

---

# **DPDispatcher**

**Deep Modeling**

**Dec 04, 2021**



**CONTENTS:**

<b>1</b>	<b>Install DPDispatcher</b>	<b>3</b>
<b>2</b>	<b>Getting Started</b>	<b>5</b>
<b>3</b>	<b>Machine parameters</b>	<b>9</b>
<b>4</b>	<b>Resources parameters</b>	<b>15</b>
<b>5</b>	<b>Task parameters</b>	<b>19</b>
<b>6</b>	<b>DPDispatcher API</b>	<b>21</b>
6.1	dpdispatcher package . . . . .	21
<b>7</b>	<b>Indices and tables</b>	<b>53</b>
	<b>Python Module Index</b>	<b>55</b>
	<b>Index</b>	<b>57</b>



DPDispatcher is a Python package used to generate HPC (High Performance Computing) scheduler systems (Slurm/PBS/LSF/dpcloudserver) jobs input scripts and submit these scripts to HPC systems and poke until they finish.

DPDispatcher will monitor (poke) until these jobs finish and download the results files (if these jobs is running on remote systems connected by SSH).



## INSTALL DPDISPATCHER

DPDispatcher can installed by pip:

```
pip install dpdispatcher
```





## GETTING STARTED

DPDispatcher provides the following classes:

- Task class, which represents a command to be run on batch job system, as well as the essential files need by the command.
- Submission class, which represents a collection of jobs defined by the HPC system. And there may be common files to be uploaded by them. DPDispatcher will create and submit these jobs when a submission instance execute `run_submission` method. This method will poke until the jobs finish and return.
- Job class, a class used by Submission class, which represents a job on the HPC system. Submission will generate jobs' submitting scripts used by HPC systems automatically with the Task and Resources
- Resources class, which represents the computing resources for each job within a submission.

You can use DPDispatcher in a Python script to submit five tasks:

```
from dpdispatcher import Machine, Resources, Task, Submission

machine = Machine.load_from_json('machine.json')
resources = Resources.load_from_json('resources.json')

task0 = Task.load_from_json('task.json')

task1 = Task(command='cat example.txt', task_work_path='dir1/', forward_files=['example.
↳txt'], backward_files=['out.txt'], outlog='out.txt')
task2 = Task(command='cat example.txt', task_work_path='dir2/', forward_files=['example.
↳txt'], backward_files=['out.txt'], outlog='out.txt')
task3 = Task(command='cat example.txt', task_work_path='dir3/', forward_files=['example.
↳txt'], backward_files=['out.txt'], outlog='out.txt')
task4 = Task(command='cat example.txt', task_work_path='dir4/', forward_files=['example.
↳txt'], backward_files=['out.txt'], outlog='out.txt')

task_list = [task0, task1, task2, task3, task4]

submission = Submission(work_base='lammps_md_300K_5GPa/',
    machine=machine,
    resources=reasources,
    task_list=task_list,
    forward_common_files=['graph.pb'],
    backward_common_files=[])
)

submission.run_submission()
```

where `machine.json` is

```
{
  "batch_type": "Slurm",
  "context_type": "SSHContext",
  "local_root": "/home/user123/workplace/22_new_project/",
  "remote_root": "/home/user123/dpdispatcher_work_dir/",
  "remote_profile": {
    "hostname": "39.106.xx.xxx",
    "username": "user123",
    "port": 22,
    "timeout": 10
  }
}
```

`resources.json` is

```
{
  "number_node": 1,
  "cpu_per_node": 4,
  "gpu_per_node": 1,
  "queue_name": "GPUV100",
  "group_size": 5
}
```

and `task.json` is

```
{
  "command": "lmp -i input.lammps",
  "task_work_path": "bct-0/",
  "forward_files": [
    "conf.lmp",
    "input.lammps"
  ],
  "backward_files": [
    "log.lammps"
  ],
  "outlog": "log",
  "errlog": "err",
}
```

You may also submit mutiple GPU jobs: complex resources example

```
resources = Resources(
    number_node=1,
    cpu_per_node=4,
    gpu_per_node=2,
    queue_name="GPU_2080Ti",
    group_size=4,
    custom_flags=[
        "#SBATCH --nice=100",
        "#SBATCH --time=24:00:00"
    ],
    strategy={
```

(continues on next page)

(continued from previous page)

```
    # used when you want to add CUDA_VISIBLE_DEVICES automatically
    "if_cuda_multi_devices": True
},
para_deg=1,
# will unload these modules before running tasks
module_unload_list=["singularity"],
# will load these modules before running tasks
module_list=["singularity/3.0.0"],
# will source the environment files before running tasks
source_list=["./slurm_test.env"],
# the envs option is used to export environment variables
# And it will generate a line like below.
# export DP_DISPATCHER_EXPORT=test_foo_bar_baz
envs={"DP_DISPATCHER_EXPORT": "test_foo_bar_baz"},
)
```

The details of parameters can be found in *Machine Parameters*, *Resources Parameters*, and *Task Parameters*.



## MACHINE PARAMETERS

**machine:**

type: dict

argument path: machine

**batch\_type:**

type: str

argument path: machine/batch\_type

The batch job system type. Option: Torque, Lebesgue, DpCloudServer, Shell, DistributedShell, PBS, LSF, Slurm

**local\_root:**

type: str

argument path: machine/local\_root

The dir where the tasks and relating files locate. Typically the project dir.

**remote\_root:**

type: str, optional

argument path: machine/remote\_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

**clean\_asynchronously:**

type: bool, optional, default: False

argument path: machine/clean\_asynchronously

Clean the remote directory asynchronously after the job finishes.

Depending on the value of *context\_type*, different sub args are accepted.

**context\_type:**

type: str (flag key)

argument path: machine/context\_type

possible choices: LazyLocalContext, SSHContext, DpCloudServerContext, LebesgueContext, HDFSContext, LocalContext

The connection used to remote machine. Option: LebesgueContext, DpCloudServerContext, LocalContext, SSHContext, LazyLocalContext, HDFSContext

When *context\_type* is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

**remote\_profile:**

type: dict, optional  
argument path: machine[LazyLocalContext]/remote\_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When *context\_type* is set to SSHContext (or its aliases sshcontext, SSH, ssh):

**remote\_profile:**

type: dict  
argument path: machine[SSHContext]/remote\_profile

The information used to maintain the connection with remote machine.

**hostname:**

type: str  
argument path: machine[SSHContext]/remote\_profile/hostname  
hostname or ip of ssh connection.

**username:**

type: str  
argument path: machine[SSHContext]/remote\_profile/username  
username of target linux system

**password:**

type: str, optional  
argument path: machine[SSHContext]/remote\_profile/password  
password of linux system

**port:**

type: int, optional, default: 22  
argument path: machine[SSHContext]/remote\_profile/port  
ssh connection port.

**key\_filename:**

type: NoneType | str, optional, default: None  
argument path: machine[SSHContext]/remote\_profile/key\_filename  
key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

**passphrase:**

type: NoneType | str, optional, default: None  
argument path: machine[SSHContext]/remote\_profile/passphrase  
passphrase of key used by ssh connection

**timeout:**

type: int, optional, default: 10  
argument path: machine[SSHContext]/remote\_profile/timeout  
timeout of ssh connection

**totp\_secret:**

type: NoneType | str, optional, default: None

argument path: machine[SSHContext]/remote\_profile/totp\_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

When *context\_type* is set to `DpCloudServerContext` (or its aliases `dpcloudservercontext`, `DpCloudServer`, `dpcloudserver`):

**remote\_profile:**

type: dict

argument path: machine[DpCloudServerContext]/remote\_profile

The information used to maintain the connection with remote machine.

**email:**

type: str

argument path: machine[DpCloudServerContext]/remote\_profile/email

Email

**password:**

type: str

argument path: machine[DpCloudServerContext]/remote\_profile/password

Password

**program\_id:**

type: int

argument path: machine[DpCloudServerContext]/remote\_profile/program\_id

Program ID

**input\_data:**

type: dict

argument path: machine[DpCloudServerContext]/remote\_profile/input\_data

Configuration of job

**job\_name:**

type: str, optional

argument path:

machine[DpCloudServerContext]/remote\_profile/input\_data/job\_name

Job name

**image\_name:**

type: str, optional

argument path:

machine[DpCloudServerContext]/remote\_profile/input\_data/image\_name

Name of the image which run the job, optional when platform is not ali/oss.

**disk\_size:**

type: str, optional

argument path:

machine[DpCloudServerContext]/remote\_profile/input\_data/disk\_size

disk size (GB), optional when platform is not ali/oss.

**scass\_type:**

type: str  
argument path:  
machine[DpCloudServerContext]/remote\_profile/input\_data/scass\_type  
machine configuration.

**platform:**

type: str  
argument path:  
machine[DpCloudServerContext]/remote\_profile/input\_data/platform  
Job run in which platform.

**log\_file:**

type: str, optional  
argument path:  
machine[DpCloudServerContext]/remote\_profile/input\_data/log\_file  
location of log file.

**checkpoint\_files:**

type: list | str, optional  
argument path:  
machine[DpCloudServerContext]/remote\_profile/input\_data/checkpoint\_files  
location of checkpoint files when it is list type. record file changes when it is string value 'sync\_files'

**checkpoint\_time:**

type: int, optional, default: 15  
argument path:  
machine[DpCloudServerContext]/remote\_profile/input\_data/checkpoint\_time  
interval of checkpoint data been stored minimum 15.

**backward\_files:**

type: list, optional  
argument path:  
machine[DpCloudServerContext]/remote\_profile/input\_data/backward\_files  
which files to be uploaded to remote resources. Upload all the files when it is None or empty.

When *context\_type* is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

**remote\_profile:**

type: dict  
argument path: machine[LebesgueContext]/remote\_profile  
The information used to maintain the connection with remote machine.

**email:**

type: str  
argument path: machine[LebesgueContext]/remote\_profile/email  
Email



**password:**

type: str

argument path: machine[LebesgueContext]/remote\_profile/password

Password

**program\_id:**

type: int

argument path: machine[LebesgueContext]/remote\_profile/program\_id

Program ID

**input\_data:**

type: dict

argument path: machine[LebesgueContext]/remote\_profile/input\_data

Configuration of job

**job\_name:**

type: str, optional

argument path: machine[LebesgueContext]/remote\_profile/input\_data/job\_name

Job name

**image\_name:**

type: str, optional

argument path: machine[LebesgueContext]/remote\_profile/input\_data/image\_name

Name of the image which run the job, optional when platform is not ali/oss.

