

---

# DPDispatcher

Deep Modeling

Jul 17, 2023



## 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>Supported contexts</b>	<b>9</b>
3.1	LazyLocal . . . . .	9
3.2	Local . . . . .	9
3.3	SSH . . . . .	9
3.4	Bohrium . . . . .	9
3.5	HDFS . . . . .	10
<b>4</b>	<b>Supported batch job systems</b>	<b>11</b>
4.1	Bash . . . . .	11
4.2	Slurm . . . . .	11
4.3	OpenPBS or PBSPro . . . . .	11
4.4	TORQUE . . . . .	12
4.5	LSF . . . . .	12
4.6	Bohrium . . . . .	12
4.7	DistributedShell . . . . .	12
4.8	Fugaku . . . . .	12
<b>5</b>	<b>Machine parameters</b>	<b>13</b>
<b>6</b>	<b>Resources parameters</b>	<b>17</b>
<b>7</b>	<b>Task parameters</b>	<b>23</b>
<b>8</b>	<b>DPDispatcher API</b>	<b>25</b>
8.1	dpdispatcher package . . . . .	25
<b>9</b>	<b>Running the DeePMD-kit on the Expanse cluster</b>	<b>85</b>
<b>10</b>	<b>Running Gaussian 16 with failure allowed</b>	<b>87</b>
<b>11</b>	<b>Running multiple MD tasks on a GPU workstation</b>	<b>89</b>
<b>12</b>	<b>Authors</b>	<b>91</b>
<b>13</b>	<b>Indices and tables</b>	<b>93</b>
	<b>Python Module Index</b>	<b>95</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",
```

(continues on next page)

(continued from previous page)

```

)
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=resources,
    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": [

```

(continues on next page)

(continued from previous page)

```

        "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={
        # 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*.



## SUPPORTED CONTEXTS

Context is the way to connect to the remote server. One needs to set `context_type` to one of the following values:

### 3.1 LazyLocal

`context_type`: LazyLocal

LazyLocal directly runs jobs in the local server and local directory.

### 3.2 Local

`context_type`: Local

Local runs jobs in the local server, but in a different directory. Files will be copied to the remote directory before jobs start and copied back after jobs finish.

### 3.3 SSH

`context_type`: SSH

SSH runs jobs in a remote server. Files will be copied to the remote directory via SSH channels before jobs start and copied back after jobs finish. To use SSH, one needs to provide necessary parameters in `remote_profile`, such as `username` and `hostname`.

It's suggested to generate [SSH keys](#) and transfer the public key to the remote server in advance, which is more secure than password authentication.

Note that SSH context is `non-login`, so `bash_profile` files will not be executed.

### 3.4 Bohrium

`context_type`: Bohrium

Bohrium is the cloud platform for scientific computing. Read Bohrium documentation for details. To use Bohrium, one needs to provide necessary parameters in `remote_profile`.

## 3.5 HDFS

*context\_type*: HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system. Read [Support DPDispatcher on Yarn](#) for details.

## SUPPORTED BATCH JOB SYSTEMS

Batch job system is a system to process batch jobs. One needs to set *batch\_type* to one of the following values:

### 4.1 Bash

*batch\_type*: Shell

When *batch\_type* is set to Shell, dpdispatcher will generate a bash script to process jobs. No extra packages are required for Shell.

Due to lack of scheduling system, Shell runs all jobs at the same time. To avoid running multiple jobs at the same time, one could set *group\_size* to 0 (means infinity) to generate only one job with multiple tasks.

### 4.2 Slurm

*batch\_type*: Slurm, SlurmJobArray

Slurm is a job scheduling system used by lots of HPCs. One needs to make sure slurm has been setup in the remote server and the related environment is activated.

When SlurmJobArray is used, dpdispatcher submits Slurm jobs with *job arrays*. In this way, several dpdispatcher *tasks* map to a Slurm job and a dpdispatcher *job* maps to a Slurm job array. Millions of Slurm jobs can be submitted quickly and Slurm can execute all Slurm jobs at the same time. One can use *group\_size* and *slurm\_job\_size* to control how many Slurm jobs are contained in a Slurm job array.

