CMS Workload Management and Rucio integration

Todor Ivanov on behalf of WMCore Team

Introduction

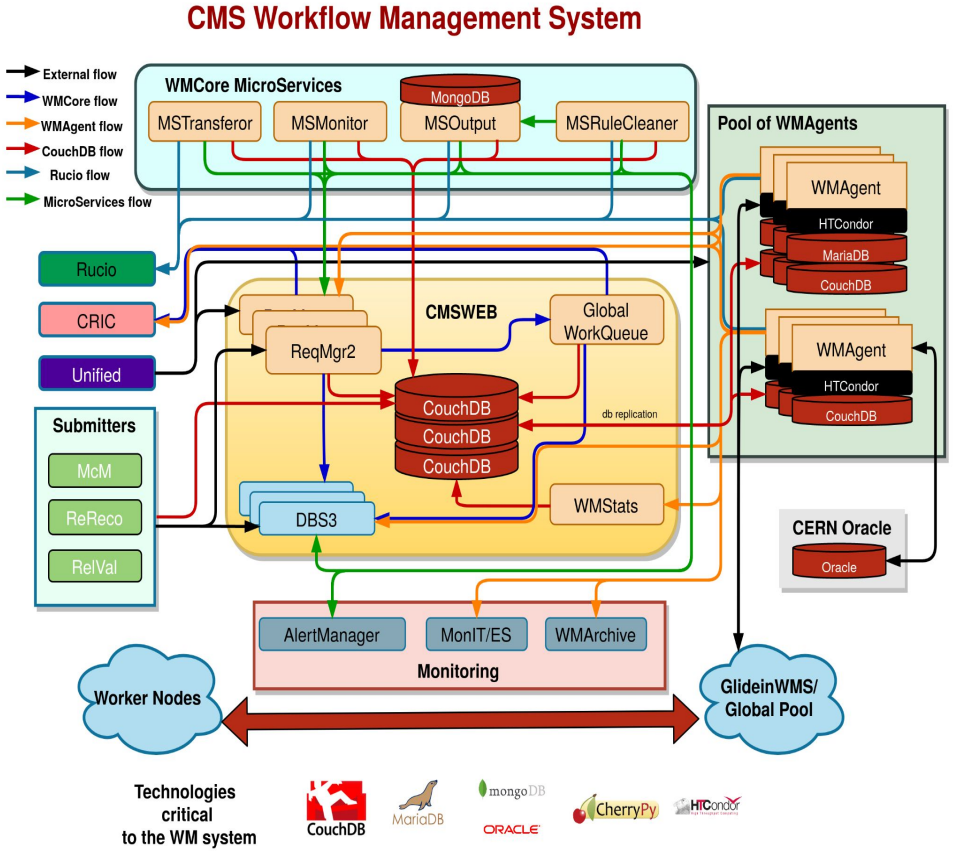
WMCore/WMAgent are the main components of the CMS Workload Management system, driving all the Monte Carlo production and Data reprocessing.

Until mid 2020 we have been using Phedex + Dynamo as our data management system. Really good solutions which were fulfilling our needs and were serving us very well throughout the years, but:

- Those were internally developed.
- Lack of broader community with wide experience outside the boundaries of the CMS experiment.
- Required dedicated development in order to fit into the new scalability standards imposed by the Experiment.

Following the tendencies in the scientific community we have recently integrated Rucio as our default data management system.

- It does have a broader community
- It is supposed to be capable of holding the pressure from the demanding new scalability requirements



WMCore components

Request Manager2:

- Provides the tools for request creation and management (REST API, Web GUI)
- Validate the requests parameters.
- Manage the request life cycle (update status, priority etc.)
- Store the request data in DB (CouchDB backend)
- Provide search tools on request data (REST API, Web GUI)