**disk\_size:**

type: str, optional

argument path: machine[LebesgueContext]/remote\_profile/input\_data/disk\_size

disk size (GB), optional when platform is not ali/oss.

**scass\_type:**

type: str

argument path: machine[LebesgueContext]/remote\_profile/input\_data/scass\_type

machine configuration.

**platform:**

type: str

argument path: machine[LebesgueContext]/remote\_profile/input\_data/platform

Job run in which platform.

**log\_file:**

type: str, optional

argument path: machine[LebesgueContext]/remote\_profile/input\_data/log\_file

location of log file.

**checkpoint\_files:**

type: list | str, optional

argument path:

machine[LebesgueContext]/remote\_profile/input\_data/checkpoint\_files

location of checkpoint files when it is list type. record file changes when it is string value 'sync\_files'

**checkpoint\_time:**

type: int, optional, default: 15

argument path:

machine[LebesgueContext]/remote\_profile/input\_data/checkpoint\_time

interval of checkpoint data been stored minimum 15.

**backward\_files:**

type: list, optional

argument path:

machine[LebesgueContext]/remote\_profile/input\_data/backward\_files

which files to be uploaded to remote resources. Upload all the files when it is None or empty.

When *context\_type* is set to `HDFSContext` (or its aliases `hdfscontext`, `HDFS`, `hdfs`):

**remote\_profile:**

type: dict, optional

argument path: machine[HDFSContext]/remote\_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When *context\_type* is set to `LocalContext` (or its aliases `localcontext`, `Local`, `local`):

**remote\_profile:**

type: dict, optional

argument path: machine[LocalContext]/remote\_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

## RESOURCES PARAMETERS

**resources:**

type: dict

argument path: `resources`

**number\_node:**

type: int

argument path: `resources/number_node`

The number of node need for each *job*

**cpu\_per\_node:**

type: int

argument path: `resources/cpu_per_node`

cpu numbers of each node assigned to each job.

**gpu\_per\_node:**

type: int

argument path: `resources/gpu_per_node`

gpu numbers of each node assigned to each job.

**queue\_name:**

type: str

argument path: `resources/queue_name`

The queue name of batch job scheduler system.

**group\_size:**

type: int

argument path: `resources/group_size`

The number of *tasks* in a *job*.

**custom\_flags:**

type: list, optional

argument path: `resources/custom_flags`

The extra lines pass to job submitting script header

**strategy:**

type: dict, optional

argument path: `resources/strategy`

strategies we use to generation job submitting scripts.

**if\_cuda\_multi\_devices:**

type: `bool`, optional, default: `True`

argument path: `resources/strategy/if_cuda_multi_devices`

**para\_deg:**

type: `int`, optional, default: `1`

argument path: `resources/para_deg`

Decide how many tasks will be run in parallel.

**source\_list:**

type: `list`, optional, default: `[]`

argument path: `resources/source_list`

The env file to be sourced before the command execution.

**module\_purge:**

type: `bool`, optional, default: `False`

argument path: `resources/module_purge`

Remove all modules on HPC system before module load (`module_list`)

**module\_unload\_list:**

type: `list`, optional, default: `[]`

argument path: `resources/module_unload_list`

The modules to be unloaded on HPC system before submitting jobs

**module\_list:**

type: `list`, optional, default: `[]`

argument path: `resources/module_list`

The modules to be loaded on HPC system before submitting jobs

**envs:**

type: `dict`, optional, default: `{}`

argument path: `resources/envs`

The environment variables to be exported on before submitting jobs

Depending on the value of *batch\_type*, different sub args are accepted.

**batch\_type:**

type: `str` (flag key)

argument path: `resources/batch_type`

possible choices: `Torque`, `Shell`, `DistributedShell`, `PBS`, `Slurm`, `DpCloudServer`, `LSF`, `Lebesgue`

The batch job system type loaded from machine/*batch\_type*.

When *batch\_type* is set to `Torque` (or its alias `torque`):

**kwargs:**

type: `dict`, optional

argument path: `resources[Torque]/kwargs`

This field is empty for this batch.

When *batch\_type* is set to `Shell` (or its alias `shell`):

**kwargs:**

type: dict, optional

argument path: `resources[Shell]/kwargs`

This field is empty for this batch.

When *batch\_type* is set to `DistributedShell` (or its alias `distributedshell`):

**kwargs:**

type: dict, optional

argument path: `resources[DistributedShell]/kwargs`

This field is empty for this batch.

When *batch\_type* is set to `PBS` (or its alias `pbs`):

**kwargs:**

type: dict, optional

argument path: `resources[PBS]/kwargs`

This field is empty for this batch.

When *batch\_type* is set to `Slurm` (or its alias `slurm`):

**kwargs:**

type: dict, optional

argument path: `resources[Slurm]/kwargs`

Extra arguments.

**custom\_gpu\_line:**

type: `NoneType` | str, optional, default: `None`

argument path: `resources[Slurm]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with `#SBATCH`

When *batch\_type* is set to `DpCloudServer` (or its alias `dpcloudserver`):

**kwargs:**

type: dict, optional

argument path: `resources[DpCloudServer]/kwargs`

This field is empty for this batch.

When *batch\_type* is set to `LSF` (or its alias `lsf`):

**kwargs:**

type: dict

argument path: `resources[LSF]/kwargs`

Extra arguments.

**gpu\_usage:**

type: bool, optional, default: `False`

argument path: `resources[LSF]/kwargs/gpu_usage`

Choosing if GPU is used in the calculation step.

**gpu\_new\_syntax:**

type: `bool`, optional, default: `False`

argument path: `resources[LSF]/kwargs/gpu_new_syntax`

For LFS  $\geq$  10.1.0.3, new option `-gpu` for `#BSUB` could be used. If `False`, and old syntax would be used.

**gpu\_exclusive:**

type: `bool`, optional, default: `True`

argument path: `resources[LSF]/kwargs/gpu_exclusive`

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

**custom\_gpu\_line:**

type: `NoneType` | `str`, optional, default: `None`

argument path: `resources[LSF]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with `#BSUB`

When *batch\_type* is set to `Lebesgue` (or its alias `lebesgue`):

**kwargs:**

type: `dict`, optional

argument path: `resources[Lebesgue]/kwargs`

This field is empty for this batch.

## **TASK PARAMETERS**

**task:**

type: dict

argument path: task

**command:**

type: str

argument path: task/command

A command to be executed of this task. The expected return code is 0.

**task\_work\_path:**

type: str

argument path: task/task\_work\_path

The dir where the command to be executed.

**forward\_files:**

type: list

argument path: task/forward\_files

The files to be uploaded in task\_work\_path before the task executed.

**backward\_files:**

type: list

argument path: task/backward\_files

The files to be download to local\_root in task\_work\_path after the task finished

**outlog:**

type: NoneType | str

argument path: task/outlog

The out log file name. redirect from stdout

**errlog:**

type: NoneType | str

argument path: task/errlog

The err log file name. redirect from stderr





## DPDISPATCHER API

### 6.1 dpdispatcher package

`dpdispatcher.info()`

#### 6.1.1 Subpackages

`dpdispatcher.dpcloudserver` package

Submodules

`dpdispatcher.dpcloudserver.api` module

`class dpdispatcher.dpcloudserver.api.API(email, password)`  
Bases: `object`

#### Methods

<code>download</code>	
<code>download_from_url</code>	
<code>get</code>	
<code>get_jobs</code>	
<code>get_tasks</code>	
<code>get_tasks_v2</code>	
<code>get_tasks_v2_list</code>	
<code>job_create</code>	
<code>job_create_v2</code>	
<code>post</code>	
<code>refresh_token</code>	
<code>upload</code>	

`download(oss_file, save_file, endpoint, bucket_name)`

`download_from_url(url, save_file)`

```
get(url, params, retry=0)

get_jobs(page=1, per_page=10)

get_tasks(job_id, page=1, per_page=10)

get_tasks_v2(job_id, group_id, page=1, per_page=10)

get_tasks_v2_list(group_id, per_page=30)

job_create(job_type, oss_path, input_data, program_id=None)

job_create_v2(job_type, oss_path, input_data, program_id=None, group_id=None)

post(url, params, retry=0)

refresh_token()

upload(oss_task_zip, zip_task_file, endpoint, bucket_name)
```

### **dpdispatcher.dpcloudserver.config module**

### **dpdispatcher.dpcloudserver.retcode module**

```
class dpdispatcher.dpcloudserver.retcode.RETCODE
    Bases: object
    DATAERR = '2002'
    DBERR = '2000'
    IOERR = '2003'
    NODATA = '2300'
    OK = '0000'
    PARAMERR = '2101'
    PWDERR = '2104'
    REQERR = '2200'
    ROLEERR = '2103'
    THIRDERR = '2001'
    TOKENINVALID = '2100'
    UNDERDEBUG = '2301'
    UNKOWNERR = '2400'
    USERERR = '2102'
```

```
VERIFYERR = '2105'
```

**dpdispatcher.dpcloudserver.temp\_test module**

**dpdispatcher.dpcloudserver.zip\_file module**

```
dpdispatcher.dpcloudserver.zip_file.unzip_file(zip_file, out_dir='./')
```

```
dpdispatcher.dpcloudserver.zip_file.zip_file_list(root_path, zip_filename, file_list=[])
```

## 6.1.2 Submodules

### 6.1.3 dpdispatcher.JobStatus module

```
class dpdispatcher.JobStatus.JobStatus(value)
```

Bases: `enum.IntEnum`

An enumeration.

```
completing = 6
```

```
finished = 5
```

```
running = 3
```

```
terminated = 4
```

```
unknown = 100
```

```
unsubmitted = 1
```

```
waiting = 2
```

### 6.1.4 dpdispatcher.base\_context module

```
class dpdispatcher.base_context.BaseContext(*args, **kwargs)
```

Bases: `object`

#### Methods

<code>machine_arginfo()</code>	Generate the machine arginfo.
<code>machine_subfields()</code>	Generate the machine subfields.

<b>bind_submission</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>download</b>	
<b>kill</b>	
<b>load_from_dict</b>	
<b>read_file</b>	
<b>upload</b>	
<b>write_file</b>	

**bind\_submission**(*submission*)

**check\_finish**(*proc*)

**clean**()

**download**(*submission*, *check\_exists=False*, *mark\_failure=True*, *back\_error=False*)

**kill**(*proc*)

**classmethod load\_from\_dict**(*context\_dict*)

**classmethod machine\_arginfo**() → dargs.dargs.Argument

Generate the machine arginfo.

