JSUB – A Tool for Job Submission and Management

Yang Yifan

IHEP

November 13, 2020

JSUB - Job submission utility bundle

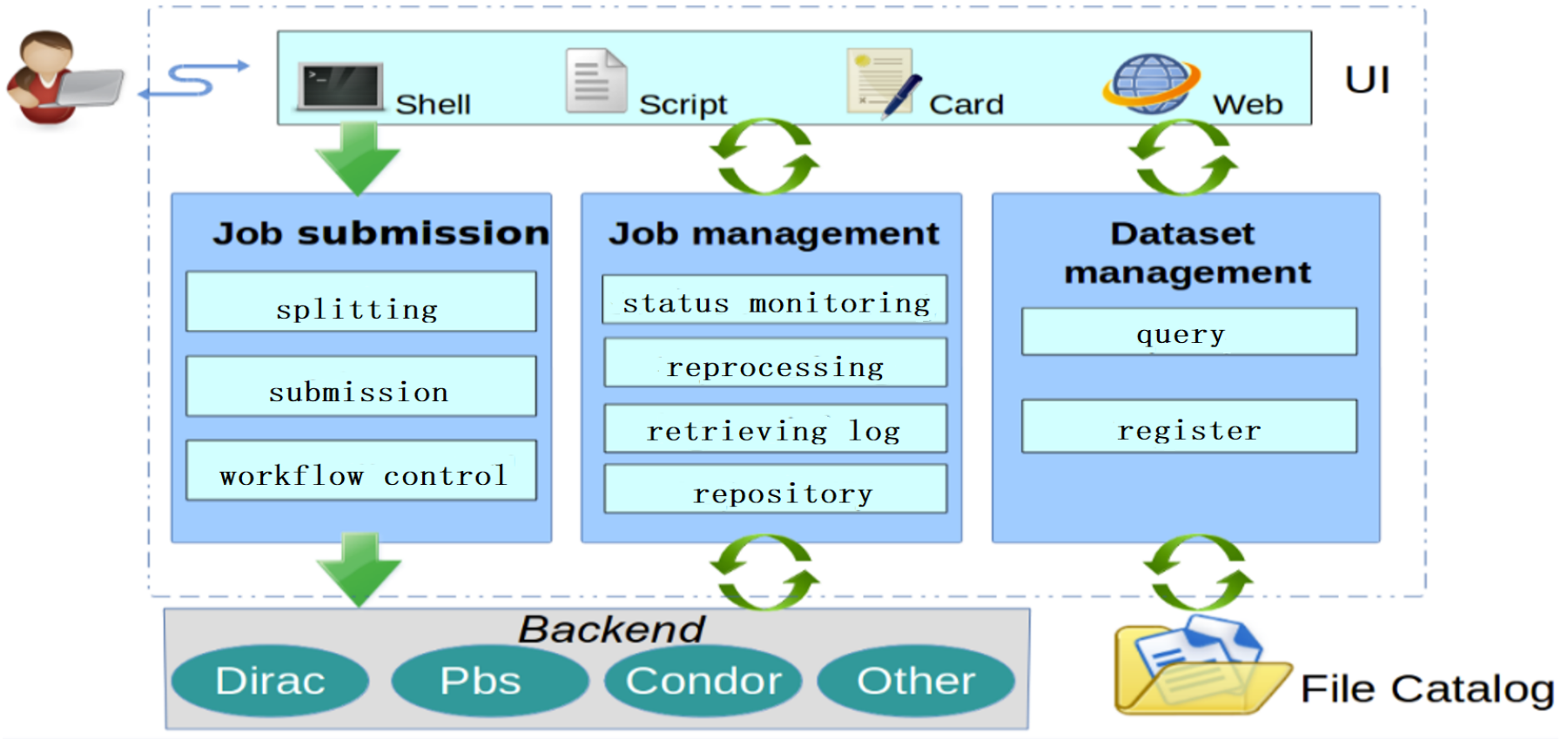
A frontend software to make users' lives easier.

