
LHCbDIRAC Documentation

Release master

LHCbDIRAC Project.

08:54 04/07/2022 Coordinated Universal Time

Contents

1	Developers Guide	3
1.1	Guide for developing LHCbDIRAC (and DIRAC, for LHCb developers)	3
1.2	Developing DIRAC and LHCbDIRAC	5
1.3	HOW TOs	6
1.4	Browsing the code running in production	6
1.5	I developed something, I want it in the next release	6
1.6	Asking for a LHCbDIRAC patch	6
2	Administrator Guide	7
2.1	LHCbDIRAC Releases	7
2.2	Renewal of certificate for ONLINE machine	12
2.3	Pit export	13
2.4	Online DIRAC	14
2.5	Renewal of the host certificate	15
2.6	Troubleshooting	15
2.7	Bookkeeping	15
2.8	Data distribution	33
2.9	ReStripping	37
2.10	Productions flushing	39
2.11	DIRAC on BOINC	41
2.12	HowTo	55
2.13	LHCbDIRAC Logs	70
2.14	Message Queues for LHCb	71
2.15	Sandbox Store	72
2.16	Data Popularity	72
2.17	Installing LHCbWebAppDIRAC	78
2.18	CERN centralized MySQL service	79
2.19	CERN centralized Elasticsearch service	79
2.20	Setup a new vbox	80
2.21	Admin Command Reference	80
2.22	Full Configuration Example	80
3	Certification	87
3.1	LHCbDIRAC Certification (development) Releases	87
3.2	DBs in certification	87
4	Code Documentation (master)	89

4.1	Systems	89
4.2	Other	99
5	User Guide	103
5.1	Bookkeeping Command Reference	103
5.2	Data Management Command Reference	103
5.3	Workload Management Command Reference	103
5.4	HOW-TO Guides	103
6	Documentation sources	107
7	Indices and tables	109



The **LHCbDIRAC** project is the LHCb Grid solution. LHCbDIRAC is DIRAC extension.

DIRAC forms a layer between a particular community and various compute resources to allow optimized, transparent and reliable usage. LHCbDIRAC specializes DIRAC for LHCb.

- DIRAC documentation: <http://dirac.readthedocs.io/en/latest/index.html>
- DIRAC hosted repository: <https://github.com/DIRACGrid>

LHCbDIRAC is the LHCb extension to DIRAC:

- LHCbDIRAC hosted repository: <https://gitlab.cern.ch/lhcb-dirac>

1.1 Guide for developing LHCbDIRAC (and DIRAC, for LHCb developers)

A short, but hopefully comprehensive guide on developing in LHCbDIRAC, referencing DIRAC development model. For what are DIRAC and LHCbDIRAC doing, look elsewhere.

LHCbDIRAC is a DIRAC extension. This means that LHCbDIRAC cannot leave independently from DIRAC. There are a number of DIRAC extensions, maintained by various communities worldwide, and LHCbDIRAC is the most important out there, and the one that receives the most support by DIRAC itself. But it also means that DIRAC and LHCbDIRAC (as all the other DIRAC extensions) have different [release cycles](#) and [versioning](#), adopts different [version control systems](#), use different [tracking systems](#), and that the [code conventions](#) may slightly differ.

DIRAC can also have other extensions, independent from a VO. All these are hosted at [github](#).

1.1.1 Pre-requisites

Within this section we just look at what is necessary to know before looking at the code.

Releases

Naming

Both DIRAC and LHCbDIRAC follow the same naming conventions for releases, inherited by the LHCb convention:

vMrNpt

where:

- **M** stands for *major version*, or simply *version*
- **N** stands for *minor version*, or simply *release*

- **t** stands for *patch version*, or simply *patch*

with a special *pre-release* naming convention: **-preX**.

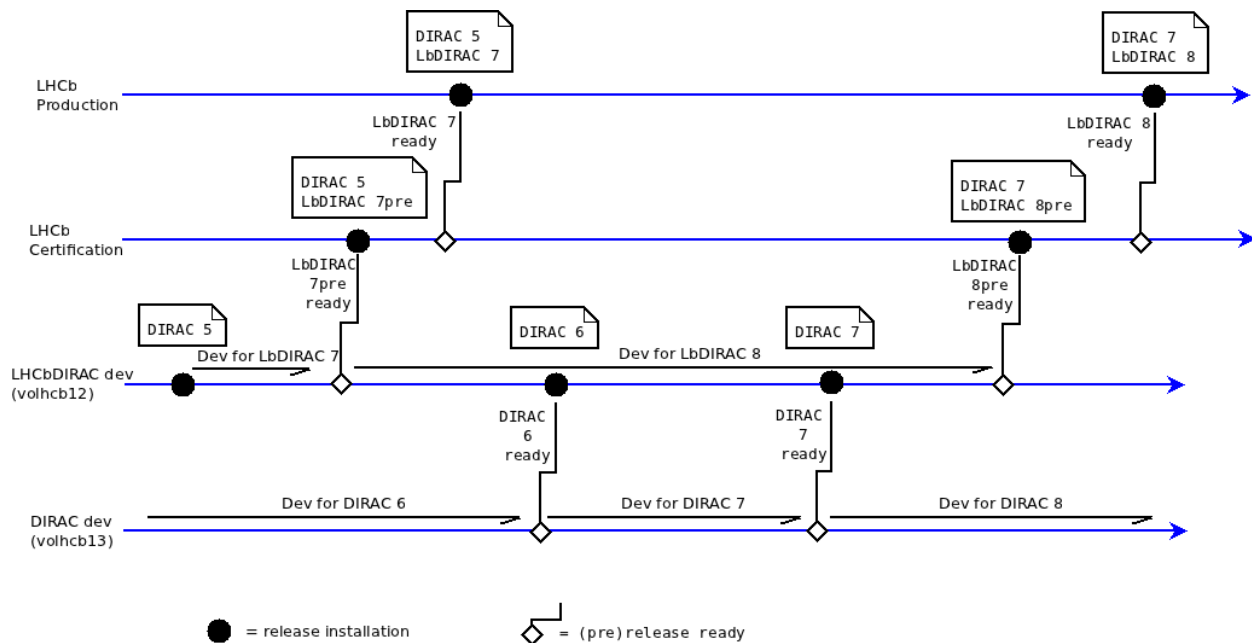
This will be clear with some examples:

- **v6r2p0** is the version 6, release 2, patch 0
- **v7r5p13** is the version 7, release 5, patch 13
- **v8r1-pre2** is the second pre-release of version 8, release 1

There are no pre-releases for patches.

Release cycle

When developing LHCbDIRAC, we need to consider that every LHCbDIRAC is developed on top of a DIRAC release. The following picture explains the model.



So, for example, there might be 2 or more LHCbDIRAC releases based on top of the same DIRAC release. Every LHCbDIRAC developers has to know which release of DIRAC its development is for. The major version of both DIRAC and LHCbDIRAC changes rarely, let's say evry 2 years. The minor version changes more frequently in LHCbDIRAC with respect to DIRAC, but there is no strict advancement scheduling for none of the 2.

A pre-release is a release candidate that goes through a certification process.

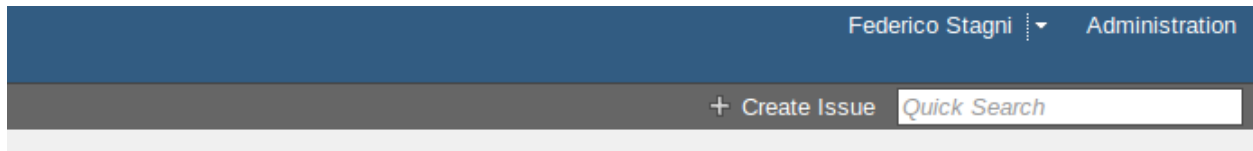
Version Control

LHCbDIRAC version control is based on GIT. GIT is a very popular distributed revision control system. The reader is supposed to be familiar with the way such systems work. The code is hosted in the [CERN GitLab](#).

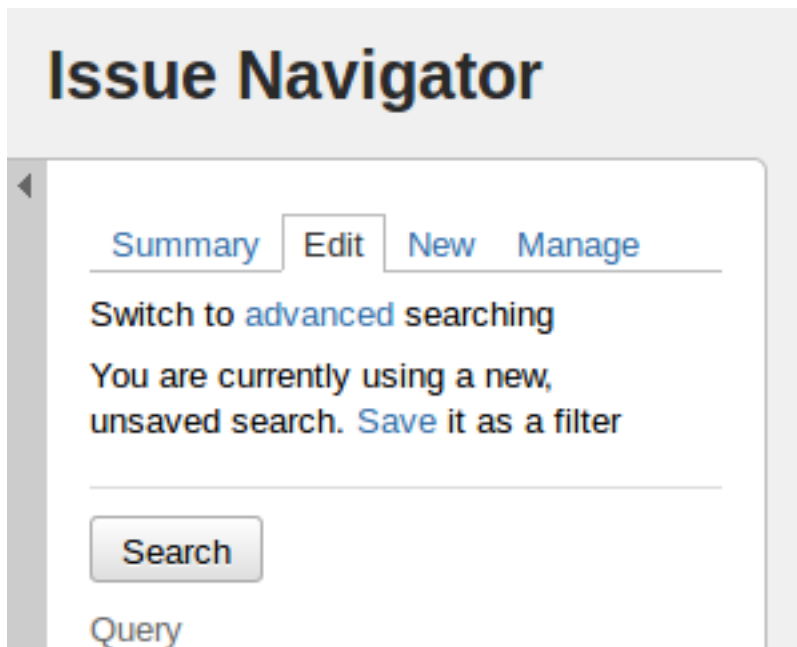
Tracking systems

The tracking system used for LHCbDIRAC is [jira](#). Jira is a fundamental tool for LHCbDIRAC, and its use is mandatory. Every development should be tracked there. Jira is a very powerful tool, but requires some time to master. Few notes/links:

- The *official documentation* is [here](#). You might also be interested in watching the first ~15 minutes of this [video](#).
- Issuing a new bug/task/story/etc. (there are many possible choices) is easy, just look at the top right of the screen:



- Remember to put a “component” when you make a new issue
- When you make a new research in the issue navigator, you can save the search: it will become useful later



Developer tools

You are free to choose the editor or IDE you prefer. If you don’t already have a preference the best starting point is [Visual Studio Code](#) which is a very popular modern editor. It has many advanced features, plugins and excellent support for developing remotely via SSH. Some other commonly used programs are [vim](#), [Emacs](#), [Sublime](#) with the [Anaconda](#) package and [PyCharm](#) ([IntelliJIDEA](#)) .

1.2 Developing DIRAC and LHCbDIRAC

Developing the code is not just about editing. You also want to “run” something, usually for testing purposes. The DIRAC way of developing can be found [here](#) and it applies also to LHCbDIRAC. Please follow carefully especially what’s [there](#)

In general, if you are developing LHCbDIRAC, you should consider that:

- everything that applies to DIRAC development, also applies to LHCbDIRAC development, so, follow carefully the links above
- every LHCbDIRAC release has a strong dependency with a DIRAC release. See <https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/blob/master/CONTRIBUTING.md> for more info.

1.3 HOW TOs

1.4 Browsing the code running in production

If you want to browse the DIRAC (and LHCbDIRAC) code running in production you'll first of all have to know which version is installed. Announcements of new deployments are done via the LHCb operations [eLog](#). The code is also always installed in the CVMFS release area (`/cvmfs/lhcb(dev).cern.ch/lhcbdirac/`) but you can normally use git to switch from one to another.

1.5 I developed something, I want it in the next release

Just open a merge request to the devel branch of LHCbDirac: all the releases (minor and major) are created branching from this branch.

1.6 Asking for a LHCbDIRAC patch

Just open a merge request to the master branch of LHCbDirac. If in a hurry, drop an e-mail to the `lhcb-dirac` mailing list.

This page is the work in progress. See more material here soon !

2.1 LHCbDIRAC Releases

The following procedure applies fully to LHCbDIRAC production releases, like patches. For pre-releases (AKA certification releases, there are some minor changes to consider).

2.1.1 Prerequisites

The release manager needs to:

- be aware of the LHCbDIRAC repository structure and branching as highlighted in the [contribution guide](#).
- have push access to the master branch of “upstream” (being part of the project “owners”)
- have DIRAC installed
- have been granted write access to <webService>
- have “lhcb_admin” or “diracAdmin” role.
- have a Proxy

The release manager of LHCbDIRAC has the triple role of:

1. creating the release
2. making basic verifications
3. deploying it in production

2.1.2 1. Creating the release

Unless otherwise specified, (patch) releases of LHCbDIRAC are usually done “on top” of the latest production release of DIRAC. The following of this guide assumes the above is true.

Releases of LHCbDIRAC are stored on the [Python Package Index \(PyPI\)](#) and created in 3 steps:

1. Merging “Merge Requests”
2. Propagating to the devel branch (for patches)
3. Creating a tag in Git and uploading it to the . This is typically handled by a manually triggered CI job.

Merging “Merge Requests”

[Merge Requests \(MR\)](#) that are targeted to the master branch and that have been approved by a reviewer are ready to be merged

Otherwise, simply click the “Accept merge request” button for each of them.

If you are making a Major release please merge devel to master follow the instruction: *1. Making a major releases.*

Propagate to the devel branch

Automatic propagation

The LHCbDIRAC repo has a webhook installed to trigger a job in [GitLab\(lhcb-dirac/sweeper\)](#) for each MR action taken on the repo. The pipeline in this repo converts all the information from GitLab hook to environment variables that are accessible for usage. For each MR that it is determined that it targets master we call the script `labelMR.py` (in `lhcb-dirac/sweeper`) to add the label `alsoTargeting:devel`. This is a hint for the subsequent procedure to cherry-pick this MR into `devel`.

The automatic cherry-picking (also referred to as sweeping), is performed in the CI job `MR_SWEEP`, which tries to cherry-pick the merged MRs into a new branch and open a new MR against `devel`. This mechanism is only triggered on MRs that have the label `alsoTargeting:devel`. Once the mechanism finished the merged MR will receive the label `sweep:done`. In the case an automatic sweep is not possible (e.g. merge conflict) the MR will receive the label `sweep:failed`, in the comment of the MR you will see a link to the job where the sweep was attempted. In the log this job you will see a hint how to reproduce the failed merge, so the you can manually resolve the problem e.g.:

```
git checkout devel
git cherry-pick -m 1 9f142d8c1
git status
```

To not overlook failed sweeps, it is advisable that you subscribe to the label `sweep:failed` in [GitLab Labels](#). If everything is successful, after merging something to master in a relatively short period of time a MR should appear that is the result of the sweep, it will have the label `sweep:from master`.

Manual propagation

Before you start doing any merging it’s good to setup the correct merge driver, you do this by adding to your `.gitconfig`

```
[merge “ours”] driver = true
```

this lets git know which files you want to ignore when you merge *master* into *devel*.

Now, you need to make sure that what’s merged in master is propagated to the devel branch. From the local fork:

```
# get the updates (this never hurts!)
git fetch upstream
# create a "newDevel" branch which from the upstream/devel branch
git checkout -b newDevel upstream/devel
# merge in newDevel the content of upstream/master
git merge upstream/master
```

The last operation may result in potential conflicts. If happens, you'll need to manually update the conflicting files (see e.g. [this guide](#)). As a general rule, prefer the master fixes to the "HEAD" (devel) fixes. Remember to add and commit once fixed.

Please fix the conflict if some files are conflicting. Do not forget to execute the following:

```
git add -A && git commit -m " message"
```

Conflicts or not, you'll need to push back to upstream:

```
# push "newDevel" to upstream/devel
git push upstream newDevel:devel
# delete your local newDevel
git checkout upstream/devel
git branch -d newDevel
# keep your repo up-to-date
git fetch upstream
```

Create/Trigger release

To create a release you go to the pipelines <https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/-/pipelines> and go the last pipeline of the branch you want to tag. At the end of the pipeline there is a manual trigger job with name *make_tag*, you click on it and you will get the following windows

LHCb Dirac > LHCbDIRAC > Jobs > #18289626

ⓘ manual Job make_tag (set VERSION to override) created 37 minutes ago by Chris Burr

This job requires a manual action

This job requires manual intervention to start. Before starting this job, you can add variables below for last-minute configuration changes.

Key	Value
VERSION	v10.4.0

Input variable key Input variable value

Specify variable values to be used in this run. The values specified in CI/CD settings will be used as default

Trigger this manual action

If you don't specify any version (with *VERSION*) only the LHCbDIRAC patch/alpha/beta/rc version will be increased for +1. This will create the release notes and tag.

Automatic procedure

When a new git tag is pushed to the repository, a gitlab-ci job takes care of testing, creating the sdist/bdist, uploading it to PyPI, and to build the docker image. You can check it in the pipeline page of the repository (<https://gitlab.cern.ch/lhcb-dirac/LHCbDIRAC/pipelines>).

It may happen that the pipeline fails. Often it is just a glitch on the runner side, if so just restart the job from the pipeline web interface. **If any of the pipelines fails, don't try to do a release manually but rather investigate why.**

2.1.3 2. Make an LHCbWebDIRAC release (if required)

LHCbWebDIRAC releases are made by simply checking the last CI pipeline was successful and then creating a tag using the GitLab web interface. GitLab CI should then test and upload this tag to PyPI.

2.1.4 3. Advertise the new release

Before you start the release you must write an Elog entry 1 hour before you start the deployment. You have to select Production and Release tick boxes.

When the intervention is over you must notify the users (reply to the Elog message).

2.1.5 4. Deploying the release

Deploying a release means deploying it for the various installations:

```
* client
* server
* pilot
```

Release for client

Releases are automatically installed on to cvmfs with the `deploy_on_cvmfs` GitLab CI job. In addition, there is a manual GitLab CI job called `set_cvmfs_prod_link`, this sets the production version link to the current deployed version. If a roll-back is required to a previous version, this job can be re-tried from an older release pipeline to re-trigger the job. A overview of the current installation health can be found at [here](#).

Note: It is normal for these jobs to show the RETRY status.

Server

Method 1 (preferred): inside the tag pipeline

The CI pipeline associate the tag pipeline has a manual job `update_instance` which you have to trigger. This will automatically apply the release to all machines that that constitute the respective instance. In the case of normal tags this is the production instance and in the case of alpha/beta/release candidate tag this is the certification instance. The update is based on the `dirac` command `dirac-admin-update-instance`. The same job will also update the pilot version via `dirac-admin-update-pilot`.

Method 2: web portal

Using the web portal:

- You cannot do all the machines at once. Select a bunch of them (between 5 and 10). Fill in the version number and click update.
- Repeat until you have them all.
- Start again selecting them by block, but this time, click on “restart” to restart the components.

Method 3: interactive via sysadmin cli

To install it on the VOBOXes from lxplus:

```
lhcb-proxy-init -g lhcb_admin
dirac-admin-sysadmin-cli --host lbvoboxXYZ.cern.ch
> update LHCbDIRAC v10.4.2
> restart *
```

Pilot

Update the pilot version from the CS, keeping 2 pilot versions, for example:

```
/Operation/LHCb-Production/Pilot/Version = v10.4.2, v10.4.1
```

The newer version should be the first in the list

for checking and updating the pilot version. Note that you'll need a proxy that can write in the CS (i.e. lhcb-admin). This script will make sure that the pilot version is update BOTH in the CS and in the json file used by pilots started in the vacuum.

2.1.6 1. Making a major releases

Making a major release requires manual intervention to update the package dependencies by editing the *setup.cfg* files of each package.

- We typically constrain LHCb(Web)DIRAC to a specific minor release series
- LHCbDIRAC has the dependency on DIRAC
- LHCbWebDIRAC has the dependency on LHCbDIRAC
- To update LHCbWebDIRAC this there will need to be a compatible (pre)release of LHCbDIRAC available first

The procedure is the same for LHCbWebDIRAC and LHCbDIRAC: you basically want `devel` to become the new `master`. In order to do so:

```
# Start from the master branch
git checkout upstream/master

# Create a commit that has two parents (master and devel) to preserve the history
# but only take the files of the parents (i.e. no changes to the files)
# and do not create the commit just yet
# (https://www.atlassian.com/git/tutorials/using-branches/merge-strategy)
git merge -s ours --no-commit upstream/devel

# Remove all the files present
git rm -r .

# Checkout all the files that are in the devel branch (notice the dot at the end)
git checkout upstream/devel .

# finish the merging
git merge --continue

# push back
git push upstream/master
```

You can then trigger the release pipeline normally, specifying a version such as `v10.4.0`.

Note about the git procedure

Ultimately, the aim is that `devel` becomes our new `master`. However, bluntly doing this would resoled in conflicts for everybody that has a clone of the LHCbDIRAC repo. Thus, this idea of doing a merge with the `-s ours` strategy. This allows to create a commit that has as parent both `master` and `devel`, and avoid any forced push. The rest of the procedure (`rm/checkout`) is to make sure that we do not keep files that were meant to be in `master` only.

