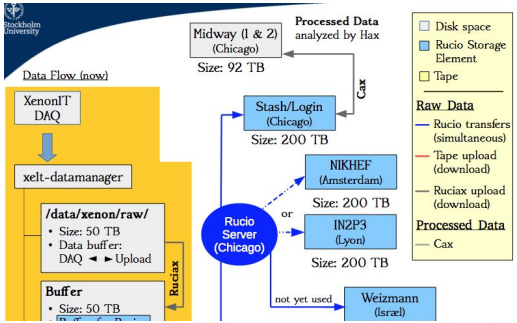
Upload:
150M files/Month
50PB/Month

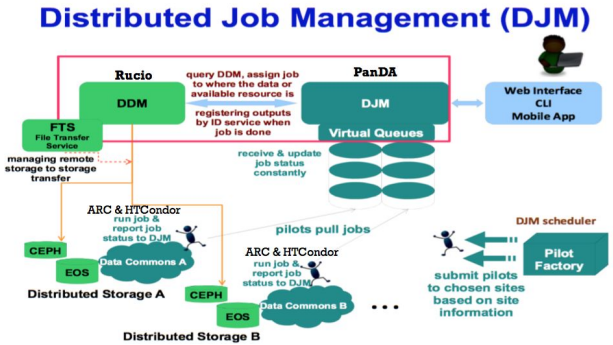
Deletion: 100M files/Month, 40 PB/Month

Rucio beyond ATLAS

- The AMS and Xenon1T experiments are already using Rucio in production:
  - Xenon1T Dark Matter Search
    - Thousands of files across 6 sites (Europe and US), using the MariaDB backend, operated by UChicago
  - AMS (Alpha Magnetic Spectrometer)
    - Millions of files across 10 sites, using the MySQL backend, operated by ASGC Taiwan
- CMS using the PostgreSQL backend operated by UChicago to evaluate Rucio
- + COMPASS, LSST and some others

→ Rucio Community Workshop: March 1-2, 2018

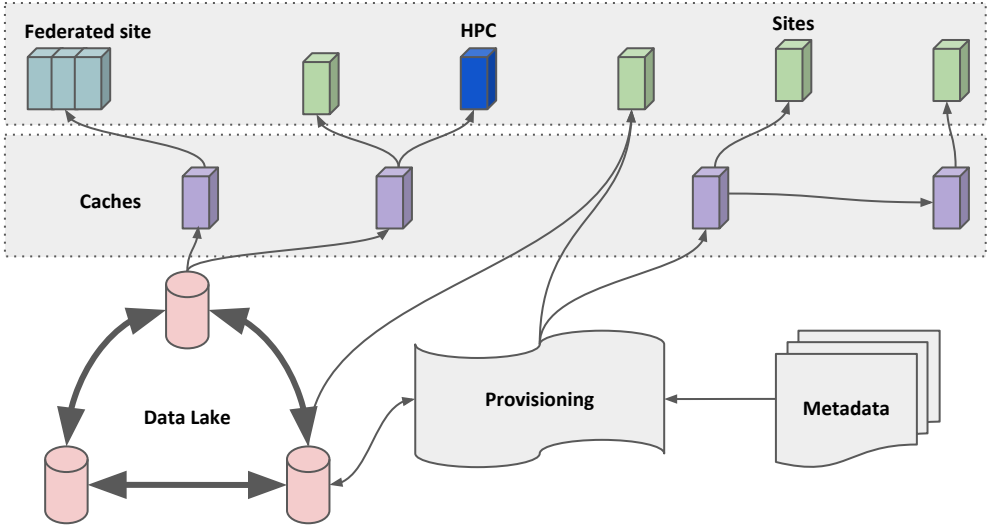




Future

Medium/Long-term planned development

- Storage integration improvements
  - E.g., Objectstores, Cloud stores
  - Storage authentication
- Object/Sub-file workflows
  - Support fine-grained computing model
  - Space & Time savings
  - Better exploitation of HPCs and time-shared or volatile resources



