Integrity Check

(Team B2Safe: Long Phan, Jülich; …)

1. Introduction

This document describes the implementation of some additional new features in the release B2Safe (version 1.2?). Beginning from concrete requirements of Communities and Projects, many contributed ideas are (being) built in EUDAT meeting and conferences. The required policies are defined as the integrity of the data must be checked, check that all copies existing on the media exist in the database, checksum generation policy, replica integrity verification policy etc. [1] so that it's necessary to analyze and develop couple approaches for integrity check process.
Integrity check is an complex process regarding tasks at local site and at remote sites to warrant that data set are (being) replicated from site to sites correctly and consistently even when data are being changed regularly. The rules can be assembled from different functions to execute the defined policies to warrant the integrity of data set at source site and replicated site. Many important parts of tasks will be processed included checksum and size of replicated data object, Parent PID (PPID) and PID of replicated data object, number of replicated data objects, access permission on data object. Besides, security issue of user access management has been also already enhanced in this release.

2. Scenarios and Approaches

2.1 Scenarios:
The first case: User wants to execute data transfer from one site to other sites. While data set is being transfered and different functions for checksum will be processed in the meantime to test whether one data object has been transfered correctly. Otherwise, this data object will be pushed into fail.log to retransfer later. This integrity check will go on during transfering every data object.

The second case: User wants to perform task checksum on data set at his local system. In this case, User will execute chain of rules to update checksum of data set at his local site.

The third case: User wants to do a extra test with checksum after transfering data object. In this case, User will perform chain of rules to determine integrity of data's checksum also size at user's location (called source_location) and checksum also size of the replicated data object at other locations are exactly the same. The data objects which fail the test would be considered as error and saved in fail.log.

Integrity Check process are performed by way PUSH. It means that Data Owner/ Community Manager (CM) at source_location, will decide when the replicated data object at target-location will be updated. CM initiates task update content of data at local site, and then trigger update contents of replicated data at target site by retransfering the renewable data objects to target-sites.

Tasks of integrity check will be processed in all 3 layers (Physical filesystem, icat irods, pid epic-server).

The first layer is supposed to be physical layer where data set will be storaged in file system (hard drive, tape, etc.). Calculating checksum data at this layer is low-level and it always returns the correct value of checksum of data object in most updated status at all time.

The second layer is irods-layer where information about data object are saved in (system) metadata of iCAT (database of irods system). Persistent State Information in iCAT Metadata Catalog „DATA_CHECKSUM“ of data will be updated with the one micro service in irods msiDataObjChksum.

The third layer is EPIC-layer where information of data object are saved in pid-record on epic-server. Firstly, checksum in iCAT 2.layer need to be updated, subsequently update checksum on 3.layer will be executed by external python-script epicclient.py using rest-(style) query (Http-get, Http-put).

It can be summarized like following: Checksum process will begin at 2.layer irods to get the most updated checksum from 1.layer and update checksum in iCAT and then in PID-record at local site. After that, user can perform chain of rules to perform integrity check of data at target sites. The following picture will illustrate integrity check process at 1 site (Location A) and between 2 sites (Location A and Location B):

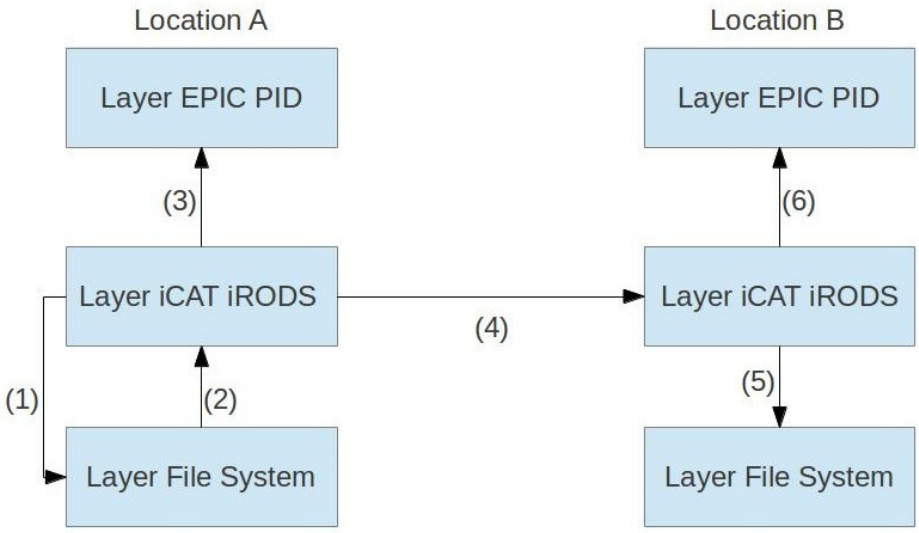

Figure - Integrity Check Process

(1) calculate checksum at Physical Layer File System (md5?) base on microservice msiDataObjChksum
(2) an update checksum at iCAT iRODS Layer (eudat.re/ EUDATiCHECKSUMget(...)),
(3) update checksum at PID Layer (eudat.re/ EUDATeCHECKSUMupdate(...)),
(4) using information from PID to interact with other target_site where the replicated data are stored,
(5) retransfer to update checksum of replicated data at target_site (replication.re/ transferSingleFile(...)),
(6) update checksum in PID at target_site by external-script epicclient (executed by operation modify).

