

---

# **DPDispatcher**

**Deep Modeling**

**Jun 22, 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>
<b>7</b>	<b>Indices and tables</b>	<b>25</b>
	<b>Python Module Index</b>	<b>27</b>
	<b>Index</b>	<b>29</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: NoneType | str, optional, default: None  
argument path: machine/remote\_profile/key\_filename  
key\_filename used by ssh connection

**passphrase:**

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

**timeout:**

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

**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: 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**

`dpdispatcher.info()`

`class dpdispatcher.JobStatus.JobStatus(value)`

An enumeration.

`completing = 6`

`finished = 5`

`running = 3`

`terminated = 4`

`unknown = 100`

`unsubmitted = 1`

`waiting = 2`

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

`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'>, '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'>, '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)

class dpdispatcher.dp_cloud_server.DpCloudServer(*args, **kwargs)

    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_context.DpCloudServerContext(*args, **kwargs)

    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)
dpdispatcher.dpdisp.main()
class dpdispatcher.lazy_local_context.LazyLocalContext(*args, **kwargs)

    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)

    read()
    readlines()
class dpdispatcher.local_context.LocalContext(*args, **kwargs)

    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)

    read()
    readlines()
class dpdispatcher.lsf.LSF(*args, **kwargs)
    LSF batch
    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)
class dpdispatcher.machine.Machine(*args, **kwargs)
    A machine is used to handle the connection with remote machines.
    context [SubClass derived from BaseContext] The context is used to maintain the connection with remote machine.
    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 = {'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'>,
'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'>}}
class dpdispatcher.pbs.PBS(*args, **kwargs)

    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.shell.Shell(*args, **kwargs)

    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.slurm.Slurm(*args, **kwargs)

    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)

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

    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.

        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)

    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.

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

Job is generated by Submission automatically. A job usually 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.

**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.

**classmethod deserialize**(*job\_dict*, *machine=None*)

convert the *job\_dict* to a Submission class object

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

**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.

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.

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

**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*)

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

**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.

**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=[])
```

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.

**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.

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

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

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

### check\_all\_finished()

check whether all the jobs in the submission.

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

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

**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()**

**get\_submission\_state()**

check whether all the jobs in the submission.

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

**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 unexpectly), 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.

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

**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()**

**upload\_jobs()**

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

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

**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

**static arginfo()**

**classmethod deserialize(task\_dict)**

convert the task\_dict to a Task class object

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

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

```
    get_hash()
    serialize()

dpdispatcher.dpcloudserver.api.download(oss_file, save_file, endpoint, bucket_name)
dpdispatcher.dpcloudserver.api.get(url, params)
dpdispatcher.dpcloudserver.api.get_jobs(page=1, per_page=10)
dpdispatcher.dpcloudserver.api.get_tasks(job_id, page=1, per_page=10)
dpdispatcher.dpcloudserver.api.job_create(job_type, oss_path, input_data)
dpdispatcher.dpcloudserver.api.login(username, password)
dpdispatcher.dpcloudserver.api.post(url, params)
dpdispatcher.dpcloudserver.api.upload(oss_task_zip, zip_task_file, endpoint, bucket_name)
class dpdispatcher.dpcloudserver.retcode.RETCODE

    DATAERR = '2002'
    DBERR = '2000'
    IOERR = '2003'
    LOGINERR = '2100'
    NODATA = '2300'
    OK = '0000'
    PARAMERR = '2101'
    PWDERR = '2104'
    REQERR = '2200'
    ROLEERR = '2103'
    THIRDERR = '2001'
    UNDERDEBUG = '2301'
    UNKOWNERR = '2400'
    USERERR = '2102'
    VERIFYERR = '2105'