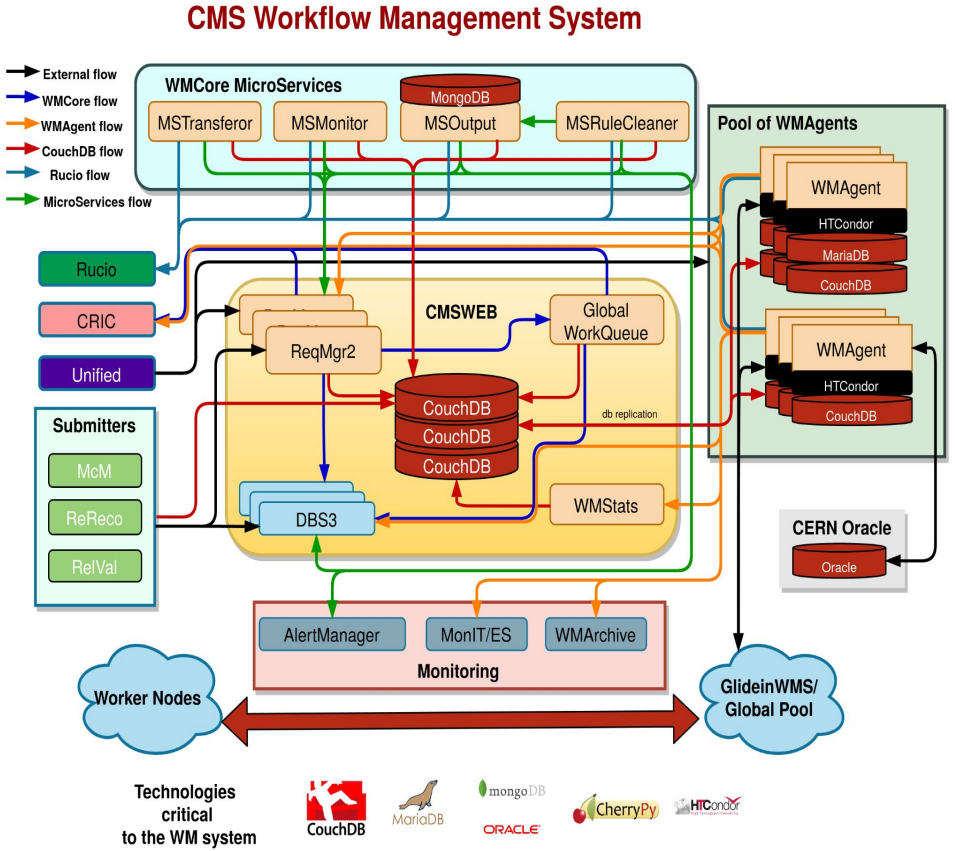
WorkQueue:

A multi-level queue. Represents work as WorkQueueElements. These elements carry key information such as input data, statistics of the amount of work measured in number of jobs, state and other operational data.

- The global WorkQueue takes work (requests) from the ReqMgr2 and
- Splits them in small pieces - WorkQueue Elements (block) in order to make it available for the local queues to acquire and posteriorly inject it into WMBS.
- Provide estimated jobs for each WQE which used for matching the resources.
- Updates locations of the WorkQueue Elements periodically
- Deletes the WQEs when request is completed

WMStats:

- Monitors current status of request and collective job. (snapshot of the current status)
- Debugging tool - provides a hierarchical view on request (from request to job level granularity to track down problems)
- Provides search tool for finding requests. (REST API, Web GUI)
- Provides error, information about the requests and the infrastructure



WMCore components interacting with Rucio

WMAgent:

The WMAgent software is a distributed component of the production system, in a nutshell its functions are:

