Contents

1. Introduction

This document focuses on using TOTCSI (TOTEM Config Splitter Improved) for easy splitting and submitting the jobs to computer cluster. It shows step by step which TOTCSI commands should the user call and which files may be of interest.

2. Prerequisites

There are few requirements to be met before starting working with TOTCSI.

- sufficiently large directory on AFS (more than 700MB) for CMSSW and TOTCSI workspace
  - TOTEM scratch space (/afs/cern.ch/exp/totem/scratch/)
  - user's work directory (i.e. /afs/cern.ch/work/l/lgrzanka) works fine (it can have up to 100GB of space)
- jobs have to be submitted to lxbatch cluster from lxplus machines

3. Map-reduce approach

TOTCSI is based on map-reduce approach, which consists of two parts (see figure 1).

- Map (parallel)
  - apply-to-all function, performing a given operation on each element of an input list
  - often gives the possibility to parallelize computing
  - it is possible thanks to the splitting of the configurations
  - example: for each job configuration (input) submit it to LFS and fetch (operation) the results (output)
- Reduce (sequential)
  - fold function, performing a given combine operation on the input list (which is often the result of a Map function)
  - almost always sequential
  - at most times it is a merge operation
  - example: merge (operation) given .root files (input) into single one (output)

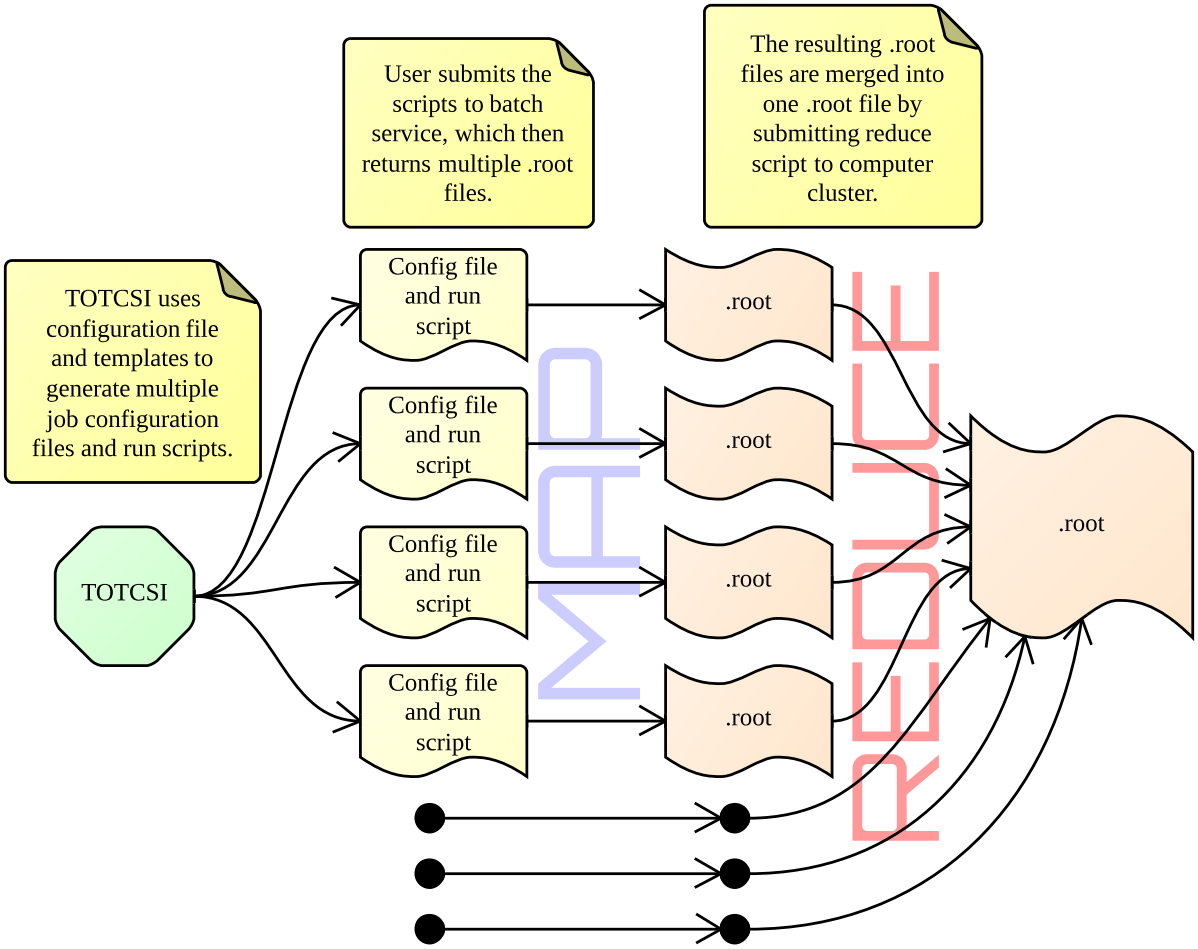

Figure 1: Map-reduce approach

4. TOTCSI versions

Current TOTCSI version is 2.0.
List of versions:

