

---

# DPDispatcher

Deep Modeling

Oct 12, 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>11</b>
<b>5</b>	<b>Task parameters</b>	<b>13</b>
<b>6</b>	<b>DPDispatcher API</b>	<b>15</b>
6.1	dpdispatcher package . . . . .	15
<b>7</b>	<b>Indices and tables</b>	<b>45</b>
	<b>Python Module Index</b>	<b>47</b>
	<b>Index</b>	<b>49</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: Slurm, PBS, LSF, Shell, DpCloudServer

**context\_type:**

type: str

argument path: machine/context\_type

The connection used to remote machine. Option: LocalContext, LazyLocalContext, SSHContext DpCloud-ServerContext

**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.

**remote\_profile:**

type: dict

argument path: machine/remote\_profile

The information used to maintain the connection with remote machine. Only needed when context is ssh.

**hostname:**

type: str

argument path: machine/remote\_profile/hostname

hostname or ip of ssh connection.

**username:**

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

**password:**

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

**port:**

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

**key\_filename:**

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

**passphrase:**

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

**timeout:**

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

**totp\_secret:**

type: str | NoneType, optional, default: None  
argument path: machine/remote\_profile/totp\_secret  
Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

**clean\_asynchronously:**

type: bool, optional, default: False  
argument path: machine/clean\_asynchronously  
Clean the remote directory asynchronously after the job finishes.

## 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\_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



## **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: str | NoneType

argument path: task/outlog

The out log file name. redirect from stdout

**errlog:**

type: str | NoneType

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

<b>download</b>	
<b>get</b>	
<b>get_jobs</b>	
<b>get_tasks</b>	
<b>get_tasks_v2</b>	
<b>job_create</b>	
<b>job_create_v2</b>	
<b>post</b>	
<b>refresh_token</b>	
<b>upload</b>	

**download**(*oss\_file, save\_file, endpoint, bucket\_name*)

**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*)

**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

<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)`

`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'>, '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'>, '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

---

<i>do_submit</i> (job)	submit th job to yarn using distributed shell
------------------------	---

---

arginfo	
bind_context	
check_finish_tag	
check_if_recover	
check_status	
default_resources	
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	
sub_script_cmd	
sub_script_head	

**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

---

<code>do_submit(job)</code>	submit a single job, assuming that no job is running there.
-----------------------------	---

---

<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>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>sub_script_cmd</code>	
<code>sub_script_head</code>	

**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*)



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

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

#### Methods

<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*)