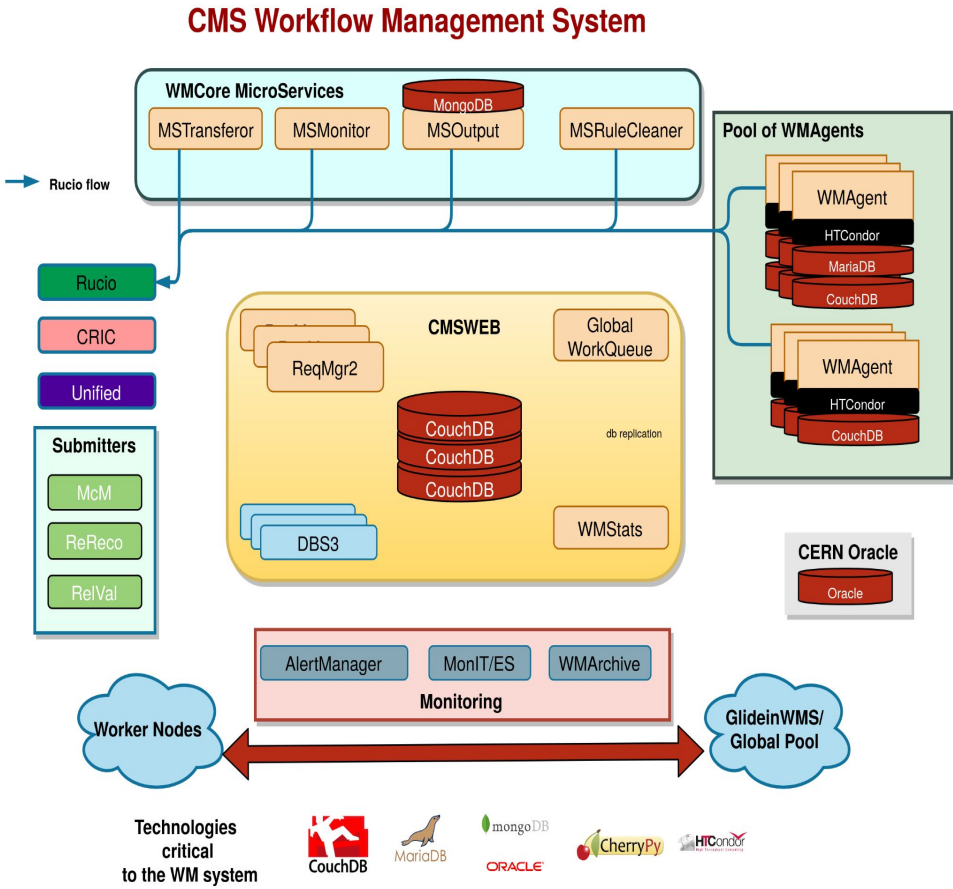
- Splitting WorkQueue elements into smaller basic work units, known as jobs.
- Creating jobs and controlling the flow of work according for the tasks defined in the workload of a request.
- Submitting jobs to a batch system (e.g. HTCondor, LSF).
- Tracking the submitted jobs and keeping tabs on their outcome.
- Registering the produced data into the CMS catalogs (i.e. DBS2/3).

MicroServices:

- MSTransferor
- MSMonitor
- MSOutput
- MSRulecleaner

GlobalWorkQueue:

- It needs Rucio for data discovery solely



High level view of Rucio usage

Rest APIs from Rucio through Rucio Client

We have our own python module, working as a wrapper on top of the RucioClient. Reasons for that are:

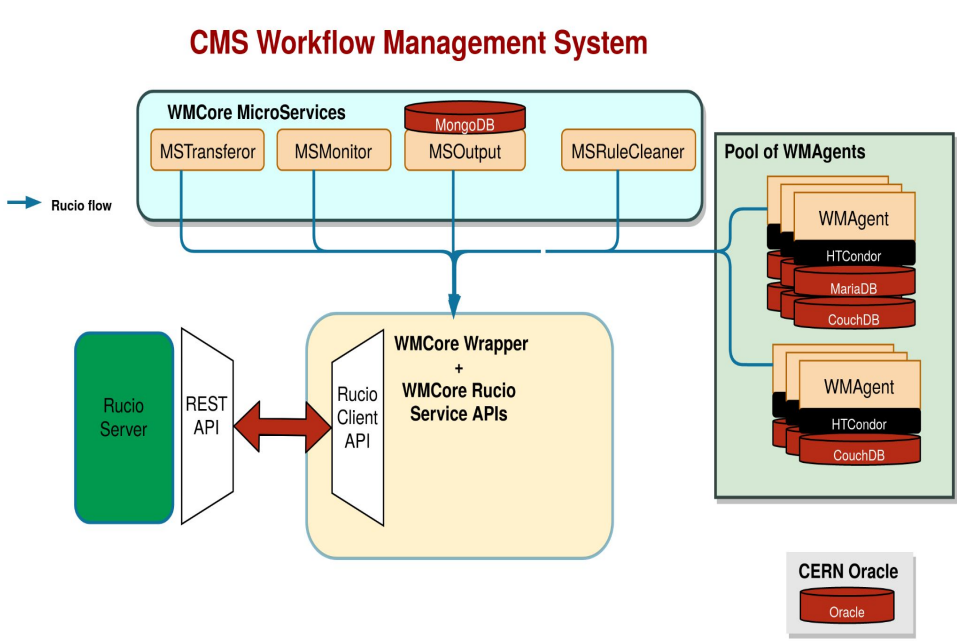
- Code maintenance - we develop/adapt only needed APIs
- Use of the caching modules provided by WMCore
- Error handling and logging provided by WMCore
- Disentangled from the mainstream RucioClient version (as long as Rucio keeps backwards compatibility)