### 4.3 OpenPBS or PBSPro

*batch\_type*: PBS

OpenPBS is an open-source job scheduling of the Linux Foundation and PBS Profession is its commercial solution. One needs to make sure OpenPBS has been setup in the remote server and the related environment is activated.

Note that do not use PBS for Torque.

## 4.4 TORQUE

*batch\_type*: Torque

The Terascale Open-source Resource and QUEue Manager (TORQUE) is a distributed resource manager based on standard OpenPBS. However, not all OpenPBS flags are still supported in TORQUE. One needs to make sure TORQUE has been setup in the remote server and the related environment is activated.

## 4.5 LSF

*batch\_type*: LSF

IBM Spectrum LSF Suites is a comprehensive workload management solution used by HPCs. One needs to make sure LSF has been setup in the remote server and the related environment is activated.

## 4.6 Bohrium

*batch\_type*: Bohrium

Bohrium is the cloud platform for scientific computing. Read Bohrium documentation for details.

## 4.7 DistributedShell

*batch\_type*: DistributedShell

DistributedShell is used to submit yarn jobs. Read Support DPDispatcher on Yarn for details.

## 4.8 Fugaku

*batch\_type*: Fugaku

Fujitsu cloud service is a job scheduling system used by Fujitsu's HPCs such as Fugaku, ITO and K computer. It should be noted that although the same job scheduling system is used, there are some differences in the details, Fugaku class cannot be directly used for other HPCs.

Read Fujitsu cloud service documentation for details.



## MACHINE PARAMETERS

---

**Note:** One can load, modify, and export the input file by using our effective web-based tool [DP-GUI](#). All parameters below can be set in DP-GUI. By clicking “SAVE JSON”, one can download the input file.

---

**machine:**

type: dict

argument path: machine

**batch\_type:**

type: str

argument path: machine/batch\_type

The batch job system type. Option: DistributedShell, Shell, Slurm, LSF, Bohrium, SlurmJobArray, Fugaku, PBS, Torque

**local\_root:**

type: str | NoneType

argument path: machine/local\_root

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

**remote\_root:**

type: str | NoneType, optional

argument path: machine/remote\_root

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

**clean\_asynchronously:**

type: bool, optional, default: False

argument path: machine/clean\_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

**context\_type:**

type: str (flag key)

argument path: machine/context\_type

possible choices: *BohriumContext*, *HDFSContext*, *LazyLocalContext*, *SSHContext*, *LocalContext*

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

When `context_type` is set to BohriumContext (or its aliases bohriumcontext, Bohrium, bohrium, DpCloudServerContext, dpcloudservercontext, DpCloudServer, dpcloudserver, LebesgueContext, lebesguecontext, Lebesgue, lebesgue):

**remote\_profile:**

type: dict

argument path: machine[BohriumContext]/remote\_profile

The information used to maintain the connection with remote machine.

**email:**

type: str, optional

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

Email

**password:**

type: str

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

Password

**program\_id:**

type: int, alias: *project\_id*

argument path:

machine[BohriumContext]/remote\_profile/program\_id

Program ID

**keep\_backup:**

type: bool, optional

argument path:

machine[BohriumContext]/remote\_profile/keep\_backup

keep download and upload zip

**input\_data:**

type: dict

argument path:

machine[BohriumContext]/remote\_profile/input\_data

Configuration of job

When `context_type` is set to HDFSContext (or its aliases hdfscontext, HDFS, hdfs):

**remote\_profile:**

type: dict, optional

argument path: machine[HDFSContext]/remote\_profile

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

When `context_type` is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

**remote\_profile:**

type: dict, optional

argument path: machine[LazyLocalContext]/remote\_profile

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

When `context_type` is set to `SSHContext` (or its aliases `sshcontext`, `SSH`, `ssh`):

**remote\_profile:**

type: dict

argument path: machine[SSHContext]/remote\_profile

The information used to maintain the connection with remote machine.

**hostname:**

type: str

argument path: machine[SSHContext]/remote\_profile/hostname

hostname or ip of ssh connection.

**username:**

type: str

argument path: machine[SSHContext]/remote\_profile/username

username of target linux system

**password:**

type: str, optional

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

(deprecated) password of linux system. Please use [SSH keys](#) instead to improve security.