- Ease the procedure of using DIRAC, and potentially other heterogeneous resources.
- Automatically manage massive jobs.
- Highly extensible for other experiments.

What JSUB can support for JUNO

- User simulation and analysis.
- Executing multi-steps in one task, including detsim/elecsim/calib/rec/ana
- User customized scripts.
- Task based monitoring and rescheduling

Functionality design of JSUB



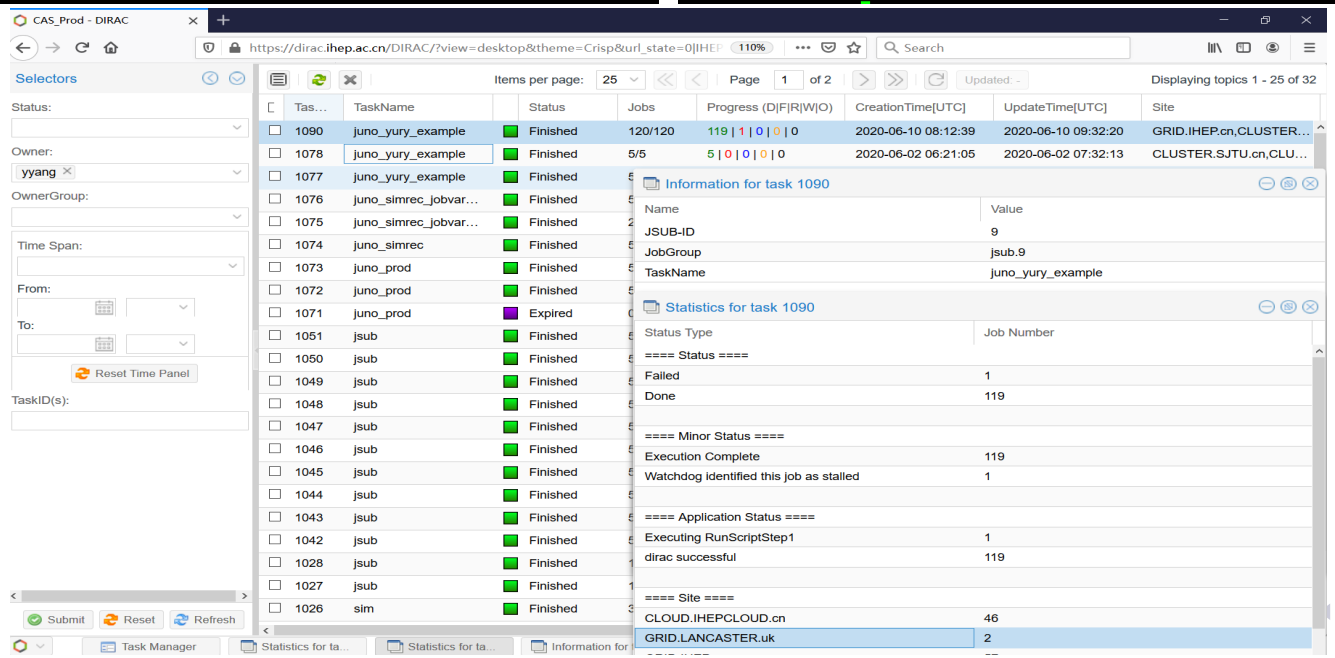
Using JSUB

```
4 taskName: juno_sim
5 experiment: juno
6 softVersion: 'J20v1r0-Pre2'
7 #softVersion: 'sl6_amd64_gcc830/Pre-Release/J20v1r0-Pre2'
8
9 backend:
10     type: dirac
11
12     ## When outputSubDir is defined, the final directory for output file would
13     outputSubDir: 'jsub_tests'
14     ## Alternatively, user may specify the full path of output LFN folder with
15 #    outputDir:'/junofs/.../jsub_tests/juno_sim'
16
17 #    site:
18 #        - CLOUD.JINRONE.ru
19 #        - CLOUD.IHEP.cn
20 #    bannedSites:
21 #        - CLOUD.JINRONE.ru
22
23 splitter:
24     ## A splitByEvent splitter generate subjobs with uniform settings.
25     ## In splitByEvent mode, filenames of input/output/userOutput are automatic
26     ## Other settings shall stay the same for all subjobs.
27     mode: splitByEvent
28     evtMaxPerJob: 1000
29     njobs: 50
30
31 workflow:
32     steps: [detsim]
33
34     detsim:
35         seed: 1 # the starting seed (in splitByEvent mode)
36
37         ## additionalArgs are put after common attributes such as output, user0
38         additionalArgs: 'gun --particles e+ --momentums 1.398'
39
```

```
(venv) yangyf:[~] > jsub --help
Usage: jsub [OPTIONS] COMMAND [ARGS]...

Options:
  --jsubrc TEXT  Configuration file to run JSUB with.
  --help         Show this message and exit.

Commands:
  create      Create a task from a task description file.
  getlog      Retrieve log files of selected subjobs.
  ls          List all tasks.
  package     Show active packages.
  remove      Delete a task.
  rename      Rename a task.
  reschedule  Reschedule selected subjobs.
  resubmit    Equivalent to 'jsub submit -r' command
  run         Create from a task profile, and submit.
  show        Show detailed description of a task.
  status      Show the backend status of a task.
  submit      Submit a task to backend.
  version     Show the version of the software.
```