Data abstraction granularity

A general misalignment between Rucio and CMS due to inheritance from the previous data management system.

- **CMS dataset** corresponds to a **Rucio container**
- **CMS block** corresponds to a **Rucio dataset**
- **CMS file** corresponds to a **Rucio file**

Difficult to alleviate. We need to constantly keep track of inherited terminology, namings and references in the code and different parts of the system as a whole.



MicroServices @ initial WF states

MSTransferor:

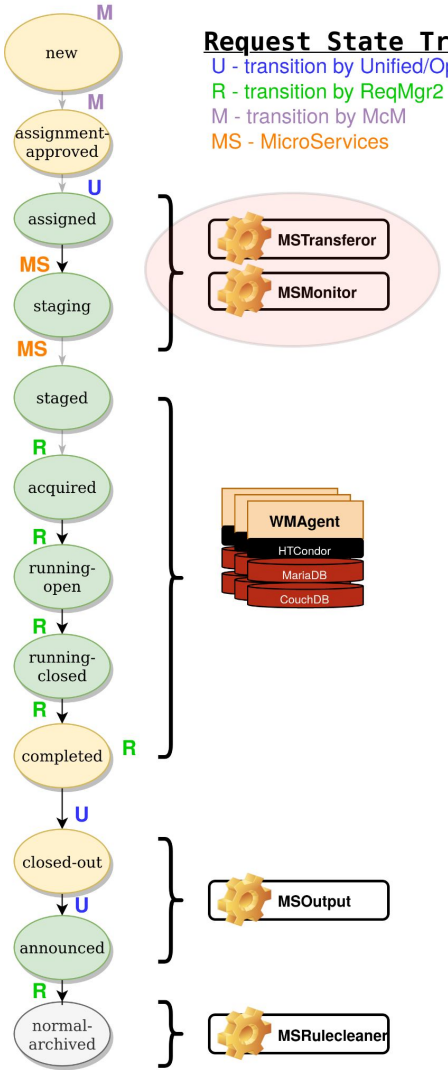
Responsible for executing all the necessary input data placement before a workflow can get acquired by Global Workqueue and the underlying agents.

Input data classification:

1. **No input data at all**: starts from the event generation step
2. **Primary input data**: it's the signal to be processed in the workflow - block level read
3. **Secondary input data**: it corresponds to the pileup dataset
   - **premix pileup:** large size, but low I/O - allows remote read
   - **classical pileup:** variable size and high I/O - no remote read allowed
4. **Parent input data**: The workflow will process events from the parent dataset

MSMonitor:

- Monitors the input data placement made by the Transferor
- Depending on the transfer status makes a request status transition to staged, allowing Global WorkQueue to fetch those requests



Rucio requirements:

* Account limits and usage are part of the input data placement algorithm
* Pileup data is usually placed as a whole at a single location. Number of replicas depend on the usage pattern
* Input rucio datasets are scattered around all the workflow allowed sites/rses.

PS.: Replication rules created against many RSEs can affect how data (lock) moves during the rule lifetime.

WMAgent @ active WF states

WorkQueueManager:

- Responsible for a continuous input data location update. For chunks of work sitting available in the local queue.