**port:**

type: int, optional, default: 22

argument path: machine[SSHContext]/remote\_profile/port

ssh connection port.

**key\_filename:**

type: str | NoneType, optional, default: None

argument path:

machine[SSHContext]/remote\_profile/key\_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

**passphrase:**

type: str | NoneType, optional, default: None

argument path: machine[SSHContext]/remote\_profile/passphrase

passphrase of key used by ssh connection

**timeout:**

type: int, optional, default: 10

argument path: machine[SSHContext]/remote\_profile/timeout

timeout of ssh connection

**totp\_secret:**

type: str | NoneType, optional, default: None

argument path:

machine[SSHContext]/remote\_profile/totp\_secret

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

**tar\_compress:**

type: bool, optional, default: True

argument path:

machine[SSHContext]/remote\_profile/tar\_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

**look\_for\_keys:**

type: bool, optional, default: True

argument path:

machine[SSHContext]/remote\_profile/look\_for\_keys

enable searching for discoverable private key files in ~/.ssh/

When `context_type` is set to `LocalContext` (or its aliases `localcontext`, `Local`, `local`):

**remote\_profile:**

type: dict, optional

argument path: machine[LocalContext]/remote\_profile

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

---

## RESOURCES PARAMETERS

---

**Note:** One can load, modify, and export the input file by using our effective web-based tool [DP-GUI](#). All parameters below can be set in DP-GUI. By clicking “SAVE JSON”, one can download the input file for.

---

### **resources:**

type: dict

argument path: resources

#### **number\_node:**

type: int, optional, default: 1

argument path: resources/number\_node

The number of node need for each *job*

#### **cpu\_per\_node:**

type: int, optional, default: 1

argument path: resources/cpu\_per\_node

cpu numbers of each node assigned to each job.

#### **gpu\_per\_node:**

type: int, optional, default: 0

argument path: resources/gpu\_per\_node

gpu numbers of each node assigned to each job.

#### **queue\_name:**

type: str, optional, default: (empty string)

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*. 0 means infinity.

**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: False  
argument path: resources/strategy/if\_cuda\_multi\_devices

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.

**ratio\_unfinished:**

type: float, optional, default: 0.0  
argument path: resources/strategy/ratio\_unfinished

The ratio of *tasks* that can be unfinished.

**para\_deg:**

type: int, optional, default: 1  
argument path: resources/para\_deg  
Decide how many tasks will be run in parallel.

**source\_list:**

type: list, optional, default: []  
argument path: resources/source\_list  
The env file to be sourced before the command execution.

**module\_purge:**

type: bool, optional, default: False  
argument path: resources/module\_purge  
Remove all modules on HPC system before module load (module\_list)

**module\_unload\_list:**

type: list, optional, default: []  
argument path: resources/module\_unload\_list  
The modules to be unloaded on HPC system before submitting jobs

**module\_list:**

type: list, optional, default: []  
argument path: resources/module\_list  
The modules to be loaded on HPC system before submitting jobs

**envs:**

type: dict, optional, default: {}  
argument path: resources/envs

The environment variables to be exported on before submitting jobs

**prepend\_script:**

type: list, optional, default: []  
argument path: resources/prepend\_script

Optional script run before jobs submitted.

**append\_script:**

type: list, optional, default: []  
argument path: resources/append\_script

Optional script run after jobs submitted.

**wait\_time:**

type: int | float, optional, default: 0  
argument path: resources/wait\_time

The waiting time in second after a single *task* submitted

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

**batch\_type:**

type: str (flag key)  
argument path: resources/batch\_type  
possible choices: *Slurm*, *SlurmJobArray*, *DistributedShell*, *PBS*, *LSF*,  
*Shell*, *Bohrium*, *Torque*, *Fugaku*

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

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

**kwargs:**

type: dict, optional  
argument path: resources[Slurm]/kwargs

Extra arguments.

**custom\_gpu\_line:**

type: str | NoneType, optional, default: None  
argument path: resources[Slurm]/kwargs/custom\_gpu\_line

Custom GPU configuration, starting with #SBATCH

When *batch\_type* is set to *SlurmJobArray* (or its alias *slurmjobarray*):

**kwargs:**

type: dict, optional  
argument path: resources[SlurmJobArray]/kwargs

Extra arguments.