- version 1.0 (released 13.09.2012):
  - include splitting and job submission for simple cases of MC simulation and data reconstruction
  - tests of existence of generated data
  - tests of configuration and templates correctness by running local cmsRun process
- version 1.1 (released 24.10.2012)
  - contains few bug fixes, corrected examples and improved documentation
  - limitation 1: few jobs per single partial raw file - not implemented yet
  - limitation 2: usage of two instances of TOTCSI in parallel is causing problems (i.e. not possible to run in a sequence: split in directory A, split in directory B and then

submit_map in directory A), split, and two submit commands needs to be run in a sequence

  - limitation 3: map tuning function, needed to adjust configuration for many runs with slightly different options - not working correctly

- version 2.0 (released 18.01.2012)
  - removed limitations 1-3, known in version 1.1
  - changed workspace structure
  - rebuilt splitting module completely
  - new set of commands replaced old ones

5. Fetching needed files

Before you can use TOTCSI you have to obtain necessary files and prepare the environment for it.

5.1. Setting up CMSSW

First fetch the CMSSW private work directory.

```
cd /afs/cern.ch/work/i/ijurkows/private
svn co svn+ssh://svn.cern.ch/reps/totem/trunk/offline/cmssw/src CMSSW_4_2_4/src
```

After that follow instruction https://twiki.cern.ch/twiki/bin/view/TOTEM/CompOfflineGettingStartedCMSSW424 to compile CMSSW.

5.2. Fetching TOTCSI

You can use TOTCSI as a service or you can copy it to your directory on AFS.

- TOTCSI as service
  - direct all the calls through
    ```
    /afs/cern.ch/exp/totem/soft/TOTCSI/totcsi
    ```
- getting a copy of TOTCSI (mostly for developers)
  - copy from AFS
    ```
    cp -Lrf /afs/cern.ch/exp/totem/soft/TOTCSI config_splitter
    ```
  - project trunk
    ```
    svn co svn+ssh://svn.cern.ch/reps/totem/trunk/offline/cmssw/tools/config_splitter
    ```
  - tag for version TOTCSI 1.0
    ```
    svn co svn+ssh://svn.cern.ch/reps/totem/tags/TOTCSI/TOTCSI_1_0 config_splitter
    ```
  - tag for version TOTCSI 1.1
    ```
    svn co svn+ssh://svn.cern.ch/reps/totem/tags/TOTCSI/TOTCSI_1_1 config_splitter
    ```
  - tag for version TOTCSI 2.0
    ```
    svn co svn+ssh://svn.cern.ch/reps/totem/tags/TOTCSI/TOTCSI_2_0 config_splitter
    ```

From now on we will consider the directory *config_splitter* to be the main TOTCSI directory.

6. Configuring TOTCSI

You can find the example configurations (basic usage) by going into:

```
cd config_splitter/examples/configurations
```

And the example templates for CMSSW by going into:

```
cd config_splitter/examples/templates
```

For advanced configuration options please go to appendices: *general configuration*, *reconstruction configuration*, *simulation configuration*. For some more example configurations see *appendix D*.

6.1. Mandatory changes in TOTCSI configuration

Some parts of the configuration have to be configured so that TOTCSI can work properly. All other configurables are optional.

- type of task user wants to do (either "Reconstruction" or "Simulation")
  ```
  config.task_type
  ```
- path to compiled CMSSW should be set according to user's preferences (the path from example configurations won't work)
  ```
  config.cmssw_dir
  ```
- path to directory for parallel1 output (although it's strongly suggested to specify the sequential output path as well), it will most likely be a CASTOR path
  ```
  config.parallel1_output
  config.sequential_output
  ```
- paths to parallel1 templates (again it's suggested to specify them for sequential phase as well)
  ```
  # for reconstruction it has be only one path
  # for simulation user can specify many paths, but same amount in every phase
  config.input_config.parallel1_path
  config.input_config.sequential_path
  ```
- for simulation the number_of_events and number_of_jobs have to be specified
  ```
  config.simulation.number_of_events = 1500
  config.simulation.number_of_jobs = 10
  ```
- path to main workspace directory, all important files will be stored under this directory in subdirectory specified by the name of configuration file
  ```
  config.workspace.root_dir
  ```

From this moment we will consider *workspace_directory* to be our root_dir. The workspace_directory will have similar constuction to the one show on figure 2.