2.2 Renewal of certificate for ONLINE machine

Login as `lhcbprod` on `lbdirac.cern.ch` and generate the certificate request


```
openssl req -new -subj /CN=lbdirc.cern.ch -out newcsr.csr -nodes -sha1
```

Open in your browser the page <http://ca.cern.ch> cut the content of *newcsr.csr* (created in the previous step) in the web page and click on the submit button. Save the Base 64 encoded certificate as a file *newcert.cer*. Copy this file to *lbdirc.cern.ch*. Then convert the certificate in the correct format.

```
openssl pkcs12 -export -inkey privkey.pem -in newcert.cer -out myCertificate.pks (You
↪ will have to type the PEM password you typed in the previous step. Type also an
↪ export password, and don't forget it. Your certificate in PKCS12 format is ready in
↪ file myCertificate.pks, you can delete the other files.)
openssl pkcs12 -in myCertificate.pks -clcerts -nokeys -out hostcert.pem
openssl pkcs12 -in myCertificate.pks -nocerts -out hostkey.pem.passwd
openssl rsa -in hostkey.pem.passwd -out hostkey.pem (remove the password)
```

If you want to test that the new host certificate is valid without any password, just do

```
dirac-proxy-init -C <cert> -K <key>
```

2.3 Pit export

2.3.1 Installation of LHCbDirac

The machine running the transfers from the pit is *lbdirc*, and is in the online network. This machine runs:

- A complete RMS: ReqManager (url: RequestManagement/onlineGateway), a ReqProxy (known only from inside) and a RequestExecutingAgent
- The RAWIntegrity system: the RAWIntegrityHandler and RAWIntegrityAgent

A special catalog is defined in the local configuration in order to keep track of the files transferred:

```
RAWIntegrity
{
  AccessType = Read-Write
  Status = Active
}
```

We also have two special configuration for StorageElements:

```
# Setting it to NULL to transfer without
# checking the checksum, since it is already done by
# the DataMover and the RAWIntegrityAgent
# It should avoid the double read on the local disk
ChecksumType=NULL
# Setting this to True is dangerous...
# If we have a SRM_FILE_BUSY, we remove the file
# But we have enough safety net for the transfers from the pit
SRMBusyFilesExist = True
```

Finally, you need to overwrite the URLs of the RMS to make sure that they use the internal RMS:

```
URLs
{
  ReqManager = dips://lbdirc.cern.ch:9140/RequestManagement/ReqManager
```

(continues on next page)

(continued from previous page)

```
ReqProxyURLs = dips://lbdirac.cern.ch:9161/RequestManagement/ReqProxy
}
```

2.3.2 Installation/update of LHCbDirac version

Instructions to install or update a new version of LHCbDirac

```
ssh lhcbprod@lbw.cern.ch
ssh store06
cd /sw/dirac/run2
source /sw/dirac/run2/bashrc
dirac-install -v -r vArBpC -t server -l LHCb -e LHCb
rm /sw/dirac/run2/pro ; ln -s versions/vArBpC_hhhhhh pro
cd /sw/dirac/run2/pro

foreach i (`ls`)
if -l $i then
  echo $i
  rm $i
  ln -s ../../$i
endif
end
```

2.3.3 Workflow

The DataMover is the Online code responsible for the interaction with the BKK (register the run, the files, set the replica flag), to request the physical transfers, and to remove the file of the Online storage when properly transferred.

The doc is visible here: https://lbdokuwiki.cern.ch/online_user:rundb_onlinetooftlinedataflow

The DataMover registers the Run and the files it already knows about in the BKK. Then it creates for each file a request with a PutAndRegister operation. The target SE is CERN-RAW, the Catalog is RAWIntegrity. The RequestExecutingAgent will execute the copy from the local online storage to CERN-RAW, and register it in the RAWIntegrity DB.

The RAWIntegrityAgent looks at all the files in the DB that are in status 'Active'.

For each of them, it will check if the file is already on tape, and if so, compare the checksum.

If the checksum is incorrect, the file remains in status 'Active', and will require manual intervention. If the checksum is correct, we attempt to register the file in the DFC only.

If the registration fails, the file goes into 'Copied' status in the DB, c If the registration works, we attempt to remove the file from the Online storage. This removal Request sends a signal to the DataMover, which will mark the file for removal (garbage collection), and the replica flag to yes in the BKK.

If the removal fails, the file status is set to 'Registered' in the DB, and will be reattempted from there at the next loop. If the removal works, the file is set to 'Done' in the DB.

2.4 Online DIRAC

This page details how the HLT farm is used to run DIRAC

The linux account used is *lhcbprod*, and all the necessary files are under */home/lhcbprod/production/*.

PVSS will start the script `/home/lhcbprod/production/launch_agent.sh`. This script mostly creates the working directory under `/localdisk1`, sets up a bit of the environment, and start the `/home/lhcbprod/production/dirac-pilot-3.sh` script.

The code for the Pilot itself is store in `/home/lhcbprod/production/Pilot3`. The content of this directory is updated every hour by a cron run on the machine cron02 as lhcbprod.

The script which updates it is `/home/lhcbprod/production/dirac-pilot3-cron.sh`.

2.5 Renewal of the host certificate

The certificate used to bootstrap the jobs is the same host certificate that is used to run the pit export installation, and is linked to the `lbdirac.cern.ch` alias.

Everyone in the `LHCB.PC-ADMINS` egroup should be able to renew it. Reminders will come many times before it expires.

Procedure:

- go to the [cern ca website](#)
- use the tab *New Grid Host Certificate*, and click *Automatic certificate generation*. Do not use any passphrase. This will give you a p12 file
- convert the p12 file in two PEM files: one for the host certificate, one for the private key. The details commands are given [here](#)
- put these two files (`hostcert.pem` and `hostkey.pem`) under `/home/lhcbprod/production/etc` (note: this location is the one used in `launch_agent.sh`)

Since the same certificate is used for the pit export, you also may want to update these. Just copy the two files under `/sw/dirac/run2/etc/grid-security/`

2.6 Troubleshooting

The online PVSS panel can be accessed via `/group/online/ecs/Shortcuts315/OTHER/ONLDIRAC/ONLDIRAC_UI_FSM.sh`

2.7 Bookkeeping

2.7.1 LHCb Bookkeeping database administration

This document contains all the information needed to manage the Bookkeeping Oracle database.

Login to the database

How-To in [lbDevOps doc](#).

Ticketing, contacts for IT-DB Oracle support

You can send a mail to <mailto:phydb.support@cern.ch> mailing list. It will be converted to a ticket.

Otherwise, directly in [service now](#).

There's also a [mattermost channel](#).

Compile oracle stored procedure

In order to compile the stored procedure you need the [sql file](#). Make sure that you are using the correct version.

1. Login the database (e.g. using `sqlplus`, but also with `sqldeveloper` or `sqlcl`)
2. in the terminal execute `@/path/to/oracle_schema_storedprocedures.sql`
3. commit;

In case of error you have to use 'show errors' command

In case of a schema change, you can find the command that needs to be executed in this sql file [oracle_schema_commands.sql](#)

Discover slow queries in the db

Note: If you are not familiar with Oracle, it is better to send a mail to <mailto:phydb.support@cern.ch> mailing list. You can write we have problem with the database and that the queries are very slow. IT/DB expert will find the slow queries and will probably tell what is the problem and try to solve.

The <https://cern.ch/session-manager> is a portal provided by IT/DB where you can logon and find the running queries. You can find the query which is running very long, get the execution plan and also can take the query and run it in `sqlplus`. So you can compare the execution plan which is in the web and in `sqlplus`.

When you login to the session manager, you have a reader and a writer account. All the select queries are running with the reader account.

```
Login form:
UserName: LHCb_DIRACBOOKKEEPING_users
Password: pass
Database: LHCBR
```

You can also check for slow queries in oem.cern.ch.

You can check the execution plan in the following way, after having logged in with the main owner account:

1. set autot traceo
2. set timing on
3. set linesize 30000
4. set pagesize 3000
5. execute the qurey

When the query will finish you will have the execution plan and you will have the real execution time as well. I propose to look the following parameters:

```
Cost (%CPU) , consistent gets, physical reads
```

For example:

```

Elapsed: 00:00:00.12
Execution Plan
-----
Plan hash value: 3340191443
-----
↪-----
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
↪Pstart| Pstop |
-----
↪-----
| 0 | SELECT STATEMENT | | 4960 | 2232K | 21217 (1) | 00:00:01 |
↪|
| 1 | NESTED LOOPS | | | | | | |
| 2 | NESTED LOOPS | | 4960 | 2232K | 21217 (1) | 00:00:01 |
↪|
| 3 | PARTITION RANGE ALL | | 4897 | 1219K | 1619 (1) |
↪00:00:01 | 1 | 20 |
| 4 | TABLE ACCESS BY LOCAL INDEX ROWID | JOBS | 4897 | 1219K | 1619 |
↪(1) | 00:00:01 | 1 | 20 |
| * 5 | INDEX RANGE SCAN | PROD_CONFIG | 4897 | | 88 (0) |
↪00:00:01 | 1 | 20 |
| 6 | PARTITION RANGE ITERATOR | | 1 | | 3 (0) |
↪00:00:01 | KEY | KEY |
| * 7 | INDEX RANGE SCAN | JOBS_REP_VIS | 1 | | 3 (0) |
↪00:00:01 | KEY | KEY |
| 8 | TABLE ACCESS BY LOCAL INDEX ROWID | FILES | 1 | 206 | 4 (0) |
↪00:00:01 | 1 | 1 |
-----
↪-----
Predicate Information (identified by operation id):
-----
5 - access("J"."PRODUCTION"=51073)
7 - access("J"."JOBID"="F"."JOBID" AND "F"."GOTREPLICA"='Yes')
Statistics
-----
46 recursive calls
0 db block gets
508 consistent gets
46 physical reads
1452 redo size
56603 bytes sent via SQL*Net to client
640 bytes received via SQL*Net from client
10 SQL*Net roundtrips to/from client
1 sorts (memory)
0 sorts (disk)
131 rows processed

```

Problems: * the cost is a big number. * the consistent gets is very high * physical reads are very high

Notes:

- You may have queries which need to read a lot of data. In this case the consistent gets and physical reads are very high numbers. In that example if the consistent gets and physical reads are very high (for example more than 10k) we have a problem. This is because the query only returned 131 rows.
- TABLE ACCESS FULL is not good. You have to make sure that the query uses an index. This is not always true.
- parallel execution: you have to make sure that the query is running parallel, the processes does not send to much

data between each other. If you run a query parallel and the consistent gets is very high then you have a problem. Contact to oracle IT/DB if you do not know what to do. . .

- CARTESIAN join: If you see that word in the execution plan, the query is wrong.

2.7.2 Steps in the Bookkeeping database

Steps are used to process/produce data. The steps are used by the Production Management system and work flow. The steps are stored in the steps table which has the following columns:

```
STEPID
STEPNAME
APPLICATIONNAME
APPLICATIONVERSION
OPTIONFILES
DDDB
CONDDB
EXTRAPACKAGES
INSERTTIMESTAMPS
VISIBLE
INPUTFILETYPES
OUTPUTFILETYPES
PROCESSINGPASS
USABLE
DQTAG
OPTIONSFORMAT
ISMULTICORE
SYSTEMCONFIG
MCTCK
```

The steps table has 3 triggers:

```
STEP_INSERT: This trigger is used to replace NULL, None to an empty string.
steps_before_insert: It checks that the processing pass contains a '/'.
step_update: The steps which are already used can not be modified.
```

Modifying steps

We may want to modify an already used steps. A step can be modified if the trigger is disabled. The following commands has to be performed in order to modify a step:

```
ALTER TRIGGER step_update disable;
UPDATE steps SET stepname='Recol6Smog for 2015 pA', processingpass='Recol6Smog' WHERE
↪stepid=129609; --an alternative is used by the StepManager page
ALTER TRIGGER step_update enable;
```

2.7.3 Processing pass in the Bookkeeping

The processing pass is a collection of steps. The processing pass is stored in the processing table:

```
ID
ParentID
Name
```

The following example illustrates how to create a step:

```
SELECT max(id)+1 FROM processing;
SELECT * FROM processing where name='Real Data';
insert into processing(id,parentid, name) values (1915,12, 'Reco16Smog');
```

In this example we have created the following processing pass: /Real Data/Reco16Smog

The following query can be used to check the step:

```
SELECT * FROM (SELECT distinct SYS_CONNECT_BY_PATH(name, '/') Path, id ID
FROM processing v START WITH id in (SELECT distinct id FROM processing where
↳ name='Real Data')
CONNECT BY NOCYCLE PRIOR id=parentid) v where v.path='/Real Data/Reco16Smog';
```

If we know the processing id, we can use the following query to found out the processing pass:

```
SELECT v.id,v.path FROM (SELECT distinct LEVEL-1 Pathlen, SYS_CONNECT_BY_PATH(name,
↳ '/') Path, id
FROM processing
WHERE LEVEL > 0 and id=1915
CONNECT BY PRIOR id=parentid order by Pathlen desc) v where rownum<=1;
```

2.7.4 Bookkeeping down time

The following services/agent needs to be stopped before the deep down time (SystemAdministrator can be used in order to manage the services):

```
RMS:
  RequestExecutingAgent
    check it really stops (may take long time)
TS:
  BookkeepingWatchAgent
  TransformationAgent - Reco, DM, MergePlus (this to be checked). This was not
↳ stopped the latest deep downtime
  TransformationCleaningAgent
  MCSimulationTestingAgent
PMS:
  ProductionStatusAgent
  RequestTrackingAgent
DMS:
  PopularityAgent
  StorageHistoryAgents(s)
```

Just before the intervention stop all Bookkeeping services.

2.7.5 Automatic updating of the productionoutputfiles

Create an oracle periodic job:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
    job_name      => 'produpdatejob',
    job_type      => 'PLSQL_BLOCK',
    job_action    => 'BEGIN BKUTILITIES.updateProdOutputFiles(); END; ',
```

(continues on next page)

(continued from previous page)

```

repeat_interval      => 'FREQ=MINUTELY; interval=10',
start_date           => systimestamp,
enabled              => TRUE
);
END;
/

```

For monitoring:

```

select JOB_NAME, STATE, LAST_START_DATE, LAST_RUN_DURATION, NEXT_RUN_DATE, RUN_COUNT,
↪FAILURE_COUNT from USER_SCHEDULER_JOBS;

```

Debugging the produpdatejob in case of failure:

- sqlplus LHCb_DIRACBOOKKEEPING/xxxxx@LHCb_DIRACBOOKKEEPING
- set serveroutput on
- exec BKUTILITIES.updateProdOutputFiles();

You will see the problematic production, which you will need to fix. For example: If the production is 22719, you can use the following queries for debug:

```

SELECT j.production, J.STEPIID, f.eventtypeid, f.filetypeid, f.gotreplica, f.
↪visibilityflag
FROM jobs j, files f WHERE
j.jobid = f.jobid AND
j.production=22719 and
f.gotreplica IS NOT NULL and
f.filetypeid NOT IN(9,17) GROUP BY j.production, J.STEPIID, f.eventtypeid, f.
↪filetypeid, f.gotreplica, f.visibilityflag Order by f.gotreplica,f.visibilityflag
↪asc;

select * from files f, jobs j where
j.jobid = f.jobid AND
j.production=22719 and
f.gotreplica IS NOT NULL and
f.eventtypeid is NULL and
f.filetypeid NOT IN(9,17);

update files set eventtypeid=90000000 where fileid in (select f.fileid from files f,
↪jobs j where j.jobid = f.jobid AND
j.production=22719 and
f.gotreplica IS NOT NULL and
f.eventtypeid is NULL and
f.filetypeid NOT IN(9,17));

commit;

```

2.7.6 Automatic updating of the prodrunview

Create an oracle periodic job:

```

BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name           => 'prodrunupdatejob',

```

(continues on next page)

(continued from previous page)

```

job_type          => 'PLSQL_BLOCK',
job_action        => 'BEGIN BKUTILITIES.updateprodrunview(); END;',
repeat_interval   => 'FREQ=MINUTELY; interval=20',
start_date        => systimestamp,
enabled           => TRUE
);
END;
/

```

For monitoring:

```

select JOB_NAME, STATE, LAST_START_DATE, LAST_RUN_DURATION, NEXT_RUN_DATE, RUN_COUNT,
↪FAILURE_COUNT from USER_SCHEDULER_JOBS;

```

Debugging the produpdatejob in case of failure:

- sqlplus LHCb_DIRACBOOKKEEPING/xxxxx@LHCb_DIRACBOOKKEEPING
- set serveroutput on
- exec BKUTILITIES.updateprodrunview();

You will see the problematic production, which you will need to fix.

2.7.7 Managing partitions

jobs and *files* tables are partitioned. *jobs* table RANGE-HASH partitioned by *production* and *configurationid*. *files* table RANGE partitioned by *jobid*.

jobs table partitions

The last partitions called *prodlast* and *runlast*. The maximum value of the *prodlast* partition is MAXVALUE. This may require to split, if we see performance degradation. This can happen if too many rows (*jobs*) belong to this partition. The recommended way to split the partition is to declare a down time, because when the partition split then the non partitioned indexes become invalid. The non partitioned indexes need to be recreated, which will block writing to the DB. The procedure for splitting the *prodlast* partition:

```

select max(production) from jobs PARTITION(prodlast) where production!=99998;
ALTER TABLE jobs SPLIT PARTITION prodlast AT (xxxxx) INTO (PARTITION prodXXX,
↪PARTITION prodlast);

```

One of the possibilities is to split the *prodlast* using the last production, which can be retrieved using the following query above. *xxxxx* is the result of the query. *prodXXX* is the last partition+1. For example:

```

select max(production) from jobs PARTITION(prodlast) where production!=99998;

```

which result is 83013

```

ALTER TABLE jobs SPLIT PARTITION prodlast AT (83013) INTO (PARTITION prod4, PARTITION
↪prodlast);

```

Rebuild the non partitioned indexes:

```

ALTER INDEX SYS_C00302478 REBUILD;
ALTER INDEX JOB_NAME_UNIQUE REBUILD;

```

files table partitions

This table is RANGE partitioned by *jobid*, which can reach the maximum value of the existing partition. If this happens, the following error will appear:

```
2018-07-30 01:12:00 UTC dirac-jobexec/UploadOutputData ERROR: Could not send_
↳Bookkeeping XML file to server:
Unable to create file /lhcb/MC/2015/SIM/00075280/0000/00075280_00009971_1.sim !_
↳ERROR: Execution failed.: (
ORA-14400: inserted partition key does not map to any partition
ORA-06512: at "LHCB_DIRACBOOKKEEPING.BOOKKEEPINGORACLEDB", line 976
ORA-06512: at line 1
```

In order to fix the issue a new partition has to be created:

```
alter table files add PARTITION SECT_0620M VALUES LESS THAN (620000000);
```

2.7.8 Database monitoring

Various queries are available in the BookkeepingSystem/DB/monitoring.sql file. They can be used for discovering problems such as database locks, broken oracle jobs, sessions, used indexes, etc.

2.7.9 ProductionOutputFiles table

The table contains data used for speeding up some queries. This table contains aggregated data used by the BkQuery. The queries are very fast because it does not require join of the two main tables *files* and *jobs*. The table is filled when a production is created. The addProduction method adds all necessary info for this table. The table contains the following columns:

```
Production
Stepid
EventtypeId
FileTypeId
Visible
GotReplica
```

The *visible* and *GotReplica* columns can be changed when all files are removed from a production, or when a file is set to invisible (i.e. when the files are archived, they are not supposed to be used by the users.) be used by the users). In order to update this table, the following stored procedure is used:

```
BKUTILITIES.updateProdOutputFiles
```

This method updates the productions with *visible* and *gotreplica* flags that have changed in the last 3 days. The procedure is run by an Oracle job. It is scheduled every 10 minutes. More details in the [LHCb Bookkeeping database administration](#) document.

2.7.10 Fill ProductionOutputFiles table

The productionoutputfiles table is used for removing the materialized views (MV). It is introduced June 2017. This document describes about how the table propagated with some meaningful data.

Note: This document can be useful if we want to know what changes applied to the db. Before we created a table for keeping track about the migration.

```
create table prods as select distinct production from jobs where production not in_
↪(select production from productionoutputfiles);
```

- This table contains all productions, which are not in the productionoutputfiles table. The productions which are entered already in this table

```
alter table prods add processed char(1) default 'N' null;
```

- In order to keep which productions are inserted to the productionoutputfiles table.

```
alter table prods add stepid char(1) default 'N' null;
```

- The all jobs which belong to this prod does not have stepid.

```
alter table prods add problematic char(1) default 'N' null;
```

- Duplicated steps in the stepscontainer table.

The migration started with the runs (production<0) and with the prods.processed='N':

```
declare
begin
FOR stcont in (select distinct ss.production from stepscontainer ss where ss.
↪production in (select p.production from prods p where p.processed='N' and p.
↪production<0)) LOOP
    DBMS_OUTPUT.put_line (stcont.production);
    FOR st in (select s.stepid, step from steps s, stepscontainer st where st.stepid=s.
↪stepid and st.production=stcont.production order by step) LOOP
        FOR f in (select distinct j.stepid,ft.name, f.eventtypeid, ft.filetypeid, f.
↪visibilityflag from jobs j, files f, filetypes ft
                    where ft.filetypeid=f.filetypeid and f.jobid=j.jobid and
                    j.production=stcont.production and j.stepid=st.stepid and f.
↪filetypeid not in (9,17) and eventtypeid is not null) LOOP
            DBMS_OUTPUT.put_line (stcont.production||'->'||st.stepid||'->'||f.filetypeid||'-
↪>'||f.visibilityflag||'->'||f.eventtypeid);
            BOOKKEEPINGORACLEDB.insertProdOutputFtypes(stcont.production, st.stepid, f.
↪filetypeid, f.visibilityflag,f.eventtypeid);
            update prods set processed='Y' where production=stcont.production;
        END LOOP;
    END LOOP;
    commit;
END LOOP;
END;
/
END LOOP;
END;
/
```

After I have noticed we have jobs without stepid. In order to fix this issue executed the following commands:

```
create table stepscontainer_2018_09_20 as select * from stepscontainer;
```

- this is used for backup, because the duplicated entries will be deleted...

To fill the stepid for the non processed runs:

```
declare
found number;
```

(continues on next page)

(continued from previous page)

```

pname varchar2(256);
prversion varchar2(256);
prev_name varchar2(256);
prev_version varchar2(256);
rep number;
begin
FOR stcont in (select p.production from prods p where p.processed='N' and p.production
↳<0) LOOP
    found:=0;
    select count(*) into found from jobs where production=stcont.production and stepid_
↳is null;
    if found>0 then
        prev_name:=null;
        prev_version:=null;
        for sts in (select stepid, step from stepscontainer where production=stcont.
↳production order by step) LOOP
            DBMS_OUTPUT.put_line ('Stepid'||sts.stepid||'Prod'||stcont.production);
            select applicationname, applicationversion into pname,prversion from steps_
↳where stepid=sts.stepid;
            if prev_name is null and prev_version is null then
                prev_name:=pname;
                prev_version:=prversion;
                --DBMS_OUTPUT.put_line ('Update:'|| stcont.production);
                update jobs set stepid=sts.stepid where programname=pname and_
↳programversion=prversion and production=stcont.production;
                update prods set stepid='Y' where production=stcont.production;
            elsif prev_name=pname and prev_version=prversion then
                DBMS_OUTPUT.put_line ('Problematic:'|| stcont.production);
                delete stepscontainer where production=stcont.production and stepid=sts.
↳stepid;
                update prods set problematic='Y' where production=stcont.production;
            else
                --DBMS_OUTPUT.put_line ('Update:'|| stcont.production);
                update jobs set stepid=sts.stepid where programname=pname and_
↳programversion=prversion and production=stcont.production;
                update prods set stepid='Y' where production=stcont.production;
                prev_name:=pname;
                prev_version:=prversion;
            END if;
        END LOOP
        commit;
    END if;
END LOOP;
END;
/

```

After executing this procedure 21309 productions are fixed:

```
select count(*) from prods where stepid='Y' and production<0;
```

Now we can add these productions to the productionoutputfiles table:

Check how many runs are processed:

```
select count(*) from prods where processed='Y' and production<0;
```

the result is 14026 Check all the runs which are not processed:

```
select count(*) from prods where stepid='Y' and processed='N' and production
<0; result is 21308
```

Note: 21309!=21308 because I did a test before executing the procedure.