**custom\_gpu\_line:**

type: str | NoneType, optional, default: None

argument path:

resources[SlurmJobArray]/kwargs/custom\_gpu\_line

Custom GPU configuration, starting with #SBATCH

**slurm\_job\_size:**

type: int, optional, default: 1

argument path:

resources[SlurmJobArray]/kwargs/slurm\_job\_size

Number of tasks in a Slurm job

When `batch_type` is set to DistributedShell (or its alias distributedshell):

**kwargs:**

type: dict, optional

argument path: resources[DistributedShell]/kwargs

This field is empty for this batch.

When `batch_type` is set to PBS (or its alias pbs):

**kwargs:**

type: dict, optional

argument path: resources[PBS]/kwargs

This field is empty for this batch.

When `batch_type` is set to LSF (or its alias lsf):

**kwargs:**

type: dict

argument path: resources[LSF]/kwargs

Extra arguments.

**gpu\_usage:**

type: bool, optional, default: False

argument path: resources[LSF]/kwargs/gpu\_usage

Choosing if GPU is used in the calculation step.

**gpu\_new\_syntax:**

type: bool, optional, default: False

argument path: resources[LSF]/kwargs/gpu\_new\_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

**gpu\_exclusive:**

type: bool, optional, default: True

argument path: resources[LSF]/kwargs/gpu\_exclusive

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



**custom\_gpu\_line:**

type: str | NoneType, optional, default: None

argument path: resources[LSF]/kwargs/custom\_gpu\_line

Custom GPU configuration, starting with #BSUB

When `batch_type` is set to Shell (or its alias shell):

**kwargs:**

type: dict, optional

argument path: resources[Shell]/kwargs

This field is empty for this batch.

When `batch_type` is set to Bohrium (or its aliases bohrium, Lebesgue, lebesgue, DpCloudServer, dpcloudserver):

**kwargs:**

type: dict, optional

argument path: resources[Bohrium]/kwargs

This field is empty for this batch.

When `batch_type` is set to Torque (or its alias torque):

**kwargs:**

type: dict, optional

argument path: resources[Torque]/kwargs

This field is empty for this batch.

When `batch_type` is set to Fugaku (or its alias fugaku):

**kwargs:**

type: dict, optional

argument path: resources[Fugaku]/kwargs

This field is empty for this batch.



## TASK PARAMETERS

---

**Note:** One can load, modify, and export the input file by using our effective web-based tool [DP-GUI](#). All parameters below can be set in DP-GUI. By clicking “SAVE JSON”, one can download the input file.

---

**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

## 8.1 dpdispatcher package

```
class dpdispatcher.DistributedShell(*args, **kwargs)
```

Bases: *Machine*

## Methods

<code>do_submit(job)</code>	Submit th job to yarn using distributed shell.
<code>kill(job)</code>	Kill the job.
<code>resources_arginfo()</code>	Generate the resources arginfo.
<code>resources_subfields()</code>	Generate the resources subfields.