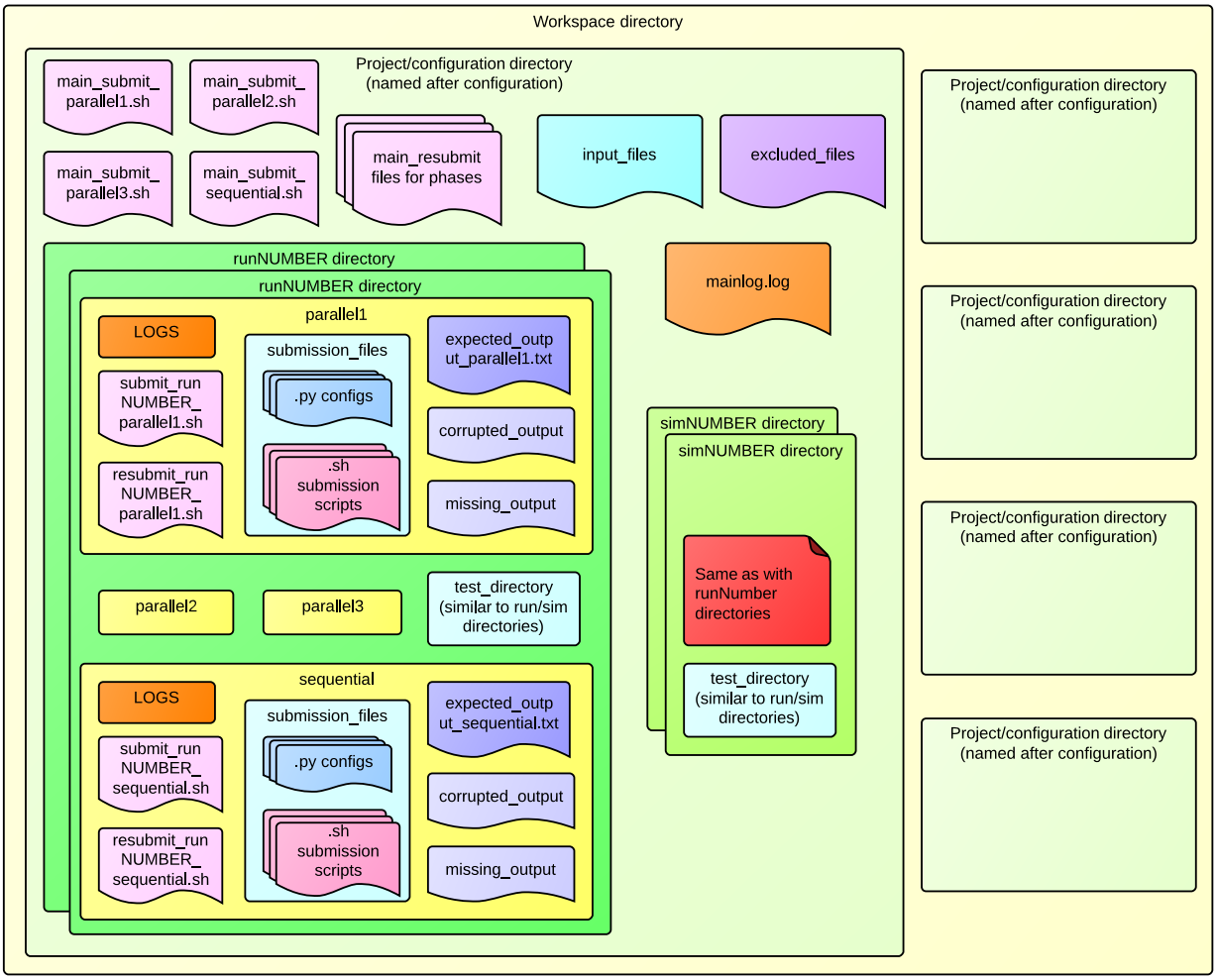

Figure 2: Workspace structure

6.2. Mandatory changes in CMSSW template

Some fragments of the job configuration file have to be changed in order to create a template understandable by TOTCSI.

- **{{number_of_events}}**
  Tag will be replaced by the number declared in config.simulation.number_of_events (simulation) or config.reconstruction.events_per_file_to_reconstruct (reconstruction). One should change

```
process.maxEvents = cms.untracked.PSet(
    input = cms.untracked.int32(-1)
)
```

  into

```
process.maxEvents = cms.untracked.PSet(
    input = cms.untracked.int32({{number_of_events}})
)
```

- **{{output|name("name_base")}}** (for standard output files)
  {{output|ntuple_name("name_base")}} (for NTuple output files)
  Tags will be replaced by the output file names generated for each job. One should change

```
process.TotemNtuplizer.outputFileName = "ntuple_8372.root"
```

to

```
process.TotemNtuplizer.outputFileName = "{{output|ntuple_name("TotemNTuple")}}"
```

- **{{input}}**
  Tag will be replaced by the appends to process.source.fileNames. One should change

  ```
  process.source.fileNames.append("/castor/cern.ch/.../map/reco_8372.part1.root"')
  ```

  to

  ```
  {{input}}
  ```

- **{{skipped_events}}**
  Tag will be replaced by the number of events to be skipped (counting from the beginnig of a file. Following line should be added in configuration template after cms.Source initialization:

  ```
  process.source.skipEvents = cms.untracked.uint32({{skipped_events}})
  ```

7. Using TOTCSI

This section will show you how to use the TOTCSI commands and what to expect of them. Basic syntax for calling the TOTCSI is:

```
./config_splitter/totcsi command [options]
```

7.1. TOTCSI launch options

There is a number of options available for TOTCSI, out of which some are optional and some mandatory.