dpdispatcher.dpcloudserver.zip_file.unzip_file(zip_file, out_dir='./')
dpdispatcher.dpcloudserver.zip_file.zip_file_list(root_path, zip_filename, file_list=[])
```

## INDICES AND TABLES

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



## PYTHON MODULE INDEX

### d

- `dpdispatcher`, 15
- `dpdispatcher.base_context`, 15
- `dpdispatcher.dp_cloud_server`, 16
- `dpdispatcher.dp_cloud_server_context`, 16
- `dpdispatcher.dpcloudserver`, 24
- `dpdispatcher.dpcloudserver.api`, 24
- `dpdispatcher.dpcloudserver.config`, 24
- `dpdispatcher.dpcloudserver.retcode`, 24
- `dpdispatcher.dpcloudserver.zip_file`, 24
- `dpdispatcher.dpdisp`, 17
- `dpdispatcher.JobStatus`, 15
- `dpdispatcher.lazy_local_context`, 17
- `dpdispatcher.local_context`, 17
- `dpdispatcher.lsf`, 18
- `dpdispatcher.machine`, 18
- `dpdispatcher.pbs`, 19
- `dpdispatcher.shell`, 19
- `dpdispatcher.slurm`, 19
- `dpdispatcher.ssh_context`, 20
- `dpdispatcher.submission`, 21





## INDEX

### A

`arginfo()` (`dpdispatcher.machine.Machine` static method), 18  
`arginfo()` (`dpdispatcher.ssh_context.SSHSession` static method), 20  
`arginfo()` (`dpdispatcher.submission.Resources` static method), 22  
`arginfo()` (`dpdispatcher.submission.Task` static method), 23

### B

`BaseContext` (class in `dpdispatcher.base_context`), 15  
`bind_context()` (`dpdispatcher.machine.Machine` method), 18  
`bind_machine()` (`dpdispatcher.submission.Submission` method), 22  
`bind_submission()` (`dpdispatcher.base_context.BaseContext` method), 15  
`bind_submission()` (`dpdispatcher.dp_cloud_server_context.DpCloudServerContext` method), 16  
`bind_submission()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 17  
`bind_submission()` (`dpdispatcher.local_context.LocalContext` method), 17  
`bind_submission()` (`dpdispatcher.ssh_context.SSHContext` method), 20  
`block_call()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 17  
`block_call()` (`dpdispatcher.local_context.LocalContext` method), 17  
`block_call()` (`dpdispatcher.ssh_context.SSHContext` method), 20  
`block_checkcall()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 17  
`block_checkcall()` (`dpdispatcher.local_context.LocalContext` method),

17

`block_checkcall()` (`dpdispatcher.ssh_context.SSHContext` method), 20

### C

`call()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 17  
`call()` (`dpdispatcher.local_context.LocalContext` method), 17  
`call()` (`dpdispatcher.ssh_context.SSHContext` method), 20  
`check_all_finished()` (`dpdispatcher.submission.Submission` method), 22  
`check_file_exists()` (`dpdispatcher.dp_cloud_server_context.DpCloudServerContext` method), 16  
`check_file_exists()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 17  
`check_file_exists()` (`dpdispatcher.local_context.LocalContext` method), 17  
`check_file_exists()` (`dpdispatcher.ssh_context.SSHContext` method), 20  
`check_finish()` (`dpdispatcher.base_context.BaseContext` method), 15  
`check_finish()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 17  
`check_finish()` (`dpdispatcher.local_context.LocalContext` method), 17  
`check_finish()` (`dpdispatcher.ssh_context.SSHContext` method), 20  
`check_finish_tag()` (`dpdispatcher.dp_cloud_server.DpCloudServer` method), 16  
`check_finish_tag()` (`dpdispatcher.lsf.LSF` method),

18  
 check\_finish\_tag() (dpdispatcher.machine.Machine method), 18  
 check\_finish\_tag() (dpdispatcher.pbs.PBS method), 19  
 check\_finish\_tag() (dpdispatcher.shell.Shell method), 19  
 check\_finish\_tag() (dpdispatcher.slurm.Slurm method), 19  
 check\_home\_file\_exists() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 16  