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>gen_command_env_cuda_devices</code>	
<code>gen_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>serialize</code>	
<code>sub_script_cmd</code>	
<code>sub_script_head</code>	

```
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*)

**dpdispatcher.DpCloudServer**

alias of *Bohrrium*

**dpdispatcher.DpCloudServerContext**

alias of *BohrriumContext*

**class dpdispatcher.Fugaku**(\*args, \*\*kwargs)

Bases: *Machine*

### Methods

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

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>gen_command_env_cuda_devices</code>	
<code>gen_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>serialize</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)`

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

Bases: [BaseContext](#)

## Methods

<code>check_file_exists(fname)</code>	Check whether the given file exists, often used in checking whether the belonging job has finished.
<code>download(submission[, check_exists, ...])</code>	Download backward files from HDFS root dir.
<code>machine_arginfo()</code>	Generate the machine arginfo.
<code>machine_subfields()</code>	Generate the machine subfields.
<code>upload(submission[, dereference])</code>	Upload forward files and forward command files to HDFS root dir.

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

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

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

Check whether the given file exists, often used in checking whether the belonging job has finished.

**Parameters**

**fname**

[string] file name to be checked

**Returns**

**status:** boolean

**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

**check\_exists**

[bool] whether to check if the file exists

**mark\_failure**

[bool] whether to mark the task as failed if the file does not exist

**back\_error**

[bool] whether to download error files

**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

**dereference**

[bool] whether to dereference symbolic links



**Returns****none****write\_file**(*fname*, *write\_str*)**class** dpdispatcher.**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**

<a href="#"><i>deserialize</i></a> ( <i>job_dict</i> [, <i>machine</i> ])	Convert the <i>job_dict</i> to a Submission class object.
<a href="#"><i>get_job_state</i></a> ()	Get the jobs.
<a href="#"><i>serialize</i></a> ([ <i>if_static</i> ])	Convert the Task class instance to a dictionary.

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

**classmethod** **deserialize**(*job\_dict*, *machine=None*)Convert the *job\_dict* to a Submission class object.**Parameters****job\_dict**

[dict] the dictionary which contains the job information

**machine**

[Machine] the machine object to execute the job

**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.LSF(\*args, \*\*kwargs)**

Bases: *Machine*

LSF batch.

## Methods

---

<i>default_resources</i> (resources)	
<i>kill</i> (job)	Kill the job.
<i>resources_arginfo</i> ()	Generate the resources arginfo.
<i>resources_subfields</i> ()	Generate the resources subfields.

---

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

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

**check\_status**(\*\**kwargs*)

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

**do\_submit**(\*\**kwargs*)

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

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

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

**kill**(*job*)

Kill the job.

#### Parameters

**job**

[Job] job

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

Generate the resources subfields.

#### Returns

**list[Argument]**

resources subfields

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

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

**class** `dpdispatcher.LazyLocalContext(*args, **kwargs)`

Bases: [BaseContext](#)

Run jobs in the local server and local directory.

### Parameters

**local\_root**

[str] The local directory to store the jobs.

**remote\_root**

[str, optional] The argument takes no effect.

**remote\_profile**

[dict, optional] The remote profile. The default is {}.

**\*args**

The arguments.

**\*\*kwargs**

The keyword arguments.

### Methods

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

<b>bind_submission</b>	
<b>block_call</b>	
<b>block_checkcall</b>	
<b>call</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>download</b>	
<b>get_job_root</b>	
<b>get_return</b>	
<b>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)

classmethod load_from_dict(context_dict)

read_file(fname)

upload(jobs, dereference=True)

write_file(fname, write_str)

```

`dpdispatcher.Lebesgue`

alias of [\*Bohrrium\*](#)

`dpdispatcher.LebesgueContext`

alias of [\*BohrriumContext\*](#)

**class** `dpdispatcher.LocalContext`(\*args, \*\*kwargs)

Bases: [\*BaseContext\*](#)

Run jobs in the local server and remote directory.

#### Parameters

##### **local\_root**

[str] The local directory to store the jobs.

##### **remote\_root**

[str] The remote directory to store the jobs.

##### **remote\_profile**

[dict, optional] The remote profile. The default is {}.

##### **\*args**

The arguments.

##### **\*\*kwargs**

The keyword arguments.

#### Methods

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

<b>bind_submission</b>	
<b>block_call</b>	
<b>block_checkcall</b>	
<b>call</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>download</b>	
<b>get_job_root</b>	
<b>get_return</b>	
<b>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**(*submission*, *check\_exists=False*, *mark\_failure=True*, *back\_error=False*)

**get\_job\_root**()

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

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

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

**upload**(*submission*)

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

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

Bases: `object`

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

#### Parameters

##### **context**

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

## Methods

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

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>gen_command_env_cuda_devices</code>	
<code>gen_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>serialize</code>	
<code>sub_script_cmd</code>	
<code>sub_script_head</code>	

```

alias: Tuple[str, ...] = ()

classmethod arginfo()

bind_context(context)

abstract check_finish_tag(**kwargs)

check_if_recover(submission)

abstract check_status(job)

default_resources(res)

classmethod deserialize(machine_dict)

abstract 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*)

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

**gen\_script\_wait**(*resources*)

**kill**(*job*)

Kill the job.

If not implemented, pass and let the user manually kill it.

**Parameters**

**job**

[Job] job

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

**classmethod load\_from\_json**(*json\_path*)

**options** = {'Bohrium', 'DistributedShell', 'Fugaku', 'LSF', 'PBS', 'Shell', 'Slurm', 'SlurmJobArray', 'Torque'}

**classmethod resources\_arginfo**() → [Argument](#)

Generate the resources arginfo.

**Returns**

**Argument**

resources arginfo

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

Generate the resources subfields.

**Returns**

**list[Argument]**

resources subfields

**serialize**(*if\_empty\_remote\_profile=False*)

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

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