- −h/−−help — this option shows you the usage options for TOTCSI (useful, when you are not sure what to call)
- −c/−−config <path>— ***mandatory*** option to which you add path to the TOTCSI configuration you will be using for this TOTCSI call (the subdirectory's name in workspace_directory will be taken from the configuration)
- −q/−−quiet — stops printing messages about current state of TOTCSI
- −d/−−debug — adds additional information when an exception occurs
- −f/−−force — allows the usage of already used directories in workspace_directory (be careful, files from directory with same name as configuration might get replaced by new ones with this option)

7.2. TOTCSI commands

There are currently 9 commands, from which 4 of are similar in design and results provided.

- readCastor (rc) — Looks through CASTOR TOTEM directory tree (except backups) and creates a file (named CASTOR_paths.txt) in workspace_directory containing paths to all .vmea files found.
- findPaths (fp) — Creates a file (named input_files) with list of paths to files, which user specified in configuration (config.reconstruction.input_data). The file will be located in workspace_directory's subdirectory named after configuration.
- generateTests (gt) — Creates sub-directories and files for testing the configurations (user can run CMSSW locally through provided shell scripts).
- generateSubmission (gs) — Generates needed CMSSW configurations and submission scripts (in workspace_directory's subdirectory named after configuration) for user to submit jobs to computer cluster.
- checkParallel1 (cp1) — Checks the output for parallel1 phase and creates files for use by generateResubmission command.
- checkParallel2 (cp2) — Checks the output for parallel2 phase and creates files for use by generateResubmission command.
- checkParallel3 (cp3) — Checks the output for parallel3 phase and creates files for use by generateResubmission command.
- checkSequential (cs) — Checks the output for sequential phase and creates files for use by generateResubmission command.
- generateResubmission (gr) — Creates resubmission scripts similar to those created by generateSubmission script, which can be used by user to resubmit jobs to computer cluster.

7.3. Examples of usage

We will now proceed with examples of usage. Our workspace directory will be named *workspace_directory*, configuration will be named *our_conf.py* and will be in **config_splitter/examples/configurations directory**, output directory for all phases *output_dir* and we will be calling TOTCSI by

```
./totcsi command [options]
```

as we are in main config_splitter directory.

If you want to see all commands just write down:

```
./totcsi --help
```

7.3.1. readCastor

This command will be needed only rarely (when something on CASTOR changes) and only for reconstruction. First call the command.

```
./totcsi -c examples/configurations/our_conf.py readCastor
```

This will take some time, after TOTCSI finishes you can look into your workspace_directory to see the CASTOR_paths.txt file. It will be used if we proceed with configuration for reconstruction.

7.3.2. findPaths

Again this command will have to be used only when doing reconstruction and even then only when the config.reconstruction.input_data has something else than direct paths to input files. Call the command.

```
./totcsi -c examples/configurations/our_conf.py findPaths
```

This will generate the input_files file in directory **workspace_directory/our_conf** . Check the directory and the file, then return to config_splitter's directory.

```
cd workspace_directory/our_conf
vi input_files
cd -
```

7.3.3. generateTests

This command will generate small CMSSW configuration files and bash scripts for local running of the cmsRun command. The files can be found in test_directories .

```
./totcsi -c examples/configurations/our_conf.py generateTests
```

The test_directories are created in subdirectories, which reside in **workspace_directory/our_conf** directory. If you are running reconstruction for run 8331, then the sub-directory's path will be **workspace_directory/our_conf/run8331** and test_directory will reside in it. test_directory has same structure as it's parent (except for test_directory, of course).

You can run the local script by entering any test_directory's sub-directory (parallel1, parallel2, parallel3 or sequential) and running the shell script that was created in there.
Say we called generateTests and are currently inside **run8331/test_directory/parallel1** directory. We can call the tests with:

```
./test_run8331_parallel1.sh
```

This will produce log files and output (if the script has no errors), which can be viewed in test_directory as well.

7.3.4. generateSubmission

This command will work similar to generateTests, however it will produce a number of bash scripts at different levels, as you can see in figure 2

```
./totcsi -c examples/configurations/our_conf.py generateSubmission
```

Go through **workspace_directory/our_conf** and see yourself what was created. The expected_output.txt files are used for later checking of the output. The created .py configurations are for CMSSW. The shell scripts residing inside the **workspace_directory/our_conf** directory will cause submission of all jobs related to given phase (which can be read from script's name). Sub-directories have in each phase folder another bash script containing a list of bsub calls. The

files which are to be submitted to computing cluster are located in submission_directories in each of phase directories.