 check\_if\_recover() (dpdispatcher.dp\_cloud\_server.DpCloudServer method), 16  
 check\_if\_recover() (dpdispatcher.machine.Machine method), 18  
 check\_status() (dpdispatcher.dp\_cloud\_server.DpCloudServer method), 16  
 check\_status() (dpdispatcher.lsf.LSF method), 18  
 check\_status() (dpdispatcher.machine.Machine method), 18  
 check\_status() (dpdispatcher.pbs.PBS method), 19  
 check\_status() (dpdispatcher.shell.Shell method), 19  
 check\_status() (dpdispatcher.slurm.Slurm method), 19  
 clean() (dpdispatcher.base\_context.BaseContext method), 15  
 clean() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 17  
 clean() (dpdispatcher.lazy\_local\_context.LazyLocalContext method), 17  
 clean() (dpdispatcher.local\_context.LocalContext method), 17  
 clean() (dpdispatcher.ssh\_context.SSHContext method), 20  
 clean\_jobs() (dpdispatcher.submission.Submission method), 22  
 close() (dpdispatcher.ssh\_context.SSHContext method), 20  
 close() (dpdispatcher.ssh\_context.SSHSession method), 20  
 completing (dpdispatcher.JobStatus.JobStatus attribute), 15

**D**  
 DATAERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 24  
 DBERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 24  
 default\_resources() (dpdispatcher.lsf.LSF method), 18  
 default\_resources() (dpdispatcher.machine.Machine method), 18  
 default\_resources() (dpdispatcher.pbs.PBS method), 19  
 default\_resources() (dpdispatcher.shell.Shell method), 19  
 default\_resources() (dpdispatcher.slurm.Slurm method), 19  
 deserialize() (dpdispatcher.submission.Job class method), 21  
 deserialize() (dpdispatcher.submission.Resources class method), 22  
 deserialize() (dpdispatcher.submission.Submission class method), 22  
 deserialize() (dpdispatcher.submission.Task class method), 23  
 do\_submit() (dpdispatcher.dp\_cloud\_server.DpCloudServer method), 16  
 do\_submit() (dpdispatcher.lsf.LSF method), 18  
 do\_submit() (dpdispatcher.machine.Machine method), 18  
 do\_submit() (dpdispatcher.pbs.PBS method), 19  
 do\_submit() (dpdispatcher.shell.Shell method), 19  
 do\_submit() (dpdispatcher.slurm.Slurm method), 19  
 download() (dpdispatcher.base\_context.BaseContext method), 15  
 download() (dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext method), 16  
 download() (dpdispatcher.lazy\_local\_context.LazyLocalContext method), 17  
 download() (dpdispatcher.local\_context.LocalContext method), 17  
 download() (dpdispatcher.ssh\_context.SSHContext method), 20  
 download() (in module dpdispatcher.dpcloudserver.api), 24  
 download\_() (dpdispatcher.local\_context.LocalContext method), 17  
 download\_jobs() (dpdispatcher.submission.Submission method), 22  
 DpCloudServer (class in dpdispatcher.dp\_cloud\_server), 16  
 DpCloudServerContext (class in dpdispatcher.dp\_cloud\_server\_context), 16  
 dpdispatcher module, 15  
 dpdispatcher.base\_context module, 15  
 dpdispatcher.dp\_cloud\_server module, 16  
 dpdispatcher.dp\_cloud\_server\_context module, 16  
 dpdispatcher.dpcloudserver

module, 24  
 dpdispatcher.dpcloudserver.api  
   module, 24  
 dpdispatcher.dpcloudserver.config  
   module, 24  
 dpdispatcher.dpcloudserver.retcode  
   module, 24  
 dpdispatcher.dpcloudserver.zip\_file  
   module, 24  
 dpdispatcher.dpdisp  
   module, 17  
 dpdispatcher.JobStatus  
   module, 15  
 dpdispatcher.lazy\_local\_context  
   module, 17  
 dpdispatcher.local\_context  
   module, 17  
 dpdispatcher.lsf  
   module, 18  
 dpdispatcher.machine  
   module, 18  
 dpdispatcher.pbs  
   module, 19  
 dpdispatcher.shell  