**Returns**

**Argument** machine arginfo

**classmethod machine\_subfields**() → List[dargs.dargs.Argument]

Generate the machine subfields.

**Returns**

**list[Argument]** machine subfields

**options** = {'DpCloudServerContext', 'HDFSContext', 'LazyLocalContext',  
'LebesgueContext', 'LocalContext', 'SSHContext'}

**read\_file**(*fname*)

```

subclasses_dict = {'DpCloudServer': <class
'dpdispatcher.dp_cloud_server_context.DpCloudServerContext'>,
'DpCloudServerContext': <class
'dpdispatcher.dp_cloud_server_context.DpCloudServerContext'>, 'HDFS': <class
'dpdispatcher.hdfs_context.HDFSContext'>, 'HDFSContext': <class
'dpdispatcher.hdfs_context.HDFSContext'>, 'LazyLocal': <class
'dpdispatcher.lazy_local_context.LazyLocalContext'>, 'LazyLocalContext': <class
'dpdispatcher.lazy_local_context.LazyLocalContext'>, 'Lebesgue': <class
'dpdispatcher.dp_cloud_server_context.LebesgueContext'>, 'LebesgueContext': <class
'dpdispatcher.dp_cloud_server_context.LebesgueContext'>, 'Local': <class
'dpdispatcher.local_context.LocalContext'>, 'LocalContext': <class
'dpdispatcher.local_context.LocalContext'>, 'SSH': <class
'dpdispatcher.ssh_context.SSHContext'>, 'SSHContext': <class
'dpdispatcher.ssh_context.SSHContext'>, 'dpcloudserver': <class
'dpdispatcher.dp_cloud_server_context.DpCloudServerContext'>,
'dpcloudservercontext': <class
'dpdispatcher.dp_cloud_server_context.DpCloudServerContext'>, 'hdfs': <class
'dpdispatcher.hdfs_context.HDFSContext'>, 'hdfscontext': <class
'dpdispatcher.hdfs_context.HDFSContext'>, 'lazylocal': <class
'dpdispatcher.lazy_local_context.LazyLocalContext'>, 'lazylocalcontext': <class
'dpdispatcher.lazy_local_context.LazyLocalContext'>, 'lebesgue': <class
'dpdispatcher.dp_cloud_server_context.LebesgueContext'>, 'lebesguecontext': <class
'dpdispatcher.dp_cloud_server_context.LebesgueContext'>, 'local': <class
'dpdispatcher.local_context.LocalContext'>, 'localcontext': <class
'dpdispatcher.local_context.LocalContext'>, 'ssh': <class
'dpdispatcher.ssh_context.SSHContext'>, 'sshcontext': <class
'dpdispatcher.ssh_context.SSHContext'>}]

upload(submission)

write_file(fname, write_str)

```

### 6.1.5 dpdispatcher.distributed\_shell module

**class** dpdispatcher.distributed\_shell.DistributedShell(\*args, \*\*kwargs)  
 Bases: [dpdispatcher.machine.Machine](#)

#### Methods

<a href="#">do_submit</a> (job)	submit th job to yarn using distributed shell
<a href="#">resources_arginfo</a> ()	Generate the resources arginfo.
<a href="#">resources_subfields</a> ()	Generate the resources subfields.

<b>arginfo</b>	
<b>bind_context</b>	
<b>check_finish_tag</b>	
<b>check_if_recover</b>	
<b>check_status</b>	
<b>default_resources</b>	
<b>deserialize</b>	
<b>gen_command_env_cuda_devices</b>	
<b>gen_script</b>	
<b>gen_script_command</b>	
<b>gen_script_custom_flags_lines</b>	
<b>gen_script_end</b>	
<b>gen_script_env</b>	
<b>gen_script_header</b>	
<b>gen_script_wait</b>	
<b>load_from_dict</b>	
<b>load_from_json</b>	
<b>serialize</b>	
<b>sub_script_cmd</b>	
<b>sub_script_head</b>	

**check\_finish\_tag**(*job*)

**check\_status**(*job*)

**do\_submit**(*job*)

submit th job to yarn using distributed shell

**Parameters**

**job** [Job class instance] job to be submitted

**Returns**

**job\_id: string** submit process id

**gen\_script\_end**(*job*)

**gen\_script\_env**(*job*)

**gen\_script\_header**(*job*)

## 6.1.6 dpdispatcher.dp\_cloud\_server module

**class** dpdispatcher.dp\_cloud\_server.DpCloudServer(\*args, \*\*kwargs)

Bases: *dpdispatcher.machine.Machine*

### Methods

<i>do_submit</i> (job)	submit a single job, assuming that no job is running there.
resources_arginfo()	Generate the resources arginfo.
resources_subfields()	Generate the resources subfields.

arginfo	
bind_context	
check_finish_tag	
check_if_recover	
check_status	
default_resources	
deserialize	
gen_command_env_cuda_devices	
gen_local_script	
gen_script	
gen_script_command	
gen_script_custom_flags_lines	
gen_script_end	
gen_script_env	
gen_script_header	
gen_script_wait	
load_from_dict	
load_from_json	
map_dp_job_state	
serialize	
sub_script_cmd	
sub_script_head	

**check\_finish\_tag**(job)

**check\_if\_recover**(submission)

**check\_status**(job)

**do\_submit**(job)

submit a single job, assuming that no job is running there.

**gen\_local\_script**(job)

**gen\_script**(job)

`gen_script_header(job)`

`static map_dp_job_state(status)`

`class dpdispatcher.dp_cloud_server.Lebesgue(*args, **kwargs)`

Bases: `dpdispatcher.dp_cloud_server.DpCloudServer`

### Methods

<code>do_submit(job)</code>	submit a single job, assuming that no job is running there.
<code>resources_arginfo()</code>	Generate the resources arginfo.
<code>resources_subfields()</code>	Generate the resources subfields.

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>gen_command_env_cuda_devices</code>	
<code>gen_local_script</code>	
<code>gen_script</code>	
<code>gen_script_command</code>	
<code>gen_script_custom_flags_lines</code>	
<code>gen_script_end</code>	
<code>gen_script_env</code>	
<code>gen_script_header</code>	
<code>gen_script_wait</code>	
<code>load_from_dict</code>	
<code>load_from_json</code>	
<code>map_dp_job_state</code>	
<code>serialize</code>	
<code>sub_script_cmd</code>	
<code>sub_script_head</code>	



### 6.1.7 dpdispatcher.dp\_cloud\_server\_context module

**class** dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext(\*args, \*\*kwargs)  
 Bases: *dpdispatcher.base\_context.BaseContext*

#### Methods

<code>machine_arginfo()</code>	Generate the machine arginfo.
<code>machine_subfields()</code>	Generate the machine subfields.

<b>bind_submission</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>check_home_file_exists</b>	
<b>clean</b>	
<b>download</b>	
<b>kill</b>	
<b>load_from_dict</b>	
<b>read_file</b>	
<b>read_home_file</b>	
<b>upload</b>	
<b>write_file</b>	
<b>write_home_file</b>	
<b>write_local_file</b>	

**bind\_submission**(*submission*)

**check\_file\_exists**(*fname*)

**check\_home\_file\_exists**(*fname*)

**clean**()

**download**(*submission*)

**kill**(*cmd\_pipes*)

**classmethod load\_from\_dict**(*context\_dict*)

**classmethod machine\_subfields**() → List[dargs.dargs.Argument]  
 Generate the machine subfields.

#### Returns

**list[Argument]** machine subfields

**read\_file**(*fname*)

```
read_home_file(fname)
```

```
upload(submission)
```

```
write_file(fname, write_str)
```

```
write_home_file(fname, write_str)
```

```
write_local_file(fname, write_str)
```

```
class dpdispatcher.dp_cloud_server_context.LebesgueContext(*args, **kwargs)
    Bases: dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext
```

### Methods

<code>machine_arginfo()</code>	Generate the machine arginfo.
<code>machine_subfields()</code>	Generate the machine subfields.

<code>bind_submission</code>	
<code>check_file_exists</code>	
<code>check_finish</code>	
<code>check_home_file_exists</code>	
<code>clean</code>	
<code>download</code>	
<code>kill</code>	
<code>load_from_dict</code>	
<code>read_file</code>	
<code>read_home_file</code>	
<code>upload</code>	
<code>write_file</code>	
<code>write_home_file</code>	
<code>write_local_file</code>	

## 6.1.8 dpdispatcher.dpdisp module

```
dpdispatcher.dpdisp.main()
```

### 6.1.9 dpdispatcher.hdfs\_cli module

**class** `dpdispatcher.hdfs_cli.HDFS`

Bases: `object`

Fundamental class for HDFS basic manipulation

#### Methods

<code>copy_from_local(local_path, to_uri)</code>	Returns: True on success Raises: on unexpected error
<code>exists(uri)</code>	Check existence of hdfs uri Returns: True on exists Raises: RuntimeError
<code>mkdir(uri)</code>	Make new hdfs directory Returns: True on success Raises: RuntimeError
<code>remove(uri)</code>	Check existence of hdfs uri Returns: True on exists Raises: RuntimeError

<code>copy_to_local</code>	
<code>move</code>	
<code>read_hdfs_file</code>	

**static** `copy_from_local(local_path, to_uri)`

Returns: True on success Raises: on unexpected error

**static** `copy_to_local(from_uri, local_path)`

**static** `exists(uri)`

Check existence of hdfs uri Returns: True on exists Raises: RuntimeError

**static** `mkdir(uri)`

Make new hdfs directory Returns: True on success Raises: RuntimeError

**static** `move(from_uri, to_uri)`

**static** `read_hdfs_file(uri)`

**static** `remove(uri)`

Check existence of hdfs uri Returns: True on exists Raises: RuntimeError

### 6.1.10 dpdispatcher.hdfs\_context module

**class** `dpdispatcher.hdfs_context.HDFSContext(*args, **kwargs)`

Bases: `dpdispatcher.base_context.BaseContext`

## Methods

<code>download(submission[, check_exists, ...])</code>	download backward files from HDFS root dir
<code>machine_arginfo()</code>	Generate the machine arginfo.
<code>machine_subfields()</code>	Generate the machine subfields.
<code>upload(submission[, dereference])</code>	upload forward files and forward command files to HDFS root dir

<b>bind_submission</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>get_job_root</b>	
<b>kill</b>	
<b>load_from_dict</b>	
<b>read_file</b>	
<b>write_file</b>	

**bind\_submission**(*submission*)

**check\_file\_exists**(*fname*)

**clean**()

**download**(*submission*, *check\_exists=False*, *mark\_failure=True*, *back\_error=False*)  
download backward files from HDFS root dir

**Parameters**

**submission** [Submission class instance] represents a collection of tasks, such as backward file names

**Returns**

**none**

**get\_job\_root**()

**classmethod load\_from\_dict**(*context\_dict*)

**read\_file**(*fname*)

**upload**(*submission*, *dereference=True*)  
upload forward files and forward command files to HDFS root dir

**Parameters**

**submission** [Submission class instance] represents a collection of tasks, such as forward file names

**Returns**

**none**

```
write_file(fname, write_str)
```

### 6.1.11 dpdispatcher.lazy\_local\_context module

```
class dpdispatcher.lazy_local_context.LazyLocalContext(*args, **kwargs)
    Bases: dpdispatcher.base_context.BaseContext
```

#### Methods

<code>machine_arginfo()</code>	Generate the machine arginfo.
<code>machine_subfields()</code>	Generate the machine subfields.