Installation/Activation

Currently, a test version has already been installed on CVMFS.
The software is installed in an isolated Pythonic virtualenv, which can be activated with the following commands:

```
source /cvmfs/dcomputing.ihep.ac.cn/frontend/jsub/activate.sh
source .../activate.sh -e juno
source .../activate.sh -v 0.3.0
```

And the environment can be deactivated with:

```
deactivate
```

Also, the source code is available on github (https://github.com/jsubpy).
The Python packages can be installed with pip.

Configuration

By default, JSUB looks for configuration file at *./.jsubrc*.
An example configuration file is shown below:

```
# JSUB Configuration File
# This configuration file is supposed to be put at ~/.jsubrc.
# The settings here would overload the default ones defined in .jsubrc in JSUB main dir.
#============================================================================

###    The packages to be loaded to JSUB. JSUB would search for extension modules according to the order given here.
package: [jsub_juno, jsub_dirac]

### Location to put task information files; may need big space for log and output files
taskDir:
  location: /junofs/users/USER_NAME/jsub

### Backend setting
backend:
  default: dirac
#  dirac:
    # Config backend settings here
#   site:
#     - CLOUD.IHEP.cn
#     - GRID.JINR.ru
#     - CLUSTER.USTC.cn
```

The most important setting here includes:

- package: from which Python modules to look for JSUB extensions.
- taskDir: where to store JSUB files including task info, logfiles, and (in some cases) runtime files as well as output files.
- backend: Universal backend settings. Can be overridden by task-specific settings.

Preparing task description files

To create a task, users should provide a task definition file (TDF), which can be in the format of YAML or JSON.
The following parts shall be addressed in task description:

- General Settings: task name, experiment, input sandbox, ...
- Backend: output folder, job group, site, banned sites, ...
- Splitter: how the task splits into subjobs.
- Workflow: list of action steps and their settings.

```yaml
taskName: juno_sim
experiment: juno
softVersion: 'J20v1r0-Pre2'
#softVersion: 'sl6_amd64_gcc830/Pre-Release/J20v1r0-Pre2'

backend:
    type: dirac

    ## When outputSubDir is defined, the final directory for output file would
    outputSubDir: 'jsub_tests'
    ## Alternatively, user may specify the full path of output LFN folder with
#    outputDir:'/junofs/.../jsub_tests/juno_sim'

#    site:
#        - CLOUD.JINRONE.ru
#        - CLOUD.IHEP.cn
#    bannedSites:
#        - CLOUD.JINRONE.ru

splitter:
    ## A splitByEvent splitter generate subjobs with uniform settings.
    ## In splitByEvent mode, filenames of input/output/userOutput are automatic
    ## Other settings shall stay the same for all subjobs.
    mode: splitByEvent
    evtMaxPerJob: 1000
    njobs: 50

workflow:
    steps: [detsim]

    detsim:
        seed: 1 # the starting seed (in splitByEvent mode)

        ## additionalArgs are put after common attributes such as output, user0
        additionalArgs: 'gun --particles e+ --momentums 1.398'
```