```
declare
begin
FOR stcont in (select distinct ss.production from stepscontainer ss where ss.
production in (select p.production from prods p where stepid='Y' and p.processed='N'
and p.production<0)) LOOP
  DBMS_OUTPUT.put_line (stcont.production);
  FOR st in (select s.stepid, step from steps s, stepscontainer st where st.stepid=s.
stepid and st.production=stcont.production order by step) LOOP
    FOR f in (select distinct j.stepid,ft.name, f.eventtypeid, ft.filetypeid, f.
visibilityflag from jobs j, files f, filetypes ft
              where ft.filetypeid=f.filetypeid and f.jobid=j.jobid and
j.production=stcont.production and j.stepid=st.stepid and f.
filetypeid not in (9,17) and eventtypeid is not null) LOOP
      DBMS_OUTPUT.put_line (stcont.production||'-'||st.stepid||'-'||f.filetypeid||'-'
||f.visibilityflag||'-'||f.eventtypeid);
      BOOKKEEPINGORACLEDB.insertProdnOutputFtypes(stcont.production, st.stepid, f.
filetypeid, f.visibilityflag,f.eventtypeid);
      update prods set processed='Y' where production=stcont.production;
    END LOOP;
  END LOOP;
  commit;
END LOOP;
END;
/
END LOOP;
END;
/
```

```
select count(*) from prods where stepid='Y' and processed='N' and production<0;
```

the result is 260. Checking one of the production -22595: this run does not has associated files.

The following script is used to fix the 260 problematic runs:

```
DECLARE
nbfiles number;
BEGIN
for prod in (select production from prods where stepid='Y' and processed='N' and
production<0)
LOOP
  select count(*) into nbfiles from jobs j, files f where j.jobid=f.jobid and j.
production=prod.production and j.production<0;
  if nbfiles = 0 then
    DBMS_OUTPUT.put_line ('DELETE:'|| prod.production);
    delete runstatus where runnumber=-1 * prod.production;
    delete jobs where production<0 and production=prod.production;
    delete productionscontainer where production=prod.production;
    delete stepscontainer where production=prod.production;
    update prods set processed='Y' where production=prod.production;
    commit;
  END IF;
END LOOP;
```

(continues on next page)

(continued from previous page)

```
END;
/
```

After checking the result:

```
SQL> select production from prods where stepid='Y' and processed='N' and production<0;

PRODUCTION
-----
-9
```

After this fix we check how many runs are not in the productionoutputfiles table:

```
SQL> select count(*) from prods p where p.processed='N' and p.production<0;

COUNT(*)
-----
155
```

After checking the runs, we noticed the stepid is okay, but the runs do not have any files. For fixing:

```
DECLARE
nbfiles number;
BEGIN
for prod in (select production from prods where processed='N' and production<0)
LOOP
    select count(*) into nbfiles from jobs j, files f where j.jobid=f.jobid and j.
    ↪production=prod.production and j.production<0;
    if nbfiles = 0 then
        DBMS_OUTPUT.put_line ('DELETE:|| prod.production);
        delete runstatus where runnumber=-1 * prod.production;
        delete jobs where production<0 and production=prod.production;
        delete productionscontainer where production=prod.production;
        delete stepscontainer where production=prod.production;
        update prods set processed='Y' where production=prod.production;
        commit;
    END IF;
END LOOP;
END;
/
```

We can check how many runs are remained:

```
SQL> select * from prods p where p.processed='N' and p.production<0;

PRODUCTION P S P
-----
-42854 N N N
-9 N Y N
```

-9 can be deleted:

```
SQL> select count(*) from jobs j, files f where j.jobid=f.jobid and j.production=-9
    ↪and f.gotreplica='Yes';

COUNT(*)
```

(continues on next page)

(continued from previous page)

```
-----
0
```

The runs are almost fixed:

```
SQL> select * from prods p where p.processed='N' and p.production<0;
```

```
PRODUCTION P S P
```

```
-----
-42854 N N N
```

Fixing the productions which are not in the stepscontainer:

```
declare
stepid number;
stnum number;
begin
for prod in (select p.production from prods p where p.processed='N' and p.production>
  ↳0 and p.production not in (select distinct ss.production from stepscontainer ss))
LOOP
  stnum:=0;
  FOR jprod in (select j.programName, j.programVersion, f.filetypeid, ft.name, f.
  ↳visibilityflag, f.eventtypeid from jobs j, files f, filetypes ft where ft.
  ↳filetypeid=f.filetypeid and j.jobid=f.jobid and j.production=prod.production and j.
  ↳stepid is null and f.filetypeid not in (9,17) and f.eventtypeid is not null group
  ↳by j.programName, j.programVersion, f.filetypeid, ft.name, f.visibilityflag, f.
  ↳eventtypeid
    Order by( CASE j.PROGRAMNAME WHEN 'Gauss' THEN '1' WHEN 'Boole' THEN '2' WHEN
  ↳'Moore' THEN '3' WHEN 'Brunel' THEN '4' WHEN 'Davinci' THEN '5' WHEN 'LHCb' THEN '6
  ↳' ELSE '7' END))
  LOOP
    stnum:=stnum+1;
    DBMS_OUTPUT.put_line ('Production:'||prod.production||' applicationname:'||
  ↳jprod.programname||' APPLICATIONVERSION:'||jprod.programversion||stnum);
    select count(*) into stepid from steps s, table(s.outputfiletypes) o where s.
  ↳applicationname=jprod.programname and s.APPLICATIONVERSION=jprod.programversion and
  ↳o.name=jprod.name and o.visible=jprod.visibilityflag and ROWNUM<2;
    if stepid>0 then
      select s.STEPID into stepid from steps s, table(s.outputfiletypes) o where s.
  ↳applicationname=jprod.programname and s.APPLICATIONVERSION=jprod.programversion and
  ↳o.name=jprod.name and o.visible=jprod.visibilityflag and ROWNUM<2;
      --DBMS_OUTPUT.put_line ('Stepid:'|| stepid);
      BOOKKEEPINGORACLEDB.insertProdOutputFtypes(prod.production, stepid, jprod.
  ↳filetypeid, jprod.visibilityflag, jprod.eventtypeid);
      update prods set processed='Y', stepid='Y' where production=prod.production;
      update jobs j set j.stepid=stepid where j.production=prod.production and j.
  ↳programname=jprod.programname and j.programversion=jprod.programversion;
      BOOKKEEPINGORACLEDB.insertStepsContainer (prod.production,stepid,stnum);
    else
      select count(*) into stepid from steps s, table(s.outputfiletypes) o where s.
  ↳applicationname=jprod.programname and s.APPLICATIONVERSION=jprod.programversion and
  ↳o.name=jprod.name and ROWNUM<2;
      if stepid > 0 then
        select s.stepid into stepid from steps s, table(s.outputfiletypes) o where s.
  ↳applicationname=jprod.programname and s.APPLICATIONVERSION=jprod.programversion and
  ↳o.name=jprod.name and ROWNUM<2;
        BOOKKEEPINGORACLEDB.insertProdOutputFtypes(prod.production, stepid, jprod.
  ↳filetypeid, jprod.visibilityflag, jprod.eventtypeid);
```

(continues on next page)

(continued from previous page)

```

        update prods set processed='Y', stepid='Y' where production=prod.production;
        update jobs j set j.stepid=stepid where j.production=prod.production and j.
        ↪programname=jprod.programname and j.programversion=jprod.programversion;
        BOOKKEEPINGORACLEDB.insertStepsContainer (prod.production, stepid, stnum);
    else
        --DBMS_OUTPUT.put_line ('insert');
        SELECT applications_index_seq.nextval into stepid from dual;
        insert into steps(stepid,applicationName,applicationversion,
        ↪processingpass)values(stepid,jprod.programname,jprod.programversion,'FixedStep');
        BOOKKEEPINGORACLEDB.insertProdOutputFtypes(prod.production, stepid, jprod.
        ↪filetypeid, jprod.visibilityflag, jprod.eventtypeid);
        update prods set processed='Y', stepid='Y' where production=prod.production;
        update jobs j set j.stepid=stepid where j.production=prod.production and j.
        ↪programname=jprod.programname and j.programversion=jprod.programversion;
        BOOKKEEPINGORACLEDB.insertStepsContainer (prod.production, stepid, stnum);
    END IF;
END IF;
commit;
END LOOP;
END LOOP;
END;
/

```

NOTE: The files which do not have event type it is not added to the productionoutputfiles...

```

SQL> select * from prods p where p.processed='N' and p.production>0 and p.production
        ↪not in (select distinct ss.production from stepscontainer ss);

PRODUCTION P S P
-----
52192 N N N

```

Added to the productionoutputfile:

```

exec BOOKKEEPINGORACLEDB.insertProdOutputFtypes(52192, 128808, 88, 'Y',11114044);
exec BOOKKEEPINGORACLEDB.insertProdOutputFtypes(52192, 129669, 121, 'Y',11114044);

```

Fix the remained productions:

```

declare
nb number;
begin
FOR stcont in (select distinct ss.production from stepscontainer ss where ss.
        ↪production in (select p.production from prods p where p.processed='N' and p.
        ↪production>0)) LOOP
    DBMS_OUTPUT.put_line (stcont.production);
    FOR st in (select s.stepid, step from steps s, stepscontainer st where st.stepid=s.
        ↪stepid and st.production=stcont.production order by step) LOOP
        select count(*) into nb from jobs j, files f, filetypeypes ft where ft.filetypeid=f.
        ↪filetypeid and f.jobid=j.jobid and j.production=stcont.production and j.stepid=st.
        ↪stepid and f.filetypeid not in (9,17) and eventtypeid is not null;
        if nb=0 then
            update jobs set stepid=st.stepid where production=stcont.production;
            commit;
        END IF;
        FOR f in (select distinct j.stepid,ft.name, f.eventtypeid, ft.filetypeid, f.
        ↪visibilityflag from jobs j, files f, filetypeypes ft

```

(continues on next page)

(continued from previous page)

```

        where ft.filetypeid=f.filetypeid and f.jobid=j.jobid and
              j.production=stcont.production and j.stepid=st.stepid and f.
↪filetypeid not in (9,17) and eventtypeid is not null) LOOP
        DBMS_OUTPUT.put_line (stcont.production||'-'>'||st.stepid||'-'>'||f.filetypeid||
↪'-'>'||f.visibilityflag||'-'>'||f.eventtypeid);
        BOOKKEEPINGORACLEDB.insertProdnOutputFtypes(stcont.production, st.stepid, f.
↪filetypeid, f.visibilityflag,f.eventtypeid);
        update prods set processed='Y' where production=stcont.production;
    END LOOP;
END LOOP;
commit;
END LOOP;
END;
/

```

```
select * from prods where processed='N';
```

```
PRODUCTION P S P
```

```
-----
```

```
24179 N N N
```

```
-42854 N N N
```

Two production are problematic. The eventtypeid is null for 24179. -42854 is not yet deleted...

2.7.11 Consistency checks

We run some consistent checks in order to make sure the productionoutputfiles table correctly filled.

```

declare
counter number;
nb number;
begin
counter:=0;
for p in (select production,EVENTTYPEID,FILETYPEID, programname, programversion,
↪simid, daqperiodid from prodview) LOOP
    if p.simid>0 then
        select count(*) into nb from productionoutputfiles prod, productionscontainer ct,
↪steps s where ct.production=prod.production and
        prod.production=p.production and prod.filetypeid=p.filetypeid and prod.
↪eventtypeid=p.eventtypeid and prod.gotreplica='Yes' and prod.Visible='Y' and
        ct.simid=p.simid and s.stepid=prod.stepid and s.applicationname=p.programname,
↪and s.applicationversion=p.programversion;
    else
        select count(*) into nb from productionoutputfiles prod, productionscontainer ct,
↪steps s where ct.production=prod.production and
        prod.production=p.production and prod.filetypeid=p.filetypeid and prod.
↪eventtypeid=p.eventtypeid and prod.gotreplica='Yes' and prod.Visible='Y' and
        ct.daqperiodid=p.daqperiodid and s.stepid=prod.stepid and s.applicationname=p.
↪programname and s.applicationversion=p.programversion;
    end if;
    if nb=0 then
        DBMS_OUTPUT.put_line (nb||' '||p.production||' '||p.EVENTTYPEID||' '||p.
↪FILETYPEID);
        counter:=counter+1;
    end if;
end if;
end if;

```

(continues on next page)

(continued from previous page)

```

    if nb>1 then
        DBMS_OUTPUT.put_line ('DOUBLE:'||nb||' '||p.production||' '||p.EVENTTYPEID||'
↪'||p.FILETYPEID);
    END IF;
END LOOP;
DBMS_OUTPUT.put_line ('COUNTER:'||counter);
END;
/

```

1035 production found.

The following script is used to fix the productions which are wrong in the productionoutputfiles tabe.

```

declare
    counter number;
    nb number;
begin
    counter:=0;
    for p in (select production,EVENTTYPEID,FILETYPEID, programname, programversion,
↪simid, daqperiodid from prodview)
        LOOP
            if p.simid>0 then
                select count(*) into nb from productionoutputfiles prod, productionscontainer ct,
↪steps s where ct.production=prod.production and
                prod.production=p.production and prod.filetypeid=p.filetypeid and prod.
↪eventtypeid=p.eventtypeid and prod.gotreplica='Yes' and prod.Visible='Y' and
                ct.simid=p.simid and s.stepid=prod.stepid;
            else
                select count(*) into nb from productionoutputfiles prod, productionscontainer ct,
↪steps s where ct.production=prod.production and
                prod.production=p.production and prod.filetypeid=p.filetypeid and prod.
↪eventtypeid=p.eventtypeid and prod.gotreplica='Yes' and prod.Visible='Y' and
                ct.daqperiodid=p.daqperiodid and s.stepid=prod.stepid;
            end if;
            if nb=0 then
                for dat in (select j.production, J.STEPID, f.eventtypeid, f.filetypeid, f.
↪gotreplica, f.visibilityflag from
                jobs j, files f where j.jobid=f.jobid and j.production=p.production and f.
↪filetypeid not in (9,17) and
                f.eventtypeid is not null GROUP BY j.production, j.stepid, f.eventtypeid, f.
↪filetypeid, f.gotreplica, f.visibilityflag Order by f.gotreplica,f.visibilityflag,
↪asc)
                    LOOP
                        select count(*) into nb from productionoutputfiles where production=dat.
↪production and
                        stepid=dat.stepid and filetypeid=dat.filetypeid and visible=dat.
↪visibilityflag and
                        eventtypeid=dat.eventtypeid and gotreplica=dat.gotreplica;
                        if nb=0 then
                            DBMS_OUTPUT.put_line (nb||' '||p.production||' '||p.EVENTTYPEID||' '||p.
↪FILETYPEID);
                            select count(*) into nb from productionoutputfiles where production=dat.
↪production and
                            stepid=dat.stepid and filetypeid=dat.filetypeid and visible=dat.
↪visibilityflag and
                            eventtypeid=dat.eventtypeid;
                            if nb=0 then

```

(continues on next page)

(continued from previous page)

```

        INSERT INTO productionoutputfiles(production, stepid, filetypeid, visible,
↪ eventtypeid,gotreplica) VALUES (dat.production,dat.stepid, dat.filetypeid, dat.
↪ visibilityflag,dat.eventtypeid, dat.gotreplica);
        else
            update productionoutputfiles set gotreplica=dat.gotreplica where
↪ production=dat.production and
                stepid=dat.stepid and filetypeid=dat.filetypeid and visible=dat.
↪ visibilityflag and
                eventtypeid=dat.eventtypeid;
            END IF;
            counter:=counter+1;
        end if;
    END LOOP;
end if;
if nb>1 then
    DBMS_OUTPUT.put_line ('DOUBLE:'||nb||' '||p.production||' '||p.EVENTTYPEID||'
↪ '||p.FILETYPEID);
    END IF;
END LOOP;
DBMS_OUTPUT.put_line ('COUNTER:'||counter);
END;
/

```


Note: The first and last file does not need to be executed. If you want to start with an empty database, you have to use them.

2.8 Data distribution

2.8.1 Policy

Archive

The default option is at *Operations/Setup/TransformationPlugins/ArchiveSEs*. It can be overwritten in each plugin. The choice is done randomly.

DST broadcast

The broadcast done by LHCbDSTBroadcast plugin is done according to the free space

2.8.2 RAW files processing and distribution

The RAW files all have a copy at CERN, and are then distributed across the Tier1. The processing is shared between CERN and the Tier1.

The selection of the site for copying the data and the site where the data will be processed (so called *RunDestination*) is done by the *RAWReplication* plugin. To do so, it uses shares that are defined in *Operations/Setup/Shares*

Selection of a Tier1 for the data distribution

The quotas are defined in *Operations/Setup/Shares/RAW*.

Since CERN has a copy of every file, it does not appear in the quota.

In practice, the absolute values are meaningless, what matters is their relative values. The total is normalized to a 100 in the code.

When choosing where a run will be copied, we look at the current status of the distribution, based on the run duration. The site which is the furthest from its objectives is selected.

Selection of a Tier1 for the data processing

Once a Tier1 has been selected to copy the RAW file, one needs to select a site where the data will be processed: either CERN or the Tier1 where the data is: the *RunDestination*. Note that the destination is chosen per Run, and will stay as is: all the production will process the run at the same location.

This is done using *Operations/Setup/Shares/CPUforRAW*. There, the values are independent: they should be between 0 and 1, and represents the fraction of data it will process compared to CERN. So if the value is 0.8, it means 80% of the data copied to that site will be processed at that site, and the 20 other percent at CERN.

This share is used by the processing plugin *DataProcessing*. The equivalent exists when reprocessing (plugin *DataReprocessing*): *Operations/Setup/Shares/CPUforReprocessing*

Change of values in the shares

Note: if a change is to be made after a transformation has already distributed a lot of files, it is better to start a new transformation.

The principle goes as follow, but is obviously better done with an Excel sheet.

From CRIC (<http://wlcg-cric.cern.ch/core/pledge/list/>), we take for each T1 the CPUpledge (in MHS06) and the TapePledge (PB). We deduce easily the CPUpledgePercent and TapePledgePercent.

From the StorageUsageSummary, we get the CurrentTapeUsage (e.g. `dirac-dms-storage-usage-summary -LCG -Site LCG.CERN.cern`)

We then have:

```
AdditionalTape = TapePledge - CurrentTape
```

From which we deduce AdditionalTapePercent.

We then compute the ratio:

```
CPU / NewTape = CPUpledgePercent / AdditionalTapePercent
```

It represents the increase of CPU pledge vs the increase of Tape with respect to the total.

We then chose a certain percentage of data which is going to be processed at CERN. Say 20%. We then get:

```
CPUShare = CPUpledgePercent*(1-0.2)
```

The next step is to assign a CPUFraction (in [0:1]) by hand following this guideline: the lower the CPU/Tape ratio, the lower the fraction processed “locally”.

The final step is to compute:

```
RAWShare = CPUShare/CPUFraction
```

It represents the percentage of data to be copied to the given T1.

Obviously, since we have an extra constraint, we have to give a degree of freedom. We normally give it to RAL with the following:

```
RALRAWShare = 100% - Sum(OtherShares)
RALCPUFraction = RALCpuShare / RALRAWShare
```

CPUFraction corresponds to *Operations/<Setup>/Shares/CPUforRAW*

RAWShare corresponds to *Operations/<Setup>/Shares/RAW*

2.8.3 MonteCarlo distribution

The distribution of MC relies on the *LHCbMCDstBroadcast* plugin. In order to know what to replicate, we use a wildcard in the bookkeeping query, and for each of the individual path, we start a transformation. To be sure not to start several time the same, we use the *-Unique* option.

The difficulty is to know for which year and which sim version to start. Gloria or Vladimir can tell you...

In order to list the BK paths that are going to be replicated:

```
for year in 2011 2012 2015 2016;
do
  for sim in Sim09b Sim09c;
  do
    dirac-dms-add-transformation --List --BK=/MC/$year//$sim/...Reco...;
  done;
done
```

List of processing passes for BK path /MC/2011//Sim09b/...Reco...