Now submit the parallel1 phase to cluster:

```
cd workspace_directory/our_conf
./main_submit_parallel1.sh
```

You will see a number of jobs being submitted to computing cluster. You can check their status by calling

```
bjobs
```

After the jobs finish, you can list them by using command:

```
nsls -l output_dir
```

7.3.5. checkParallel1, checkParallel2, checkParallel3, checkSequential

All four of check functions have similar target. They check the expected_output files for given phase and look for the files in the output location. If the output files are corrupted, their list is written to corrupted_output document inside specific (for that output) sub-directory of **workspace_directory/our_conf**. For example for run 8331 if we had corrupted file in phase parallel1 the corrupted_output document would reside in **workspace_directory/our_conf/run8331/parallel1** directory. Same behaviour have two another files created during this TOTCSI command – missing_output (list of output files that are missing for specific sub-directory) and configs_for_resubmission which is used in generateResubmission command.

Now call the checkParallel1 command:

```
./totcsi -c examples/configurations/our_conf.py checkParallel1
```

The command will show you the formatted list of files that are missing or corrupted (if there are any). If everything went okay you can try submitting files again, killing some of the jobs with command

```
bkill <job_number>
```

and calling checkParallel1 again. See what the results are now.

7.3.6. generateResubmission

Last command creates the two levels of resubmission shell scripts (if of course they are needed) based on configs_for_resubmission files provided by the checkParallel1, checkParallel2, checkParallel3 or checkSequential commands. The scripts act exactly in the same way as the scripts from generateSubmission command, however they will consist only of lines which are related to missing or corrupted data's configurations.

Call the command:

```
./totcsi -c examples/configurations/our_conf.py generateResubmission
```

After TOTCSI finishes it's work see if there are any new files in **workspace_directory/our_conf**. If during any of "check" commands there was information regarding the missing or corrupted file, then there should be new resubmission script in workspace. Say

we have missing two output files of parallel1 phase in run 8331 and one output file of parallel2 phase in run 8332. If checkParallel1 and checkParallel2 were called, then the generateResubmission should produce two scripts in **workspace_directory/our_conf** – main_resubmit_parallel1.sh and main_resubmit_parallel2.sh, resubmit_run8331_parallel1.sh file in **workspace_directory/our_conf/run8331/parallel1** and resubmit_run8332_parallel2.sh in **workspace_directory/our_conf/run8332/parallel2**.

Try calling any of the main_resubmit scripts, if they are present (if not, you can always try out killing jobs after submission and calling check after that). If we have missing files in parallel1 phase call:

```
cd workspace_directory/our_conf
./main_resubmit_parallel1.sh
```

You will see that the amount of jobs submitted to cluster will be same as the number of configurations in all configs_for_resubmission files of all parallel1 phase sub-directories.

Appendices

A. General configurables

A.1. General parameters

A.1.1. config.task_type

Type of task to be run

Values:

- "Reconstruction"
- "Simulation"

It has to be set by user!

```
# Example:
config.task_type = "Reconstruction"
```

A.1.2. config.cmssw_dir

CMSSW location (full path), beginning with "/"

Has to be given by user!

```
# Example:
config.cmssw_dir = "/afs/cern.ch/exp/totem/scratch... \
                       ...Release/raw_data_reco/rev5679/CMSSW_4_2_4"
```

A.1.3. config.parallel1_output

Output directory for files (logs, results, configurations) for parallel1 phase

Can be CASTOR, AFS and EOS path

It has to be set by user!

```
# Example:
config.parallel1_output = "/castor/cern.ch/user/.../Parallel1OutputFolder"
```

A.1.4. config.parallel2_output

Output directory for files (logs, results, configurations) for parallel2 phase

Can be CASTOR, AFS and EOS path

It has to be set by user!

```
# Example:
config.parallel2_output = "/castor/cern.ch/user/.../Parallel2OutputFolder"
```

A.1.5. config.parallel3_output

Output directory for files (logs, results, configurations) for parallel3 phase

Can be CASTOR, AFS and EOS path

It has to be set by user!

```
# Example:
config.parallel3_output = "/castor/cern.ch/user/.../Parallel3OutputFolder"
```

A.1.6. config.sequential_output

Output directory for files (logs, results, configurations) for sequential phase

Can be CASTOR, AFS and EOS path

It has to be set by user!

```
# Example:
config.sequential_output = "/castor/cern.ch/user/.../SequentialOutputFolder"
```

A.1.7. config.save_using_pool

Flag saying whether output should be saved first to pool and then copied to output location (might help with errors regarding CASTOR saving scripts) or directly to output location.