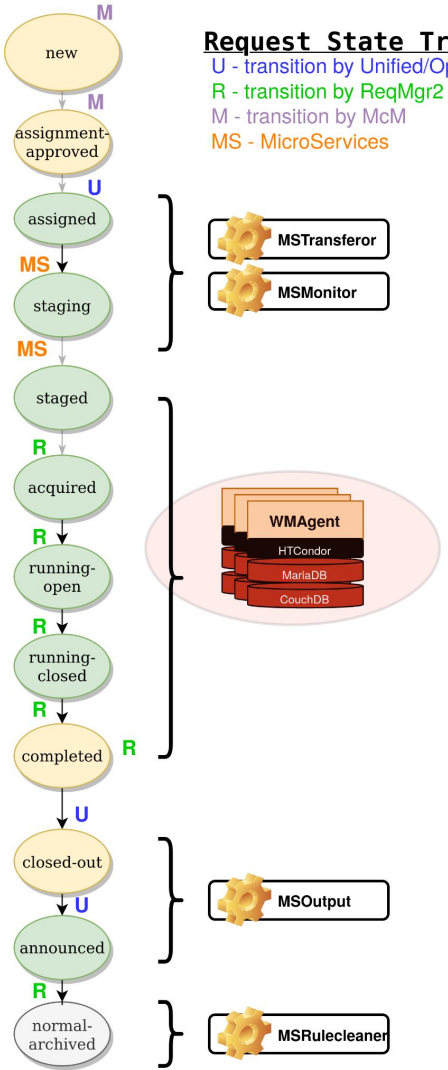
RucioInjector:

Takes care of output data produced. Responsible for:

- Insert containers (with the cli.add_container API)
- Insert rucio datasets (with cli.add_dataset). If successful, attach it to the concerning container (with cli.attach_dids)
- Account and lock rucio datasets at their origin location via a replication rule (with cli.add_replication_rule)

  NOTE: this can be done in bulk. If a DuplicateRule is raised, we need them to discover which of the datasets already has such rule (slow process!)

- Insert replicas (with cli.add_replicas). If successful, attach it to the concerning rucio dataset
- After some threshold, rucio datasets are also closed such that they do not receive any new replicas (with cli.close)
- T0 specific: rucio datasets replicated to their final destination - verified through cli.list_content (rucio datasets in a container) and cli.list_dataset_replicas_bulk - can have their origin location rule deleted (with cli.delete_replication_rule)
- Optional: containers might get a rule as well. Some Tape RSEs require to be approved (with cli.add_replication_rule), and rules are created as such.



Rucio requirements:

* Where data gets created and injected into Rucio, considering the origin data location
* Current relationship is: 1 CT → 1..N DSETS → 1..N replicas
* Replicas are injected into a dataset as they are being produced
* Dataset lock is made for proper accounting and data safety

PS.: attaching DIDs could be part of add_did APIs.

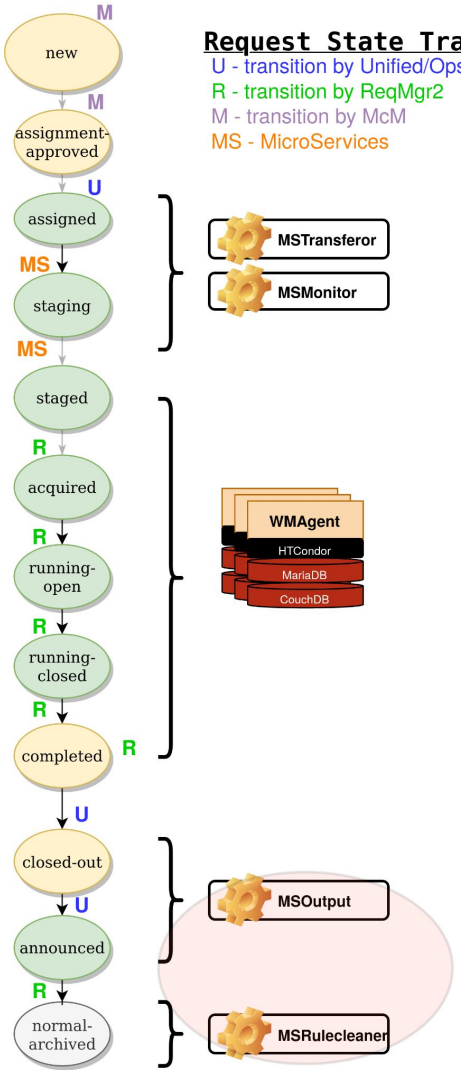
MicroServices @ final WF states

MSOutput:

- Creates Disk Replica for output data placement.
- Creates tape rules for long term data preservation.