Task description - general settings and backend settings

General settings

- taskName
- experiment: decides the parser of this yaml file; experiment-specific parsers expand user settings with auto-fill, while "common" parser is for raw format.
- softVersion: the version of the experiment-specific application.

Backend settings

JSUB supports backend such as DIRAC, IHEPCondor, and local (login nodes), and these backend each can use different settings. For example, on DIRAC backend, users may define the sites (or banned sites) to send jobs to, target SE to send output data, and job groups, etc.

Task description - splitter

A splitter defines how a JSUB task can be splitted into multiple subjobs that each can run on a single backend worknode and how the values of subjob-specific variables should be assigned.
Currently, JSUB supports two splitter modes – splitByEvent and splitByJobvars.

SplitByEvent splitter

- This splitter is for jobs that use uniform settings.
- Users only need to define the number of jobs and events per job, and the splitter would handle the seeds and filenames.

```
10 splitter:
11     mode: splitByEvent
12     evtMaxPerJob: 1000
13     njobs: 50
14
15 workflow:
16     detsim:
17         seed: 1 # the starting value
```

Task description - splitter

SplitByJobvars splitter

The splitter uses jobvar extensions to generate parameter lists, combine these parameters into job variable sets. The number of subjobs are decided by the length of value sets list.

This splitter gives user more control over the details of subjobs by allowing referencing jobvar values in settings.

```
splitter:
    mode: splitByJobvars # In this mode, a task is splitted into subjobs depending on job variable lists
    maxSubJobs: 500     # the resulted number of subjobs won't exceed this
    evtMaxPerJob: 5000
    jobvarLists:
        nuclear:
            type: enumerate
            list: ['U-238','Th-232','K-40','Pb-210','C-14','Kr-85']
            group: nuclear  # 6 nuclear
        subjob:
            type: range
            first: 1
            step: 1
            length: 20
            group: same_nuclear # 20 subjob for each nuclear
        seed:
            type: range
            first: 1
            step: 1
            #a unique seed for each of 120 subjobs

workflow:
    detsim:
        seed: '$(seed)'     # jobvars can be referenced in workflow setting
        output: '$(nuclear).$(subjob).detsim.root'
        userOutput: '$(nuclear).$(subjob).user.detsim.root'
        additionalArgs: 'gendecay --nuclear $(nuclear) --volume pTarget --material LS'
```

Task description - workflow settings

Workflow settings

This part describes the list of action steps in the task workflow and their settings.

With SplitByJobvars splitter, some settings in the workflow may reference the value of jobvars.

```
25 workflow:
26     steps: [junosoft, detsim]
27
28     junosoft:
29         type: juno_software     # for step name other than reserved step-type keywords, it's necessary to d
30         software: offline/InstallArea/amd64_linux26/bin/Muon.exe        #under $JUNOTOP
31         args: '-seed $(seed) -o Muon.$(seed).txt -n 5 -v Rock -mult 1 -music_dir  ${JUNOTOP}/data/Generator
32
33     detsim:
34         fullArgs: '--seed $(seed) --output detsim-$(seed).root --user-output detsim-$(seed).user.root --pmt
   -detoption Acrylic hepevt --exe Muon --file Muon.$(seed).txt'
```

Job management with command lines

Given a TDF, users can create a JSUB task with the following command line, and a task ID would be returned after successful creation.

```
jsub create <TDF>
```

The task can then be submitted to backend for running.

```
jsub submit <task-id>
```