**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)
```

### 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>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**

<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>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

---

*default\_resources*(resources)

---

<i>do_submit</i> (job)	submit a single job, assuming that no job is running there.
------------------------	---

---

<b>arginfo</b>	
<b>bind_context</b>	
<b>check_finish_tag</b>	
<b>check_if_recover</b>	
<b>check_status</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>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*)

**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.
-----------------------------	---

---

<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>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>sub_script_cmd</b>	
<b>sub_script_head</b>	

`static arginfo()`

`bind_context(context)`

`check_finish_tag(**kwargs)`

`check_if_recover(submission)`

`check_status(job)`

`default_resources(res)`

`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)
```

```
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'>, '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'>, '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</a> ( <i>job</i> )	submit a single job, assuming that no job is running there.
--	---

---

<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>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>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*)

**class** dpdispatcher.pbs.Torque(\*args, \*\*kwargs)  
Bases: [dpdispatcher.pbs.PBS](#)

## Methods

<b>do_submit</b> ( <i>job</i> )	submit a single job, assuming that no job is running there.
---------------------------------	---

<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>gen_command_env_cuda_devices</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>sub_script_cmd</code>	
<code>sub_script_head</code>	

`check_status(job)`

### 6.1.16 dpdispatcher.shell module

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

#### Methods

---

<code>do_submit(job)</code>	submit a single job, assuming that no job is running there.
-----------------------------	---

---

<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>gen_command_env_cuda_devices</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>sub_script_cmd</code>	
<code>sub_script_head</code>	

`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

---

<code>do_submit(job[, retry, max_retry])</code>	submit a single job, assuming that no job is running there.
---	---

---

<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>gen_command_env_cuda_devices</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>sub_script_cmd</code>	
<code>sub_script_head</code>	

`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)`

### 6.1.18 dpdispatcher.ssh\_context module

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

Bases: `dpdispatcher.base_context.BaseContext`

#### Attributes

`sftp`

`ssh`

## Methods

---

<code>block_checkcall(cmd[, asynchronously, ...])</code>	Run command with arguments.
--	-----------------------------

---

<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*)

**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

---

<a href="#">exec_command</a> ( <i>cmd[, retry]</i> )	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_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] path-like, the base directory of the local tasks

**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.

continues on next page

Table 14 – continued from previous page

<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, if_none_local_root])</code>	convert the Submission class instance to a dictionary.
<code>update_submission_state()</code>	check whether all the jobs in the submission.

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

**`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*, *if\_none\_local\_root=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 comand 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](#), 15
- [dpdispatcher.base\\_context](#), 17