```
/Sim09b/Reco14c
/Sim09b/Reco14c/Stripping21r1NoPrescalingFlagged
/Sim09b/Trig0x40760037/Reco14c
/Sim09b/Trig0x40760037/Reco14c/Stripping20r1Filtered
/Sim09b/Trig0x40760037/Reco14c/Stripping20r1NoPrescalingFlagged
/Sim09b/Trig0x40760037/Reco14c/Stripping21r1Filtered
/Sim09b/Trig0x40760037/Reco14c/Stripping21r1NoPrescalingFlagged
/Sim09b/Trig0x40760037/Reco14c/Stripping21r1p1Filtered
/Sim09b/Trig0x40760037/Reco14c/Stripping21r1p1NoPrescalingFlagged
```

In order to actually start these replications:

```
for year in 2011 2012 2015 2016;
do
  for sim in Sim09b Sim09c;
  do
    dirac-dms-add-transformation --Plugin LHCbMCDSTBroadcastRandom --BK=/MC/$year//
    ↪$sim/...Reco... --Unique --Start;
  done;
done
```

2.8.4 Standing Transformations

it is useful to have some transformations always at hand where you can just add a few files. Here are a few:

```
# Replicate files to freezer
dirac-dms-add-transformation --Plugin ReplicateDataset --Destination CERN-FREEZER-EOS_
↪--Name 'Replicate-to-Freezer' --Force

# Replicate to local buffer
dirac-dms-add-transformation --Plugin=ReplicateToLocalSE --Dest=Tier1-Buffer --Name
↪'Replicate-to-local-Buffer' --Force

# Replicate to RAW
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=CERN-RAW --Name
↪'Replicate-to-CERN-RAW' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=GRIDKA-RAW --Name
↪'Replicate-to-GRIDKA-RAW' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=RAL-RAW --Name
↪'Replicate-to-RAL-RAW' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=IN2P3-RAW --Name
↪'Replicate-to-IN2P3-RAW' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=CNAF-RAW --Name
↪'Replicate-to-CNAF-RAW' --Force
```

(continues on next page)

(continued from previous page)

```

dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=RRCKI-RAW --Name
↳ 'Replicate-to-RRCKI-RAW' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=PIC-RAW --Name
↳ 'Replicate-to-PIC-RAW' --Force

# Replicate to DST
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=CERN-DST-EOS --Name
↳ 'Replicate-to-CERN-DST-EOS' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=GRIDKA-DST --Name
↳ 'Replicate-to-GRIDKA-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=RAL-DST --Name
↳ 'Replicate-to-RAL-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=IN2P3-DST --Name
↳ 'Replicate-to-IN2P3-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=CNAF-DST --Name
↳ 'Replicate-to-CNAF-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=RRCKI-DST --Name
↳ 'Replicate-to-RRCKI-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=PIC-DST --Name
↳ 'Replicate-to-PIC-DST' --Force

# Replicate to MC-DST
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=CERN_MC-DST-EOS --Name
↳ 'Replicate-to-CERN_MC-DST-EOS' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=GRIDKA_MC-DST --Name
↳ 'Replicate-to-GRIDKA_MC-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=RAL_MC-DST --Name
↳ 'Replicate-to-RAL_MC-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=IN2P3_MC-DST --Name
↳ 'Replicate-to-IN2P3_MC-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=CNAF_MC-DST --Name
↳ 'Replicate-to-CNAF_MC-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=RRCKI_MC-DST --Name
↳ 'Replicate-to-RRCKI_MC-DST' --Force
dirac-dms-add-transformation --Plugin=ReplicateDataset --Dest=PIC_MC-DST --Name
↳ 'Replicate-to-PIC_MC-DST' --Force

# To reduce number of replicas, based on data popularity
dirac-dms-add-transformation --Plugin ReduceReplicas --Number 1 --Name 'Reduce-to-1-
↳ replica' --Force
dirac-dms-add-transformation --Plugin ReduceReplicas --Number 2 --Name 'Reduce-to-2-
↳ replicas' --Force
dirac-dms-add-transformation --Plugin ReduceReplicas --Number 3 --Name 'Reduce-to-3-
↳ replicas' --Force

# remove from Tier1-Buffer
dirac-dms-add-transformation --Plugin RemoveReplicas --From Tier1-Buffer --Name
↳ 'Remove-from-Tier1-Buffer' --Force
dirac-dms-add-transformation --Plugin RemoveReplicas --From Tier1-DST --Name 'Remove-
↳ from-Tier1-DST' --Force

```

(continues on next page)

(continued from previous page)

```
# destroy dataset
dirac-dms-add-transformation --Plugin=DestroyDataset --Name 'Destroy-dataset' --Force
```

As a reminder, to add files in these transformations:

```
dirac-transformation-add-files <transName> [--Term | --File <file> | --LFN <lfn>]
```

2.9 ReStripping

2.9.1 Pre-Staging

We do the prestaging of the RDST and the RAW files before a re-stripping campaign. Both files are staged on the BUFFER storage of the run destination. A rough estimation of the last exercise is that we can stage about 100TB a day.

The staging of these two can be launched with a single command:

```
dirac-dms-add-transformation --Start --BK=<BKPATH> --Runs=<RUNRANGES> --
↳Plugin=ReplicateWithAncestors --DQFlags=OK,UNCHECKED --Dest=Tier1-Buffer
```

For example

```
dirac-dms-add-transformation --Start --BK=/LHCb/Collision17//RealData/Reco17//RDST --
↳Runs=191782:193500 --Plugin=ReplicateWithAncestors --DQFlags=OK,UNCHECKED --
↳Dest=Tier1-Buffer
```

In practice, the transformation plugin only knows about the RDST, and then adds the ancestors to the tasks.

Important parameters

A few operation parameters can be used to tune the staging. They are in *Operations/<setup>/TransformationPlugins/ReplicateWithAncestors*

TargetFilesAtDestination

This number sets an upper limit to the number of files allowed in total per **directory**. We look at the LFNs in the transformation, and take the directory of level 4 (in that case */lhcb/LHCb/Collision17/RDST*).

In order to know how many files are on an SE, we look into the StorageUsageDB for that specific directory.

Each SE has a share which is the TargetFilesAtDestination divided by the RAWShare of this site. The number of files in the SE should not be greater than this Share.

Since the StorageUsageDB is refreshed only once a day, we add the number of files Processed and Assigned in the last 12 hours to get a better estimate.

Example of logs in the TransformationAgent:

```
(V) [NoThread] [-9999] Get storage usage for directories /lhcb/LHCb/Collision16/RDST
(V) [NoThread] [-9999] Current storage usage per SE:
(V) [NoThread] [-9999] CERN-BUFFER: 2
(V) [NoThread] [-9999] CNAF-BUFFER: 41702
```

(continues on next page)

(continued from previous page)

```

(V) [NoThread] [-9999]      RAL-BUFFER: 29
(V) [NoThread] [-9999]      RRCKI-BUFFER: 16
(V) [NoThread] [-9999] Shares per SE for 400000 files:
(V) [NoThread] [-9999]      CERN-BUFFER: 79888.0
(V) [NoThread] [-9999]      CNAF-BUFFER: 57600.0
(V) [NoThread] [-9999]      GRIDKA-BUFFER: 52920.0
(V) [NoThread] [-9999]      IN2P3-BUFFER: 37080.0
(V) [NoThread] [-9999]      PIC-BUFFER: 19840.0
(V) [NoThread] [-9999]      RAL-BUFFER: 93152.0
(V) [NoThread] [-9999]      RRCKI-BUFFER: 26400.0
(V) [NoThread] [-9999]      SARA-BUFFER: 33120.0
(V) [NoThread] [-9999] Number of files Assigned or recently Processed (12 hours):
(V) [NoThread] [-9999]      CERN-BUFFER: 0
(V) [NoThread] [-9999]      CNAF-BUFFER: 15897
(V) [NoThread] [-9999]      GRIDKA-BUFFER: 0
(V) [NoThread] [-9999]      IN2P3-BUFFER: 0
(V) [NoThread] [-9999]      PIC-BUFFER: 0
(V) [NoThread] [-9999]      RAL-BUFFER: 0
(V) [NoThread] [-9999]      RRCKI-BUFFER: 0
(V) [NoThread] [-9999]      SARA-BUFFER: 0
[NoThread] [-9999] Maximum number of files per SE:
[NoThread] [-9999]      CERN-BUFFER: 79886
[NoThread] [-9999]      CNAF-BUFFER: 0
[NoThread] [-9999]      GRIDKA-BUFFER: 52919
[NoThread] [-9999]      IN2P3-BUFFER: 37080
[NoThread] [-9999]      PIC-BUFFER: 19840
[NoThread] [-9999]      RAL-BUFFER: 93122
[NoThread] [-9999]      RRCKI-BUFFER: 26383
[NoThread] [-9999]      SARA-BUFFER: 33120

```

MinFreeSpace

This is a water mark, in TB, and per SE. The information is taken from <CacheFeederAgent> Note that it is not very smart: if when we check, we are below the watermark, we do not create tasks. if we are above, we create them all, even if we will be well below after ! It is just a safeguard

2.9.2 Input Removal

Once a given RDST has been processed, it can be removed from BUFFER, as well as the associated raw file. This can be done with a single transformation:

```

dirac-dms-add-transformation --Plugin=RemoveReplicasWithAncestors --FromSE=Tier1-
↳Buffer --BK=<BKPATH> --ProcessingPass=PROCESSINGPASS> --DQFlags=OK,UNCHECKED --Runs=
↳<RUNRANGES> --Start

```

For example

```

dirac-dms-add-transformation --Plugin=RemoveReplicasWithAncestors --FromSE=Tier1-
↳Buffer --BK=/LHCb/Collision17/Beam6500GeV-VeloClosed-MagDown/RealData/Reco17//RDST -
↳-ProcessingPass=Stripping29r2 --DQFlags=OK,UNCHECKED --Runs=199386:200000 --Start

```

A few important points:

- the BKPath must contain the condition because that is what is used to find the production in which we check the descendants
- the processing pass is the output of the production
- Although not strictly needed, it is good practice to have one removal transformation per production, with the same run ranges.
- the production manager should extend the run ranges of both the production and the removal transformation

2.9.3 Output Replication

In the current computing model, the output is replicated on an archive and at a second DST storage. This is done using the LHCbDSTBroadcast plugin

```
dirac-dms-add-transformation --Plugin=LHCbDSTBroadcast --BK=<BKPATH> --Except
↪<USELESSSTREAM> --Start
```

For example

```
dirac-dms-add-transformation --Plugin=LHCbDSTBroadcast --BK=/LHCb/Collision15//
↪RealData/Reco15a/Stripping24r1//ALL.DST,ALL.MDST --Except CALIBRATION.DST --Start
```

Typical useless streams are normally *CALIBRATION.DST* and *MDST.DST*

2.9.4 Productions check

The productions need to be checked for consistency and from the Datamanagement point of view.

For the DataManagement, please see *DM checks at the end of Stripping* and *DM checks at the end of Merging*.

Also, some files might need to be cleaned manually because they were flagged bad during the production, see *Files unused in productions*.

2.10 Productions flushing

2.10.1 Flushing a transformation

Transformations normally have grouping factors: total size of the input files, number of files, etc. There are cases when the grouping conditions cannot be reached, for example if there are not enough files in the run to reach the threshold defined. In that case, the transformation can be *flushed*, meaning create tasks anyway with whatever is there.

The flushing is a manual operation that only has an impact on the files present at the moment of triggering it, meaning that if new files arrive later, they will accumulate again: a transformation does not stay in “flush mode”.

2.10.2 Flushing a run in a transformation

Many transformations have a grouping by Run on top of a running by size/files. The same as described previously can happen: within a given run, the grouping conditions cannot be reached. In that case, it is possible to flush the run. There are two major differences compared to flushing a transformation:

1. Flushing a run is definitive
2. The procedure *can* be automatic

1. Flushing a run is definitive

Once a run is set to flush, it will stay in this state. This means that if new files arrive after flushing the run, they will not be accumulated, and a new task will be create for each and every file that arrives. This is not what you want normally. You can check here how to make sure that you can manually flush a run in the Merging: [Flushing runs](#)

2. Automatic run flushing for Merging

The principle always consists in going back to the RAW files of a run, and making sure that all of them have descendants in the current production. In practice, we count the number of RAW ancestors of the files in the production, and compare it with the number of RAW files declared in the BKK. These two numbers must match. This count is done by stream.

The only runs that are considered for flushing are the runs marked as ‘finished’ in the bookkeeping.

However, it might happen that a run does not get flushed. This normally shows an issue at the Stripping level. Consider the following example, with a Run that contains 3 raw files:

RAW file	RDST file	Stripping output
A.RAW	A.RDST	A.stream1, A.stream2, A.stream3
B.RAW	B.RDST	B.stream2, B.stream3, B.stream4
C.RAW	C.RDST	C.stream3, c.stream4

So, when looking at the ancestors per stream, we find:

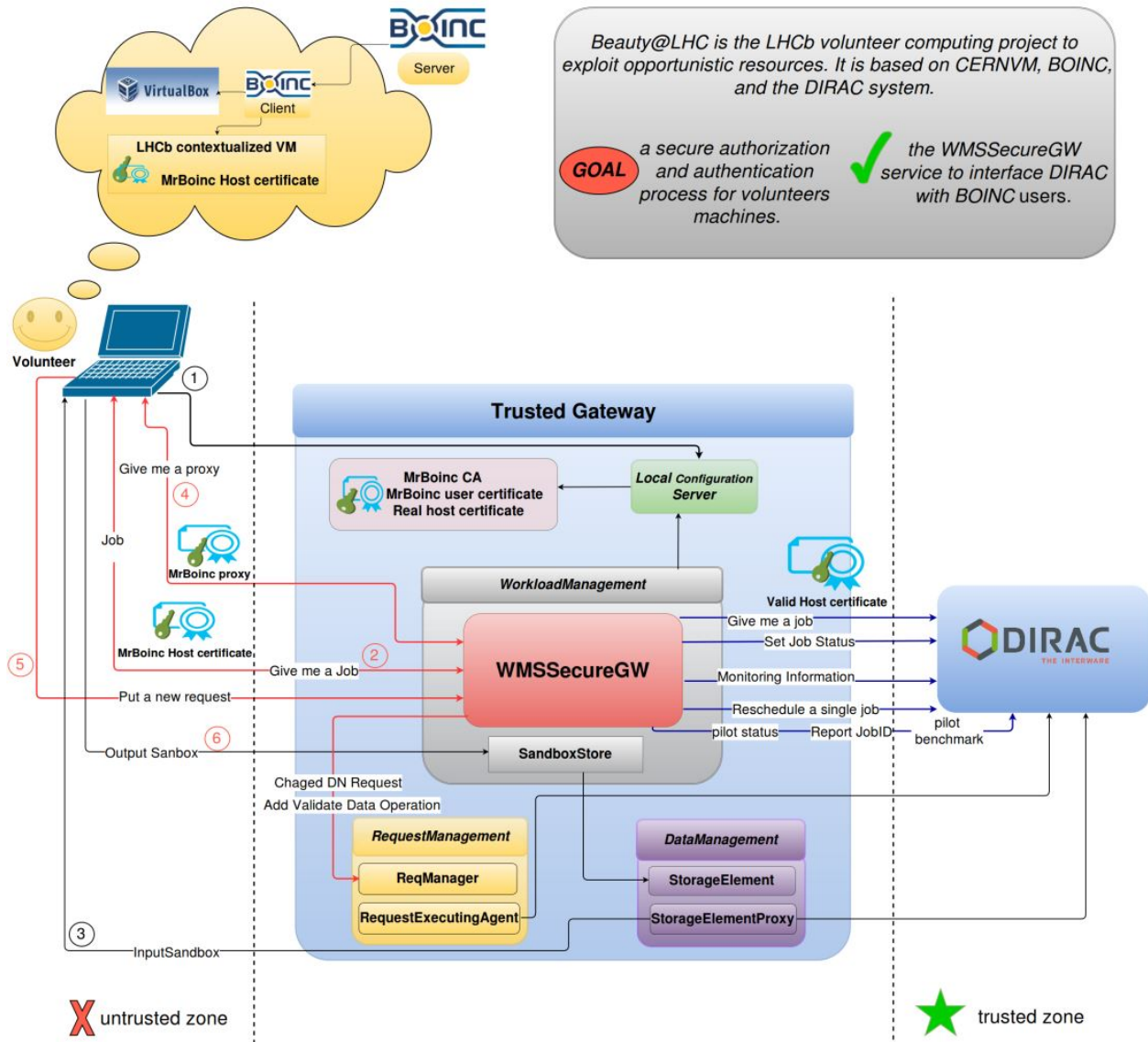
Stream	Nb of RAW ancestors
stream1	1
stream2	2
stream3	3
stream4	2

In that case, the flushing of the run will be triggered by stream3, since it finds the 3 ancestors. However, if in the stripping production, one file is never stripped because problematic, no stream will ever have all the raw files as ancestors, and the run will never be flushed. Hence, the run status in the merging is a good way to check the stripping :-)

Note that the script transformation-debug is more clever that the plugin, and can warn of such situations.

2.11 DIRAC on BOINC

2.11.1 Design of the system



Generic principle:

Two worlds, the trusted one (real Dirac system) and the untrusted one (Boinc client). The security relies on the gateway. In the untrusted world, we use only self sign certificates from a local CA. The BOINC gateway has two dirac configuration files: one for the trusted world, one for the untrusted world which is the one that is served by the local CS and that the jobs will see.

In the BOINC Gateway there are the following components:

- **DataManagement:**
 - StorageElement to upload the output of the Job

- StorageElementProxy to fetch the input sandbox
- **RequestManagement:**
 - After the job execution, to forward to the trusted world the data after carefully checking it
- SandboxStore for the outputSandbox, which uses the local StorageElement
- **Configuration server:**
 - running the BOINC setup.
 - define all the URLs of all the systems to WMSecureGW
- **WMSecureGW:**
 - accept calls targeted at all systems (WMS, Monitoring, RMS, Accounting, etc) and redirect them to the trusted world.

The BOINC users only (almost, see below) talk to the WMSecureGW, which redirects all the calls to the trusted world, using real host certificate. From the BOINC user perspective, it is just like they are talking to the real system.

Steps for running a job:

0. The machine boots with a MrBOINC host certificate integrated in the image
1. The machine starts a pilot that contacts the BOINC configuration server.
2. The pilot asks for a job to the WMSecureGW, which will ask one to the trusted world, and return it
3. The pilot download the InputSandbox through a call to the StorageElementProxy (How is the authentication done ?)
4. The pilot asks for a proxy to the WMSecureGW. It will receive a MrBoinc proxy.
5. After the job has finished, the job will upload the output to the StorageElement on the BOINC gateway (as failover) and send a RMS Request to the WMSecureGW (see details in gw setup)
6. The output sandbox is uploaded to the BOINC gateway

The Request will be executed by the local RequestExecutingAgent, which will transfer all the output data to the trusted world.

Authentication and Authorization

Input sandbox download

The download goes through a StorageElementProxy. This means that if the real Sandbox SE is a dirac service, we will use the host certificate of the boinc gateway. This is the only case that can work. If the SE were an outside service (eos, anything else), we would use the MrBoinc proxy, and then fail.

Output data upload

The data is uploaded as MrBOINC to the BOINC-SE. The final copy to the real system will be done using the DN inserted to perform the replication request (see below).

Request modification

When a request is done, it is intercepted by the WMSecureGW, which changes the ownerDN and add a first Operation which is WMSecureOutput. For the time being, the ownerDN is always the one defined in *Operations/<setup>/BoincShifter/ProductionManager*. This should however change and be based on the actual job (see todo)

2.11.2 Configuration of the system

There are in facts two distinct configurations:

- The configuration used by the services and agents running on the gateway
- The configuration used by the jobs

Template files are available for you to use.

Configuration of services and agents

This configuration corresponds to the local configuration of the gateway as a vbox: the traditional *dirac.cfg*. An example of such file can be found [here](#).

First of all, because the components on the gateway will interact with the real system, this configuration file has to refer to the real CS and the setup used should be the same as the one the jobs will run it (so in fact, your production setup normally).

Systems

Even if all the services and agents definition could be left in the central CS if standard, we rather recommend to redefine them locally to have a self contained configuration file. For a list of components, please see [here](#).

We strongly recommend to define a database section totally different from the production one.

As for the URLs, you do not need to define most of them as the calls should go directly to the real system, however there are a few exceptions:

- *Framework/BoincProxyManager*: This URL should point at your local proxy manager. The reason is that the jobs will contact the WMSecureGW to get a proxy. And this gateway will in fact query *Framework/BoincProxyManager* to retrieve the MrBoinc Proxy, and not *Framework/ProxyManager* not to leak any real proxy.
- *RequestManagement/ReqManager*: This URL should point at your local ReqManager. The reason is that any request that the job will try to put will go through the gateway, which will modify it slightly (see REFERENCE TO REQUEST MODIFICATION), and then forward it to the local ReqManager, to be executed by the local RequestExecutingAgent. No request from a BOINC Job should ever reach the central system.
- *RequestManagement/ReqProxyURLs*: this needs to be redefined to the local ReqProxy, otherwise the requests might enter the real system if they go through a central ReqProxy.

RequestExecutingAgent

There is a special Operation to define in the RequestExecutingAgent: *WMSecureOutputData*. It can point to any module you like. This Operation will be inserted in the first position whenever a job puts a Request. Typically, it is a safety operation, where you can check anything you like (provenance, nature of the output data, etc).

StorageElementProxy

The StorageElementProxy will try getting the MrBoinc proxy when the job asks for the input sandbox. So for this service, we need to redefine the ProxyManager as pointing to the WMSSecureGW.

ReqProxy

The ReqProxy must go through the WMSSecureGW in order to upload the request with the right ownerDN otherwise it will come up with MrBoincUser. So for this service, we need to redefine the ReqManager as pointing to the WMSSecureGW.

Operations

To make sure to override anything that could be defined in the central CS, we need to put it in the setup specific section, not in the *Defaults* one.

- *BoincShifter/ProductionManager*: This is used by the WMSSecureGW to reassign the ownership of the requests coming from BOINC jobs. This will hopefully disappear in a future version.
- *BoincShifter/BoincShifter*: This is used by the WMSSecureGW to return the MrBoinc user proxy. The user and group should match what you define in Operations

Resources

StorageElements

Here you need to only define BOINC-SE as your local storage element. Technically, you could avoid it because it must be defined anyway in the real CS. All the access must be set to Active.

Registry

The DefaultGroup should be boinc_users.

Users

Only MrBoinc needs to be defined. The DN is the one of the user certificate you self generated (see [Create/Renew the MrBoinc User certificate and proxy](#))

CAUTION: not sure now, probably need to redefine all the shifter users.

Groups

The groups are not merged with the central CS, but replaced. You should define a boinc_users group, with NormalUser as property, and add only MrBoinc in it.

Hosts

MrBoincHost should be defined as the certificate shipped with the VM image

Configuration for the jobs

This configuration is served to the jobs by the Configuration server running on the gateway. It basically just defines what the job needs in order to run.

The setup used should be the same as the real one (like LHCb-Certification), but you have to redefine the setup as using a different instance for each system (like boincInstance)

Systems

We only define the URLs, and all of them must point to the WMSSecureGW. There is currently one exception: the DataStore, which is not handled by the WMSSecureGW.

LocalSite

This is your BOINC site name. It has to be consistent with what goes in the central CS (see below)

Operations

CAUTION: check which protocol really needs modification * DataManagement: The protocol lists (RegistrationProtocols, ThirdPartyProtocols, WriteProtocols, AccessProtocols) need to be redefined in order to include 'proxy' * ResourceStatus/Config: disable RSS all together. * Services/Catalogs: define the catalogs you want to use.

Resources

- FileCatalog: define the catalogs you want to use.
- Sites: only the BOINC site is needed
- StorageElement: all the SEs a job might need to write to or read from. Most of the definition can be fake, but they need to be there. What matters is that the configuration of BOINC-SE, which is used as failover, is correct.
- StorageElementGroups: all the groups that might potentially be used by your jobs. Important is to redefine the failover as BOINC-SE
- Computing: this gives the OS compatibility. Take it from your real system. Hopefully, pilot3 will get rid of that.

Configuration to put in the central CS

Some changes are needed on your real system to have jobs flowing in BOINC.

In order to have the matcher send jobs to BOINC, you need to define the site just like in the BOINC-Conf.cfg:

```
Resources
{
  Sites
  {
    BOINC
    {
      BOINC.World.org
      {
```

(continues on next page)

(continued from previous page)

```

CE = Boinc-World-CE.org
CEs
{
  Boinc-World-CE.org
  {
    CETYPE = Boinc
    Queues
    {
      MaxCPUTime = 100000
      Boinc.World.Queue
      {
        MaxCPUTime = 100000
      }
    }
  }
}
# This special CE is for sending test jobs with a special tag
Boinc-World-TestCE.org
{
  CETYPE = Boinc
  Tag = BoincTestJobs
  Queues
  {
    MaxCPUTime = 100000
    Boinc.World.TestQueue
    {
      MaxCPUTime = 100000
    }
  }
}
}
}
}
}
}
}
}
}
}

```

You might also want to define some running limits (typically, at the moment, you only want MC running there):

```

Operations
{
  <Setup>
  {
    JobScheduling
    {
      RunningLimit
      {
        BOINC.World.org
        {
          JobType
          {
            User = 0
            MCSimulation = 500
          }
        }
      }
    }
  }
}
}
}
}
}
}
}
}
}

```

You have to define the BOINC SE, just like it is in the gateway `dirac.cfg`, without the file protocol. The reason is that the REA of the gateway will have RSS enabled, so RSS must know this SE. Define an always banned RSS rule for it, so RSS does not bother trying to test it:

```
Operations
{
  <Setup>
  {
    ResourceStatus
    {
      Policies
      {
        Ban_BOINC_SE
        {
          policyType = AlwaysBanned
          matchParams
          {
            name = BOINC-SE
            statusType = ReadAccess
            statusType += WriteAccess
            statusType += CheckAccess
            elementType = StorageElement
          }
        }
      }
    }
  }
}
```

2.11.3 Gateway setup

Certificate handling

First of all, a real certificate, trusted by your real system must be present on this machine. This is the one that will be used by the services and agents running on it. However, the untrusted world runs with self signed certificates. So we need to have our own CA.

Generate the CA certs

see <https://jamielinux.com/docs/openssl-certificate-authority/index.html> for detailed guide.

Work into `/path/to/boincCertificate` and there:

```
mkdir -p ca/newcerts ca/certs ca/crl
touch ca/index.txt
echo 1000 > ca/serial
echo 1000 > ca/crlnumber
```

Create the `openssl_config_ca.cnf` file (see below) in the `ca/` directory. You might need to edit it some path in the file.

Create the CA key

```
cd ca
# Type one of the two command below
openssl genrsa -aes256 -out ca.key.pem 4096      # for encrypted key
openssl genrsa -out ca.key.pem 4096              # for unencrypted key

chmod 400 ca.key.pem
```

Create the CA root certificate

```
openssl req -config openssl_config_ca.cnf -key ca.key.pem -new -x509 -days 7300 -
↳sha256 -extensions v3_ca -out ca.cert.pem
```

On the gateway machine, the CA certificate should be copied (*sym-link*) (ca.cert.pem) in /etc/grid-security/certificates. And the index hash link should be created

```
caHash=$(openssl x509 -in ca.cert.pem -noout -hash)
ln -s /etc/grid-security/certificates/ca.cert.pem /etc/grid-security/certificates/"
↳$caHash".0
```

Create/Renew the MrBoinc User certificate and proxy

Then we need a user certificate (MrBoinc User) self-signed by our own CA to be used in the untrusted world to obtain a MrBoinc user proxy.

Work into /path/to/boincCertificate and there:

```
mkdir MrBoinc
```

Create/use the openssl_config_user.cnf file (see below) in the MrBoinc/ directory.

Create the MrBoinc User private key

```
openssl genrsa -out MrBoinc/userkey.pem 4096
chmod 400 MrBoinc/userkey.pem
```

Generate the certificate request

```
openssl req -config MrBoinc/openssl_config_user.cnf -key MrBoinc/userkey.pem -new -
↳out MrBoinc/request.csr.pem
```

Create the MrBoinc User certificate, valid for 375 days

```
openssl ca -config ca/openssl_config_ca.cnf \
    -extensions usr_cert \
    -in MrBoinc/request.csr.pem \
    -out MrBoinc/usercert.pem
```

The MrBoinc user certificate must be then saved on the gateway machine under `~/ .globus`.

MrBoinc Proxy generation

From the gateway machine, become the dirac user, and source the bashrc. Then:

```
lhcb-proxy-init \
  -o '/Systems/Framework/<your Setup>/URLs/ProxyManager=dips://<your gw machine>
↪:9152/Framework/ProxyManager'\
  -o '/DIRAC/Configuration/Servers=dips://<your gw machine>:9135/Configuration/Server
↪'\
  -U
#e.g.
lhcb-proxy-init \
  -o '/Systems/Framework/Certification/URLs/ProxyManager=dips://lbboincertif.cern.
↪ch:9152/Framework/ProxyManager'\
  -o '/DIRAC/Configuration/Servers=dips://lbboincertif.cern.ch:9135/Configuration/
↪Server'\
  -U
```

MrBoinc Proxy generation (DEPRECATED)

This procedure must be done after the gateway is setup

1. From lxplus: Ban the BOINC site
 - `lb-run LHCbDirac/prod bash --norc`
 - `lhcb-proxy-init -g lhcb_admin`
 - `dirac-admin-ban-site BOINC.World.org "boinc_proxy_renewal"`
2. From lbvobox46: you need to upload the new proxy in the proxydb of lbvobox46. As dirac user, do
 - `lhcb-proxy-init -o /Systems/Framework/Production/URLs/ProxyManager="dips://lbvobox46.cern.ch:9152/Framework/ProxyManager" -o /DIRAC/Configuration/Servers="dips://lbvobox46.cern.ch:9135/Configuration/Server" -g lhcb_pilot -v 1500:00 -U`
3. Allow the site from lxplus
 - `dirac-admin-allow-site BOINC.World.org "boinc_proxy_updated"`

2.11.4 Gateway DIRAC Installation

Just follow what is described in the documentation: <http://dirac.readthedocs.io/en/latest/AdministratorGuide/InstallingDIRACService/index.html#install-primary-server>

Use the `install.cfg` provided in the `cfgTemplates`

Then turn every component off in order to edit by hand the configuration files:

```
runsvctrl d startup/*
```

Use the provided configuration files

Take the `dirac.cfg` and `BOINC-conf.cfg` in the template, and adapt them. Remember that the `dirac.cfg` is used only by agents and services, and overwrites or adds little on top of the central CS. The `BOINC-Conf.cfg` is what is served to the BOINC VMs.

The advantage of using these templates is that it is much easier and faster. But hopefully, they will stay up to date...

Generate the `dirac.cfg` (in progress...)

Move content of the `BOINCCert.cfg` to `dirac.cfg` : * all of Registry * remove website part * copy all the system definition

In `dirac.cfg`: * replace `boincInstance` with the real instance you want to map to (`boincInstance -> certification`) * replace `DIRAC/Configuration/Servers` with the real server * `LocalSite`: name the gateway * Registry: add `MrBoincHost` (after generating it) * `DIRAC/Security/UseServerCertificate = True` * In the URLs, `ProxyManager -> BoincProxyManager`

Edit the file:

- add the primary server URL
- change the setup to point to your real system and not local one
- run `generateRunitEnv`
- create the database by hand (`ReqDB`, `proxyDB`, `sandboxstoreDB`)
- Edit `dirac.cfg`

2.11.5 BOINC Image

The client side interaction is handled by CERN IT (Laurence Field atm). I am not sure how their whole system works, but the main point is that the client ends up booting an image: * contextualized by a `user_data` file which basically just specify the `cvmfs` path to use, and contain the `MrBoincHost` certificate and key (see below) * Start a pilot bootstrap script available here: <https://gitlab.cern.ch/vc/vm/raw/master/bin/lhcb-pilot>

2.11.6 Generate the MrBoinc host certificate

This certificate/key needs to be given to IT to add in the contextualization file.

Work into `/path/to/boincCertificate` and there:

```
mkdir MrBoincHost
```

Create the `openssl_config_host.cnf` file in the `MrBoincHost/` directory:

```
# Generate the key
openssl genrsa -out MrBoincHost/hostkey.pem 4096
chmod 400 MrBoincHost/hostkey.pem

# Create the request
openssl req -config MrBoincHost/openssl_config_host.cnf -key MrBoincHost/hostkey.pem \
  -new -sha256 -out MrBoincHost/request.csr.pem

# Sign it
openssl ca -config ca/openssl_config_ca.cnf \
  -extensions server_cert \
```

(continues on next page)

(continued from previous page)

```
-in MrBoincHost/request.csr.pem \
-out MrBoincHost/hostcert.pem
```

This self-signed host certificate (MrBoinc Host) must then be saved on any BOINC VM in `/etc/grid-security`. Do not forget to add it to the list of trusted host for ProxyDelegation in the `dirac.cfg` of the gateway

2.11.7 Test

The BOINCDirac repository is here: <https://github.com/DIRACGrid/BoincDIRAC>

This is *generic*, right now (04/2017) is not used by LHCb. All the code used by LHCb is within LHCbDIRAC.

- VM creation and contextualization files (lhcb_pilot, create_image): <https://gitlab.cern.ch/vc/vm/tree/master/bin>
- Pilot v3 (the one used in VMs) code repository: <https://github.com/DIRACGrid/Pilot>

Start a pilot on VM to test the BOINC gateway service:

1. MrBoinc Host certificate under `/etc/grid-security` needed (or somewhere else)
2. Get the pilot files:

```
curl --insecure -L -O https://lhcb-portal-dirac.cern.ch/pilot/pilot.json
curl --insecure -L -O https://lhcb-portal-dirac.cern.ch/pilot/pilotTools.
↪py
curl --insecure -L -O https://lhcb-portal-dirac.cern.ch/pilot/
↪pilotCommands.py
curl --insecure -L -O https://lhcb-portal-dirac.cern.ch/pilot/
↪LHCbPilotCommands.py
curl --insecure -L -O https://lhcb-portal-dirac.cern.ch/pilot/dirac-
↪pilot.py
curl --insecure -L -O https://lhcb-portal-dirac.cern.ch/pilot/dirac-
↪install.py
```

4. Define the environment variables needed:

```
export X509_CERT_DIR=/cvmfs/lhcb.cern.ch/etc/grid-security/certificates/
export X509_VOMS_DIR=/cvmfs/lhcb.cern.ch/etc/grid-security/vomsdir
```

3. start the pilot with:

```
#run the dirac-pilot script
python ./dirac-pilot.py \
  --debug \
  --setup=<YOUR SETUP (LHCb-Certification)> \
  --pilotCFGFile=pilot.json \
  -l LHCb \
  -o LbRunOnly \
  --Name=<YOUR CE NAME (BOINCCert-World-CE.org)> \
  --Queue=<YOUR QUEUE NAME (BoincCert.World.Queue)> \
  --MaxCycles=1 \
  --name=<YOUR BOINC SITE (BOINCCert.World.org)> \
  --cert \
  --certLocation=<CERTIFICATE LOCATION (/etc/grid-security/) \
  --commandExtensions LHCbPilot \
  --configurationServer <YOUR BOINC CONFIG SERVER (dips://lbboinccertif.
↪cern.ch:9135/Configuration/Server)>
```

2.11.8 TO-DO

Stalled jobs

In the current setup:

- The VM is running for a set amount of time (currently 24 hours)
- The JobAgent has no limit on its number of cycles

BOINC will kill the VM at the end of the set time, and any job that was running at the time will stall.

Some possible solutions are:

- 1) Put a limit of one cycle on the JobAgent, and shutdown the VM at the end of this cycle. This will end the BOINC task, and a new one will begin.
- 2) Make use of the planned \$MACHINEFEATURES and \$JOBFEAUTURES to signal that we have 24 hours of computing available and let the jobs adapt.

InputData

If we need input data, there is currently no way: the real storages are define so that the upload fail and we create a request. The downloading would need to go throught a proxy. Maybe we could have a read only proxy, and the result would be the same as for the sandbox?

SandboxStore

At the moment, the output sandbox are not used at all. They are uplaoded on the BOINC gateway, and stay there forever. We can: * not upload them (point to WMSSecureGW and always return S_OK) * forward them (either directly (not good) or by adding an extra step in the SecureOutput)

StorageElement

The disk often fills up because things are not cleaned.

Find the files and sort them by date:

```
find -type f -printf '%a %p\n' | sort -k1
```

Find files older than 60 days and sort:

```
find -mtime +60 -type f -printf '%a %p\n' | sort -k1
```

We can assume that after being there 60 days, they will not be touched anymore...

To remove, append to your find command:

```
-exec rm {} \;
```

Alternative (quicker find):

```
find -mtime +60 -type f -print0 | xargs -0 ls -lt
```

to keep only the path, append::

```
awk -F' ' '{print $9}'
```


to remove, append to the previous::

```
xargs rm -f
```

Accounting

BOINC credits

The BOINC accounting system is active and grants credits to the volunteers based on the time the VM ran and their CPU power.

This is not in most cases the actual work done on DIRAC jobs. This is maybe not an issue, as it still credit volunteers for the time they give us (after all, it's our problem if we don't use it).

Note that BOINC only gives credit at the end of a succesfull run. This has two constraints:

- VMs must have a set run time
- A failed or cancelled VM run on BOINC side will not grant credits, regardless of what happened inside the VM (eg. successful jobs)

An alternate method of granting credits on the run exist but is considered deprecated by the BOINC developers. Having this credit system enabled is pretty much mandatory to attract people within the BOINC community, especially the competitive minded volunteers.

LHCb accounting

Volunteers accounting

The Test4Theory project uses a second accounting method, based on the number of event produced. While not a good metric in our case as all events are not the same, a similar system could be implemented.

Currently, BOINC VMs hostname are set to boinc[host id]. This enables a basic job accounting, since the host-name will appear in the parameters of the jobs. Philippe Charpentier made a script to select jobs based on this: `WorkloadManagementSystem/scripts/dirac-wms-get-wn.py`. Because the hostname is not a primary key, this is slow. A web interface could be made available to volunteers to access this information.

Alternative? Use a noSQL db to store ~json~ data with job parameters (yet another accounting)

New VM contextualization / merge BOINC-specific changes ?

Quite a lot of BOINC-specific additions have been introduced in the contextualization and would need to be properly merged in the new contextualization. They currently reside in a branch at: http://svnweb.cern.ch/world/wsvn/dirac/LHCbVMDIRAC/branches/LHCbVMDIRAC_v0_BOINC_branch/.

Web page

The project web page is pretty much just the default BOINC page, this would probably need some additions.

Security aspects

The VM is now accessible as boinc:boinc:boinc. See the amiconfig file.

The certificate is in the image that is created. Use the SSH contextualization?

WMSecureGW development

Some developments are clearly needed

- Do something sensible in the SecureOutput operation
- Set the owner DN to the real owner of the job, not the shifter
- Keep a mapping of “boincID <-> JobID” if possible

Other stuff

Laurence said that we could submit directly to his HTCondor, so we would not need anymore his intervention. Much better. We would need a MrBoinc pilot proxy, and a MrBoinc user proxy (could technicaly be the same) And then a site director on the BOINC gateway Luca should do that, with the help of Andrew.

2.11.9 HOW-TO

Process Completed Jobs

From the gateway (lbboinc01.cern.ch) have a look at the requests

- `dirac-rms-request --Job <JobIDs>`
- check if output files are registered and/or do exists at SE with `dirac-dms-pfn-exists <LFNs>`
- check if output files that do not exists @CERN have descendants with (on lxplus) `dirac-bookkeeping-get-file-descendants --All --Depth=100 <LFNs>`
- **if files have already descendants**
 - update the job status to ‘Done’
 - remove the files @BOINC-SE-01 if needed with `dirac-dms-remove-files -o "/DIRAC/Security/UseServerCertificate=No" <LFN>`, using a valid Proxy (must be in `/tmp/x509up_uNNNNN` and exported to the env `export X509_USER_PROXY=/tmp/x509up_uNNNNN`)
- **if files are @CERN**
 - update the job status to ‘Done’
 - DO NOT REMOVE the files!
- **if files have neither descendants nor are @CERN nor @BOINC-SE**
 - KILL the job
- dispose the LOG files properly (simply remove them if still registered @BOINC)

Remove files (triggered by Vlad)

From lbboinc01.cern.ch

- double check if files are registered in the FC and/or do exists at the SE with `dirac-dms-lfn-replicas <LFNs>` and/or `dirac-dms-pfn-exists <LFNs>`
- if files do not exists (anymore) @CERN-BUFFER, double check if they have descendants with (on lxplus) `dirac-bookkeeping-get-file-descendants --All --Depth=100 <LFNs>`
- double check the status of the Job(s) and update it accordingly
- if everything is ok, then

Using a valid Proxy (must be in `/tmp/x509up_uNNNNN` and exported to the env `export X509_USER_PROXY=/tmp/x509up_uNNNNN`):

- `dirac-dms-remove-files -o "/DIRAC/Security/UseServerCertificate=No" <LFN>`

this will remove file(s) both from SE and catalog.

If the file(s) is not registered in the catalog but exists on SE (or for checking if it does exist in the SE)

- use the script `remove-LFN-from-SE.py`

remove-LFN-from-SE.py

```
from DIRAC.Core.Base.Script import parseCommandLine
parseCommandLine()
from DIRAC.Resources.Storage.StorageElement import StorageElement
se = StorageElement('BOINC-SE-01')
lfns = [<list of LFNs>]
#se.exists(lfns)
#se.removeFile(lfns)
```

2.12 HowTo

2.12.1 HowTo

Productions

These advices can be applied to all sort of productions.

Hospital

When?

When all files have been processed except some that cannot make it due to either excessive CPU or memory (e.g. at IN2P3 jobs stalled)

How?

In the CS: /Operations/LHCb-Production/Hospital set 2 or 3 options:

```
Transformations: list of productions to be processed at the hospital queue
HospitalSite: for example CLOUD.CERN.cern (or any other site without strict
↳ limitations)
HospitalCE: if needed, define a specific CE (e.g. one with more memory or CPU)
```

It is also possible to define some "Clinics" for specific purposes:

```
Within /Operations/LHCb-Production/Hospital/Clinics define one section per clinic,
↳ with a dummy name (e.g. Stripping)
For each clinic, define a list of transformations, a site and possibly a CE. If Site
↳ or CE is not defines the site / CE set for hospital is used.
```

```
Transformations: list of productions to be processed at this clinic
ClinicSite: for example CLOUD.CERN.cern (or any other site without strict limitations)
ClinicCE: if needed, define a specific CE (e.g. one with more memory or CPU)
```

Then:

reset the files Unused. They will be brokered to the designated hospital site, wherever the input data is.

Example:

```
[localhost] ~ $ dirac-transformation-debug 71500 --Status MaxReset --Info jobs
Transformation 71500 (Active) of type DataStripping (plugin ByRunWithFlush,
↳ GroupSize: 1) in Real Data/Reco17/Stripping29r2
BKQuery: {'StartRun': 199386L, 'ConfigName': 'LHCb', 'EndRun': 200350L, 'EventType':
↳ 90000000L, 'FileType': 'RDST', 'ProcessingPass': 'Real Data/Reco17', 'Visible': 'Yes
↳ ', 'DataQualityFlag': ['OK', 'UNCHECKED'], 'ConfigVersion':
'Collision17', 'DataTakingConditions': 'Beam6500GeV-VeloClosed-MagDown'}
```

2 files found with status ['MaxReset']

```
1 LFNs: ['/lhcb/LHCb/Collision17/RDST/00066581/0004/00066581_00043347_1.rdst'] :
↳ Status of corresponding 5 jobs (sorted):
Jobs: 200059983, 200171070, 200415532, 201337455, 201397489
Sites (CPU): LCG.IN2P3.fr (3609 s), LCG.IN2P3.fr (3635 s), LCG.IN2P3.fr (3626 s), LCG.
↳ IN2P3.fr (3617 s), LCG.IN2P3.fr (3649 s)
5 jobs terminated with status: Failed; Job stalled: pilot not running; DaVinci step
↳ 1
1 jobs stalled with last line: (LCG.IN2P3.fr) EventSelector SUCCESS Reading
↳ Event record 3001. Record number within stream 1: 3001
1 jobs stalled with last line: (LCG.IN2P3.fr) EventSelector SUCCESS Reading
↳ Event record 4001. Record number within stream 1: 4001
2 jobs stalled with last line: (LCG.IN2P3.fr) EventSelector SUCCESS Reading
↳ Event record 5001. Record number within stream 1: 5001
1 jobs stalled with last line: (LCG.IN2P3.fr) EventSelector SUCCESS Reading
↳ Event record 3001. Record number within stream 1: 3001
```

(continues on next page)

(continued from previous page)