The job statuses of a task can be queryed.

```
jsub status <task-id> // statistics
jsub status <task-id> -s <status> // list subjobs in given status
```

The brief info of all tasks can be listed.

```
jsub ls
jsub ls -u // to update backend status info
```

Job Management with command lines

Bad jobs can be rescheduled, or the whole task can be resubmitted.

```
jsub reschedule <task-id> [-dfrw] // for given states
jsub resubmit <task-id> // resubmit the whole task
```

Log files of selected jobs can be downloaded from DIRAC server.

```
jsub getlog <task-id> -i SUB_IDS [-n NJOBS]
jsub getlog <task-id> -s STATUS
```

JSUB is ready for user test

- Available on IHEP-CVMFS.

/cvmfs/dcomputing.ihep.ac.cn/frontend/jsub/

- Testing examples are distributed.

/cvmfs/dcomputing.ihep.ac.cn/frontend/jsub/0.3.0/install/jsub/examples

- More details on guide

https://jsubpy.github.io/

Thanks!

Backup

Yaml example with DetSim

```yaml
## ----------------------------------------
## A simple example showing the basics about running a JUNO simulation task
## ----------------------------------------
taskName: juno_sim
experiment: juno
softVersion: 'J20v1r0-Pre2'
#softVersion: 'sl6_amd64_gcc830/Pre-Release/J20v1r0-Pre2'

backend:
    type: dirac

    ## When outputSubDir is defined, the final directory for output file would be: /<junofs-user-home>/<outputSubDir>/<taskName>
    outputSubDir: 'jsub_tests'
    ## Alternatively, user may specify the full path of output LFN folder with outputDir
#    outputDir:'/junofs/.../jsub_tests/juno_sim'

#    site:
#       - CLOUD.JINRONE.ru
#       - CLOUD.IHEP.cn
#    bannedSites:
#       - CLOUD.JINRONE.ru

splitter:
    ## A splitByEvent splitter generate subjobs with uniform settings.
    ## In splitByEvent mode, filenames of input/output/userOutput are automatic, and the seeds are incremental by 1 by default.
    ## Other settings shall stay the same for all subjobs.
    mode: splitByEvent
    evtMaxPerJob: 1000
    njobs: 50

workflow:
    steps: [detsim]

    detsim:
        seed: 1 # the starting seed (in splitByEvent mode)

        ## additionalArgs are put after common attributes such as output, userOutput, input, seed, evtmax, and rate.
        additionalArgs: 'gun --particles e+ --momentums 1.398'
```

Multi-steps of sim/rec

```yaml
## ---------------------------------
## An example of having multi-steps in the workflow
## ---------------------------------
taskName: juno_simrec
experiment: juno
softVersion: 'J20v1r0-Pre2'

backend:
    type: dirac
    outputSubDir: 'jsub_tests'

splitter:
    mode: splitByEvent
    evtMaxPerJob: 1000
    njobs: 50


workflow:
    steps: [detsim, elecsim, calib, rec]

    detsim:
        seed: 1 ## when in splitByEvent mode, the seeds are incremental by default
        particles: e+
        momentums: 1.398    #MeV
        additionalArgs: ''
        ## when gun params are defined:  full_args= '--seed X --output X --user-output X $(additionalArgs) gun --particles X ...'

    elecsim:
        ## when detsim and elecsim are both in the workflow, the output of detsim automatically feeds into elecsim
        seed: 10
        rate: 1
        additionalArgs: ''

    #calib:        # if a step only uses default settings, it's description can be skipped
    #    additionalArgs: ''