```

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

```

```
class dpdispatcher.PBS(*args, **kwargs)
```

Bases: [Machine](#)

## Methods

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

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

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

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

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

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

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

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

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

**kill**(*job*)

Kill the job.

#### Parameters

**job**

[Job] job

```
class dpdispatcher.Resources(number_node, cpu_per_node, gpu_per_node, queue_name, group_size, *,  
                           custom_flags=[], strategy={'if_cuda_multi_devices': False, 'ratio_unfinished':  
                           0.0}, para_deg=1, module_unload_list=[], module_purge=False,  
                           module_list=[], source_list=[], envs={}, prepend_script=[],  
                           append_script=[], wait_time=0, **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.

**ratio\_unfinished**

[float] The ratio of *task* that can be unfinished.

**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.

**wait\_time**  
 [int] The waiting time in second after a single task submitted. Default: 0.

## Methods

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

**static arginfo**(*detail\_kwargs=True*)

**classmethod deserialize**(*resources\_dict*)

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

**classmethod load\_from\_json**(*json\_file*)

**serialize**()

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

Bases: [BaseContext](#)

### Attributes

**sftp**  
**ssh**

## Methods

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

<b>bind_submission</b>	
<b>block_call</b>	
<b>call</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>close</b>	
<b>download</b>	
<b>get_job_root</b>	
<b>get_return</b>	
<b>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.

**stderr\_whitelist**

[list of str, optional, default=None] If not None, the stderr will be checked against the whitelist. If the stderr contains any of the strings in the whitelist, the command will be considered successful.

**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*)

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

**classmethod** `machine_subfields()` → `List[Argument]`

Generate the machine subfields.

**Returns**

`list[Argument]`

machine subfields

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

**property** `sftp`

**property** `ssh`

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

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

**class** `dpdispatcher.Shell(*args, **kwargs)`

Bases: `Machine`

## Methods

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

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>gen_command_env_cuda_devices</code>	
<code>gen_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>serialize</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*)

**kill**(*job*)

Kill the job.

#### Parameters

**job**

[Job] job

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

Bases: [Machine](#)

#### Methods

<a href="#">kill</a> ( <i>job</i> )	Kill the job.
<a href="#">resources_arginfo</a> ()	Generate the resources arginfo.
<a href="#">resources_subfields</a> ()	Generate the resources subfields.

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

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

**check\_status**(\*\*kwargs)

**default\_resources**(resources)

**do\_submit**(\*\*kwargs)

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

**gen\_script**(job)

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

**kill**(job)

Kill the job.

#### Parameters

**job**

[Job] job

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

Generate the resources subfields.

#### Returns

list[Argument]

resources subfields

**class dpdispatcher.Submission**(work\_base, machine=None, resources=None, forward\_common\_files=[], backward\_common\_files=[], \*, task\_list=[])

Bases: `object`

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

#### Parameters

**work\_base**

[Path] the base directory of the local tasks. It is usually the dir name of project .

**machine**

[Machine] machine class object (for example, PBS, Slurm, Shell) to execute the jobs. The machine can still be bound after the instantiation with the `bind_submission` method.

**resources**

[Resources] the machine resources (cpu or gpu) used to generate the slurm/pbs script

**forward\_common\_files**

[list] the common files to be uploaded to other computers before the jobs begin

**backward\_common\_files**

[list] the common files to be downloaded from other computers after the jobs finish

**task\_list**

[list of Task] a list of tasks to be run.