```

1 LFNs: ['/lhcb/LHCb/Collision17/RDST/00066581/0007/00066581_00074330_1.rdst'] :_
↪Status of corresponding 7 jobs (sorted):
Jobs: 200339756, 200636702, 200762354, 200913317, 200945856, 201337457, 201397490
Sites (CPU): LCG.IN2P3.fr (32 s), LCG.IN2P3.fr (44 s), LCG.IN2P3.fr (46 s), LCG.IN2P3.
↪fr (45 s), LCG.IN2P3.fr (44 s), LCG.IN2P3.fr (3362 s), LCG.IN2P3.fr (38 s)
7 jobs terminated with status: Failed; Job stalled: pilot not running; DaVinci step_
↪1
1 jobs stalled with last line: (LCG.IN2P3.fr) dirac-jobexec INFO: DIRAC JobID_
↪200339756 is running at site LCG.IN2P3.fr
1 jobs stalled with last line: (LCG.IN2P3.fr) dirac-jobexec/Subprocess VERBOSE:_
↪systemCall: ['lb-run', '--use-grid', '-c', 'best', '--use=AppConfig v3r [...]'
1 jobs stalled with last line: (LCG.IN2P3.fr) dirac-jobexec/Subprocess VERBOSE:_
↪systemCall: ['lb-run', '--use-grid', '-c', 'best', '--use=AppConfig v3r [...]'
1 jobs stalled with last line: (LCG.IN2P3.fr) dirac-jobexec/Subprocess VERBOSE:_
↪systemCall: ['lb-run', '--use-grid', '-c', 'best', '--use=AppConfig v3r [...]'
1 jobs stalled with last line: (LCG.IN2P3.fr) dirac-jobexec/Subprocess VERBOSE:_
↪systemCall: ['lb-run', '--use-grid', '-c', 'best', '--use=AppConfig v3r [...]'
1 jobs stalled with last line: (LCG.IN2P3.fr) EventSelector SUCCESS Reading_
↪Event record 2001. Record number within stream 1: 2001
1 jobs stalled with last line: (LCG.IN2P3.fr) dirac-jobexec/Subprocess VERBOSE:_
↪systemCall: ['lb-run', '--use-grid', '-c', 'best', '--use=AppConfig v3r [...]'

```

So 2 files have to be hospitalised. Reset them Unused:

```

[localhost] ~ $ dirac-transformation-reset-files 71500 --Status MaxReset
2 files were set Unused in transformation 71500

```

Stripping

DM checks at the end of Stripping

The procedure is very similar to the Merging production: *DM checks at the end of Merging*

Troubleshooting

files Assigned with no jobs.

The first action to take is always to check the descendants. Then several cases may arise.

Jobs killed, data upload failing

This can be seen from the log files. In that case, we clean whatever the old job managed to produce, and we re-strip the input files

```

[lxplus079] ~ $ grep InFCNotInBK CheckDescendantsResults_70268.txt | dirac-dms-check-
↪fc2bkk --FixFC
Got 16 LFNs
Checking replicas for 16 files : found 16 files with replicas and 0 without in 0.2_
↪seconds
Getting 16 files metadata from BK : completed in 0.1 seconds
>>>>

```

(continues on next page)

(continued from previous page)

```

16 files are in the FC but have replica = NO in BK
===== Now checking 16 files from FC to SE =====
Checking replicas for 16 files : found 16 files with replicas and 0 without in 5.2_
↳seconds
Get FC metadata for 16 files to be checked: : completed in 0.1 seconds
Check existence and compare checksum file by file...
Getting checksum of 16 replicas in 2 SEs
0. At CERN-BUFFER (13 files) : completed in 2.1 seconds
1. At PIC-BUFFER (3 files) : completed in 1.3 seconds
Verifying checksum of 16 files
No files in FC not in BK -> OK!
No missing replicas at sites -> OK!
No replicas have a bad checksum -> OK!
All files exist and have a correct checksum -> OK!
===== Completed, 16 files are in the FC and SE but have replica = NO in BK =====
0 files are visible, 16 files are invisible
/lhcb/LHCb/Collision16/BHADRON.MDST/00070268/0006/00070268_00061489_1.Bhadron.mdst :_
↳Visi N
/lhcb/LHCb/Collision16/BHADRON.MDST/00070268/0006/00070268_00063156_1.Bhadron.mdst :_
↳Visi N
/lhcb/LHCb/Collision16/BHADRONCOMPLETEEVENT.DST/00070268/0006/00070268_00063156_1.
↳BhadronCompleteEvent.dst : Visi N
/lhcb/LHCb/Collision16/CALIBRATION.DST/00070268/0006/00070268_00060532_1.Calibration.
↳dst : Visi N
/lhcb/LHCb/Collision16/CALIBRATION.DST/00070268/0006/00070268_00063183_1.Calibration.
↳dst : Visi N
/lhcb/LHCb/Collision16/CHARM.MDST/00070268/0006/00070268_00060532_1.Charm.mdst : Visi_
↳N
/lhcb/LHCb/Collision16/CHARM.MDST/00070268/0006/00070268_00063156_1.Charm.mdst : Visi_
↳N
/lhcb/LHCb/Collision16/CHARM.MDST/00070268/0006/00070268_00063183_1.Charm.mdst : Visi_
↳N
/lhcb/LHCb/Collision16/CHARMCOMPLETEEVENT.DST/00070268/0006/00070268_00063156_1.
↳CharmCompleteEvent.dst : Visi N
/lhcb/LHCb/Collision16/CHARMCOMPLETEEVENT.DST/00070268/0006/00070268_00063183_1.
↳CharmCompleteEvent.dst : Visi N
/lhcb/LHCb/Collision16/EW.DST/00070268/0001/00070268_00016205_1.EW.dst : Visi N
/lhcb/LHCb/Collision16/EW.DST/00070268/0006/00070268_00061489_1.EW.dst : Visi N
/lhcb/LHCb/Collision16/EW.DST/00070268/0006/00070268_00063156_1.EW.dst : Visi N
/lhcb/LHCb/Collision16/FTAG.DST/00070268/0001/00070268_00016205_1.FTAG.dst : Visi N
/lhcb/LHCb/Collision16/LEPTONIC.MDST/00070268/0001/00070268_00016205_1.Leptonic.mdst_
↳: Visi N
/lhcb/LHCb/Collision16/LEPTONIC.MDST/00070268/0006/00070268_00063156_1.Leptonic.mdst_
↳: Visi N
Full list of files: grep InFCButBKNo CheckFC2BK-1.txt
Going to fix them, by removing from the FC and storage
Removing 16 files : completed in 22.1 seconds
Successfully removed 16 files
<<<<
No files in FC not in BK -> OK!

```

RDST files should be reset Unused automatically by the DRA:

```

NotProcWithDesc /lhcb/LHCb/Collision16/RDST/00051872/0000/00051872_00004324_1.rdst
NotProcWithDesc /lhcb/LHCb/Collision16/RDST/00051872/0007/00051872_00073248_1.rdst
NotProcWithDesc /lhcb/LHCb/Collision16/RDST/00051872/0007/00051872_00075929_1.rdst
NotProcWithDesc /lhcb/LHCb/Collision16/RDST/00051872/0013/00051872_00134216_1.rdst

```

(continues on next page)

(continued from previous page)

```
NotProcWithDesc /lhcb/LHCb/Collision16/RDST/00051872/0013/00051872_00138382_1.rdst
```

Merging

DM checks at the end of Merging

When the Merging productions for a given Stripping are over, there are a couple of situations that might arise:

1. Output file is registered in the FC but is not registered in the BKK
2. Output file is registered in the FC but does not have the replica flag = No
3. Output file is registered in the BKK but not in the FC

First case

Nothing can be done, we do not have all the info, we should just remove the file. If it is a debug file, it will be cleaned anyway when cleaning the DEBUG storage.

Second case

Either the file is at its final destination, and in that case the replica flag can just be toggled (*-FixBK* in *dirac-dms-check-fc2bkk*), or the file is in a failover. In the later case, it is enough to replicate the file to its run destination (*dirac-dms-replicate-to-run-destination*) and remove the replica on the failover storage.

Third case

If the file physically exists, the file can just be registered in the FC.

Examples

The 3rd case is noticeable only in the output replication transformation, because it will mark these files as Missing-InFC. For the first two cases, the best is to use *dirac-dms-check-fc2bkk*.

For example

```
[LHCbDirac prod] diracos $ dirac-dms-check-fc2bkk --Prod 69080,69076,68774,68772
Processing production 69080
Getting files from 18 directories : found 5348 files with replicas and 0 without in
↳13.9 seconds
Getting 5348 files metadata from BK : completed in 1.8 seconds
>>>>
1 files are in the FC but have replica = NO in BK
===== Now checking 1 files from FC to SE =====
Checking replicas for 1 files : found 1 files with replicas and 0 without in 1.1
↳seconds
Get FC metadata for 1 files to be checked: : completed in 0.1 seconds
Check existence and compare checksum file by file...
Getting checksum of 1 replicas in 1 SEs
0. At RAL-DST (1 files) : completed in 1.7 seconds
```

(continues on next page)

(continued from previous page)

```

Verifying checksum of 1 files
No files in FC not in BK -> OK!
No missing replicas at sites -> OK!
No replicas have a bad checksum -> OK!
All files exist and have a correct checksum -> OK!
===== Completed, 1 files are in the FC and SE but have replica = NO in BK =====
1 files are visible, 0 files are invisible
/lhcb/LHCb/Collision15/BHADRONCOMPLETEEVENT.DST/00069080/0000/00069080_00003151_1.
↳bhadroncompleteevent.dst :
Visi Y
Full list of files:      grep InFCButBKNo CheckFC2BK-2.txt
Use --FixBK to fix it (set the replica flag) or --FixFC (for removing from FC and
↳storage)
<<<<
No files in FC not in BK -> OK!
Processed production 69080
Processing production 69076
Getting files from 18 directories : found 5789 files with replicas and 0 without in
↳10.3 seconds
Getting 5789 files metadata from BK : completed in 2.9 seconds
No files in FC with replica = NO in BK -> OK!
No files in FC not in BK -> OK!
Processed production 69076
Processing production 68774
Getting files from 18 directories : found 7510 files with replicas and 0 without in
↳12.7 seconds
Getting 7510 files metadata from BK : completed in 2.8 seconds
No files in FC with replica = NO in BK -> OK!
No files in FC not in BK -> OK!
Processed production 68774
Processing production 68772
Getting files from 18 directories : found 10702 files with replicas and 0 without in
↳14.8 seconds
Getting 10702 files metadata from BK : completed in 4.2 seconds
No files in FC with replica = NO in BK -> OK!
>>>>
1 files are in the FC but are NOT in BK:
/lhcb/debug/Collision15/LEPTONIC.MDST/00068772/0000/00068772_00003806_1.leptonic.mdst
Full list of files:      grep InFCNotInBK CheckFC2BK-3.txt
Use --FixFC to fix it (remove from FC and storage)
<<<<

[LHCbDirac prod] diracos $ dirac-dms-check-fc2bkk --
LFN=/lhcb/LHCb/Collision15/BHADRONCOMPLETEEVENT.DST/00069080/0000/00069080_00003151_1.
↳bhadroncompleteevent.dst
--FixBK
Checking replicas for 1 files : found 1 files with replicas and 0 without in 0.3
↳seconds
Getting 1 files metadata from BK : completed in 0.0 seconds
>>>>
1 files are in the FC but have replica = NO in BK
===== Now checking 1 files from FC to SE =====
Checking replicas for 1 files : found 1 files with replicas and 0 without in 4.8
↳seconds

```

(continues on next page)

(continued from previous page)

```

Get FC metadata for 1 files to be checked: : completed in 0.4 seconds
Check existence and compare checksum file by file...
Getting checksum of 1 replicas in 1 SEs
0. At RAL-DST (1 files) : completed in 1.0 seconds
Verifying checksum of 1 files
No files in FC not in BK -> OK!
No missing replicas at sites -> OK!
No replicas have a bad checksum -> OK!
All files exist and have a correct checksum -> OK!
===== Completed, 1 files are in the FC and SE but have replica = NO in BK =====
1 files are visible, 0 files are invisible
/lhcb/LHCb/Collision15/BHADRONCOMPLETEEVENT.DST/00069080/0000/00069080_00003151_1.
->bhadroncompleteevent.dst :
Visi Y
Full list of files:      grep InFCButBKNo CheckFC2BK-4.txt
Going to fix them, setting the replica flag
      Successfully added replica flag to 1 files
<<<<
No files in FC not in BK -> OK!

[LHCbDirac prod] diracos $ dirac-dms-remove-files
/lhcb/debug/Collision15/LEPTONIC.MDST/00068772/0000/00068772_00003806_1.leptonic.mdst
Removing 1 files : completed in 8.1 seconds
Successfully removed 1 files

```

jobs failing during finalize

Problem:

If a Merge job fails during finalisation, its input files may not be removed... In addition its output files may be incorrectly uploaded or registered Therefore starting from the left non-merged files one may find anomalies and fix them. This requires getting invisible files in the DataStripping productions and checking their descendants in the Merge production

Examples:

Get the descendants of the DataStripping production (here 69528) that still have replicas, and check their descendants in the Merging production (here 69529)

```

[localhost] ~ $ dirac-bookkeeping-get-files --Production 69528 --Visi No | dirac-
->production-check-descendants 69529
Got 59 LFNs
Processing Merge production 69529
Looking for descendants of type ['EW.DST', 'BHADRON.MDST', 'SEMILEPTONIC.DST',
->'DIMUON.DST', 'CALIBRATION.DST', 'FTAG.DST', 'CHARMCOMPLETEEVENT.DST',
->'BHADRONCOMPLETEEVENT.DST', 'CHARM.MDST', 'LEPTONIC.MDST']
Getting files from the TransformationSystem...
Found 59 processed files and 0 non processed files (1.2 seconds)
Now getting daughters for 59 processed mothers in production 69529 (depth 1) :
->completed in 5.9 seconds
Checking replicas for 2 files : found 2 files with replicas and 0 without in 0.4
->seconds
Checking FC for 2 file found in FC and not in BK |
->|Checking replicas for 2 files (not in Failover) : found 0 files with
->replicas and 0 without in 0.5 seconds

```

(continues on next page)

(continued from previous page)

```

: found 2 in Failover in 0.5 seconds

Results:
2 descendants were found in Failover and have no replica flag in BK
All files:
/lhcb/LHCb/Collision16/DIMUON.DST/00069529/0001/00069529_00012853_1.dimuon.dst
/lhcb/LHCb/Collision16/BHADRONCOMPLETEEVENT.DST/00069529/0001/00069529_00012813_1.
↳ bhadroncompleteevent.dst
You should check whether they are in a failover request by looking at their job_
↳ status and in the RMS...
To list them:      grep InFailover CheckDescendantsResults_69529-1.txt
2 unique daughters found with real descendants
No processed LFNs with multiple descendants found -> OK!
No processed LFNs without descendants found -> OK!
No non processed LFNs with multiple descendants found -> OK!
No non processed LFNs with descendants found -> OK!
Complete list of files is in CheckDescendantsResults_69529-1.txt
Processed production 69529 in 9.4 seconds

```

After checking at the RMS whether they have matching Requests, and if so what happened to it, we can replicate them to final destination and then remove from Failover

```

[localhost] ~ $ grep InFailover CheckDescendantsResults_69529-1.txt | dirac-dms-
↳ replicate-to-run-destination --RemoveSource --SE Tier1-DST
Got 2 LFNs
Replicating 2 files to CERN-DST-EOS
Successful :
    CERN-DST-EOS :
        /lhcb/LHCb/Collision16/BHADRONCOMPLETEEVENT.DST/00069529/0001/00069529_
↳ 00012813_1.bhadroncompleteevent.dst :
            register : 0.757441997528
            replicate : 655.287761927
        /lhcb/LHCb/Collision16/DIMUON.DST/00069529/0001/00069529_00012853_1.dimuon.
↳ dst :
            register : 0.632274866104
            replicate : 46.3780457973

```

Finally, Check again and remove non-merged files

```

[localhost] ~ $ dirac-dms-remove-files --Last
Got 59 LFNs
Removing 59 files : completed in 103.1 seconds
59 files in status Processed in transformation 69529: status unchanged
Successfully removed 59 files

```

Flushing runs

When a file is problematic in the Stripping production, or if a RAW file was not processed in the Reco, the run cannot be flushed automatically (Number of ancestors != number of RAW in the run). We list the runs in the Stripping productions (here 71498) that have problematic files, and we flush them in the Merging (here 71499)

```

[localhost] ~ $ dirac-transformation-debug 71498 --Status Problematic --Info files |_
↳ dirac-bookkeeping-file-path --GroupBy RunNumber --Summary
--List

```

(continues on next page)

(continued from previous page)

```

Got 29 LFNs
Successful :
  RunNumber 201413 : 1 files
  RunNumber 201423 : 1 files
  RunNumber 201467 : 1 files
  RunNumber 201602 : 1 files
  RunNumber 201643 : 1 files
  RunNumber 201647 : 1 files
  RunNumber 201664 : 1 files
  RunNumber 201719 : 1 files
  RunNumber 201745 : 2 files
  RunNumber 201749 : 1 files
  RunNumber 201822 : 1 files
  RunNumber 201833 : 1 files
  RunNumber 201864 : 1 files
  RunNumber 201873 : 1 files
  RunNumber 201983 : 1 files
  RunNumber 202031 : 1 files
  RunNumber 202717 : 1 files
  RunNumber 202722 : 1 files
  RunNumber 202752 : 1 files
  RunNumber 202773 : 1 files
  RunNumber 202809 : 1 files
  RunNumber 202825 : 1 files
  RunNumber 202835 : 2 files
  RunNumber 202860 : 1 files
  RunNumber 202869 : 1 files
  RunNumber 202873 : 1 files
  RunNumber 202887 : 1 files

List of RunNumber values
201413,201423,201467,201602,201643,201647,201664,201719,201745,201749,201822,201833,
↳201864,201873,201983,202031,202717,202722,2027
52,202773,202809,202825,202835,202860,202869,202873,202887

```

Then flush the runs in the merging production

```

[localhost] ~ $ dirac-transformation-flush-runs 71499 --Runs
201413,201423,201467,201602,201643,201647,201664,201719,201745,201749,201822,201833,
↳201864,201873,201983,202031,202717,202722,2027
52,202773,202809,202825,202835,202860,202869,202873,202887
Runs being flushed in transformation 71499:
201413,201423,201467,201602,201643,201647,201664,201719,201745,201749,201822,201833,
↳201864,201873,201983,202031,202717,202722,2027
52,202773,202809,202825,202835,202860,202869,202873,202887
27 runs set to Flush in transformation 71499

```

Then, starting from the runs that are not flushed in the Merging, we can check if some RAW files do not have descendant

```

dirac-bookkeeping-run-files <runNumber> | grep FULL | dirac-bookkeeping-get-file-
↳descendants

```

The files that are marked as NotProcessed or NoDescendants are in runs that will need to be flushed by hand

Another way of understanding why a run is not flushed is by using `dirac-transformation-debug`. But this takes a looooong while

```
dirac-transformation-debug --Status=Unused --Info=flush <mergingProd>
```

What to do if some files have been processed multiply within a production ?

Here the example is for stripping <STRIPPING> and merging <MERGING>

0. Quickly stop the merging productions as well as the stripping productions!

Good to also stop the removal productions if any

1. Check whether their output was merged

```
grep ProcMultDesc CheckDescendantsResults_<STRIPPING>.txt | dirac-bookkeeping-get-  
↪file-descendants --Prod <STRIPPING> --All | dirac-transformation-debug <MERGING>
```

Look at whether the multiple descendants have been merged, in which case they are set Processed in the final list. You can also see all files that are eventually already part of merging jobs (status Assigned).

If the number of RDST files is small and only 1 or 2 streams have been merged, one may afford just removing the merged files (i.e. the output of the merging jobs that ran): select the Processed files above and do:

```
dirac-bookkeeping-get-file-descendants --Term  
<paste list of Processed files>
```

Else, one needs to remove the whole runs that are affected :(

2. Get the list of runs

```
grep ProcMultDesc CheckDescendantsResults_<STRIPPING>.txt | dirac-bookkeeping-file-  
↪path --Summary --GroupBy RunNumber --List
```

This prints out how many files per run and at the end the list of affected runs... Now starts the fun!

3. Get the list of files to be removed

3.0 Check whether some files are being merged

```
dirac-transformation-debug <MERGING> --Status Assigned --Run <list of runs>
```

As long as you have files Assigned, better do nothing. if you are in a hurry, you may kill the corresponding jobs and proceed, but this may cause troubles...

3.1 Merged files

```
dirac-bookkeeping-get-files --Prod <MERGING> --Run <list of runs> --Output toRemove  
↪<STRIPPING>-merged.txt
```

3.2 Non-merged files

```
dirac-bookkeeping-get-files --Prod <STRIPPING> --Run <list of runs> --Output toRemove
↳<STRIPPING>-nonMerged.txt
```

if this fails (there is a bug in the BK, but could be fixed ;-)

```
dirac-loop --Item <list of runs> `dirac-bookkeeping-get-files --Prod <STRIPPING>
↳--Run @arg@:@arg@ --Visi No' > toRemove<STRIPPING>-nonMerged.txt
```

3.3 Remove these files

Firstly enable the xxx-ARCHIVE SEs for removal:

```
dirac-dms-replica-stats --File toRemove<STRIPPING>-merged.txt
```

will give you the list of ARCHIVE SEs concerned...

```
dirac-admin-enable-se --AllowRemove <list of ARCHIVE SEs >

dirac-transformation-add-files --File toRemove<STRIPPING>-merged.txt,toRemove
↳<STRIPPING>-nonMerged.txt Remove-all-replicas-CleanTrans
```

This is unfortunately not enough as you should set Removed the files that were Processed already, since the transformation above would not change status of Processed files...:

```
dirac-bookkeeping-get-file-ancestors --File toRemove<STRIPPING>-merged.txt | dirac-
↳transformation-reset-files --NewStatus Removed <MERGING>
```

4. Get the list of RDST files to reprocess and reset them

4.0 Check whether some files are being processed

```
dirac-transformation-debug <STRIPPING> --Status Assigned --Run <list of runs>
```

As long as you have files Assigned, better do nothing If you are in a hurry, you may kill the corresponding jobs and proceed, to your own risk!

4.1 Get list of RDST files

```
dirac-bookkeeping-get-files --BK <BK path> --Run <list of runs> --Output
↳rdstToReprocess-<STRIPPING>.txt
```

4.2 Re-stage these files and their ancestors

```
dirac-transformation-add-files --File rdstToReprocess-<STRIPPING>.txt PreStage-with-
↳ancestors
```

or if this transformation doesn't exist:

```
dirac-dms-add-transformation --Plugin ReplicateWithAncestors --File rdstToReprocess-  
↳<STRIPPING>.txt --Name PreStage-with-ancestors --Start
```

4.3 Reset RDST files Unused in the stripping production

Get the status of those files in the stripping production, in case they are already set Problematic or NotProcessed:

```
dirac-transformation-debug --File rdstToReprocess-<STRIPPING>.txt --Status_  
↳Problematic,NotProcessed <STRIPPING>
```

Save the list of files that are in either of these statuses in order to reset them later

When you are sure no file is Assigned:

```
dirac-transformation-debug --File rdstToReprocess-<STRIPPING>.txt --Status Assigned  
↳<STRIPPING>
```

returns no files!