   module, 19  
 dpdispatcher.slurm  
   module, 19  
 dpdispatcher.ssh\_context  
   module, 20  
 dpdispatcher.submission  
   module, 21

## E

ensure\_alive() (*dpdispatcher.ssh\_context.SSHSession*  
   *method*), 20  
 exec\_command() (*dpdispatcher.ssh\_context.SSHSession*  
   *method*), 20

## F

finished (*dpdispatcher.JobStatus.JobStatus* attribute),  
   15

## G

gen\_command\_env\_cuda\_devices() (*dpdis-*  
   *patcher.machine.Machine* *method*), 18  
 gen\_local\_script() (*dpdis-*  
   *patcher.dp\_cloud\_server.DpCloudServer*  
   *method*), 16  
 gen\_script() (*dpdispatcher.dp\_cloud\_server.DpCloudServer*  
   *method*), 16  
 gen\_script() (*dpdispatcher.lsf.LSF* *method*), 18  
 gen\_script() (*dpdispatcher.machine.Machine* *method*),  
   18  
 gen\_script() (*dpdispatcher.pbs.PBS* *method*), 19

gen\_script() (*dpdispatcher.shell.Shell* *method*), 19  
 gen\_script() (*dpdispatcher.slurm.Slurm* *method*), 20  
 gen\_script\_command() (*dpdis-*  
   *patcher.machine.Machine* *method*), 18  
 gen\_script\_custom\_flags\_lines() (*dpdis-*  
   *patcher.machine.Machine* *method*), 18  
 gen\_script\_end() (*dpdispatcher.machine.Machine*  
   *method*), 19  
 gen\_script\_env() (*dpdispatcher.machine.Machine*  
   *method*), 19  
 gen\_script\_header() (*dpdis-*  
   *patcher.dp\_cloud\_server.DpCloudServer*  
   *method*), 16  
 gen\_script\_header() (*dpdispatcher.lsf.LSF* *method*),  
   18  
 gen\_script\_header() (*dpdispatcher.machine.Machine*  
   *method*), 19  
 gen\_script\_header() (*dpdispatcher.pbs.PBS* *method*),  
   19  
 gen\_script\_header() (*dpdispatcher.shell.Shell*  
   *method*), 19  
 gen\_script\_header() (*dpdispatcher.slurm.Slurm*  
   *method*), 20  
 gen\_script\_wait() (*dpdispatcher.machine.Machine*  
   *method*), 19  
 generate\_jobs() (*dpdis-*  
   *patcher.submission.Submission* *method*),  
   22  
 get() (*in module dpdispatcher.dpcloudserver.api*), 24  
 get\_hash() (*dpdispatcher.submission.Job* *method*), 21  
 get\_hash() (*dpdispatcher.submission.Submission*  
   *method*), 23  
 get\_hash() (*dpdispatcher.submission.Task* *method*), 23  
 get\_job\_root() (*dpdis-*  
   *patcher.lazy\_local\_context.LazyLocalContext*  
   *method*), 17  
 get\_job\_root() (*dpdis-*  
   *patcher.local\_context.LocalContext* *method*),  
   17  
 get\_job\_root() (*dpdispatcher.ssh\_context.SSHContext*  
   *method*), 20  
 get\_job\_state() (*dpdispatcher.submission.Job*  
   *method*), 21  
 get\_jobs() (*in module dpdispatcher.dpcloudserver.api*),  
   24  
 get\_return() (*dpdispatcher.lazy\_local\_context.LazyLocalContext*  
   *method*), 17  
 get\_return() (*dpdispatcher.local\_context.LocalContext*  
   *method*), 18  
 get\_return() (*dpdispatcher.ssh\_context.SSHContext*  
   *method*), 20  
 get\_ssh\_client() (*dpdis-*  
   *patcher.ssh\_context.SSHSession* *method*),  
   20

`get_submission_state()` (*dpdispatcher.submission.Submission* method), 23  
`get_tasks()` (in module *dpdispatcher.dpcloudserver.api*), 24  
**H**  
`handle_unexpected_job_state()` (*dpdispatcher.submission.Job* method), 21  
`handle_unexpected_submission_state()` (*dpdispatcher.submission.Submission* method), 23  
**I**  
`info()` (in module *dpdispatcher*), 15  
`IOERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24  
**J**  
`Job` (class in *dpdispatcher.submission*), 21  
`job_create()` (in module *dpdispatcher.dpcloudserver.api*), 24  
`job_to_json()` (*dpdispatcher.submission.Job* method), 21  
`JobStatus` (class in *dpdispatcher.JobStatus*), 15  
**K**  
`kill()` (*dpdispatcher.base\_context.BaseContext* method), 15  