<b>bind_submission</b>	
<b>block_call</b>	
<b>block_checkcall</b>	
<b>call</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>download</b>	
<b>get_job_root</b>	
<b>get_return</b>	
<b>kill</b>	
<b>load_from_dict</b>	
<b>read_file</b>	
<b>upload</b>	
<b>write_file</b>	

```
bind_submission(submission)
```

```
block_call(cmd)
```

```
block_checkcall(cmd)
```

```
call(cmd)
```

```
check_file_exists(fname)
```

```
check_finish(proc)
```

```
clean()
```

```
download(jobs, check_exists=False, mark_failure=True, back_error=False)
```

**get\_job\_root()**

**get\_return(*proc*)**

**kill(*proc*)**

**classmethod load\_from\_dict(*context\_dict*)**

**read\_file(*fname*)**

**upload(*jobs*, *dereference=True*)**

**write\_file(*fname*, *write\_str*)**

**class** dpdispatcher.lazy\_local\_context.SPRetObj(*ret*)  
 Bases: [object](#)

### Methods

<b>read</b>	
<b>readlines</b>	

**read()**

**readlines()**

## 6.1.12 dpdispatcher.local\_context module

**class** dpdispatcher.local\_context.LocalContext(*\*args*, *\*\*kwargs*)  
 Bases: [dpdispatcher.base\\_context.BaseContext](#)

### Methods

<b>machine_arginfo()</b>	Generate the machine arginfo.
<b>machine_subfields()</b>	Generate the machine subfields.

<b>bind_submission</b>	
<b>block_call</b>	
<b>block_checkcall</b>	
<b>call</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>download</b>	
<b>download_</b>	
<b>get_job_root</b>	
<b>get_return</b>	
<b>kill</b>	
<b>load_from_dict</b>	
<b>read_file</b>	
<b>upload</b>	
<b>upload_</b>	
<b>write_file</b>	

**bind\_submission**(*submission*)

**block\_call**(*cmd*)

**block\_checkcall**(*cmd*)

**call**(*cmd*)

**check\_file\_exists**(*fname*)

**check\_finish**(*proc*)

**clean**()

**download**(*submission*, *check\_exists=False*, *mark\_failure=True*, *back\_error=False*)

**download\_**(*job\_dirs*, *remote\_down\_files*, *check\_exists=False*, *mark\_failure=True*, *back\_error=False*)

**get\_job\_root**()

**get\_return**(*proc*)

**kill**(*proc*)

**classmethod load\_from\_dict**(*context\_dict*)

**read\_file**(*fname*)

**upload**(*submission*)

**upload\_**(*job\_dirs*, *local\_up\_files*, *dereference=True*)

**write\_file**(*fname*, *write\_str*)

**class** `dpdispatcher.local_context.SPRetObj`(*ret*)  
 Bases: `object`

### Methods

<b>read</b>	
<b>readlines</b>	

**read**()

**readlines**()

## 6.1.13 dpdispatcher.lsf module

**class** `dpdispatcher.lsf.LSF`(\*args, \*\*kwargs)  
 Bases: `dpdispatcher.machine.Machine`  
 LSF batch

### Methods

<hr/>	
<code>default_resources</code> (resources)	
<hr/>	
<code>do_submit</code> (job)	submit a single job, assuming that no job is running there.
<hr/>	
<code>resources_arginfo</code> ()	Generate the resources arginfo.
<hr/>	
<code>resources_subfields</code> ()	Generate the resources subfields.
<hr/>	



<b>arginfo</b>	
<b>bind_context</b>	
<b>check_finish_tag</b>	
<b>check_if_recover</b>	
<b>check_status</b>	
<b>deserialize</b>	
<b>gen_command_env_cuda_devices</b>	
<b>gen_script</b>	
<b>gen_script_command</b>	
<b>gen_script_custom_flags_lines</b>	
<b>gen_script_end</b>	
<b>gen_script_env</b>	
<b>gen_script_header</b>	
<b>gen_script_wait</b>	
<b>load_from_dict</b>	
<b>load_from_json</b>	
<b>serialize</b>	
<b>sub_script_cmd</b>	
<b>sub_script_head</b>	

**check\_finish\_tag**(*job*)

**check\_status**(*job*)

**default\_resources**(*resources*)

**do\_submit**(*job*)

submit a single job, assuming that no job is running there.

**gen\_script**(*job*)

**gen\_script\_header**(*job*)

**classmethod resources\_subfields**() → List[dargs.dargs.Argument]

Generate the resources subfields.

**Returns**

**list**[Argument] resources subfields

**sub\_script\_cmd**(*res*)

**sub\_script\_head**(*res*)

### 6.1.14 dpdispatcher.machine module

**class** dpdispatcher.machine.Machine(\*args, \*\*kwargs)

Bases: `object`

A machine is used to handle the connection with remote machines.

#### Parameters

**context** [SubClass derived from BaseContext] The context is used to maintain the connection with remote machine.

#### Methods

<code>do_submit(job)</code>	submit a single job, assuming that no job is running there.
<code>resources_arginfo()</code>	Generate the resources arginfo.
<code>resources_subfields()</code>	Generate the resources subfields.

<b>arginfo</b>	
<b>bind_context</b>	
<b>check_finish_tag</b>	
<b>check_if_recover</b>	
<b>check_status</b>	
<b>default_resources</b>	
<b>deserialize</b>	
<b>gen_command_env_cuda_devices</b>	
<b>gen_script</b>	
<b>gen_script_command</b>	
<b>gen_script_custom_flags_lines</b>	
<b>gen_script_end</b>	
<b>gen_script_env</b>	
<b>gen_script_header</b>	
<b>gen_script_wait</b>	
<b>load_from_dict</b>	
<b>load_from_json</b>	
<b>serialize</b>	
<b>sub_script_cmd</b>	
<b>sub_script_head</b>	

**classmethod** `arginfo()`

**bind\_context**(*context*)

**check\_finish\_tag**(\*\*kwargs)

**check\_if\_recover**(*submission*)

**check\_status**(*job*)

---

```

default_resources(res)

classmethod deserialize(machine_dict)

do_submit(job)
    submit a single job, assuming that no job is running there.

gen_command_env_cuda_devices(resources)

gen_script(job)

gen_script_command(job)

gen_script_custom_flags_lines(job)

gen_script_end(job)

gen_script_env(job)

gen_script_header(job)

gen_script_wait(resources)

classmethod load_from_dict(machine_dict)

classmethod load_from_json(json_path)

options = {'DistributedShell', 'DpCloudServer', 'LSF', 'Lebesgue', 'PBS', 'Shell',
'Slurm', 'Torque'}

classmethod resources_arginfo() → dargs.dargs.Argument
    Generate the resources arginfo.

    Returns
        Argument resources arginfo

classmethod resources_subfields() → List[dargs.dargs.Argument]
    Generate the resources subfields.

    Returns
        list[Argument] resources subfields

serialize(if_empty_remote_profile=False)

sub_script_cmd(res)

sub_script_head(res)

```

```

subclasses_dict = {'DistributedShell': <class
'dpdispatcher.distributed_shell.DistributedShell'>, 'DpCloudServer': <class
'dpdispatcher.dp_cloud_server.DpCloudServer'>, 'LSF': <class
'dpdispatcher.lsf.LSF'>, 'Lebesgue': <class
'dpdispatcher.dp_cloud_server.Lebesgue'>, 'PBS': <class 'dpdispatcher.pbs.PBS'>,
'Shell': <class 'dpdispatcher.shell.Shell'>, 'Slurm': <class
'dpdispatcher.slurm.Slurm'>, 'Torque': <class 'dpdispatcher.pbs.Torque'>,
'distributedshell': <class 'dpdispatcher.distributed_shell.DistributedShell'>,
'dpcloudserver': <class 'dpdispatcher.dp_cloud_server.DpCloudServer'>, 'lebesgue':
<class 'dpdispatcher.dp_cloud_server.Lebesgue'>, 'lsf': <class
'dpdispatcher.lsf.LSF'>, 'pbs': <class 'dpdispatcher.pbs.PBS'>, 'shell': <class
'dpdispatcher.shell.Shell'>, 'slurm': <class 'dpdispatcher.slurm.Slurm'>, 'torque':
<class 'dpdispatcher.pbs.Torque'>}}

```

### 6.1.15 dpdispatcher.pbs module

**class** dpdispatcher.pbs.PBS(\*args, \*\*kwargs)

Bases: [dpdispatcher.machine.Machine](#)

#### Methods

<a href="#">do_submit(job)</a>	submit a single job, assuming that no job is running there.
<a href="#">resources_arginfo()</a>	Generate the resources arginfo.
<a href="#">resources_subfields()</a>	Generate the resources subfields.

<a href="#">arginfo</a>	
<a href="#">bind_context</a>	
<a href="#">check_finish_tag</a>	
<a href="#">check_if_recover</a>	
<a href="#">check_status</a>	
<a href="#">default_resources</a>	
<a href="#">deserialize</a>	
<a href="#">gen_command_env_cuda_devices</a>	
<a href="#">gen_script</a>	
<a href="#">gen_script_command</a>	
<a href="#">gen_script_custom_flags_lines</a>	
<a href="#">gen_script_end</a>	
<a href="#">gen_script_env</a>	
<a href="#">gen_script_header</a>	
<a href="#">gen_script_wait</a>	
<a href="#">load_from_dict</a>	
<a href="#">load_from_json</a>	
<a href="#">serialize</a>	
<a href="#">sub_script_cmd</a>	
<a href="#">sub_script_head</a>	

[check\\_finish\\_tag\(job\)](#)

**check\_status**(*job*)

**default\_resources**(*resources*)

**do\_submit**(*job*)

submit a single job, assuming that no job is running there.