The following ways to do above steps:
i) Pre-Transfer:
User will perform different checksum steps on physical layer, irods layer and epic layer at local site. The applied functions are EUDATeiPIDeiChecksum... ,

ii) Transfer:
During data transfer, comparing checksum and size of the replicating data will be performed synchron. It means that function checkError(...) is being integrated into the workflow of data replication and catch possible errors and push all of error-data into fail.log automatically.

iii) Post-Transfer:
User will perform action loop on a collection to test whether all data objects under this collection have been transfered/ replicated correctly. All data objects which have been tracked out and confirmed as error. There are two options:

1. All detected error-data will be pushed into fail.log (demonstrated with queue-log in logging_mechanism) with function checkError(..). All error-data in queue.log would be retransfered again by user with one function transferUsingFailLog(..).

2. or all detected error-data objects can be retransfered immediately instead of logging them in queue.log with function checkReplicas(..). While transfering data, checksum at target-site will be updated as well.

   Note: replace line transferSingleFile(x,y) in function transferCollection(...) (replication.re) with checkError(x,y) case iii-1) or checkReplicas(x,y) case iii-2)

One (optional) possible extension is applying cron-job to execute integrity check process periodic or at specific time.

2.2 Approaches (Implementations by functions/ features):
- The common rulebase now are (being) reorganized with different rulebases:
Main Functions:
rulebase „**eudat.re**“ contains main functions regarding processing in PIDs, iCAT and replication of single data object from source_location to target_location.
The concrete new functions which have contributed to Integrity Check process:

- CheckReplicas : check error checksum/size of data object at source and destination. Error-data will be triggered to transfer right after error-detection.
- CheckError : check error checksum/ size of data object at source and destination. Error-data will be pushed into fail.log, enable to investigate the causes of error and retransfer. Otherwise it's possible that an error-data object will be repeated many time to transfer without success.
- Logging_mechanism: is built based on data structure Queue and executed by python-script log.manager.py to do different operations (push, pop, queuesize, log) to data objects. This new logging mechanism is very useful to sort out only the important information about data transfer from normal rodsLog of irods, support monitoring data transfer with b2safe.log and have control on data objects which have been confirmed as error during process transfer.
- The functions which demonstrate basic operations on PID EUDATi... and iCAT EUDATe... to maintain newest state information of data object in PID and in iCAT.

User Authentication Management:
This release supplements one new feature to control the user authentication „Who are allowed to execute external scripts“ (ex. epicclient.py) on remote-server to do operations with PIDs on their data. This feature is supported by new iRODS hook acPreProcForExecCmd (version irods 3.3). There are right now 2 ways:
- The first chosen way is using one python-script (**log.manager.py**). This script applied one predefined json.file which stores information about users and their allowed actions. This json.file is saved at target-location to decide who are allowed to do what on remote-server.
- **eudat-authZ-filter.re**: contains optional function for extension of first choice by checking the data Owner in iCAT and user who is accessing into data. Only data owner has permissions to do actions on their data and pid of data. It's possible for data owner to assign access right to other user. This feature is enabled by exploiting AVU (user)-metadata of iCAT.

Processing Error during transfer:
catchError.re: This rule set contains functions to catch and process the errors which could possibly happen during transfer. Currently the possible recognized errors which have been checked and processed during data transfer are error checksum, error size, data object without pid, data object without update pid at source location, user without access permission on data object. The processing of the errors play one important role into integrity check in distributed data environment. Especially when data set are often organized as hierarchical collection structure and could contains data of many work groups or users.

Wrapper for data transfer:
replication.re: This rule set contains a group of functions with the purpose as a wrapper of data transfer, check Error and update Logging, begin with main-function transferSingleFile(x,y) from x_source to y_destination. Some additional functions regarding collection management (beta-version) like different kind of implementations for transfering the whole Collection, or support an overview on collection with hierarchical data structure are (being) developed and tested.