- [dpdispatcher.distributed\\_shell](#), 19
- [dpdispatcher.dp\\_cloud\\_server](#), 20
- [dpdispatcher.dp\\_cloud\\_server\\_context](#), 21
- [dpdispatcher.dpcloudserver](#), 15
- [dpdispatcher.dpcloudserver.api](#), 15
- [dpdispatcher.dpcloudserver.config](#), 16
- [dpdispatcher.dpcloudserver.retcode](#), 16
- [dpdispatcher.dpcloudserver.zip\\_file](#), 17
- [dpdispatcher.dpdisp](#), 22
- [dpdispatcher.hdfs\\_cli](#), 22
- [dpdispatcher.hdfs\\_context](#), 23
- [dpdispatcher.JobStatus](#), 17
- [dpdispatcher.lazy\\_local\\_context](#), 24
- [dpdispatcher.local\\_context](#), 25
- [dpdispatcher.lsf](#), 27
- [dpdispatcher.machine](#), 28
- [dpdispatcher.pbs](#), 30
- [dpdispatcher.shell](#), 32
- [dpdispatcher.slurm](#), 33
- [dpdispatcher.ssh\\_context](#), 34
- [dpdispatcher.submission](#), 37
- [dpdispatcher.utils](#), 42



## INDEX

### A

API (class in *dpdispatcher.dpcloudserver.api*), 15

*arginfo()* (*dpdispatcher.machine.Machine* static method), 29

*arginfo()* (*dpdispatcher.ssh\_context.SSHSession* static method), 36

*arginfo()* (*dpdispatcher.submission.Resources* static method), 39

*arginfo()* (*dpdispatcher.submission.Task* static method), 42

### B

*BaseContext* (class in *dpdispatcher.base\_context*), 17

*bind\_context()* (*dpdispatcher.machine.Machine* method), 29

*bind\_machine()* (*dpdispatcher.submission.Submission* method), 40

*bind\_submission()* (*dpdispatcher.base\_context.BaseContext* method), 17

*bind\_submission()* (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 21

*bind\_submission()* (*dpdispatcher.hdfs\_context.HDFSContext* method), 23

*bind\_submission()* (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 24

*bind\_submission()* (*dpdispatcher.local\_context.LocalContext* method), 26

*bind\_submission()* (*dpdispatcher.ssh\_context.SSHContext* method), 35

*block\_call()* (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 24

*block\_call()* (*dpdispatcher.local\_context.LocalContext* method), 26

*block\_call()* (*dpdispatcher.ssh\_context.SSHContext* method), 35

*block\_checkcall()* (*dpdis-*

*patcher.lazy\_local\_context.LazyLocalContext* method), 24

*block\_checkcall()* (*dpdispatcher.local\_context.LocalContext* method), 26

*block\_checkcall()* (*dpdispatcher.ssh\_context.SSHContext* method), 35

### C

*call()* (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 24

*call()* (*dpdispatcher.local\_context.LocalContext* method), 26

*call()* (*dpdispatcher.ssh\_context.SSHContext* method), 35

*check\_all\_finished()* (*dpdispatcher.submission.Submission* method), 40

*check\_file\_exists()* (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 21

*check\_file\_exists()* (*dpdispatcher.hdfs\_context.HDFSContext* method), 23

*check\_file\_exists()* (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 24

*check\_file\_exists()* (*dpdispatcher.local\_context.LocalContext* method), 26

*check\_file\_exists()* (*dpdispatcher.ssh\_context.SSHContext* method), 35

*check\_finish()* (*dpdispatcher.base\_context.BaseContext* method), 17

*check\_finish()* (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 24

*check\_finish()* (*dpdispatcher.local\_context.LocalContext* method),

26  
 check\_finish() (dpdispatcher.ssh\_context.SSHContext method), 35  
 check\_finish\_tag() (dpdispatcher.distributed\_shell.DistributedShell method), 19  
 check\_finish\_tag() (dpdispatcher.dp\_cloud\_server.DpCloudServer method), 20  
 check\_finish\_tag() (dpdispatcher.lsf.LSF method), 28  
 check\_finish\_tag() (dpdispatcher.machine.Machine method), 29  
 check\_finish\_tag() (dpdispatcher.pbs.PBS method), 31  
 check\_finish\_tag() (dpdispatcher.shell.Shell method), 33  
 check\_finish\_tag() (dpdispatcher.slurm.Slurm method), 34  
 check\_home\_file\_exists() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 21  
 check\_if\_recover() (dpdispatcher.dp\_cloud\_server.DpCloudServer method), 20  