Values:

- True - save with a use of pool (Default)
- False - try to directly save to output location

```
# Default:
config.save_using_pool = True
```

A.2. TOTCSI workspace parameters

A.2.1. config.workspace.root_dir

Base directory for all configuration files of TOTCSI

Can be both relative and full path.

```
# Default:
config.workspace.root_dir = "."
```

A.2.2. config.workspace.shell_template

Path to shell template. It is filled by TOTCSI and then submitted to computing cluster as job.

```
# Default:
config.workspace.shell_template = "totcsi/templates/job_template.sh"
```

Default template looks like:

```
#!/bin/sh
function run_and_exit_on_error {
 cmd="$1"
 eval $cmd
 retcode=$?
 if [ $retcode -ne 0 ]; then
  echo "executing command [ " $cmd " ] failed, error code " $retcode " ,
    exiting"
  exit $retcode
 else
  echo "executing command [ " $cmd " ] ok"
 fi
}
function run_and_skip_on_error {
 cmd="$1"
 eval $cmd
 retcode=$?
 if [ $? -ne 0 ]; then
  echo "executing command [ " $cmd " ] failed, error code " $retcode " ,
    skipping"
 else
  echo "executing command [ " $cmd " ] ok"
 fi
}
export RFIO_USE_CASTOR_V2=YES
export STAGE_HOST={{stage_host}}
export STAGE_SVCCLASS=default
export SCRAM_ARCH=slc5_amd64_gcc434
source /afs/cern.ch/cms/cmsset_default.sh
run_and_exit_on_error "cd {{CMSSW_path}}"
run_and_exit_on_error "eval `scram runtime -sh`"
run_and_exit_on_error "cd -"
run_and_skip_on_error "{{rm_type}} {{file_to_remove}}"
run_and_skip_on_error "{{rm_type}} {{output_log_to_remove}}"
run_and_skip_on_error "rm -f {{log_to_remove}}"
run_and_skip_on_error "{{mkdir_type}} {{dir_to_create}}"
run_and_exit_on_error "cmsRun {{python_conf_file}}"
{{optional_copy_command}}
```

A.3. Input parameters

Input templates configuration in which user gives full paths to parallel and sequential templates.

```
# Example for simulation
# (3 simulation with one parallel phase and one sequential phase each):
config.input_config.parallel1_path =
             ['pythiaDD_cfg.py', 'pythiaSD_cfg.py', 'pythiaMB_cfg.py']
config.input_config.sequential_path =
             ['DDval_cfg.py', 'SDval_cfg.py', 'MBval_cfg.py']

# Example for reconstruction
# (chaining of 2 parallel operations and then sequential validation):
config.input_config.parallel1_path = ["rawtodigi.cfg"]
config.input_config.parallel2_path = ["digi_to_tracks.cfg"]
config.input_config.sequential_path = ["validate.cfg"]
```

A.3.1. config.input_config.parallel1_path

List of paths to parallel1 templates (production)

Has to be given by user!

```
# Example (production digitalized data for second parallel phase):
config.input_config.parallel1_path = ["rawtodigi.cfg"]
```

A.3.2. config.input_config.parallel2_path

Same as parallel1_path, but for parallel 2 phase (second parallel phase)

Allows chaining of parallel operations (will take as input files produced from parallel1)

```
# Example:
config.input_config.parallel2_path = ["digi_to_tracks.cfg"]
# takes data produced in parallel1 phase
```

A.3.3. config.input_config.parallel3_path

Same as parallel2_path, but for parallel 3 phase (third parallel phase)

Allows chaining of parallel operations (will take as input files produced from parallel2)

```
# Example:
config.input_config.parallel3_path = ["tracks_to_something_new.cfg"]
# takes data produced in parallel2 phase
```

A.3.4. config.input_config.sequential_path

List of paths to sequential templates (validation)

```
# Example:
config.input_config.sequential_path = ["validate.cfg"]
# takes data produced in last parallel phase specified
```

A.4. bsub parameters

A.4.1. config.bsub - default bsub parameters

Same configurations can be used for bsub_parallel1, bsub_parallel2, bsub_parallel3 and bsub_sequential. If there are no specific configurations for parallel1-3/sequential phases then the configuration from config.bsub is used for them.

config.bsub.pool_size

Pool size for the job (integer)

```
# Default:
config.bsub.pool_size = 30000
```

config.bsub.swap_size

Swap size for the job (integer)

```
# Default:
config.bsub.swap_size = 2000
```

config.bsub.queue

Type of the queue to be used
Values (sorted by decreasing priority) :

- 8nm - 8 minutes of CPU time available on cluster
- 1nh - 1 hour of CPU time available on cluster
- 8nh - 8 hours of CPU time available on cluster
- 1nd - 1 day of CPU time available on cluster
- 2nd - 2 days of CPU time available on cluster
- 1nw - 1 week of CPU time available on cluster
- 2nw - 2 weeks of CPU time available on cluster

For more information run the bqueues command on lxplus.

```
# Default:
config.bsub.queue = "1nh"
```

config.bsub.mail

Should the mail be sent after the job finishes

```
# Default:
config.bsub.mail = False
```