**gen\_script**(*job*)

**gen\_script\_header**(*job*)

**class** dpdispatcher.pbs.Torque(\*args, \*\*kwargs)

Bases: [dpdispatcher.pbs.PBS](#)

### Methods

do_submit( <i>job</i> )	submit a single job, assuming that no job is running there.
resources_arginfo()	Generate the resources arginfo.
resources_subfields()	Generate the resources subfields.

arginfo	
bind_context	
check_finish_tag	
check_if_recover	
check_status	
default_resources	
deserialize	
gen_command_env_cuda_devices	
gen_script	
gen_script_command	
gen_script_custom_flags_lines	
gen_script_end	
gen_script_env	
gen_script_header	
gen_script_wait	
load_from_dict	
load_from_json	
serialize	
sub_script_cmd	
sub_script_head	

**check\_status**(*job*)

### 6.1.16 dpdispatcher.shell module

**class** dpdispatcher.shell.**Shell**(\*args, \*\*kwargs)  
Bases: *dpdispatcher.machine.Machine*

#### Methods

<i>do_submit</i> (job)	submit a single job, assuming that no job is running there.
<i>resources_arginfo</i> ()	Generate the resources arginfo.
<i>resources_subfields</i> ()	Generate the resources subfields.

<b>arginfo</b>	
<b>bind_context</b>	
<b>check_finish_tag</b>	
<b>check_if_recover</b>	
<b>check_status</b>	
<b>default_resources</b>	
<b>deserialize</b>	
<b>gen_command_env_cuda_devices</b>	
<b>gen_script</b>	
<b>gen_script_command</b>	
<b>gen_script_custom_flags_lines</b>	
<b>gen_script_end</b>	
<b>gen_script_env</b>	
<b>gen_script_header</b>	
<b>gen_script_wait</b>	
<b>load_from_dict</b>	
<b>load_from_json</b>	
<b>serialize</b>	
<b>sub_script_cmd</b>	
<b>sub_script_head</b>	

**check\_finish\_tag**(job)

**check\_status**(job)

**default\_resources**(resources)

**do\_submit**(job)  
submit a single job, assuming that no job is running there.

**gen\_script**(job)

**gen\_script\_header**(job)

### 6.1.17 dpdispatcher.slurm module

**class** dpdispatcher.slurm.**Slurm**(\*args, \*\*kwargs)

Bases: *dpdispatcher.machine.Machine*

#### Methods

<i>do_submit</i> (job[, retry, max_retry])	submit a single job, assuming that no job is running there.
<i>resources_arginfo</i> ()	Generate the resources arginfo.
<i>resources_subfields</i> ()	Generate the resources subfields.

<b>arginfo</b>	
<b>bind_context</b>	
<b>check_finish_tag</b>	
<b>check_if_recover</b>	
<b>check_status</b>	
<b>default_resources</b>	
<b>deserialize</b>	
<b>gen_command_env_cuda_devices</b>	
<b>gen_script</b>	
<b>gen_script_command</b>	
<b>gen_script_custom_flags_lines</b>	
<b>gen_script_end</b>	
<b>gen_script_env</b>	
<b>gen_script_header</b>	
<b>gen_script_wait</b>	
<b>load_from_dict</b>	
<b>load_from_json</b>	
<b>serialize</b>	
<b>sub_script_cmd</b>	
<b>sub_script_head</b>	

**check\_finish\_tag**(job)

**check\_status**(job, retry=0, max\_retry=3)

**default\_resources**(resources)

**do\_submit**(job, retry=0, max\_retry=3)  
submit a single job, assuming that no job is running there.

**gen\_script**(job)

**gen\_script\_header**(job)

**classmethod resources\_subfields**() → List[dargs.dargs.Argument]  
Generate the resources subfields.

**Returns**

**list[Argument]** resources subfields

## 6.1.18 dpdispatcher.ssh\_context module

**class** `dpdispatcher.ssh_context.SSHContext(*args, **kwargs)`

Bases: `dpdispatcher.base_context.BaseContext`

**Attributes**

**sftp**

**ssh**

**Methods**

<code><i>block_checkcall</i>(cmd[, asynchronously, ...])</code>	Run command with arguments.
<code><i>machine_arginfo</i>()</code>	Generate the machine arginfo.
<code><i>machine_subfields</i>()</code>	Generate the machine subfields.

<b>bind_submission</b>	
<b>block_call</b>	
<b>call</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>close</b>	
<b>download</b>	
<b>get_job_root</b>	
<b>get_return</b>	
<b>kill</b>	
<b>load_from_dict</b>	
<b>read_file</b>	
<b>upload</b>	
<b>write_file</b>	

**bind\_submission**(*submission*)

**block\_call**(*cmd*)

**block\_checkcall**(*cmd*, *asynchronously=False*, *stderr\_whitelist=None*)

Run command with arguments. Wait for command to complete. If the return code was zero then return, otherwise raise RuntimeError.

**Parameters**

**cmd:** **str** The command to run.

**asynchronously:** **bool**, **optional**, **default=False** Run command asynchronously. If True, *nohup* will be used to run the command.



**call**(*cmd*)

**check\_file\_exists**(*fname*)

**check\_finish**(*cmd\_pipes*)

**clean**()

**close**()

**download**(*submission*, *check\_exists=False*, *mark\_failure=True*, *back\_error=False*)

**get\_job\_root**()

**get\_return**(*cmd\_pipes*)

**kill**(*cmd\_pipes*)

**classmethod load\_from\_dict**(*context\_dict*)

**classmethod machine\_subfields**() → List[dargs.dargs.Argument]

Generate the machine subfields.

**Returns**

**list[Argument]** machine subfields

**read\_file**(*fname*)

**property sftp**

**property ssh**

**upload**(*submission*, *dereference=True*)

**write\_file**(*fname*, *write\_str*)

**class** dpdispatcher.ssh\_context.**SSHSession**(*hostname*, *username*, *password=None*, *port=22*,  
*key\_filename=None*, *passphrase=None*, *timeout=10*,  
*totp\_secret=None*)

Bases: [object](#)

**Attributes**

[sftp](#) Returns sftp.

## Methods

---

<code>exec_command(cmd[, retry])</code>	Calling self.ssh.exec_command but has an exception check.
---	---

---

<b>arginfo</b>	
<b>close</b>	
<b>ensure_alive</b>	
<b>get_ssh_client</b>	

**static arginfo()**

**close()**

**ensure\_alive**(*max\_check=10, sleep\_time=10*)

**exec\_command**(*cmd, retry=0*)

Calling self.ssh.exec\_command but has an exception check.

**get\_ssh\_client()**

**property sftp**

Returns sftp. Open a new one if not existing.

## 6.1.19 dpdispatcher.submission module

**class** dpdispatcher.submission.Job(*job\_task\_list, \*, resources, machine=None*)

Bases: `object`

Job is generated by Submission automatically. A job ususally has many tasks and it may request computing resources from job scheduler systems. Each Job can generate a script file to be submitted to the job scheduler system or executed locally.

### Parameters

**job\_task\_list** [list of Task] the tasks belonging to the job

**resources** [Resources] the machine resources. Passed from Submission when it constructs jobs.

**machine** [machine] machine object to execute the job. Passed from Submission when it constructs jobs.

## Methods

<code>deserialize(job_dict[, machine])</code>	convert the job_dict to a Submission class object
<code>get_job_state()</code>	get the jobs.
<code>serialize([if_static])</code>	convert the Task class instance to a dictionary.

<code>get_hash</code>	
<code>handle_unexpected_job_state</code>	
<code>job_to_json</code>	
<code>register_job_id</code>	
<code>submit_job</code>	

**classmethod** `deserialize(job_dict, machine=None)`

convert the job\_dict to a Submission class object

### Parameters

**submission\_dict** [dict] path-like, the base directory of the local tasks

### Returns

**submission** [Job] the Job class instance converted from the job\_dict

**get\_hash()**

**get\_job\_state()**

get the jobs. Usually, this method will query the database of slurm or pbs job scheduler system and get the results.

## Notes

this method will not submit or resubmit the jobs if the job is unsubmitted.

**handle\_unexpected\_job\_state()**

**job\_to\_json()**

**register\_job\_id(job\_id)**

**serialize(if\_static=False)**

convert the Task class instance to a dictionary.

### Parameters

**if\_static** [bool] whether dump the job runtime information (job\_id, job\_state, fail\_count, job\_uuid etc.) to the dictionary.

### Returns

**task\_dict** [dict] the dictionary converted from the Task class instance

**submit\_job()**

```
class dpdispatcher.submission.Resources(number_node, cpu_per_node, gpu_per_node, queue_name,
                                       group_size, *, custom_flags=[],
                                       strategy={'if_cuda_multi_devices': False}, para_deg=1,
                                       module_unload_list=[], module_purge=False, module_list=[],
                                       source_list=[], envs={}, **kwargs)
```

Bases: `object`

Resources is used to describe the machine resources we need to do calculations.

### Parameters

**number\_node** [int] The number of node need for each *job*.

**cpu\_per\_node** [int] cpu numbers of each node.

**gpu\_per\_node** [int] gpu numbers of each node.

**queue\_name** [str] The queue name of batch job scheduler system.

**group\_size** [int] The number of *tasks* in a *job*.

**custom\_flags** [list of Str] The extra lines pass to job submitting script header

**strategy** [dict] strategies we use to generation job submitting scripts. `if_cuda_multi_devices` :  
bool

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS. If true, dpdispatcher will manually export environment variable `CUDA_VISIBLE_DEVICES` to different task. Usually, this option will be used with `Task.task_need_resources` variable simultaneously.

**para\_deg** [int] Decide how many tasks will be run in parallel. Usually run with *strategy*['if\_cuda\_multi\_devices']

**source\_list** [list of Path] The env file to be sourced before the command execution.

### Methods

<b>arginfo</b>	
<b>deserialize</b>	
<b>load_from_dict</b>	
<b>load_from_json</b>	
<b>serialize</b>	

**static** `arginfo()`

**classmethod** `deserialize(resources_dict)`

**classmethod** `load_from_dict(resources_dict)`

**classmethod** `load_from_json(json_file)`

**serialize()**

```
class dpdispatcher.submission.Submission(work_base, machine=None, resources=None,
                                         forward_common_files=[], backward_common_files=[], *,
                                         task_list=[])
```

Bases: `object`

A submission represents a collection of tasks. These tasks usually locate at a common directory. And these Tasks may share common files to be uploaded and downloaded.