 check\_if\_recover() (dpdispatcher.machine.Machine method), 29  
 check\_status() (dpdispatcher.distributed\_shell.DistributedShell method), 19  
 check\_status() (dpdispatcher.dp\_cloud\_server.DpCloudServer method), 20  
 check\_status() (dpdispatcher.lsf.LSF method), 28  
 check\_status() (dpdispatcher.machine.Machine method), 29  
 check\_status() (dpdispatcher.pbs.PBS method), 31  
 check\_status() (dpdispatcher.pbs.Torque method), 32  
 check\_status() (dpdispatcher.shell.Shell method), 33  
 check\_status() (dpdispatcher.slurm.Slurm method), 34  
 clean() (dpdispatcher.base\_context.BaseContext method), 18  
 clean() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 21  
 clean() (dpdispatcher.hdfs\_context.HDFSContext method), 23  
 clean() (dpdispatcher.lazy\_local\_context.LazyLocalContext method), 24  
 clean() (dpdispatcher.local\_context.LocalContext method), 26  
 clean() (dpdispatcher.ssh\_context.SSHContext method), 35  
 clean\_jobs() (dpdispatcher.submission.Submission method), 40  
 close() (dpdispatcher.ssh\_context.SSHContext method), 35  
 close() (dpdispatcher.ssh\_context.SSHSession method), 36  
 completing (dpdispatcher.JobStatus.JobStatus attribute), 17  
 copy\_from\_local() (dpdispatcher.hdfs\_cli.HDFS static method), 22  
 copy\_to\_local() (dpdispatcher.hdfs\_cli.HDFS static method), 22

## D

DATAERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 16  
 DBERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 16  
 default\_resources() (dpdispatcher.lsf.LSF method), 28  
 default\_resources() (dpdispatcher.machine.Machine method), 29  
 default\_resources() (dpdispatcher.pbs.PBS method), 31  
 default\_resources() (dpdispatcher.shell.Shell method), 33  
 default\_resources() (dpdispatcher.slurm.Slurm method), 34  
 deserialize() (dpdispatcher.submission.Job class method), 37  
 deserialize() (dpdispatcher.submission.Resources class method), 39  
 deserialize() (dpdispatcher.submission.Submission class method), 40  
 deserialize() (dpdispatcher.submission.Task class method), 42  
 DistributedShell (class in dpdispatcher.distributed\_shell), 19  
 do\_submit() (dpdispatcher.distributed\_shell.DistributedShell method), 19  
 do\_submit() (dpdispatcher.dp\_cloud\_server.DpCloudServer method), 20  
 do\_submit() (dpdispatcher.lsf.LSF method), 28  
 do\_submit() (dpdispatcher.machine.Machine method), 29  
 do\_submit() (dpdispatcher.pbs.PBS method), 31  
 do\_submit() (dpdispatcher.shell.Shell method), 33  
 do\_submit() (dpdispatcher.slurm.Slurm method), 34  
 download() (dpdispatcher.base\_context.BaseContext method), 18  
 download() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 21  
 download() (dpdispatcher.dpcloudserver.api.API method), 15

`download()` (*dpdispatcher.hdfs\_context.HDFSContext* module), 23  
`download()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext* module), 24  
`download()` (*dpdispatcher.local\_context.LocalContext* module), 26  
`download()` (*dpdispatcher.ssh\_context.SSHContext* module), 35  
`download_()` (*dpdispatcher.local\_context.LocalContext* module), 26  
`download_jobs()` (*dpdispatcher.submission.Submission* module), 40  
`DpCloudServer` (class in *dpdispatcher.dp\_cloud\_server*), 20  
`DpCloudServerContext` (class in *dpdispatcher.dp\_cloud\_server\_context*), 21  
`dpdispatcher` module, 15  
`dpdispatcher.base_context` module, 17  
`dpdispatcher.distributed_shell` module, 19  
`dpdispatcher.dp_cloud_server` module, 20  
`dpdispatcher.dp_cloud_server_context` module, 21  
`dpdispatcher.dpcloudserver` module, 15  
`dpdispatcher.dpcloudserver.api` module, 15  
`dpdispatcher.dpcloudserver.config` module, 16  
`dpdispatcher.dpcloudserver.retcode` module, 16  
`dpdispatcher.dpcloudserver.zip_file` module, 17  
`dpdispatcher.dpdisp` module, 22  
`dpdispatcher.hdfs_cli` module, 22  
`dpdispatcher.hdfs_context` module, 23  
`dpdispatcher.JobStatus` module, 17  
`dpdispatcher.lazy_local_context` module, 24  
`dpdispatcher.local_context` module, 25  
`dpdispatcher.lsf` module, 27  
`dpdispatcher.machine` module, 28  