- Network has proven to be cheaper and better than expected
  - Remote data access over wide area network
  - Exploit client locality w.r.t data locality/placement (CDN-style)
  - Improve network usage with SDNs

https://indico.cern.ch/event/676472/

Rucio Community Workshop: March 1-2, 2018

- Looking for user stories from scientific collaborations/experiments/communities
  - HEP, neutrinos, astronomy, biology, medical science, environmental and earth sciences, …
    - Collaboration scale, size, requirements, data model, metadata schema, access pattern, etc.
    - Data curation and characterisation (data quality)
    - Workflow from data production/acquisition to scientific results and preservation
- Rucio will benefit from third party services integration and new R&D paradigms
  - Integration with workload management systems and processing capabilities
  - Complement Rucio with more generic metadata services with customisable schemas
  - More innovative machine learning techniques to optimize system performances
  - Smart placement of data w.r.t. data centre and network capabilities
- We would like to share Rucio with you and work together
  - We will be happy to share our expertise and operational experience for a resilient service
  - Feeding back into the code base existing experience in supporting more scientific collaborations

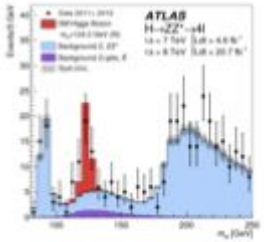
More information

Website	http://rucio.cern.ch
Documentation	https://rucio.readthedocs.io
Repository	https://github.com/rucio/
Continuous Integration	https://travis-ci.org/rucio/
Images	https://hub.docker.com/r/rucio/
Online support	https://rucio.slack.com/messages/#support/
Developer contact	rucio-dev@cern.ch