#### Parameters

- work\_base** [Path] the base directory of the local tasks. It is usually the dir name of project .
- machine** [Machine] machine class object (for example, PBS, Slurm, Shell) to execute the jobs.  
The machine can still be bound after the instantiation with the `bind_submission` method.
- resources** [Resources] the machine resources (cpu or gpu) used to generate the slurm/pbs script
- forward\_common\_files** [list] the common files to be uploaded to other computers before the jobs begin
- backward\_common\_files** [list] the common files to be downloaded from other computers after the jobs finish
- task\_list** [list of Task] a list of tasks to be run.

#### Methods

<code>bind_machine(machine)</code>	bind this submission to a machine.
<code>check_all_finished()</code>	check whether all the jobs in the submission.
<code>deserialize(submission_dict[, machine])</code>	convert the submission_dict to a Submission class object
<code>generate_jobs()</code>	After tasks register to the self.belonging_tasks, This method generate the jobs and add these jobs to self.belonging_jobs.
<code>handle_unexpected_submission_state()</code>	handle unexpected job state of the submission.
<code>run_submission(*[, exit_on_submit, clean])</code>	main method to execute the submission.
<code>serialize([if_static])</code>	convert the Submission class instance to a dictionary.
<code>update_submission_state()</code>	check whether all the jobs in the submission.

<b>clean_jobs</b>	
<b>download_jobs</b>	
<b>get_hash</b>	
<b>register_task</b>	
<b>register_task_list</b>	
<b>submission_from_json</b>	
<b>submission_to_json</b>	
<b>try_recover_from_json</b>	
<b>upload_jobs</b>	

**bind\_machine**(*machine*)

bind this submission to a machine. update the machine's context remote\_root and local\_root.

#### Parameters

- machine** [Machine] the machine to bind with

**check\_all\_finished()**

check whether all the jobs in the submission.

**Notes**

This method will not handle unexpected job state in the submission.

**clean\_jobs()****classmethod deserialize(submission\_dict, machine=None)**

convert the submission\_dict to a Submission class object

**Parameters**

**submission\_dict** [dict] path-like, the base directory of the local tasks

**Returns**

**submission** [Submission] the Submission class instance converted from the submission\_dict

**download\_jobs()****generate\_jobs()**

After tasks register to the self.belonging\_tasks, This method generate the jobs and add these jobs to self.belonging\_jobs. The jobs are generated by the tasks randomly, and there are self.resources.group\_size tasks in a task. Why we randomly shuffle the tasks is under the consideration of load balance. The random seed is a constant (to be concrete, 42). And this insures that the jobs are equal when we re-run the program.

**get\_hash()****handle\_unexpected\_submission\_state()**

handle unexpected job state of the submission. If the job state is unsubmitted, submit the job. If the job state is terminated (killed unexpectedly), resubmit the job. If the job state is unknown, raise an error.

**register\_task(task)****register\_task\_list(task\_list)****run\_submission(\*, exit\_on\_submit=False, clean=True)**

main method to execute the submission. First, check whether old Submission exists on the remote machine, and try to recover from it. Second, upload the local files to the remote machine where the tasks to be executed. Third, run the submission defined previously. Forth, wait until the tasks in the submission finished and download the result file to local directory. if exit\_on\_submit is True, submission will exit.

**serialize(if\_static=False)**

convert the Submission class instance to a dictionary.

**Parameters**

**if\_static** [bool] whether dump the job runtime information (like job\_id, job\_state, fail\_count) to the dictionary.

**Returns**

**submission\_dict** [dict] the dictionary converted from the Submission class instance

**classmethod submission\_from\_json(json\_file\_name='submission.json')**

**submission\_to\_json()**

**try\_recover\_from\_json()**

**update\_submission\_state()**

check whether all the jobs in the submission.

### Notes

this method will not handle unexpected (like resubmit terminated) job state in the submission.

**upload\_jobs()**

**class dpdispatcher.submission.Task**(*command, task\_work\_path, forward\_files=[], backward\_files=[], outlog='log', errlog='err'*)

Bases: `object`

A task is a sequential command to be executed, as well as the files it depends on to transmit forward and backward.

### Parameters

**command** [Str] the command to be executed.

**task\_work\_path** [Path] the directory of each file where the files are dependent on.

**forward\_files** [list of Path] the files to be transmitted to remote machine before the command execute.

**backward\_files** [list of Path] the files to be transmitted from remote machine after the command finished.

**outlog** [Str] the filename to which command redirect stdout

**errlog** [Str] the filename to which command redirect stderr

### Methods

---

<code>deserialize(task_dict)</code>	convert the task_dict to a Task class object
-------------------------------------	--

---

<b>arginfo</b>	
<b>get_hash</b>	
<b>load_from_json</b>	
<b>serialize</b>	

**static arginfo()**

**classmethod deserialize**(*task\_dict*)

convert the task\_dict to a Task class object

### Parameters

**task\_dict** [dict] the dictionary which contains the task information

### Returns

```
_____
    task [Task] the Task class instance converted from the task_dict

get_hash()

classmethod load_from_json(json_file)

serialize()
```

### 6.1.20 dpdispatcher.utils module

`dpdispatcher.utils.generate_totp(secret: str, period: int = 30, token_length: int = 6) → int`  
Generate time-based one time password (TOTP) from the secret.

Some HPCs use TOTP for two-factor authentication for safety.

#### Parameters

**secret: str** The encoded secret provided by the HPC. It's usually extracted from a 2D code and base32 encoded.

**period: int, default=30** Time period where the code is valid in seconds.

**token\_length: int, default=6** The token length.

#### Returns

**token: int** The generated token.

#### References

<https://github.com/lepture/otpauth/blob/49914d83d36dbcd33c9e26f65002b21ce09a6303/otpauth.py#L143-L160>

`dpdispatcher.utils.get_sha256(filename)`  
Get sha256 of a file.

#### Parameters

**filename: str** The filename.

#### Returns

**sha256: str** The sha256.

`dpdispatcher.utils.run_cmd_with_all_output(cmd)`



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### d

- `dpdispatcher`, 21
- `dpdispatcher.base_context`, 23
- `dpdispatcher.distributed_shell`, 25
- `dpdispatcher.dp_cloud_server`, 27
- `dpdispatcher.dp_cloud_server_context`, 29
- `dpdispatcher.dpcloudserver`, 21
- `dpdispatcher.dpcloudserver.api`, 21
- `dpdispatcher.dpcloudserver.config`, 22
- `dpdispatcher.dpcloudserver.retcode`, 22
- `dpdispatcher.dpcloudserver.zip_file`, 23
- `dpdispatcher.dpdisp`, 30
- `dpdispatcher.hdfs_cli`, 31
- `dpdispatcher.hdfs_context`, 31
- `dpdispatcher.JobStatus`, 23
- `dpdispatcher.lazy_local_context`, 33
- `dpdispatcher.local_context`, 34
- `dpdispatcher.lsf`, 36
- `dpdispatcher.machine`, 38
- `dpdispatcher.pbs`, 40
- `dpdispatcher.shell`, 42
- `dpdispatcher.slurm`, 43
- `dpdispatcher.ssh_context`, 44
- `dpdispatcher.submission`, 46
- `dpdispatcher.utils`, 52



## A

API (class in *dpdispatcher.dpcloudserver.api*), 21  
 arginfo() (*dpdispatcher.machine.Machine* class method), 38  
 arginfo() (*dpdispatcher.ssh\_context.SSHSession* static method), 46  
 arginfo() (*dpdispatcher.submission.Resources* static method), 48  
 arginfo() (*dpdispatcher.submission.Task* static method), 51

## B

BaseContext (class in *dpdispatcher.base\_context*), 23  
 bind\_context() (*dpdispatcher.machine.Machine* method), 38  
 bind\_machine() (*dpdispatcher.submission.Submission* method), 49  
 bind\_submission() (*dpdispatcher.base\_context.BaseContext* method), 24  
 bind\_submission() (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 29  
 bind\_submission() (*dpdispatcher.hdfs\_context.HDFSContext* method), 32  
 bind\_submission() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 33  
 bind\_submission() (*dpdispatcher.local\_context.LocalContext* method), 35  
 bind\_submission() (*dpdispatcher.ssh\_context.SSHContext* method), 44  
 block\_call() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 33  
 block\_call() (*dpdispatcher.local\_context.LocalContext* method), 35  
 block\_call() (*dpdispatcher.ssh\_context.SSHContext* method), 44  
 block\_checkcall() (*dpdis-*

*patcher.lazy\_local\_context.LazyLocalContext* method), 33  
 block\_checkcall() (*dpdispatcher.local\_context.LocalContext* method), 35  
 block\_checkcall() (*dpdispatcher.ssh\_context.SSHContext* method), 44

## C

call() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 33  
 call() (*dpdispatcher.local\_context.LocalContext* method), 35  
 call() (*dpdispatcher.ssh\_context.SSHContext* method), 44  
 check\_all\_finished() (*dpdispatcher.submission.Submission* method), 49  
 check\_file\_exists() (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 29  
 check\_file\_exists() (*dpdispatcher.hdfs\_context.HDFSContext* method), 32  
 check\_file\_exists() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 33  
 check\_file\_exists() (*dpdispatcher.local\_context.LocalContext* method), 35  
 check\_file\_exists() (*dpdispatcher.ssh\_context.SSHContext* method), 45  
 check\_finish() (*dpdispatcher.base\_context.BaseContext* method), 24  
 check\_finish() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 33  
 check\_finish() (*dpdispatcher.local\_context.LocalContext* method),

35

`check_finish()` (*dpdispatcher.ssh\_context.SSHContext* method), 45

`check_finish_tag()` (*dpdispatcher.distributed\_shell.DistributedShell* method), 26

`check_finish_tag()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer* method), 27

`check_finish_tag()` (*dpdispatcher.lsf.LSF* method), 37

`check_finish_tag()` (*dpdispatcher.machine.Machine* method), 38

`check_finish_tag()` (*dpdispatcher.pbs.PBS* method), 40

`check_finish_tag()` (*dpdispatcher.shell.Shell* method), 42

`check_finish_tag()` (*dpdispatcher.slurm.Slurm* method), 43

`check_home_file_exits()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 29

`check_if_recover()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer* method), 27