`kill()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 16  
`kill()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 17  
`kill()` (*dpdispatcher.local\_context.LocalContext* method), 18  
`kill()` (*dpdispatcher.ssh\_context.SSHContext* method), 20  
**L**  
`LazyLocalContext` (class in *dpdispatcher.lazy\_local\_context*), 17  
`load_from_dict()` (*dpdispatcher.base\_context.BaseContext* class method), 15  
`load_from_dict()` (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* class method), 16  
`load_from_dict()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext* class method), 17  
`load_from_dict()` (*dpdispatcher.local\_context.LocalContext* class method), 18  
`load_from_dict()` (*dpdispatcher.machine.Machine* class method), 19  
`load_from_dict()` (*dpdispatcher.ssh\_context.SSHContext* class method), 20  
`load_from_dict()` (*dpdispatcher.submission.Resources* class method), 22  
`load_from_json()` (*dpdispatcher.machine.Machine* class method), 19  
`load_from_json()` (*dpdispatcher.submission.Resources* class method), 22  
`LocalContext` (class in *dpdispatcher.local\_context*), 17  
`login()` (in module *dpdispatcher.dpcloudserver.api*), 24  
`LOGINERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24  
`LSF` (class in *dpdispatcher.lsf*), 18  
**M**  
`Machine` (class in *dpdispatcher.machine*), 18  
`main()` (in module *dpdispatcher.dpdisp*), 17  
`map_dp_job_state()` (*dpdispatcher.dp\_cloud\_server.DpCloudServer* static method), 16  
**module**  
*dpdispatcher*, 15  
*dpdispatcher.base\_context*, 15  
*dpdispatcher.dp\_cloud\_server*, 16  
*dpdispatcher.dp\_cloud\_server\_context*, 16  
*dpdispatcher.dpcloudserver*, 24  
*dpdispatcher.dpcloudserver.api*, 24  
*dpdispatcher.dpcloudserver.config*, 24  
*dpdispatcher.dpcloudserver.retcode*, 24  
*dpdispatcher.dpcloudserver.zip\_file*, 24  
*dpdispatcher.dpdisp*, 17  
*dpdispatcher.JobStatus*, 15  
*dpdispatcher.lazy\_local\_context*, 17  
*dpdispatcher.local\_context*, 17  
*dpdispatcher.lsf*, 18  
*dpdispatcher.machine*, 18  
*dpdispatcher.pbs*, 19  
*dpdispatcher.shell*, 19  
*dpdispatcher.slurm*, 19  
*dpdispatcher.ssh\_context*, 20  
*dpdispatcher.submission*, 21  
**N**  
`NODATA` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24  
**O**  
`OK` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24

## P

PARAMERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24

PBS (class in *dpdispatcher.pbs*), 19

post() (in module *dpdispatcher.dpcloudserver.api*), 24

PWDERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24

## R

read() (*dpdispatcher.lazy\_local\_context.SPRetObj* method), 17

read() (*dpdispatcher.local\_context.SPRetObj* method), 18

read\_file() (*dpdispatcher.base\_context.BaseContext* method), 15

read\_file() (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 16

read\_file() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 17

read\_file() (*dpdispatcher.local\_context.LocalContext* method), 18

read\_file() (*dpdispatcher.ssh\_context.SSHContext* method), 20

read\_home\_file() (*dpdispatcher.dp\_cloud\_server\_context.DpCloudServerContext* method), 16

readlines() (*dpdispatcher.lazy\_local\_context.SPRetObj* method), 17

readlines() (*dpdispatcher.local\_context.SPRetObj* method), 18

register\_job\_id() (*dpdispatcher.submission.Job* method), 21

register\_task() (*dpdispatcher.submission.Submission* method), 23

register\_task\_list() (*dpdispatcher.submission.Submission* method), 23

REQERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24

Resources (class in *dpdispatcher.submission*), 21