`dpdispatcher.pbs` module, 30  
`dpdispatcher.shell` module, 32  
`dpdispatcher.slurm` module, 33  
`dpdispatcher.ssh_context` module, 34  
`dpdispatcher.submission` module, 37  
`dpdispatcher.utils` module, 42  
**E**  
`ensure_alive()` (*dpdispatcher.ssh\_context.SSHSession* module), 36  
`exec_command()` (*dpdispatcher.ssh\_context.SSHSession* module), 36  
`exists()` (*dpdispatcher.hdfs\_cli.HDFS* static method), 22  
**F**  
`finished` (*dpdispatcher.JobStatus.JobStatus* attribute), 17  
**G**  
`gen_command_env_cuda_devices()` (*dpdispatcher.machine.Machine* method), 29  
`gen_local_script()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer* method), 20  
`gen_script()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer* method), 20  
`gen_script()` (*dpdispatcher.lsf.LSF* method), 28  
`gen_script()` (*dpdispatcher.machine.Machine* method), 29  
`gen_script()` (*dpdispatcher.pbs.PBS* method), 31  
`gen_script()` (*dpdispatcher.shell.Shell* method), 33  
`gen_script()` (*dpdispatcher.slurm.Slurm* method), 34  
`gen_script_command()` (*dpdispatcher.machine.Machine* method), 29  
`gen_script_custom_flags_lines()` (*dpdispatcher.machine.Machine* method), 29  
`gen_script_end()` (*dpdispatcher.distributed\_shell.DistributedShell* method), 19  
`gen_script_end()` (*dpdispatcher.machine.Machine* method), 29  
`gen_script_env()` (*dpdispatcher.distributed\_shell.DistributedShell* method), 19  
`gen_script_env()` (*dpdispatcher.machine.Machine* method), 30

- `gen_script_header()` (*dpdispatcher.distributed\_shell.DistributedShell method*), 19
- `gen_script_header()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer method*), 20
- `gen_script_header()` (*dpdispatcher.lsf.LSF method*), 28
- `gen_script_header()` (*dpdispatcher.machine.Machine method*), 30
- `gen_script_header()` (*dpdispatcher.pbs.PBS method*), 31
- `gen_script_header()` (*dpdispatcher.shell.Shell method*), 33
- `gen_script_header()` (*dpdispatcher.slurm.Slurm method*), 34
- `gen_script_wait()` (*dpdispatcher.machine.Machine method*), 30
- `generate_jobs()` (*dpdispatcher.submission.Submission method*), 40
- `generate_totp()` (*in module dpdispatcher.utils*), 42
- `get()` (*dpdispatcher.dpcloudserver.api.API method*), 15
- `get_hash()` (*dpdispatcher.submission.Job method*), 37
- `get_hash()` (*dpdispatcher.submission.Submission method*), 40
- `get_hash()` (*dpdispatcher.submission.Task method*), 42
- `get_job_root()` (*dpdispatcher.hdfs\_context.HDFSContext method*), 23
- `get_job_root()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 24
- `get_job_root()` (*dpdispatcher.local\_context.LocalContext method*), 26
- `get_job_root()` (*dpdispatcher.ssh\_context.SSHContext method*), 35
- `get_job_state()` (*dpdispatcher.submission.Job method*), 37
- `get_jobs()` (*dpdispatcher.dpcloudserver.api.API method*), 15
- `get_return()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 25
- `get_return()` (*dpdispatcher.local\_context.LocalContext method*), 26
- `get_return()` (*dpdispatcher.ssh\_context.SSHContext method*), 35
- `get_sha256()` (*in module dpdispatcher.utils*), 43
- `get_ssh_client()` (*dpdispatcher.ssh\_context.SSHSession method*), 36
- `get_tasks()` (*dpdispatcher.dpcloudserver.api.API method*), 15
- `get_tasks_v2()` (*dpdispatcher.dpcloudserver.api.API method*), 16
- ## H
- `handle_unexpected_job_state()` (*dpdispatcher.submission.Job method*), 38
- `handle_unexpected_submission_state()` (*dpdispatcher.submission.Submission method*), 40
- HDFS (*class in dpdispatcher.hdfs\_cli*), 22
- HDFSContext (*class in dpdispatcher.hdfs\_context*), 23

## I

`info()` (*in module dpdispatcher*), 15

IOERR (*dpdispatcher.dpcloudserver.retcode.RETCODE attribute*), 16

## J

Job (*class in dpdispatcher.submission*), 37

`job_create()` (*dpdispatcher.dpcloudserver.api.API method*), 16

`job_create_v2()` (*dpdispatcher.dpcloudserver.api.API method*), 16

`job_to_json()` (*dpdispatcher.submission.Job method*), 38

JobStatus (*class in dpdispatcher.JobStatus*), 17

## K

`kill()` (*dpdispatcher.base\_context.BaseContext method*), 18