`check_if_recover()` (*dpdispatcher.machine.Machine* method), 38

`check_status()` (*dpdispatcher.distributed\_shell.DistributedShell* method), 26

`check_status()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer* method), 27

`check_status()` (*dpdispatcher.lsf.LSF* method), 37

`check_status()` (*dpdispatcher.machine.Machine* method), 38

`check_status()` (*dpdispatcher.pbs.PBS* method), 40

`check_status()` (*dpdispatcher.pbs.Torque* method), 41

`check_status()` (*dpdispatcher.shell.Shell* method), 42

`check_status()` (*dpdispatcher.slurm.Slurm* method), 43

`clean()` (*dpdispatcher.base\_context.BaseContext* method), 24

`clean()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 29

`clean()` (*dpdispatcher.hdfs\_context.HDFSContext* method), 32

`clean()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 33

`clean()` (*dpdispatcher.local\_context.LocalContext* method), 35

`clean()` (*dpdispatcher.ssh\_context.SSHContext* method), 45

`clean_jobs()` (*dpdispatcher.submission.Submission* method), 50

`close()` (*dpdispatcher.ssh\_context.SSHContext* method), 45

`close()` (*dpdispatcher.ssh\_context.SSHSession* method), 46

`completing` (*dpdispatcher.JobStatus.JobStatus* attribute), 23

`copy_from_local()` (*dpdispatcher.hdfs\_cli.HDFS* static method), 31

`copy_to_local()` (*dpdispatcher.hdfs\_cli.HDFS* static method), 31

## D

`DATAERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 22

`DBERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 22

`default_resources()` (*dpdispatcher.lsf.LSF* method), 37

`default_resources()` (*dpdispatcher.machine.Machine* class method), 38

`default_resources()` (*dpdispatcher.pbs.PBS* method), 41

`default_resources()` (*dpdispatcher.shell.Shell* method), 42

`default_resources()` (*dpdispatcher.slurm.Slurm* method), 43

`deserialize()` (*dpdispatcher.machine.Machine* class method), 39

`deserialize()` (*dpdispatcher.submission.Job* class method), 47

`deserialize()` (*dpdispatcher.submission.Resources* class method), 48

`deserialize()` (*dpdispatcher.submission.Submission* class method), 50

`deserialize()` (*dpdispatcher.submission.Task* class method), 51

`DistributedShell` (class in *dpdispatcher.distributed\_shell*), 25

`do_submit()` (*dpdispatcher.distributed\_shell.DistributedShell* method), 26

`do_submit()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer* method), 27

`do_submit()` (*dpdispatcher.lsf.LSF* method), 37

`do_submit()` (*dpdispatcher.machine.Machine* method), 39

`do_submit()` (*dpdispatcher.pbs.PBS* method), 41

`do_submit()` (*dpdispatcher.shell.Shell* method), 42

`do_submit()` (*dpdispatcher.slurm.Slurm* method), 43

`download()` (*dpdispatcher.base\_context.BaseContext* method), 24

`download()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 29

`download()` (*dpdispatcher.dpcloudserver.api.API method*), 21  
`download()` (*dpdispatcher.hdfs\_context.HDFSContext method*), 32  
`download()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 33  
`download()` (*dpdispatcher.local\_context.LocalContext method*), 35  
`download()` (*dpdispatcher.ssh\_context.SSHContext method*), 45  
`download_()` (*dpdispatcher.local\_context.LocalContext method*), 35  
`download_from_url()` (*dpdispatcher.dpcloudserver.api.API method*), 21  
`download_jobs()` (*dpdispatcher.submission.Submission method*), 50  
`DpCloudServer` (class in *dpdispatcher.dp\_cloud\_server*), 27  
`DpCloudServerContext` (class in *dpdispatcher.dp\_cloud\_server\_context*), 29  
`dpdispatcher` module, 21  
`dpdispatcher.base_context` module, 23  
`dpdispatcher.distributed_shell` module, 25  
`dpdispatcher.dp_cloud_server` module, 27  
`dpdispatcher.dp_cloud_server_context` module, 29  
`dpdispatcher.dpcloudserver` module, 21  
`dpdispatcher.dpcloudserver.api` module, 21  
`dpdispatcher.dpcloudserver.config` module, 22  
`dpdispatcher.dpcloudserver.retcode` module, 22  
`dpdispatcher.dpcloudserver.zip_file` module, 23  
`dpdispatcher.dpdisp` module, 30  
`dpdispatcher.hdfs_cli` module, 31  
`dpdispatcher.hdfs_context` module, 31  
`dpdispatcher.JobStatus` module, 23  
`dpdispatcher.lazy_local_context` module, 33  
`dpdispatcher.local_context` module, 34  
`dpdispatcher.lsf` module, 36  
`dpdispatcher.machine` module, 38  
`dpdispatcher.pbs` module, 40  
`dpdispatcher.shell` module, 42  
`dpdispatcher.slurm` module, 43  
`dpdispatcher.ssh_context` module, 44  
`dpdispatcher.submission` module, 46  
`dpdispatcher.utils` module, 52  
**E**  
`ensure_alive()` (*dpdispatcher.ssh\_context.SSHSession method*), 46  
`exec_command()` (*dpdispatcher.ssh\_context.SSHSession method*), 46  
`exists()` (*dpdispatcher.hdfs\_cli.HDFS static method*), 31  
**F**  
`finished` (*dpdispatcher.JobStatus.JobStatus attribute*), 23  
**G**  
`gen_command_env_cuda_devices()` (*dpdispatcher.machine.Machine method*), 39  
`gen_local_script()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer method*), 27  
`gen_script()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer method*), 27  
`gen_script()` (*dpdispatcher.lsf.LSF method*), 37  
`gen_script()` (*dpdispatcher.machine.Machine method*), 39  
`gen_script()` (*dpdispatcher.pbs.PBS method*), 41  
`gen_script()` (*dpdispatcher.shell.Shell method*), 42  
`gen_script()` (*dpdispatcher.slurm.Slurm method*), 43  
`gen_script_command()` (*dpdispatcher.machine.Machine method*), 39  
`gen_script_custom_flags_lines()` (*dpdispatcher.machine.Machine method*), 39  
`gen_script_end()` (*dpdispatcher.distributed\_shell.DistributedShell method*), 26  
`gen_script_end()` (*dpdispatcher.machine.Machine method*), 39

`gen_script_env()` (*dpdispatcher.distributed\_shell.DistributedShell method*), 26  
`gen_script_env()` (*dpdispatcher.machine.Machine method*), 39  
`gen_script_header()` (*dpdispatcher.distributed\_shell.DistributedShell method*), 26  
`gen_script_header()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer method*), 27  
`gen_script_header()` (*dpdispatcher.lsf.LSF method*), 37  
`gen_script_header()` (*dpdispatcher.machine.Machine method*), 39  
`gen_script_header()` (*dpdispatcher.pbs.PBS method*), 41  
`gen_script_header()` (*dpdispatcher.shell.Shell method*), 42  
`gen_script_header()` (*dpdispatcher.slurm.Slurm method*), 43  
`gen_script_wait()` (*dpdispatcher.machine.Machine method*), 39  
`generate_jobs()` (*dpdispatcher.submission.Submission method*), 50  
`generate_totp()` (*in module dpdispatcher.utils*), 52  
`get()` (*dpdispatcher.dpcloudserver.api.API method*), 21  
`get_hash()` (*dpdispatcher.submission.Job method*), 47  
`get_hash()` (*dpdispatcher.submission.Submission method*), 50  
`get_hash()` (*dpdispatcher.submission.Task method*), 52  
`get_job_root()` (*dpdispatcher.hdfs\_context.HDFSContext method*), 32  
`get_job_root()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 33  
`get_job_root()` (*dpdispatcher.local\_context.LocalContext method*), 35  
`get_job_root()` (*dpdispatcher.ssh\_context.SSHContext method*), 45  
`get_job_state()` (*dpdispatcher.submission.Job method*), 47  
`get_jobs()` (*dpdispatcher.dpcloudserver.api.API method*), 22  
`get_return()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 34  
`get_return()` (*dpdispatcher.local\_context.LocalContext method*), 35  
`get_return()` (*dpdispatcher.ssh\_context.SSHContext method*), 45  
`get_sha256()` (*in module dpdispatcher.utils*), 52  
`get_ssh_client()` (*dpdispatcher.ssh\_context.SSHSession method*), 46  
`get_tasks()` (*dpdispatcher.dpcloudserver.api.API method*), 22  
`get_tasks_v2()` (*dpdispatcher.dpcloudserver.api.API method*), 22  
`get_tasks_v2_list()` (*dpdispatcher.dpcloudserver.api.API method*), 22  

## H

`handle_unexpected_job_state()` (*dpdispatcher.submission.Job method*), 47  
`handle_unexpected_submission_state()` (*dpdispatcher.submission.Submission method*), 50  
HDFS (*class in dpdispatcher.hdfs\_cli*), 31  
HDFSContext (*class in dpdispatcher.hdfs\_context*), 31  

## I

`info()` (*in module dpdispatcher*), 21  
IOERR (*dpdispatcher.dpcloudserver.retcode.RETCODE attribute*), 22  

## J

Job (*class in dpdispatcher.submission*), 46  
`job_create()` (*dpdispatcher.dpcloudserver.api.API method*), 22  
`job_create_v2()` (*dpdispatcher.dpcloudserver.api.API method*), 22  
`job_to_json()` (*dpdispatcher.submission.Job method*), 47  
JobStatus (*class in dpdispatcher.JobStatus*), 23  

## K

`kill()` (*dpdispatcher.base\_context.BaseContext method*), 24  
`kill()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method*), 29  
`kill()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 34  
`kill()` (*dpdispatcher.local\_context.LocalContext method*), 35  
`kill()` (*dpdispatcher.ssh\_context.SSHContext method*), 45  

## L

LazyLocalContext (*class in dpdispatcher.lazy\_local\_context*), 33  
Lebesgue (*class in dpdispatcher.dp\_cloud\_server*), 28  
LebesgueContext (*class in dpdispatcher.dp\_cloud\_server\_context*), 30