RETCODE (class in *dpdispatcher.dpcloudserver.retcode*), 24

ROLEERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24

run\_submission() (*dpdispatcher.submission.Submission* method), 23

running (*dpdispatcher.JobStatus.JobStatus* attribute), 15

## S

serialize() (*dpdispatcher.submission.Job* method), 21

serialize() (*dpdispatcher.submission.Resources* method), 22

serialize() (*dpdispatcher.submission.Submission* method), 23

serialize() (*dpdispatcher.submission.Task* method), 24

sftp (*dpdispatcher.ssh\_context.SSHContext* property), 20

sftp (*dpdispatcher.ssh\_context.SSHSession* property), 20

Shell (class in *dpdispatcher.shell*), 19

Slurm (class in *dpdispatcher.slurm*), 19

SPRetObj (class in *dpdispatcher.lazy\_local\_context*), 17

SPRetObj (class in *dpdispatcher.local\_context*), 18

ssh (*dpdispatcher.ssh\_context.SSHContext* property), 20

SSHContext (class in *dpdispatcher.ssh\_context*), 20

SSHSession (class in *dpdispatcher.ssh\_context*), 20

sub\_script\_cmd() (*dpdispatcher.lsf.LSF* method), 18

sub\_script\_cmd() (*dpdispatcher.machine.Machine* method), 19

sub\_script\_head() (*dpdispatcher.lsf.LSF* method), 18

sub\_script\_head() (*dpdispatcher.machine.Machine* method), 19

subclasses\_dict (*dpdispatcher.base\_context.BaseContext* attribute), 15

subclasses\_dict (*dpdispatcher.machine.MachineContext* attribute), 19

Submission (class in *dpdispatcher.submission*), 22

submission\_from\_json() (*dpdispatcher.submission.Submission* class method), 23

submission\_to\_json() (*dpdispatcher.submission.Submission* method), 23

submit\_job() (*dpdispatcher.submission.Job* method), 21

## T

Task (class in *dpdispatcher.submission*), 23

terminated (*dpdispatcher.JobStatus.JobStatus* attribute), 15

THIRDERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24

try\_recover\_from\_json() (*dpdispatcher.submission.Submission* method), 23

## U

UNDERDEBUG (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24

unknown (*dpdispatcher.JobStatus.JobStatus* attribute), 15

UNKOWNERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 24

unsubmitted (*dpdispatcher.JobStatus.JobStatus* attribute), 15



`unzip_file()` (in module `dpdispatcher.dpcloudserver.zip_file`), 24

`upload()` (`dpdispatcher.base_context.BaseContext` method), 16

`upload()` (`dpdispatcher.dp_cloud_server_context.DpCloudServerContext` method), 16

`upload()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 17

`upload()` (`dpdispatcher.local_context.LocalContext` method), 18

`upload()` (`dpdispatcher.ssh_context.SSHContext` method), 20

`upload()` (in module `dpdispatcher.dpcloudserver.api`), 24

`upload_()` (`dpdispatcher.local_context.LocalContext` method), 18

`upload_jobs()` (`dpdispatcher.submission.Submission` method), 23

`USERERR` (`dpdispatcher.dpcloudserver.retcode.RETCODE` attribute), 24

## V

`VERIFYERR` (`dpdispatcher.dpcloudserver.retcode.RETCODE` attribute), 24

## W

`waiting` (`dpdispatcher.JobStatus.JobStatus` attribute), 15

`write_file()` (`dpdispatcher.base_context.BaseContext` method), 16

`write_file()` (`dpdispatcher.dp_cloud_server_context.DpCloudServerContext` method), 17

`write_file()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 17

`write_file()` (`dpdispatcher.local_context.LocalContext` method), 18

`write_file()` (`dpdispatcher.ssh_context.SSHContext` method), 20

`write_home_file()` (`dpdispatcher.dp_cloud_server_context.DpCloudServerContext` method), 17

`write_local_file()` (`dpdispatcher.dp_cloud_server_context.DpCloudServerContext` method), 17

## Z

`zip_file_list()` (in module `dpdispatcher.dpcloudserver.zip_file`), 24