`kill()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method*), 21

`kill()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 25

`kill()` (*dpdispatcher.local\_context.LocalContext method*), 26

`kill()` (*dpdispatcher.ssh\_context.SSHContext method*), 36

## L

LazyLocalContext (*class in dpdispatcher.lazy\_local\_context*), 24

`load_from_dict()` (*dpdispatcher.base\_context.BaseContext class method*), 18

`load_from_dict()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext class method*), 21

`load_from_dict()` (*dpdispatcher.hdfs\_context.HDFSContext class method*), 23

`load_from_dict()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext class method*), 25



load\_from\_dict() (dpdispatcher.local\_context.LocalContext class method), 26

load\_from\_dict() (dpdispatcher.machine.Machine class method), 30

load\_from\_dict() (dpdispatcher.ssh\_context.SSHContext class method), 36

load\_from\_dict() (dpdispatcher.submission.Resources class method), 39

load\_from\_json() (dpdispatcher.machine.Machine class method), 30

load\_from\_json() (dpdispatcher.submission.Resources class method), 39

load\_from\_json() (dpdispatcher.submission.Task class method), 42

LocalContext (class in dpdispatcher.local\_context), 25

LSF (class in dpdispatcher.lsf), 27

## M

Machine (class in dpdispatcher.machine), 28

main() (in module dpdispatcher.dpdisp), 22

map\_dp\_job\_state() (dpdispatcher.dp\_cloud\_server.DpCloudServer static method), 20

mkdir() (dpdispatcher.hdfs\_cli.HDFS static method), 22

module

- dpdispatcher, 15
- dpdispatcher.base\_context, 17
- dpdispatcher.distributed\_shell, 19
- dpdispatcher.dp\_cloud\_server, 20
- dpdispatcher.dp\_cloud\_server\_context, 21
- dpdispatcher.dpcloudserver, 15
- dpdispatcher.dpcloudserver.api, 15
- dpdispatcher.dpcloudserver.config, 16
- dpdispatcher.dpcloudserver.retcode, 16
- dpdispatcher.dpcloudserver.zip\_file, 17
- dpdispatcher.dpdisp, 22
- dpdispatcher.hdfs\_cli, 22
- dpdispatcher.hdfs\_context, 23
- dpdispatcher.JobStatus, 17
- dpdispatcher.lazy\_local\_context, 24
- dpdispatcher.local\_context, 25
- dpdispatcher.lsf, 27
- dpdispatcher.machine, 28
- dpdispatcher.pbs, 30
- dpdispatcher.shell, 32
- dpdispatcher.slurm, 33
- dpdispatcher.ssh\_context, 34
- dpdispatcher.submission, 37
- dpdispatcher.utils, 42

move() (dpdispatcher.hdfs\_cli.HDFS static method), 22

## N

NODATA (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 16

## O

OK (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 16

## P

PARAMERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 16

PBS (class in dpdispatcher.pbs), 30

post() (dpdispatcher.dpcloudserver.api.API method), 16

PWDERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 16

## R

read() (dpdispatcher.lazy\_local\_context.SPRetObj method), 25

read() (dpdispatcher.local\_context.SPRetObj method), 27

read\_file() (dpdispatcher.base\_context.BaseContext method), 18

read\_file() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 21

read\_file() (dpdispatcher.hdfs\_context.HDFSContext method), 23

read\_file() (dpdispatcher.lazy\_local\_context.LazyLocalContext method), 25

read\_file() (dpdispatcher.local\_context.LocalContext method), 26

read\_file() (dpdispatcher.ssh\_context.SSHContext method), 36

read\_hdfs\_file() (dpdispatcher.hdfs\_cli.HDFS static method), 22

read\_home\_file() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 21

readlines() (dpdispatcher.lazy\_local\_context.SPRetObj method), 25

readlines() (dpdispatcher.local\_context.SPRetObj method), 27

refresh\_token() (dpdispatcher.dpcloudserver.api.API method), 16

register\_job\_id() (dpdispatcher.submission.Job method), 38

register\_task() (dpdispatcher.submission.Submission method), 41

register\_task\_list() (dpdispatcher.submission.Submission method), 41

remove() (dpdispatcher.hdfs\_cli.HDFS static method), 22

REQERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 16  
 Resources (*class in dpdispatcher.submission*), 38  
 RETCODE (*class in dpdispatcher.dpcloudserver.retcode*), 16  
 ROLEERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 16  
 run\_cmd\_with\_all\_output() (*in module dpdispatcher.utils*), 43  
 run\_submission() (*dpdispatcher.submission.Submission* method), 41  
 running (*dpdispatcher.JobStatus.JobStatus* attribute), 17

## S

serialize() (*dpdispatcher.submission.Job* method), 38  
 serialize() (*dpdispatcher.submission.Resources* method), 39  
 serialize() (*dpdispatcher.submission.Submission* method), 41  
 serialize() (*dpdispatcher.submission.Task* method), 42  
 sftp (*dpdispatcher.ssh\_context.SSHContext* property), 36  
 sftp (*dpdispatcher.ssh\_context.SSHSession* property), 36  
 Shell (*class in dpdispatcher.shell*), 32  
 Slurm (*class in dpdispatcher.slurm*), 33  
 SPRetObj (*class in dpdispatcher.lazy\_local\_context*), 25  
 SPRetObj (*class in dpdispatcher.local\_context*), 27  
 ssh (*dpdispatcher.ssh\_context.SSHContext* property), 36  
 SSHContext (*class in dpdispatcher.ssh\_context*), 34  
 SSHSession (*class in dpdispatcher.ssh\_context*), 36  
 sub\_script\_cmd() (*dpdispatcher.lsf.LSF* method), 28  
 sub\_script\_cmd() (*dpdispatcher.machine.Machine* method), 30  
 sub\_script\_head() (*dpdispatcher.lsf.LSF* method), 28  
 sub\_script\_head() (*dpdispatcher.machine.Machine* method), 30  
 subclasses\_dict (*dpdispatcher.base\_context.BaseContext* attribute), 18  
 subclasses\_dict (*dpdispatcher.machine.Machine* attribute), 30  
 Submission (*class in dpdispatcher.submission*), 39  
 submission\_from\_json() (*dpdispatcher.submission.Submission* class method), 41  
 submission\_to\_json() (*dpdispatcher.submission.Submission* method), 41  
 submit\_job() (*dpdispatcher.submission.Job* method), 38

## T

Task (*class in dpdispatcher.submission*), 41  
 terminated (*dpdispatcher.JobStatus.JobStatus* attribute), 17  
 THIRDERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 16  
 TOKENINVALID (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 16  
 Torque (*class in dpdispatcher.pbs*), 31  
 try\_recover\_from\_json() (*dpdispatcher.submission.Submission* method), 41

## U

UNDERDEBUG (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 16  
 unknown (*dpdispatcher.JobStatus.JobStatus* attribute), 17  
 UNKOWNERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 16  
 unsubmitted (*dpdispatcher.JobStatus.JobStatus* attribute), 17  
 unzip\_file() (*in module dpdispatcher.dpcloudserver.zip\_file*), 17  
 update\_submission\_state() (*dpdispatcher.submission.Submission* method), 41  
 upload() (*dpdispatcher.base\_context.BaseContext* method), 18  
 upload() (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 21  
 upload() (*dpdispatcher.dpcloudserver.api.API* method), 16  
 upload() (*dpdispatcher.hdfs\_context.HDFSContext* method), 23  
 upload() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 25  
 upload() (*dpdispatcher.local\_context.LocalContext* method), 27  
 upload() (*dpdispatcher.ssh\_context.SSHContext* method), 36  
 upload\_() (*dpdispatcher.local\_context.LocalContext* method), 27  
 upload\_jobs() (*dpdispatcher.submission.Submission* method), 41  
 USERERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 16

## V

VERIFYERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 16

## W

waiting (*dpdispatcher.JobStatus.JobStatus* attribute), 17

`write_file()` (*dpdispatcher.base\_context.BaseContext*  
method), 18

`write_file()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext*  
method), 21

`write_file()` (*dpdispatcher.hdfs\_context.HDFSContext*  
method), 24

`write_file()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext*  
method), 25

`write_file()` (*dpdispatcher.local\_context.LocalContext*  
method), 27

`write_file()` (*dpdispatcher.ssh\_context.SSHContext*  
method), 36

`write_home_file()` (*dpdis-*  
*patcher.dp\_cloud\_server\_context.DpCloudServerContext*  
method), 21

`write_local_file()` (*dpdis-*  
*patcher.dp\_cloud\_server\_context.DpCloudServerContext*  
method), 21

## Z

`zip_file_list()` (in module *dpdis-*  
*patcher.dpcloudserver.zip\_file*), 17