```
dirac-transformation-reset-files --File rdstToReprocess-<STRIPPING>.txt <STRIPPING>
```

and then reset Problematic files you have saved as such

```
dirac-transformation-reset-files --NewStatus Problematic --Term <STRIPPING>  
<paste the list of Problematic >  
  
dirac-transformation-reset-files --NewStatus NotProcessed --Term <STRIPPING>  
<paste the list of NotProcessed>
```

5. Make a few more checks

From time to time better do a further check that all is OK as you may have also other errors like files in FC without BK flag, in which case you should removed them:

```
dirac-production-check-descendants --File rdstToReprocess-<STRIPPING>.txt <STRIPPING>
```

6. Restart productions

Restart the stripping production first, do a few more checks as in 5. and when confident, restart the merging production

7. Not to be forgotten

At the end of the productions, one should remove from Tier1-Buffer the RDST files that have been re-processed as well as their RAW ancestors. This is an additional duty for the DM cleaning that anyway must take place. If all is OK it is enough to just remove all files and ancestors from Buffer. . .

2.12.2 DataManagement

Checking unfinished runs

The run is declared finished by the online data mover once all the files have been transferred. it is good to check from time to time that no runs are left behind:

We start from the Reconstruction production (here 67722)

```
[localhost] ~ $ dirac-bookkeeping-run-information --Prod 67722 --Info Finished --
↳ByValue
Found 347 runs
Successful :
    No : (36 runs) - 201402,201403,201640,201971,201972,202804,202805,202821,[.....↳
↳more that are active...]
```

You can also give a run range

```
[LHCbDirac prod] SRPM $ dirac-bookkeeping-run-information 205957:212325 --Information↳
↳Finished --ByValue
Successful :
    No : (1 runs) - 212325
    Yes : (2129 runs) - 205957,205958,[...]
```

A bit of cleaning

From StorageUsagePlot

From the StorageUsage plots, we can see that there are sometimes files left on BUFFER after a production has been finished.

To find them

```
[lxplus021] ~ $ dirac-dms-find-lfns --Path /lhcb/data/2015/RAW/SMOGPHY/LHCb/LEAD15/ --
↳SE Tier1-Buffer | dirac-dms-replica-stats
Got 3 LFNs
[...]
```

Very often, it is because the run has been flagged BAD. This can be checked as follows:

```
dirac-bookkeeping-file-path --GroupBy dataqualityflag --Summary <LFNS>
```

We make sure they were not processed before removing them

```
[lxplus021] ~ $ dirac-bookkeeping-get-file-descendants --Last
Got 3 LFNs
Getting descendants for 3 files (depth 1) : completed in 0.3 seconds
NotProcessed :
    /lhcb/data/2015/RAW/SMOGPHY/LHCb/LEAD15/169028/169028_0000000546.raw
    /lhcb/data/2015/RAW/SMOGPHY/LHCb/LEAD15/169030/169030_0000000354.raw
    /lhcb/data/2015/RAW/SMOGPHY/LHCb/LEAD15/169034/169034_0000000323.raw
[lxplus021] ~ $ dirac-dms-remove-replicas --Last --SE Tier1-Buffer
Got 3 LFNs
Removing replicas : completed in 8.6 seconds
Successfully removed 3 replicas from IN2P3-BUFFER
```

Files unused in productions

If a run is flagged BAD during the processing, some files may have been added to a production, but then never get used. A very similar process can be done. In case of stripping, we can also go up to the parent files to remove the RA files from buffer:

```
[localhost] ~ $ dirac-transformation-debug 71500 --Status Unused --Info files | dirac-
↪bookkeeping-get-file-ancestors | dirac-dms-replica-stats
Getting ancestors for 41 files (depth 1) : completed in 12.0 seconds
Got 108 LFNs
Getting replicas for 108 LFNs : completed in 5.8 seconds
108 files found with replicas

Replica statistics:
  0 archive replicas: 108 files
-----
  0 other replicas: 0 files
  1 other replicas: 41 files
  2 other replicas: 67 files
-----

SE statistics:
  CERN-RAW: 67 files
  CNAF-RAW: 67 files
  CNAF-RDST: 41 files

Sites statistics:
  LCG.CERN.cern: 67 files
  LCG.CNAF.it: 108 files
[localhost] ~ $ dirac-bookkeeping-file-path --Last --GroupBy dataqualityflag --Summary
Got 108 LFNs
Successful :
  DataqualityFlag BAD : 108 files
[localhost] ~ $ dirac-transformation-reset-files --New Removed --Last 71500
Got 108 LFNs
41 files were set Removed in transformation 71500
```

Files problematic in productions

When a file goes problematic in a production, it can be removed from buffer. If it is for the stripping, also its raw ancestor can be removed. Example for a Stripping (here, all the prods for a given Stripping)

```
[localhost] ~ $ dirac-transformation-debug 69077,69073,68675,68486,69079,69075,68773,
↪68771 --Status Problematic --Info files | dirac-dms-replica-stats
Got 28 LFNs
Getting replicas for 28 LFNs : completed in 4.7 seconds
28 files found with replicas

Replica statistics:
  0 archive replicas: 28 files
-----
  0 other replicas: 0 files
  1 other replicas: 0 files
  2 other replicas: 28 files
-----
```

(continues on next page)

(continued from previous page)

```
[...]

[localhost] ~ $ dirac-bookkeeping-get-file-ancestors --Last | dirac-dms-remove-
↳replicas --SE Tier1-Buffer
Getting ancestors for 28 files (depth 1) : completed in 6.5 seconds
Got 56 LFNs
Removing replicas : completed in 228.5 seconds
Successfully removed 12 replicas from CERN-BUFFER
Successfully removed 2 replicas from SARA-BUFFER
Successfully removed 6 replicas from RRCKI-BUFFER
Successfully removed 4 replicas from GRIDKA-BUFFER
Successfully removed 8 replicas from IN2P3-BUFFER
Successfully removed 24 replicas from RAL-BUFFER
```

We can then set these files as Removed in the removal transformation (setting them Done would not be very clean...)

```
[localhost] ~ $ dirac-transformation-reset-files --NewStatus Removed --Last 69128,
↳69127,68831,68829
Got 56 LFNs
6 files were set Removed in transformation 69128
7 files were set Removed in transformation 69127
8 files were set Removed in transformation 68831
7 files were set Removed in transformation 68829
```

From DEBUG

Here we can just clean regularly what is in debug...

```
dirac-dms-list-directory --Dir /lhcb/debug/ --Rec --Days 7 --NoDirectories | dirac-
↳transformation-add-files Remove-all-replicas
```

2.12.3 Containers

LHCb applications are ran in a containerised environment by passing the `--allow-containers` argument to `lb-run`. Before activating this feature the `GaudiExecution/lbRunOptions` configuration option under `Operations` must be set to include this argument, e.g. `--siteroot=/cvmfs/lhcb.cern.ch/lib/--allow-containers`.

Containerised LHCb applications can then be enabled or disabled at three levels of granularity. In order of preference these are: * For a single compute element by setting `/Resources/Sites/${SITE_TYPE}/${SITE}/CEs/${COMPUTE_ELEMENT}/AllowContainers` to `yes`. * For an entire site by setting `/Resources/Sites/${SITE_TYPE}/${SITE}/AllowContainers` to `yes`. * Globally by setting the `Operations` option `GaudiExecution/AllowContainers` to `yes`.

Currently `LbPlatformUtils` only supports [Singularity](#) however future releases may allow the `AllowContainers` option to be used to set which container technologies can be used.

2.13 LHCbDIRAC Logs

2.13.1 Job logs aka LogSE

The LogSE is used to store the log files from the production jobs. it is defined in */Operations/defaults/LogStorage/LogSE = LogSE-EOS*.

The content is exposed via the normal EOS protocols, but also via a [CERN web service](#). Any member of the lhcb-geoc group can [manage this web service](#). The reason for having this web service is to be able to use htaccess and php magic.

The aim of the php and htaccess magic is to transparently move from a log directory per job to one zip archive per job, and still expose the content the same way, allowing for easy browsing.

The *.htaccess*, *listzip.php* and *extract.php* are stored at the root of the logSE. The php scripts are shared with the Core soft team and available in [this repo](#).

The *.htaccess* content is pasted below:

```
# Option mandatory for CERN website exploration to work
Options +Indexes

# Allows rewrite rules
RewriteEngine On

# this is a clever trick to avoid RewriteBase
# see http://stackoverflow.com/a/21063276
# basically, {ENV:BASE} eval to the location of the htaccess
RewriteCond "%{REQUEST_URI}::$1" "^(.*?/)(.*)::\2$"
RewriteRule "^(.*)$" "-" [E=BASE:%1]

# These rules expect path that follows the convention
# <something>/LOG/<numbers>/<numbers>/<numbers>
# In practice, what we have is
# MC/2018/LOG/<prodNumber>/<first digits of the prod number>/<jobNumber>
# What we want to be able to do is to expose the exact same way the log files of a
↪ job:
# * stored in a directory <jobNumber>
# * stored in a zip file <jobNumber.zip>, containing itself a <jumbNumber> directory

# Aim: list the zip file as if it was a directory
# If the URL targets is a non existing directory
RewriteCond "%{DOCUMENT_ROOT}%{REQUEST_URI}" !-d
# if the url is if the form "<something>/LOG/<int>/<int>/<int>/ (Note the "/*: at the
↪ end)
# we redirect to "{ENV:BASE}/listzip.php?zip=<something>/LOG/<int>/<int>/<int>.zip
RewriteRule "^(.*?/LOG/[0-9]+/[0-9]+/[0-9]+)/$" "%{ENV:BASE}listzip.php?zip=$1.zip"
↪ [PT,B,L]

# Aim: extract artifacts from specific zip files
# If we have a URL targetting a non existing file
RewriteCond "%{DOCUMENT_ROOT}%{REQUEST_URI}" !-f
# if the url is if the form "<something>/LOG/<int>/<int>/<int>/<a path>
# we redirect to "{ENV:BASE}/extract.php?zip=<something>/LOG/<int>/<int>/<int>.zip&
↪ path=<int>/<a path>
RewriteRule "^(.*?/LOG/[0-9]+/[0-9]+/[0-9]+)/(.*)" "%{ENV:BASE}extract.php?zip=$1.
↪ zip&path=$2/$3" [PT,B,L]
```

2.13.2 Centralized logging

TL;DR

All the logs (up to the VERBOSE level) from the Agents and services are visible on [this kibana instance](#).

More details

Each and every component send their logs at the VERBOSE level in a message queue. This is configured using the [message queue backend](#), and the queue is described in the [MQServices resources](#)

The logs are then consumed by a logstash server (*lbdirclogstash{01,02}.cern.ch*), and forwarded to ElasticSearch. This is configured in the [ai-puppet-hostgroup-volhcb repository](#).

Data storage

We are using Elasticsearch (ES) to store the data, which is provided by IT [CERN centralized ES service](#). The ES configuration can be found in [it-elasticsearch-project repository](#).

Data Visualization

Kibana is used to visualize the data, which is accessible [in this link](#). To access the Kibana you have to be a member of lhcb-geoc egroup. There are predefined dash boards which you can access under Dash boards menu.

2.14 Message Queues for LHCb

The IT Message Queue service has setup two ActiveMQ brokers for us, configured the same way: *lhcb-test-mb* and *lhcb-mb*.

2.14.1 Queues and topics

This is the destinations that we are allowed:

```
/queue/lhcb.certification
/queue/lhcb.dirac.{path}
/queue/lhcb.jenkins
/queue/lhcb.test.{name}
/topic/lhcb.dirac.{path}
```

where *{path}* can be anything and *{name}* any word without dots.

2.14.2 Authentication & Authorizations

Authentication can be done via user name, or via host DN.

For convenience, we use a *dirac* user for every interaction. The IT guys have a list of our machines, but we have to ask them to update it every time we add a machine, so it is not very convenient.

2.14.3 Monitoring

We can see how are our queues doing on this [monitoring link](#)

cluster can be either *lhcb* or *lhcb-test*

2.15 Sandbox Store

Our Sandbox store is hosted on a VM (currently lbvobox112) and the files are stored on an external volume (currently mounted under */opt/dirac/sandboxStore/*)

2.15.1 How To resize the sandbox store volume

Resize the sandbox volume since it is full. The normal procedure can be found here: https://clouddocs.web.cern.ch/clouddocs/details/working_with_volumes.html

The volume has not been resized since the volume was moved to a 5TB volume with better IO performance.

2.16 Data Popularity

Whenever production data is accessed by a user job, we keep that information in order to estimate the popularity of the dataset.

- Freeing disk space
- Making data set consistent in terms of number of replicas
- Produce plots for the RRB

2.16.1 Components

The popularity analysis relies on a lot of components

StorageUsageDB

Despite its name, that is where both the StorageUsageAgent and the PopularityAgent stores their data. It is exposed via the StorageUsageHandler and the DataUsageHandler

StorageUsageAgent

This agent scans the DFC and stores the size and number of files per directory and per StorageElement in the StorageUsageDB.

StorageHistoryAgent

This agent crawls the StorageUsageDB, convert each directory into a bookkeeping path and fill in the following accounting:

- Storage: space used/free per storage and/or directory

- Data storage: spaced used per bookkeeping path
- user storage: like Storage, but for user directories

DataUsageHandler

This service is called by the jobs to declare their use of a given directory. It is stored per directory and per day.

PopularityAgent

This agent goes through the StorageUsageDB and creates accounting entries for the popularity. It also caches the BK dictionary for each directory in the StorageUsageDB.

DataPop server

Yandex provided service that consumes our popularity CSV and make prediction on which dataset to remove. It is ran on our mesos cluster: <https://lbmesosms02.cern.ch/marathon/ui/#/apps/%2Fdatapopserver>

PopularityAnalysisAgent

This agents creates two files:

- one CSV containing a summary of the popularity (see *popularity.csv file*).
- one CSV, generated from the first one through the DataPop server

2.16.2 Popularity scanning

This file is produced by the PopularityAgent and stored on its work directory.

To produce the *popularity.csv* file, the scanning follows this algorithm:

- list all the directories of the DFC
- Convert each visible directory into a BK dictionary, getting the information from the StorageUsage (if it is cached), or from the BK itself
- For each directory:
 - Get the number of PFNs and size per SE from the StorageUsageDB
 - Get the day by day usage of the directories and group them by week from the DataUsage (which is the StorageUsageDB..)
- Sum the size and the number of files per:
 - directory
 - storage type (Archive, tape, disk)
 - StorageElement
 - Site
- Assigns the dataset to a given storage type (see bellow)

2.16.3 popularity.csv file

This file is the output of all the processing chain. The fields are the following:

- Name: full Bookkeeping path (e.g. */LHCb/Collision11/Beam3500GeV-VeloClosed-MagDown/RealData/Reco14/Stripping20r1/90000000/SEMILEPTONIC.DST*)
- Configuration: Configuration part, so DataType + Activity (*/LHCb/Collision11*)
- ProcessingPass: guess... (*/RealData/Reco14/Stripping20r1*)
- FileType: guess again (*SEMILEPTONIC.DST*)
- Type: a number depending on the type of data:
 - 0: MC
 - 1: Real Data
 - 2: Dev simulation
 - 3: upgrade simulation
- Creation-week: “week number” when this file was created (see below for details) (e.g. *104680*)
- NbLFN: number of LFNs in the dataset
- LFNSize: size of the dataset in TB
- NbDisk: number of replicas on disk. Careful: if all LFNs have two replicas, you will have $NbDisk=2*NbLFN$.
- DiskSize: effective size on disk of the dataset in TB (also related to the number of replicas)
- NbTape: number of replicas on tape, which is not archive
- TapeSize: effective size on tape of the dataset in TB.
- NbArchived: number of replicas on Archive storage.
- ArchivedSize: effective size on Archive storage in TB
- CERN, CNAF,.. (all T1 sites): disk space used at the various sites.
- NbReplicas: average number of replicas on disk ($NbDisk/LFN$)
- NbArchReps: average number of replicas on Archive ($NbArchived/LFN$)
- Storage: one of the following:
 - Active: If the production creating this file is either idle, completed or active
 - Archived: if there are on disk copies, only archive
 - Tape: if dataset is on RAW or RDST
 - Disk: otherwise
- FirstUsage: first time the dataset was used (in “week number”)
- LastUsage: last time the dataset was used (in “week number”)
- Now: current week number
- 1, 2, etc: number of access since k weeks ago. Note that these numbers are cumulative, that means that what was accessed 1 week ago is also counted in what was included 2 weeks ago.

Week number

This allows to have an easy way to compare the age of datasets. It is defined as $year * 52 + week\ number$

2.16.4 Popularity Analysis

The analysis is performed based on an Excel spreadsheet available in the repository. This spreadsheet takes the raw data contained in the *popularity.csv* file and makes statistics on them.

Spreadsheet setup

For some obscure reason, the spreadsheet needs a bit of manual actions.

You first need to manually copy all the data from the popularity.csv file to the “Popularity data” tab of the spreadsheet, starting line 2. You can then extend the “Popularity Formulas” tab to the same number of lines. It would be nice if Excel was doing that automatically, but...

Spreadsheet content

The spreadsheet is divided into several tabs.

Popularity Data

Just a copy paste of the popularity.csv

Dataset statistics

The dataset statistics tab is a frequency table. The first column is the bin size, while all the others are the values inside the given bin.

The aim of this tab is to give a global overview of the number of replicas, archives, size, etc used per dataset

Popularity formulas

This tab crunches the popularity.csv data. It uses some value from other tabs as parameters. In particular:

- A1, name StorageType: type of storage we are doing statistics on. Normally, disk
- A2, named NbOfWeeks: taken from *PopularityPlots.L16*. It is the number of weeks on which we do our statistics

The fields are the following

- Disk: useless, place holder for A2
- Name: like popularity.csv
- Configuration: like popularity.csv
- ProcessingPass: like popularity.csv
- Storage: like popularity.csv
- NbLFN: like popularity.csv

- FileType: like popularity.csv
- Disk Real Data: DiskSize of popularity.csv if the dataset is real data
- Disk MC: DiskSize of popularity.csv if the dataset is MC or Dev
- Usage: takes the number of usage at NbOfWeeks from the popularity data
- Norm. Usage: defined as $Usage/NbLFN$ if the *Storage* is *StorageType*, -1 otherwise
- AgeWeeks: What it says ($Now - creationWeek$), if the *Storage* is *StorageType*. -1 otherwise
- Age Real Data: same as *AgeWeeks* if the dataset is real data, but in years (so divided by 52)
- Age MC: same as *AgeWeeks* if the dataset is MC or Dev, but in years
- Last usage in weeks: number of weeks since it has not been used ($Now - LastUsage$). Caution ! LastUsage is a week number in the popularity data
- Usage span: Number of weeks during which the dataset was used
- Age at last Usage: in years, only if the *Storage* is *StorageType*, -1 otherwise
- Age at first usage: in years, only if the *Storage* is *StorageType*, -1 otherwise
- Age of unused datasets: in years, if the data was never used the last *NbOfWeeks* weeks and if the *Storage* is *StorageType*. -1 otherwise
- Age of used datasets: in years, if the data was used the last *NbOfWeeks* weeks and if the *Storage* is *StorageType*. -1 otherwise
- Nb Replicas UnusedOld: Number of replicas of the dataset if it is unused and its older than *NbOfWeeks* ($Age\ of\ unused\ dataset > NbOfWeeks/52$) (folks from the Scrutiny group want that)
- OverSize: see below
- Archives Real Data: for real data on *StorageType*, this is the number of ArchReps (see *Nb ArchReps* below). -1 otherwise
- Archives MC: For MC or Dev data on *StorageType*, this is the number of ArchReps (see *Nb ArchReps* below). -1 otherwise
- (Rep-1)/Arch: see below
- (Rep-2)/Arch: see below
- Nb Replicas: like popularity.csv
- Nb ArchReps: like popularity.csv

A bit of math

There are a few formulas in the popularity that are useful to discriminate badly replicated datasets. Here is how:

In a dataset of *NbLFN* files, *N* will be correctly replicated to disks, and *n* will not be:

$$NbLFN = N + n$$

If we make the assumption that a file is either replicated the correct number of time, or not at all, you have:

$$\begin{aligned} NbReplicas &= (k*N + n) / (N + n) \\ NbArchRep &= N / (N + n) \end{aligned}$$

where k is the target number of replicas.

In the case where data has 2 disk copies and one archive, you can then compute the following:

```
(NbReplicas - 1)/NbArchRep = 1
(NbReplicas - 2)/NbArchRep = -n / N
```

This helps finding pathological datasets, as in ideal case, these values will respectively be 1 and 0.

In the old case where data has 3 disk copies and one archive, you can then compute the following:

```
(NbReplicas - 1)/NbArchRep = 2
(NbReplicas - 2)/NbArchRep = (N - n) / N
```

Ideally these values will respectively be 2 and 1.

Any other values would show that the dataset is not perfectly replicated.

Another interesting value to compute is *OverSize*. This is basically an estimate of how much space (TB) is uselessly consumed if we assume that a dataset that wasn't used during the *NbOfWeeks* period should have only 1 replicas:

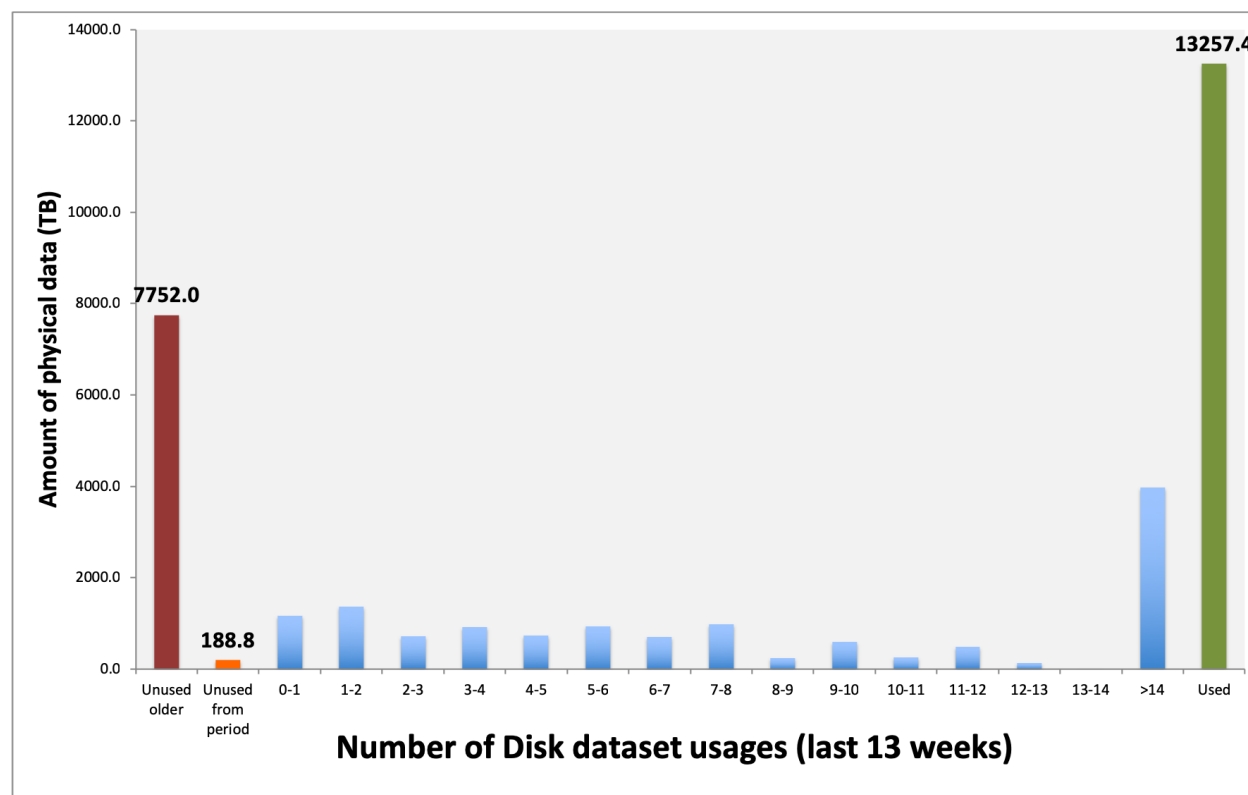
```
OverSize = (DiskSize)*([Nb Replicas UnusedOld]-1)/[Nb Replicas UnusedOld]
```

Popularity plots

This tab contains a lot of plots. It is a frequency table, just like the Dataset statistics tab, but containing data regarding the popularity and number of accesses.

RRD plots

The RRD plots are interested in a plot like the one bellow.



This translate how many TB on disks have been used 1, 2, ..., 14 times in the last n weeks. Note that these is physical size, so the number of replicas counts! There are two special bins:

- Unused older: these datasets were created before n weeks ago, and were not used in the last n weeks
- Unused from period: these datasets were created during the last n weeks

For a reason which is known to them only, but certainly is very well justified, they want these plots for 13, 26 and 52 weeks.

2.17 Installing LHCbWebAppDIRAC

The installation requires two steps:

1. Install the machine:

The machine should be installed using the `webportal` puppet template. LHCbWebAppDIRAC is using `Nginx` for better performance, which is also puppetized. The main configuration file used to install `Nginx` can be found in [this gitlab repository](#). The `site.conf` configuration file is used for handling the user requests and pass to the Tornado based LHCbWebAppDIRAC component. The configuration file can be found in [this repository](#).

1. Installing LHCbWebAppDIRAC extension:

```
cd /home/dirac
curl -O https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/Core/
scripts/install_site.sh
chmod +x install_site.sh
Edit install_site.sh add the Release = <version>
./install_site.sh install.cfg
```

Note: install.cfg file must exist in the /home/dirac directory, and it is also generated by puppet.

NOTES:

- You may need to execute the commands above more than once (for example if you change the certificate).
- Most probably, it may not work. Ask Joel to create the correct dirac.cfg file. The dirac.cfg file content must be the same as the existing web machine.

2.18 CERN centralized MySQL service

This document contains all information needed to manage the MySQL DBs. The service is provided by CERN IT in the “DBoD” service. General info can be found in this [documentation](#).

2.18.1 Communication channels:

CERN IT ES [homepage](#)

1. Tickets: open a [snow ticket](#).
2. Mattermost channel: LHCb [specific channel](#) or [IT general channel](#) (used also for Oracle).

Login to the database

How-To in [lbDevOps](#) doc.

2.19 CERN centralized Elasticsearch service

This document contains all information needed to manage the ES. The ES is provided by CERN IT. General info can be found in this [documentation](#).

2.19.1 Communication channels:

CERN IT ES [homepage](#)

1. Tickets: open a [snow ticket](#).
2. Mattermost channel: LHCb [specific channel](#) or [IT general channel](#).

2.19.2 Elasticsearch instances

We are using three production instances:

1. es-lhcb-monitoring for monitoring WMS and ComponentMonitoring
2. es-lhcb-dirac-logs for centralized Monitoring
3. es-lhcb-mcstats for MC statistics

and one for certification:

1. es-lhcb-dev for everything

The above are all ES6 instances.

2.19.3 Elasticsearch performance monitoring

IT/ES provides monitoring tool for monitoring ES instances. You can access [in the following link](#) (accessible from inside the CERN network).

2.19.4 Kibana

Kibana is used for visualize the data. IT/ES provides a Kibana end point for each ES instance. You can access using <https://instance/kibana> for example: <https://es-lhcb-monitoring.cern.ch/kibana>

All kibana interfaces are CERN internal web pages. If you are outside of the CERN network you need to follow [these instructions](#).

Note: You can access to kibana, if you are in one of the group: lhcb-dirac, lhcb-geoc, lhcb-gridshifters

2.19.5 Managing ES templates

Each ES instance has a dedicated template, what you can found in the [repository](#) by searching lhcb. For example for the logs.

2.19.6 Curator

Curator can be used for easily manage ES data. It can be used in different purpose. We are using it for deleting indexes that are older a certain date. To setup Curator you need to use the ES template repository (see Managing ES templates section.) and create *curator4.actions* file. For example: [deleting indexes older a certain period](#).

2.19.7 Re-indexing existing index

You may need to re-index indexes from one cluster to another. You can use the [following script to reindex](#).

2.20 Setup a new vbox

In order to install a vbox you need a configuration file so called install.cfg. If the machine correctly created, this configuration file will be in /home/dirac directory.

Please follow [this instructions](#) and use the configuration file from /home/dirac/install.cfg

Make sure the dirac.cfg file is correctly created in the machine.

2.21 Admin Command Reference

In this subsection the Admin commands are collected

2.22 Full Configuration Example

Below is a complete example configuration with anotations for some sections:

```

Operations
{
  Defaults
  {
    Bookkeeping
    {
      AnOption = True
    }
  }
}
Systems
{
  Bookkeeping
  {
    Services
    {
      BookkeepingManager
      {
        Port = 9167
        Authorization
        {
          Default = all
        }
      }
    }
  }
}
DataManagement
{
  Services
  {
    DataIntegrity
    {
      Port = 9150
      Authorization
      {
        Default = all
      }
    }
    RAWIntegrity
    {
      Port = 9198
      Authorization
      {
        Default = all
      }
    }
    StorageUsage
    {
      BasePath = storage
      Port = 9151
      Authorization
      {
        Default = all
      }
    }
    DataUsage
    {

```

(continues on next page)

(continued from previous page)

```

    Port = 9164
    Authorization
    {
        Default = all
    }
}
Agents
{
    PopularityAgent
    {
        PollingTime = 43200
    }
    PopularityAnalysisAgent
    {
        #every week
        PollingTime = 604800
        AnalysisPeriod = 910
        TopDirectory = '/lhcb'
        DataPopularityURL = 'http://localhost:5000'
        MailRecipients = 'lhcb-datamanagement@cern.ch'
        MailEnabled = True
        SavedSpaceTarget = 100
        MinReplicas = 1
        MaxReplicas = 7
    }
    RAWIntegrityAgent
    {
        PollingTime = 60
    }
    SEUsageAgent
    {
        PollingTime = 43200
        MaximumDelaySinceStorageDump = 86400
    }
    StorageHistoryAgent
    {
        PollingTime = 43200
    }
    StorageUsageAgent
    {
        PollingTime = 43200
        BaseDirectory = /lhcb
        Ignore = /lhcb/user
        Ignore += /lhcb/test
    }
    TargzJobLogAgent
    {
        PollingTime = 3600
        Actions = Jobs
        JobAgeDays = 14
        ProdAgeDays = 14
        ProductionGlob = 000?????
    }
    UserStorageQuotaAgent
    {
        PollingTime = 172800
    }
}

```

(continues on next page)

(continued from previous page)

```

    }
    UserStorageUsageAgent
    {
        PollingTime = 43200
        ProxyLocation = runit/DataManagement/UserStorageUsageAgent/proxy
        UseProxies = True
        BaseDirectory = /lhcb/user
    }
}
}
ProductionManagement
{
    Services
    {
        MCStatsElasticDB
        {
            Port = 9177
            Authorization
            {
                Default = authenticated
            }
        }
    }
    ProductionRequest
    {
        Port = 9188
        Authorization
        {
            Default = all
        }
    }
    TornadoAnalysisProductions
    {
        Protocol = https
        Authorization
        {
            Default = authenticated
            registerTransformations = ProductionManagement
            deregisterTransformations = ProductionManagement
            registerRequests = ProductionManagement
            archiveSamples = ProductionManagement
            setState = ProductionManagement
        }
    }
}
Agents
{
    APSyncAgent
    {
        PollingTime = 300
    }
    ProductionStatusAgent
    {
        PollingTime = 120
    }
    RequestTrackingAgent
    {
        PollingTime = 120
    }
}

```

(continues on next page)

(continued from previous page)

```

    }
    NotifyAgent
    {
        PollingTime = 1800
    }
}
ResourceStatus
{
    Agents
    {
        NagiosTopologyAgent
        {
            PollingTime = 3000
            DryRun = True
        }
        LHCbPRProxyAgent
        {
            PollingTime = 3600
        }
        ShiftDBAgent
        {
            PollingTime = 3600
        }
        EmailAgent
        {
            PollingTime = 1800
        }
    }
}
Transformation
{
    Services
    {
        TransformationManager
        {
            Port = 9131
            HandlerPath = LHCbDIRAC/TransformationSystem/Service/
↪TransformationManagerHandler.py
            Authorization
            {
                Default = all
            }
        }
    }
    Agents
    {
        DataRecoveryAgent
        {
            PollingTime = 120
        }
        BookkeepingWatchAgent
        {
            PollingTime = 120
        }
        MCSimulationTestingAgent
        {

```

(continues on next page)

(continued from previous page)

```
        PollingTime = 300
    }
}
WorkloadManagement
{
    Services
    {
        WMSecureGW
        {
            Port = 3424
            HandlerPath = LHCbDIRAC/WorkloadManagementSystem/Service/WMSecureGWHandler.py
            Authorization
            {
                Default = all
            }
        }
    }
    Agents
    {
        BKInputDataAgent
        {
            PollingTime = 30
        }
        AncestorFilesAgent
        {
            PollingTime = 30
        }
    }
}
}
```


3.1 LHCbDIRAC Certification (development) Releases

See *LHCbDIRAC Releases*.

3.2 DBs in certification

How-To in lbDevOps doc.

3.2.1 ElasticSearch

All ES data for certification is stored in **es-lhcb-dev.cern.ch** at port 9203, and visualized via kibana in [`https://es-lhcb-dev.cern.ch/kibana/app/kibana#/home?_g=\(\)`](https://es-lhcb-dev.cern.ch/kibana/app/kibana#/home?_g=())

<TODO: expand on connection details>

4.1 Systems

4.1.1 AccountingSystem

SubPackages

Client

SubPackages

Types

Modules

DataStorage

JobStep

Popularity

Storage

UserStorage

private

SubPackages

Plotters

Modules

DataStoragePlotter

JobStepPlotter

PopularityPlotter

StoragePlotter

UserStoragePlotter

4.1.2 BookkeepingSystem

SubPackages

Client

Modules

BKQuery

BaseESClient

BaseESManager

BookkeepingClient

Help

IndentMaker

JEncoder

LHCB_BKKDBClient

LHCBBookkeepingManager

ScriptExecutors

objects

DB

Modules

DataTakingConditionInterpreter

OracleBookkeepingDB

OracleDB

Utilities

Service

SubPackages

XMLReader

SubPackages

Job

Modules

DataTakingConditions

File

FileParam

InputFile

Job

JobConfiguration

JobOption

JobParameters

Quality

QualityParameters

SimulationConditions

Replica

Modules

FileReplica

Replica

ReplicaParam

Modules

JobReader

ReplicaReader

XMLFilesReaderManager

Modules

BookkeepingManagerHandler

4.1.3 ConfigurationSystem

SubPackages

Client

SubPackages

Helpers

Modules

Resources

4.1.4 DataManagementSystem

SubPackages

Agent

SubPackages

RequestOperations

Modules

LogUpload

WMSSecureOutputData

Modules

PopularityAgent

PopularityAnalysisAgent

RAWIntegrityAgent

SEUsageAgent

StorageHistoryAgent

StorageUsageAgent

UserStorageQuotaAgent

UserStorageUsageAgent

Client

Modules

AddTransformation

CheckExecutors

ConsistencyChecks

DMScript

DataIntegrityClient

DataUsageClient

ScanPopularity

ScriptExecutors

StorageUsageClient

DB

Modules

RAWIntegrityDB

StorageUsageDB

Service

Modules

DataUsageHandler

RAWIntegrityHandler

RunDBInterfaceHandler

StorageUsageHandler

Utilities

Modules

FCUtilities

private

SubPackages

FTS3Plugins

Modules

LHCbFTS3Plugin

4.1.5 FrameworkSystem

SubPackages

DB

Modules

ProxyDB

4.1.6 ProductionManagementSystem

SubPackages

Agent

Modules

APSyncAgent

DataProcessingProgressAgent

NotifyAgent

ProductionStatusAgent

RequestTrackingAgent

Client

Modules

AnalysisProductionsClient

MCStatsClient

ProcessingProgress

Production

ProductionRequest

ProductionRequestClient

DB

Modules

AnalysisProductionsDB

AnalysisProductionsObjects

ElasticApplicationSummaryDB

ElasticGeneratorLogDB

ElasticMCBooleLogErrorsDB

ElasticMCGaussLogErrorsDB

ElasticMCStatsDBBase

ElasticPrMonDB

ProductionRequestDB

Service

Modules

MCStatsElasticDBHandler

ProductionRequestHandler

TornadoAnalysisProductionsHandler

Utilities

Modules

APUtils

ModelCompatibility

Models

StateMachine

Utils

4.1.7 ResourceStatusSystem

SubPackages

Agent

Modules

LHCbPRProxyAgent

NagiosTopologyAgent

ShiftDBAgent

Policy

Modules

Configurations

DownHillPropagationPolicy

JobWebSummaryEfficiencyPolicy

TransferQualityPolicy

4.1.8 TransformationSystem

SubPackages

Agent

Modules

BookkeepingWatchAgent

DataRecoveryAgent

MCExtensionAgent

MCSimulationTestingAgent

TransformationAgent

TransformationCleaningAgent

TransformationPlugin

ValidateOutputDataAgent

WorkflowTaskAgent

Client

SubPackages

BodyPlugin

Modules

DataChallengeReplicationBody

RAWReplicationBody

Modules

TaskManager

Transformation

TransformationClient

TransformationDebug

DB

Modules

TransformationDB

Service

Modules

TransformationManagerHandler

Utilities

Modules

PluginScript

PluginUtilities

ScriptUtilities

StateMachine

4.1.9 WorkloadManagementSystem

SubPackages

Agent

Modules

AncestorFilesAgent

SiteDirector

DB

Modules

JobDB

Service

Modules

WMSSecureGWHandler

4.2 Other

4.2.1 Core

SubPackages

Utilities

Modules

File

GangaDataFile

GeneratorLog

HTML

InputDataResolution

LogErr

NagiosConnector

OutputDataPolicy

ProductionData

ProductionOptions

ResolveSE

RunApplication

XMLErr

XMLSummaries

XMLTreeParser

4.2.2 Interfaces

SubPackages

API

Modules

DiracLHCb

DiracProduction

LHCbJob

4.2.3 Resources

SubPackages

Catalog

Modules

BookkeepingDBClient

PoolXMLFile

RAWIntegrityClient

Storage

Modules

LHCbOnlineStorage

4.2.4 Workflow

SubPackages

Modules

Modules

AnalyseFileAccess

AnalyseXMLSummary

BookkeepingReport

CreateDataFile

ErrorLogging

FailoverRequest

FileUsage

GaudiApplication

LHCbScript

MergeMDF

ModuleBase

ModulesUtilities

RemoveInputData

StepAccounting

UploadLogFile

UploadMC

UploadOutputData

UserJobFinalization

`mock_Commons`

5.1 Bookkeeping Command Reference

In this subsection the Bookkeeping commands are collected

5.2 Data Management Command Reference

In this subsection the Data Management commands are collected

5.3 Workload Management Command Reference

In this subsection the Workload Management commands are collected

5.4 HOW-TO Guides

This section lists how-to and FAQ

5.4.1 DataManagement

For an introduction about DataManagement concepts, please see the [introduction](#)

All the commands mentioned below can accept several StorageElements and LFNs as parameters. Please use *-help* for more details.

Basics

Check if a file is corrupted

This question normally arises when a job spits lines like:

```
Error in <TBufferFile::CheckByteCount>: object of class LHCb::PackedRelation read too
↳few bytes: 2 instead of 1061133141
Error in <TBufferFile::CheckByteCount>: Byte count probably corrupted around buffer
↳position 21698:
    1061133141 for a possible maximum of -6

Error in <TBufferFile::ReadClassBuffer>: class: DataObject, attempting to access a
↳wrong version: -25634, object skipped at offset 4044
```

Or something like:

```
R_unzipLZMA: error 9 in lzma_code
Error in <TBasket::ReadBasketBuffers>: fNbytes = 28617, fKeylen = 92, fObjlen =
↳103039, noutot = 0, nout=0, nin=28525, nbuf=103039
```

We know that there are bugs in some applications that produce files that it then can't read, but from the pure data management point of view, we consider the file good if the checksum stored in the catalog and the actual file checksum match. To check it, we need to download the file locally, compute the checksum, and compare it with the DFC.

For example:

```
# Copy the file locally
# (the URL can be obtained from the failing job, or from dirac-dms-lfn-accessURL, or
↳one can even download the file with dirac-dms-get-file)

bash-4.2$ gfal-copy root://xrootd.grid.surfsara.nl//pnfs/grid.sara.nl/data/lhcb/LHCb-
↳Disk/lhcb/LHCb/Collision18/CHARMCHARGED.MDST/00077052/0000/00077052_00008134_1.
↳charmcharged.mdst .      Copying root://xrootd.grid.surfsara.nl//pnfs/grid.sara.nl/
↳data/lhcb/LHCb-Disk/lhcb/LHCb/Collision18/CHARMCHARGED.MDST/00077052/0000/00077052_
↳00008134_1.charmcharged.mdst
[...]
```

```
Copying root://xrootd.grid.surfsara.nl//pnfs/grid.sara.nl/data/lhcb/LHCb-Disk/lhcb/
↳LHCb/Collision18/CHARMCHARGED.MDST/00077052/0000/00077052_00008134_1.charmcharged.
↳mdst... 69s 100% [====>]Copying root://xrootd.grid.surfsara.nl//pnfs/grid.sara.nl/
↳data/lhcb/LHCb-Disk/lhcb/LHCb/Collision18/CHARMCHARGED.MDST/00077052/0000/00077052_
↳00008134_1.charmcharged.mdst [DONE] after 69s

# compute the checksum
bash-4.2$ xrdadler32 00077052_00008134_1.charmcharged.mdst
7f84828f 00077052_00008134_1.charmcharged.mdst

# Compare it with the DFC
bash-4.2$ dirac-dms-lfn-metadata /lhcb/LHCb/Collision18/CHARMCHARGED.MDST/00077052/
↳0000/00077052_00008134_1.charmcharged.mdst
Successful :
    /lhcb/LHCb/Collision18/CHARMCHARGED.MDST/00077052/0000/00077052_00008134_1.
↳charmcharged.mdst :
        Checksum : 7f84828f
        ChecksumType : Adler32
        CreationDate : 2018-08-16 02:53:48
        FileID : 368851163
```

(continues on next page)

(continued from previous page)

```
GID : 2749
GUID : 4606A403-E5A0-E811-ACFA-001B21B993CC
Mode : 775
ModificationDate : 2018-08-16 02:53:48
Owner : fstagni
OwnerGroup : lhcb_data
Size : 5059895291
Status : AprioriGood
UID : 19727
```

If the checksums don't match, the file needs to be recovered. If it is your own user file, do as you please (remove, recreate, replicate, etc). If it is centrally managed, please contact `lhcb-datamanagement` mailing list

CHAPTER 6

Documentation sources

<i>Certification</i> How to run the certification process	<i>Developers Guide</i> Adding new functionality to DIRAC
<i>Administrator Guide</i> Administration of the DIRAC services (server installations)	<i>Code Documentation (master)</i> Code reference
<i>User Guide</i> Using the DIRAC services	

CHAPTER 7

Indices and tables

- `genindex`
- `search`