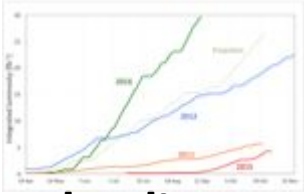
Backup

LHC Upgrade Timeline

Higgs discovery in Run-1

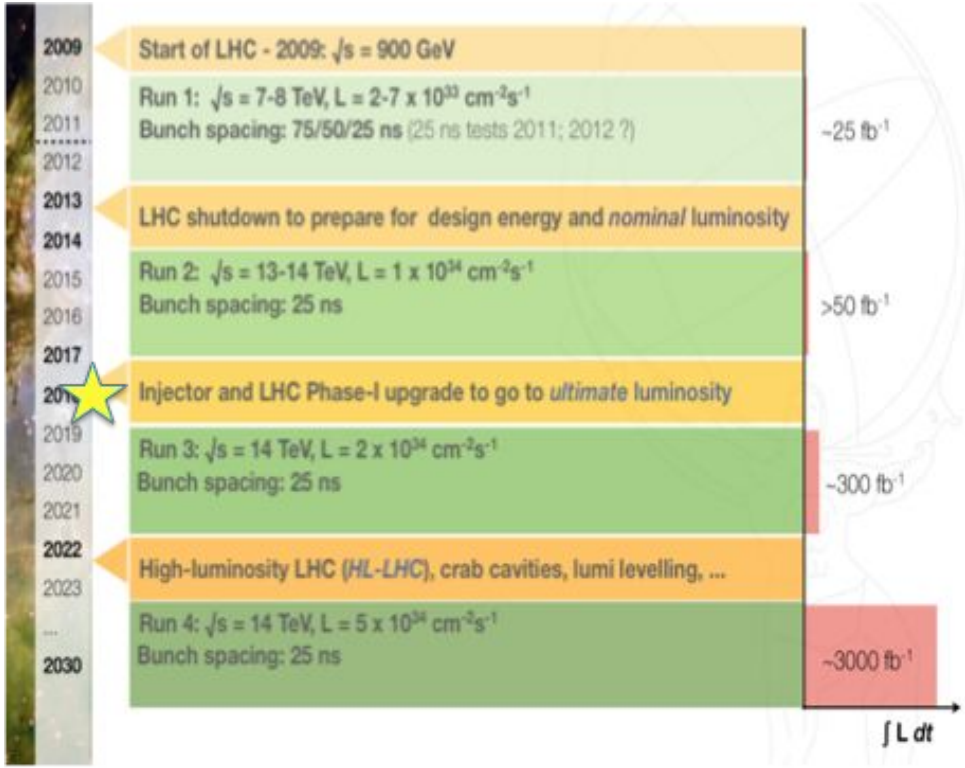


We are here: Run-2



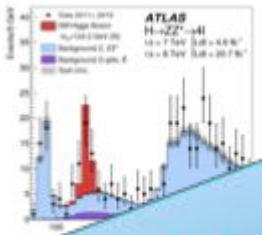
High Luminosity:

the HL-LHC challenge

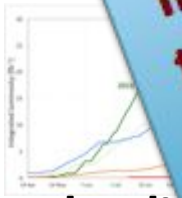


LHC Upgrade Timeline

Higgs discovery in Run-1

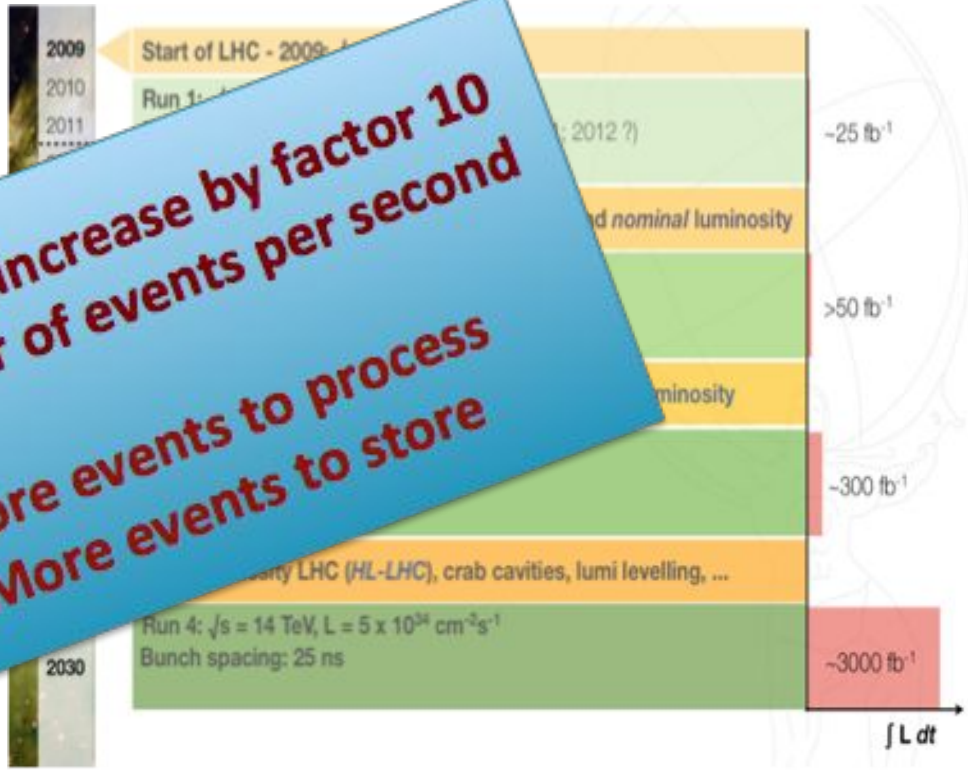


We are here



High Luminosity

the HL-LHC challe



In 10 years, increase by factor 10 the number of events per second

→ More events to process

→ More events to store

LHC Upgrade Timeline

Higgs discovery in Run-1

We are here

High Luminosity

the HL-LHC cha