A.4.2. config.bsub_parallel1

bsub configuration for first parallel phase. You can specify same parameters as for config.bsub - default bsub parameters, and they will override the default settings.

```
# Example (changing all parameters from default):
config.bsub_parallel1.pool_size = 25000
config.bsub_parallel1.swap_size = 1200
config.bsub_parallel1.queue = "1nd"
config.bsub_parallel1.mail = True
```

A.4.3. config.bsub_parallel2

bsub configuration for second parallel phase. You can specify same parameters as for config.bsub - default bsub parameters, and they will override the default settings.

```
# Example (changing all parameters from default):
config.bsub_parallel2.pool_size = 25000
config.bsub_parallel2.swap_size = 1200
config.bsub_parallel2.queue = "1nd"
config.bsub_parallel2.mail = True
```

A.4.4. config.bsub_parallel3

bsub configuration for third parallel phase. You can specify same parameters as for config.bsub - default bsub parameters, and they will override the default settings.

```
# Example (changing all parameters from default):
config.bsub_parallel3.pool_size = 25000
config.bsub_parallel3.swap_size = 1200
config.bsub_parallel3.queue = "1nd"
config.bsub_parallel3.mail = True
```

A.4.5. config.bsub_sequential

bsub configuration for sequential phase. You can specify same parameters as for config.bsub - default bsub parameters, and they will override the default settings.

```
# Example (changing all parameters from default):
config.bsub_sequential.pool_size = 25000
config.bsub_sequential.swap_size = 1200
config.bsub_sequential.queue = "1nd"
config.bsub_sequential.mail = True
```

B. Reconstruction configurables

B.1. Reconstruction parameters

B.1.1. config.reconstruction.input_data

Files to be taken into reconstruction.

Values:

- Path (relative or non-relative) to file containing in each line one path to input data
- List of paths pointing directly to input data
- List of run numbers from which file paths will be extracted

Those values can be appended to each other, so for example user can specify some of the input data by direct paths and some by run numbers.

```
# Example with file:
config.reconstruction.input_data = paths_from_file("input.txt")
# Example with run numbers:
config.reconstruction.input_data = paths_from_run_numbers([6011, 6012]
                                                          + range(6020, 6030))
```

B.1.2. config.reconstruction.files_to_exclude

Files that should not be reconstructed (they will be excluded from the input_data). Has same format as input_data.

```
# Example with file:
config.reconstruction.files_to_exclude = paths_from_file("exclude.txt")
```

B.1.3. config.reconstruction.events_per_file_to_reconstruct

Special option which gives user the ability to only reconstruct some of the events from each run file in order to get a peek at data stored in files. The given number of events at the start of each file is taken into consideration.

Values:

- -1 - meaning all events (Default)
- Positive number - amount of events per file to reconstruct (from the beginning of the file)

```
# Default:
config.reconstruction.events_per_file_to_reconstruct = -1
```

B.1.4. config.reconstruction.estimated_number_of_events_per_file

How many events are in one input file (estimation made by user).

This information is needed for splitting one file into multiple jobs. If user wants to use the reconstruction.events_per_job option this has to be specified. It takes a positive integer as value.

```
# Example:
config.reconstruction.estimated_number_of_events_per_file = 50000
# Meaning that each file should have around 50000 events in it
# (last file of a run can have less)
```

B.1.5. config.reconstruction.events_per_job

How many events can be processed in one job at once.

Values:

- -1 - no maximum value for the amount of events per job (Default)
- Positive number - maximal amount of events processed in one job

```
# Default:
config.reconstruction.events_per_job = -1
```

B.1.6. config.reconstruction.files_per_sequential_operation

How many files should be used in one sequential operation.

Values:

- -1 - as many files used in one operation as possible (all files with sequential_operation_per_run set to False)
- Positive number - number of files that will be used in one sequential operation

```
# Default:
config.reconstruction.files_per_sequential_operation = -1
```

B.1.7. config.reconstruction.parallel1_tuning_function

Function for managing runs' templates. User specifies function taking templates as input, which is a mapping with key = run_number and value = configuration_file. The function then makes changes in configuration so that each run has configuration tuned for it's reconstruction.

By default there are no changes in given templates.

WARNING

This function has to return the changed templates!

While appending care not to add unnecessary spaces/tabulations as it will cause errors in splitted configuration!

```
# Example:
def tune_par_configuration(par_templates):
 # Appending
 for run in range(6908, 6928):
  par_templates[run] += '''
toberemoved = []
for xmlfile in process.XMLIdealGeometryESSource.geomXMLFiles:
 if xmlfile.endswith("RP_Dist_Beam_Cent.xml"):
  toberemoved.append(xmlfile)
for xmlfile in toberemoved:
 process.XMLIdealGeometryESSource.geomXMLFiles.remove(xmlfile)
process.XMLIdealGeometryESSource.geomXMLFiles.append("Geometry/TotemRPData/
                          data/2010_10_29-30_offsets/RP_Dist_Beam_Cent.xml")
process.TotemRPGeometryESModule = cms.ESProducer("TotemRPGeometryESModule")
'''

 # Substitution
 for run in range(6190, 6213):
  par_templates[run] = Template(par_templates[run]).safe_substitute(GEOMETRY=
      "Geometry/TotemRPData/data/2010_10_29-30_offsets/RP_Dist_Beam_Cent.xml")
 return par_templates

config.reconstruction.parallel1_tuning_function = tune_par_configuration
```

B.1.8. config.reconstruction.parallel2_tuning_function

Works same as config.reconstruction.parallel1_tuning_function, just for parallel2 phase (second parallel operation).