[... some variation of Collection management (beta version):
+ transferCollection: is supposed to be standard way to transfer one collection from site A to site B.
+ transferCollectionStressMemory: is supposed to be optional choice, this function will probably get problem out of memory when using foreach-loop on collection with version irods3.2.
+ transferCollectionAVU: is supposed to be another way to transfer collection, it exploits function AVU of icat to create a log.file and use this file as link to transfer path of data object into database and transfer it. After transfering whole collection, all saved in icat links should return clear result. But this function need to be assessed when it's probably not good way to do operations (create/ read/ write/ delete) AVU many times into iCAT. Another good view of this way is using AVU as cache point to retransfer the rest of data set.
+ transferCollectionWithoutRecursion: introduce a way to transfer only data objects at root without going into subcollection recursively. It's useful when user needs to transfer many data objects at root.
+ transferUsingFailLog: introduce a way to transfer only data objects saved in queue.fail.log. This function is useful when user does not need to replicate whole collection only to transfer small data sets which was failed at last data transfer. ...]

3. Conclusion:

3.1) Enhancement in this release

- Support Integrity check which are performed at local site and between different sites based on irods-rules and microservices and interlinking mechanism of pid to ensure that the other sites will also have the replicated data set in updated format.
- Another interesting features of this new release beside integrity checksum are logging_mechanism and ability to retransfer all errors-data from fail.log instead of having to retransfer all of data sets. It helps reduce a lot of (redundant) data transfers and overload of network. Therefore, it can minimize/ optimize number of data transfer, avoid to repeat transfering the correct replicated data objects. This approach is especial much more useful when transfering big data set regularly.
- Enhancement of security problem in execution of script on remote-server.
- (in beta, still being tested) like wrapper to possibly transfer whole collection from site A to site B in different ways, get an overview of collection, support user to check status of replicated collection after transfering.

3.2) Challenges and approaches

Issue transfering a whole big data set of many hundreds thousands data objects and the total size up to hundreds Terabyte with demand to ensure their integrity is extremly difficult, especially in context of distributed environment between data centers at different cities and when data set are being continuously updated by many users from different groups with the different IT-infrastructure. Architecture iRODS has facilitated these problems by grouping all different data storages into common virtual storage collection. When many servers at different locations have installed iRODS, they can be federated each other to create the common (virtual) data grid for exchange information and data [2][3]. However, processing of functions in EUDAT ruleset needs to go through Rule Engine and its iCAT where are considered as one of bottlenecks in scalability and performance [4]. Therefore, processing tasks would cause additional needed running time and also memory of current system. Especially process irodsAgent which conducts calling rule consumes specific number of memory during execution as well as transfering big data set between EUDAT sites.

[… *From assessment of testing transfering many thousands data objects, rule contains EUDAT-functions can runs without broken when process irodsAgent reachs about 1/3 available memory of current system* ...]

To ensure a complete task with executed irods-rule without broken by out of memory, user should consider to transfer data set that coressponds with available IT-infrastructure (ex. Memory, network condition) and build up strategies to transfer data set. The strategies can be synthesized by different rules (ex. Transfer smaller data set, transfer using fail log, keep monitoring on transfered data set).

The approaches could be extended/ developed in next release:
- Recognization the available (limited) IT-infrastructure automatically to transfer a corresponding data set
- Saving the broken point to be able to continue transfering from broken point instead of transfering from beginning (ex. exploit AVU and enhance function of transferCollectionAVU).
- Enhancement of function catchError to process new errors. We still need to find out all of possible errors which could happen during workflow data transfer and improve functionality of logging_mechanism as well as policies to process these errors. It's very neccessary since one unknown error causes the executing-rule down, make data transfer unable to finish and the connection is broken, even logging_mechanism in this case won't work correctly and lead to tasks of unexpected replication again and again. Developers of iRODS has suggested a list of all possible errors [5] which would happen inside irods, so our tasks is keep working on this list to find out what kind of errors needed as well as adapting methods processing error (in catchError.re) for our data replication.

With these new features recently in this release, especially feature Integrity Check has contributed a lot to tasks of Data Transfer and Replication in iRODS environment. We've been walking one step further and step by step on the way achieving our goal from B2Safe to (B2)Safe.

References

[1] https://confluence.csc.fi/display/Eudat/Integrity%20Checking%20Policies
[2] https://wiki.irods.org/index.php/federation
[3] Reagan W. Moore, San Diego Supercomputer Center, University of California, San Diego „Evolution of Data Grid Concepts“
[4] Arcot Rajasekar, Reagan Moore University of North Carolina, Chapel Hill, NC ; Michael Wan & Wayne Schroeder University of California at San Diego, La Jolla, CA „Universal View and Open Policy: Paradigms for Collaboration in Data Grids“
[5] https://wiki.irods.org/index.php/iRODS_Error_Codes