MSRulecleaner:

- Cleans any transient disk rule created by previous stages
  - From both MSTransferror and WMAgent
  - Recognises the type of rule to deal with by the rule creator
- Archives the workflow



Rucio requirements:

* Final output data placement logic depends on a general and campaign-level policy.
* Rules are made with a weight expression and ALL data is grouped in the same RSE
* Disk rules have a well defined lifetime
* Most of the output data gets at least one Disk replica and one permanent Tape rule.

* Once input data is no longer needed, their rules locking them is deleted.
* Once output data has been safely archived on Tape, its intermediate rules can be deleted too.

Possible Improvements

For WMAgent:

- Provide a single API that performs an atomic operation for adding a DID and attaching it to another DID
- A better way of handling DuplicateRule Exception when inserting items in bulk. We would like to avoid manual and expensive resolution on the client side - Either a clear message, pointing to which are the duplicates or an additional API giving the returning only the duplicates in a single call: https://github.com/rucio/rucio/issues/3807

For Microservices:

- Support container input type for get_dataset_locks API: https://github.com/rucio/rucio/issues/4807
- Extend get_dataset_locks API to support requests in bulk - thus, client providing a list of datasets to be discovered - we need to find out all the RSEs holding parts of it: https://github.com/rucio/rucio/issues/3982
- Evaluate dataset replication state in the get_dataset_locks API - given a rule on a container, we would want to check which datasets are locked, where, and whether all the files have already been transferred: https://github.com/rucio/rucio/issues/4034
- Properly and on a timely manner update timestamps for Stuck rules: https://github.com/rucio/rucio/issues/4592
- Static locks per RSE for multi-RSE rules - We want to be sure Rucio does not move/transfer the data from an RSE while our agent/htcondor is delaying processing at a single site out of the many covered by that same rule.

Summary

- WMCore has walked the long path of integrating Rucio as its data management system successfully
- We try to benefit to the best from Rucio functionalities, completely relying on Rucio to do the heavy lifting, while we just deal with rule manipulation.
- Maintaining a low traffic to the service is a hard job for us - we through a lot of work to Rucio and any capability for aggregation of queries or responses is quite welcome
- Rucio is still a new avenue for us - plenty more to be discovered and explored.

Backup slides:

MicroServices

MSRulecleaner:

This is a single threaded stateless service.

Responsible for:

- Cleaning all Rucio rules (block and container level)
- Archival of all cleaned Workflows

Basic abstraction: Central abstraction element in the model is the Functional Pipeline - a reduce action on functions taking objects as input instead of a reduce action on objects directly.

Backend database: None

Provides:

- Separation of the service architecture from the functional implementation
- Allowing to have the same set of functions cleaning multiple types of data sources and granularity by just rearranging them in the proper set of pipelines.
- Flexible mechanism allowing to call external modules in the middle of the service logic leaving the rest of the code untouched

Requirements:

- All the functions need to have the same type of object as input
- All the functions need to return the object on exit

MicroServices

MSOutput:

This is a multithreaded Stateful service.

Responsible for:

- executing all the necessary output data placement after a workflow is completed.

Output data classification:
Based on both data tier and type of workflow producing the data.

1. **Release Validation:** Well defined destination map based on data tier
2. **Standard:** Destination map should be based on campaign map

Basic abstraction: Producer - Consumer model

Backend database: MongoDB

Provides:

- Stateful service capable of keeping truck of any partially completed actions on a workflow basis
- A database with “flat” and fixed types document representing a workflow
- Takes care of both Disk + Tape data placement

Requirements:

- All the the records in the database (even though a NoSQL database) to have a fixed schema - this is an artificially implied constraint in order to provide better database scalability