B.1.9. config.reconstruction.parallel3_tuning_function

Works same as config.reconstruction.parallel1_tuning_function, just for parallel3 phase (third parallel operation).

B.1.10. config.reconstruction.sequential_tuning_function

Works same as config.reconstruction.parallel1_tuning_function, just for sequential phase.

C. Simulation configurables

C.1. Simulation parameters

C.1.1. config.simulation.number_of_events

Number of events to be generated (integer number) per job

It has to be set by user!

```
 # Example:
 config.simulation.number_of_events = 1500
```

C.1.2. config.simulation.number_of_jobs

Number of jobs generating the simulation data (integer number)

Each job generates number of events specified by config.simulation.number_of_events

It has to be set by user!

```
# Example:
config.simulation.number_of_jobs = 10
```

D. Configuration examples

D.1. Reconstruction

```
from totcsi.configuration_logic import *

config = TOTCSIConfiguration()
config.task_type = "Reconstruction"
config.cmssw_dir = "/afs/cern.ch/user/i/ijurkows/private/cmssw/CMSSW_4_2_4"
config.parallel1_output = "/castor/cern.ch/user/i/ijurkows/output"
# save right away (user should often stay with default setting)
config.save_using_pool = False

# setting the splitter workspace
config.workspace.root_dir = "/afs/cern.ch/user/i/ijurkows/private/Workspace"

config.input_config.parallel1_path = "/afs/cern.ch/user/i/ijurkows/public/\
    configurations_for_parallel1/run8331_cfg.py"
config.input_config.sequential_path = "/afs/cern.ch/user/i/ijurkows/public/\
    configurations_for_sequential/ntuplize_cfg.py"
config.bsub.pool_size = 20000
config.bsub_parallel1.pool_size = 26000
# input.txt - file with path to data in each line
config.reconstruction.input_data = paths_from_file("input.txt")
                                   + paths_from_run_numbers([8331])
config.reconstruction.files_to_exclude =
    ["/castor/cern.ch/totem/LHCRawData/2012/Physics/run_8331.001.vmea",
    "/castor/cern.ch/totem/LHCRawData/2012/Physics/run_8331.005.vmea"]

def tune_parallel1_configuration(parallel1_templates):
   # appending
   for run in range(8330, 8331):
      parallel1_templates[run] += '''
toberemoved = []
for xmlfile in process.XMLIdealGeometryESSource.geomXMLFiles:
   if xmlfile.endswith("RP_Dist_Beam_Cent.xml"):
      toberemoved.append(xmlfile)
for xmlfile in toberemoved:
   process.XMLIdealGeometryESSource.geomXMLFiles.remove(xmlfile)
process.XMLIdealGeometryESSource.geomXMLFiles.append( ...
                ..."Geometry/TotemRPData/data/2010_10_29-30_offsets/RP_Dist_Beam_Cent.xml")
process.TotemRPGeometryESModule = cms.ESProducer("TotemRPGeometryESModule")
'''
   # substitution
   for run in range(8331, 8332):
       parallel1_templates[run] = Template(parallel1_templates[run]).safe_substitute(GEOMETRY=
                       "Geometry/TotemRPData/data/2010_10_29-30_offsets/RP_Dist_Beam_Cent.xml")
   return parallel1_templates
config.reconstruction.parallel1_tuning_function = tune_parallel1_configuration
```

D.2. Simulation

```
from totcsi.configuration_logic import *

config = TOTCSIConfiguration()
config.task_type = "Simulation"
config.cmssw_dir = "/afs/cern.ch/user/i/ijurkows/private/cmssw/CMSSW_4_2_4"
config.map_output = "/castor/cern.ch/user/i/ijurkows/output"
# setting the splitter workspace
config.workspace.root_dir = "/afs/cern.ch/user/i/ijurkows/private/TOTCSI_Workspace"

config.input_config.map_path = "/afs/cern.ch/user/i/ijurkows/public/\
    configurations_for_map/RPElasticbeta90energy3p5TeV220October2011a3_cfg.py"
config.input_config.reduce_path = "/afs/cern.ch/user/i/ijurkows/public/\
    configurations_for_reduce/valRPElasticbeta90energy3p5TeV220October2011a3_cfg.py"
config.bsub.pool_size = 20000
config.bsub_parallel1.pool_size = 26000
config.simulation.number_of_events = 2000
config.simulation.number_of_jobs = 100
```