`load_from_dict()` (*dpdispatcher.base\_context.BaseContext* class method), 24  
`load_from_dict()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* class method), 29  
`load_from_dict()` (*dpdispatcher.hdfs\_context.HDFSContext* class method), 32  
`load_from_dict()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext* class method), 34  
`load_from_dict()` (*dpdispatcher.local\_context.LocalContext* class method), 35  
`load_from_dict()` (*dpdispatcher.machine.Machine* class method), 39  
`load_from_dict()` (*dpdispatcher.ssh\_context.SSHContext* class method), 45  
`load_from_dict()` (*dpdispatcher.submission.Resources* class method), 48  
`load_from_json()` (*dpdispatcher.machine.Machine* class method), 39  
`load_from_json()` (*dpdispatcher.submission.Resources* class method), 48  
`load_from_json()` (*dpdispatcher.submission.Task* class method), 52  
`LocalContext` (class in *dpdispatcher.local\_context*), 34  
`LSF` (class in *dpdispatcher.lsf*), 36

## M

`Machine` (class in *dpdispatcher.machine*), 38  
`machine_arginfo()` (*dpdispatcher.base\_context.BaseContext* class method), 24  
`machine_subfields()` (*dpdispatcher.base\_context.BaseContext* class method), 24  
`machine_subfields()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* class method), 29  
`machine_subfields()` (*dpdispatcher.ssh\_context.SSHContext* class method), 45  
`main()` (in module *dpdispatcher.dpdisp*), 30  
`map_dp_job_state()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer* static method), 28  
`mkdir()` (*dpdispatcher.hdfs\_cli.HDFS* static method), 31  
module  
    *dpdispatcher*, 21

*dpdispatcher.base\_context*, 23  
*dpdispatcher.distributed\_shell*, 25  
*dpdispatcher.dp\_cloud\_server*, 27  
*dpdispatcher.dp\_cloud\_server\_context*, 29  
*dpdispatcher.dpcloudserver*, 21  
*dpdispatcher.dpcloudserver.api*, 21  
*dpdispatcher.dpcloudserver.config*, 22  
*dpdispatcher.dpcloudserver.retcode*, 22  
*dpdispatcher.dpcloudserver.zip\_file*, 23  
*dpdispatcher.dpdisp*, 30  
*dpdispatcher.hdfs\_cli*, 31  
*dpdispatcher.hdfs\_context*, 31  
*dpdispatcher.JobStatus*, 23  
*dpdispatcher.lazy\_local\_context*, 33  
*dpdispatcher.local\_context*, 34  
*dpdispatcher.lsf*, 36  
*dpdispatcher.machine*, 38  
*dpdispatcher.pbs*, 40  
*dpdispatcher.shell*, 42  
*dpdispatcher.slurm*, 43  
*dpdispatcher.ssh\_context*, 44  
*dpdispatcher.submission*, 46  
*dpdispatcher.utils*, 52

`move()` (*dpdispatcher.hdfs\_cli.HDFS* static method), 31

## N

`NODATA` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 22

## O

`OK` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 22  
`options` (*dpdispatcher.base\_context.BaseContext* attribute), 24  
`options` (*dpdispatcher.machine.Machine* attribute), 39

## P

`PARAMERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 22  
`PBS` (class in *dpdispatcher.pbs*), 40  
`post()` (*dpdispatcher.dpcloudserver.api.API* method), 22  
`PWDERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 22

## R

`read()` (*dpdispatcher.lazy\_local\_context.SPRetObj* method), 34  
`read()` (*dpdispatcher.local\_context.SPRetObj* method), 36  
`read_file()` (*dpdispatcher.base\_context.BaseContext* method), 24  
`read_file()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 29

[read\\_file\(\)](#) ([dpdispatcher.hdfs\\_context.HDFSContext](#) method), 32  
[read\\_file\(\)](#) ([dpdispatcher.lazy\\_local\\_context.LazyLocalContext](#) method), 34  
[read\\_file\(\)](#) ([dpdispatcher.local\\_context.LocalContext](#) method), 35  
[read\\_file\(\)](#) ([dpdispatcher.ssh\\_context.SSHContext](#) method), 45  
[read\\_hdfs\\_file\(\)](#) ([dpdispatcher.hdfs\\_cli.HDFS](#) static method), 31  
[read\\_home\\_file\(\)](#) ([dpdispatcher.dp\\_cloud\\_server\\_context.DpCloudServerContext](#) method), 29  
[readlines\(\)](#) ([dpdispatcher.lazy\\_local\\_context.SPRetObj](#) method), 34  
[readlines\(\)](#) ([dpdispatcher.local\\_context.SPRetObj](#) method), 36  
[refresh\\_token\(\)](#) ([dpdispatcher.dpcloudserver.api.API](#) method), 22  
[register\\_job\\_id\(\)](#) ([dpdispatcher.submission.Job](#) method), 47  
[register\\_task\(\)](#) ([dpdispatcher.submission.Submission](#) method), 50  
[register\\_task\\_list\(\)](#) ([dpdispatcher.submission.Submission](#) method), 50  
[remove\(\)](#) ([dpdispatcher.hdfs\\_cli.HDFS](#) static method), 31  
[REQERR](#) ([dpdispatcher.dpcloudserver.retcode.RETCODE](#) attribute), 22  
[Resources](#) (class in [dpdispatcher.submission](#)), 47  
[resources\\_arginfo\(\)](#) ([dpdispatcher.machine.Machine](#) class method), 39  
[resources\\_subfields\(\)](#) ([dpdispatcher.lsf.LSF](#) class method), 37  
[resources\\_subfields\(\)](#) ([dpdispatcher.machine.Machine](#) class method), 39  
[resources\\_subfields\(\)](#) ([dpdispatcher.slurm.Slurm](#) class method), 43  
[RETCODE](#) (class in [dpdispatcher.dpcloudserver.retcode](#)), 22  
[ROLEERR](#) ([dpdispatcher.dpcloudserver.retcode.RETCODE](#) attribute), 22  
[run\\_cmd\\_with\\_all\\_output\(\)](#) (in module [dpdispatcher.utils](#)), 52  
[run\\_submission\(\)](#) ([dpdispatcher.submission.Submission](#) method), 50  
[running](#) ([dpdispatcher.JobStatus.JobStatus](#) attribute), 23

**S**

[serialize\(\)](#) ([dpdispatcher.machine.Machine](#) method), 39  
[serialize\(\)](#) ([dpdispatcher.submission.Job](#) method), 47  
[serialize\(\)](#) ([dpdispatcher.submission.Resources](#) method), 48  
[serialize\(\)](#) ([dpdispatcher.submission.Submission](#) method), 50  
[serialize\(\)](#) ([dpdispatcher.submission.Task](#) method), 52  
[sftp](#) ([dpdispatcher.ssh\\_context.SSHContext](#) property), 45  
[sftp](#) ([dpdispatcher.ssh\\_context.SSHSession](#) property), 46  
[Shell](#) (class in [dpdispatcher.shell](#)), 42  
[Slurm](#) (class in [dpdispatcher.slurm](#)), 43  
[SPRetObj](#) (class in [dpdispatcher.lazy\\_local\\_context](#)), 34  
[SPRetObj](#) (class in [dpdispatcher.local\\_context](#)), 36  
[ssh](#) ([dpdispatcher.ssh\\_context.SSHContext](#) property), 45  
[SSHContext](#) (class in [dpdispatcher.ssh\\_context](#)), 44  
[SSHSession](#) (class in [dpdispatcher.ssh\\_context](#)), 45  
[sub\\_script\\_cmd\(\)](#) ([dpdispatcher.lsf.LSF](#) method), 37  
[sub\\_script\\_cmd\(\)](#) ([dpdispatcher.machine.Machine](#) method), 39  
[sub\\_script\\_head\(\)](#) ([dpdispatcher.lsf.LSF](#) method), 37  
[sub\\_script\\_head\(\)](#) ([dpdispatcher.machine.Machine](#) method), 39  
[subclasses\\_dict](#) ([dpdispatcher.base\\_context.BaseContext](#) attribute), 24  
[subclasses\\_dict](#) ([dpdispatcher.machine.Machine](#) attribute), 39  
[Submission](#) (class in [dpdispatcher.submission](#)), 48  
[submission\\_from\\_json\(\)](#) ([dpdispatcher.submission.Submission](#) class method), 50  
[submission\\_to\\_json\(\)](#) ([dpdispatcher.submission.Submission](#) method), 50  
[submit\\_job\(\)](#) ([dpdispatcher.submission.Job](#) method), 47

**T**

[Task](#) (class in [dpdispatcher.submission](#)), 51  
[terminated](#) ([dpdispatcher.JobStatus.JobStatus](#) attribute), 23  
[THIRDERR](#) ([dpdispatcher.dpcloudserver.retcode.RETCODE](#) attribute), 22  
[TOKENINVALID](#) ([dpdispatcher.dpcloudserver.retcode.RETCODE](#) attribute), 22  
[Torque](#) (class in [dpdispatcher.pbs](#)), 41  
[try\\_recover\\_from\\_json\(\)](#) ([dpdispatcher.submission.Submission](#) method), 51

**U**

[UNDERDEBUG](#) ([dpdispatcher.dpcloudserver.retcode.RETCODE](#)

attribute), 22  
 unknown (dpdispatcher.JobStatus.JobStatus attribute), 23  
 UNKNOWNERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 22  
 unsubmitted (dpdispatcher.JobStatus.JobStatus attribute), 23  
 unzip\_file() (in module dpdispatcher.dpcloudserver.zip\_file), 23  
 update\_submission\_state() (dpdispatcher.submission.Submission method), 51  
 upload() (dpdispatcher.base\_context.BaseContext method), 25  
 upload() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 30  
 upload() (dpdispatcher.dpcloudserver.api.API method), 22  
 upload() (dpdispatcher.hdfs\_context.HDFSContext method), 32  
 upload() (dpdispatcher.lazy\_local\_context.LazyLocalContext method), 34  
 upload() (dpdispatcher.local\_context.LocalContext method), 35  
 upload() (dpdispatcher.ssh\_context.SSHContext method), 45  
 upload\_() (dpdispatcher.local\_context.LocalContext method), 36  
 upload\_jobs() (dpdispatcher.submission.Submission method), 51  
 USERERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 22

## V

VERIFYERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 22

## W

waiting (dpdispatcher.JobStatus.JobStatus attribute), 23  
 write\_file() (dpdispatcher.base\_context.BaseContext method), 25  
 write\_file() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 30  
 write\_file() (dpdispatcher.hdfs\_context.HDFSContext method), 32  
 write\_file() (dpdispatcher.lazy\_local\_context.LazyLocalContext method), 34  
 write\_file() (dpdispatcher.local\_context.LocalContext method), 36  
 write\_file() (dpdispatcher.ssh\_context.SSHContext method), 45  
 write\_home\_file() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 30