```

Job variable splitter

```yaml
splitter:
    ## A splitByJobvars generate job variable lists and combine them into sets. For each variable set, the splitter generates one subjob
    mode: splitByJobvars
    maxSubJobs: 500     ## the resulted number of subjobs won't exceed this number
    evtMaxPerJob: 5000
    jobvarLists:
        ## The jobvar lists are grouped.
        ## For jobvars in the same group, the length of their common var-set list is decided by the shortest jobvar list.
        ## For jobvar sets in different groups, the combining result is their Cartesian product.
        ## Jobvars without group attribute would make a final common var-set list with the combining result of all jobvar groups.

        ## In this example, there shall be 6*20=120 jobs, each with a unique seed.
        nuclear:
            type: enumerate
            list: ['U-238','Th-232','K-40','Pb-210','C-14','Kr-85']
            group: nuclear
        subjob:
            type: range
            first: 1    ## default 1
            step: 1     ## default 1
            length: 20  ## default 100000
            group: same_nuclear
        seed:
            type: range
            first: 1
            step: 1

workflow:
    steps: [detsim]

    detsim:
        ## The values of jobvars can be referenced in workflow setting.
        seed: '$(seed)'
        output: '$(nuclear).$(subjob).detsim.root'
        userOutput: '$(nuclear).$(subjob).user.detsim.root'
        additionalArgs: 'gendecay --nuclear $(nuclear) --volume pTarget --material LS'

        ## fullArgs = seed + ... + additionalArgs
        #fullArgs: '--evtmax 5000 --seed $(seed) --output $(nuclear).$(subjob).detsim.root --user-output $(nuclear).$(subjob).user.detsim
erial LS'
```

Elecsim (input data from DFC)

```yaml
## ------------------------------------------------
## A example with juno elecsim, showcasing how to get input
## ------------------------------------------------
taskName: juno_elecsim
experiment: juno
softVersion: 'sl6_amd64_gcc830/Pre-Release/J20v1r0-Pre2'

backend:
    type: dirac
    outputSubDir: 'jsub_tests/'

splitter:
    ## For jobs with input, splitByJobvars splitter is necessary so that the input filenames can be referenced in workflow setting
    mode: splitByJobvars
    maxSubJobs: 500
    evtMaxPerJob: 5000
    jobvarLists:
        ## One way to assign input file is to list the filenames in a text file.
#       input_filename:
#           type: lines_in_file
#           file: './lfnlist.example'
        ## Another way is to filter LFN list in a given folder, using dirac-dms-find-lfns command
        input_filename:
           type: find_lfns
           path: '/juno/user/.../test'
           metaspec:    ' "Size>1000" "CreationDate>2010-01-01" ' # metadata index specification
        seed:
            type: range

workflow:
    steps: [elecsim]

    elecsim:
        seed: '$(seed)'  # jobvars can be referenced in workflow setting
        input: '$(input_filename)'
        rate: 10
        output: 'elecsim.$(seed).root'
        userOutput: 'elecsim.user.$(seed).root'
        additionalArgs: ''
```

User Analysis

```yaml
taskName: juno_custom_Alg
experiment: juno
softVersion: 'J20v1r0-Pre2'

backend:
    type: dirac
    outputSubDir: 'jsub_tests'

splitter:
    mode: splitByJobvars
    maxSubJobs: 20
    evtMaxPerJob: 1000

    jobvarLists:
        idx:
            type: range
            length: 10

workflow:
    steps: [myAlg]

    myAlg:
        type: 'juno_alg'
        # Users shall provide a job configuration file template for the algorithm and the referenced DLLs.
        # These files would be put into input sandbox.
        # The folders of Sniper.LoadDll() in the config would be auto-redirected.
        soFile:
            - './JsubDummyAlg/amd64_linux26/libJsubDummyAlg.so'
            - './JsubHelloWorld/amd64_linux26/libJsubHelloWorld.so'
#       jobConfig: './JsubDummyAlg/share/run.py'   # local position
        jobConfig: './JsubHelloWorld/share/run.py'   # local position

        # Users may use case-sensitive text replacement to set subjob-dependent parameters.
        textReplace:
            # keyword: replacement
            OUTPUT1: 'a/output.$(idx).1.root'
            OUTPUT2: 'b/output.$(idx).2.root'
        # what files to be uploaded as output data, for (dirac backend)
        outputUpload:
            - '*root'
```