## Methods

<code>async_run_submission(**kwargs)</code>	Async interface of run_submission.
<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>check_ratio_unfinished(ratio_unfinished)</code>	Calculate the ratio of unfinished tasks in the submission.
<code>deserialize(submission_dict[, machine])</code>	Convert the submission_dict to a Submission class object.
<code>generate_jobs()</code>	After tasks register to the self.belonging_tasks, This method generate the jobs and add these jobs to self.belonging_jobs.
<code>handle_unexpected_submission_state()</code>	Handle unexpected job state of the submission.
<code>run_submission(*[, dry_run, exit_on_submit, ...])</code>	Main method to execute the submission.
<code>serialize([if_static])</code>	Convert the Submission class instance to a dictionary.
<code>update_submission_state()</code>	Check whether all the jobs in the submission.

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

**async async\_run\_submission(\*\*kwargs)**

Async interface of run\_submission.

## Examples

```
>>> import asyncio
>>> from dpdispatcher import Machine, Resource, Submission
>>> async def run_jobs():
...     background_task = set()
...     # task1
...     task1 = Task(...)
...     submission1 = Submission(..., task_list=[task1])
...     background_task = asyncio.create_task(
...         submission1.async_run_submission(check_interval=2, clean=False)
...     )
...     # task2
...     task2 = Task(...)
...     submission2 = Submission(..., task_list=[task1])
...     background_task = asyncio.create_task(
...         submission2.async_run_submission(check_interval=2, clean=False)
...     )
```

(continues on next page)



(continued from previous page)

```

...     )
...     background_tasks.add(background_task)
...     result = await asyncio.gather(*background_tasks)
...     return result
>>> run_jobs()

```

May raise Error if pass *clean=True* explicitly when submit to pbs or slurm.

**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.

**check\_ratio\_unfinished**(*ratio\_unfinished: float*) → bool

Calculate the ratio of unfinished tasks in the submission.

**Parameters**

**ratio\_unfinished**

[float] the ratio of unfinished tasks in the submission

**Returns**

**bool**

whether the ratio of unfinished tasks in the submission is larger than ratio\_unfinished

**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

**machine**

[Machine] Machine class Object to execute the jobs

**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)****remove\_unfinished\_tasks()****run\_submission(\*, dry\_run=False, exit\_on\_submit=False, clean=True, check\_interval=30)**

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 dry\_run is True, submission will be uploaded but not be executed and exit. If exit\_on\_submit is True, submission will exit.

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

Convert the Submission class instance to a dictionary.

**Parameters****if\_static**

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

**Returns****submission\_dict**

[dict] the dictionary converted from the Submission class instance

**classmethod submission\_from\_json(json\_file\_name='submission.json')****submission\_to\_json()****try\_download\_result()****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.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.
<code>get_task_state(context)</code>	Get the task state by checking the tag file.

<b>arginfo</b>	
<b>get_hash</b>	
<b>load_from_dict</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()**

**get\_task\_state**(*context*)

Get the task state by checking the tag file.

**Parameters**

**context**

[Context] the context of the task

**classmethod** **load\_from\_dict**(*task\_dict: dict*) → *Task*

**classmethod** **load\_from\_json**(*json\_file*)

**serialize()**

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

Bases: *PBS*

**Methods**

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

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

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

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

**dpdispatcher.info**()

Show basic information about dpdispatcher, its location and version.

### 8.1.1 Subpackages

#### dpdispatcher.dpcloudserver package

**class** dpdispatcher.dpcloudserver.**Client**(*email=None, password=None, debug=False, base\_url='https://bohrium.dp.tech/'*)

Bases: `object`

#### Methods

<b>download</b>	
<b>download_from_url</b>	
<b>get</b>	
<b>get_job_detail</b>	
<b>get_job_result_url</b>	
<b>get_log</b>	
<b>get_tasks_list</b>	
<b>job_create</b>	
<b>post</b>	
<b>refresh_token</b>	
<b>upload</b>	

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

**download\_from\_url**(*url, save\_file*)

**get**(*url, header=None, params=None, retry=5*)

**get\_job\_detail**(*job\_id*)

**get\_job\_result\_url**(*job\_id*)

**get\_log**(*job\_id*)

**get\_tasks\_list**(*group\_id, per\_page=30*)

**job\_create**(*job\_type, oss\_path, input\_data, program\_id=None, group\_id=None*)

**post**(*url, data=None, header=None, params=None, retry=5*)

**refresh\_token**(*retry=3*)

**upload**(*oss\_task\_zip, zip\_task\_file, endpoint, bucket\_name*)

#### Submodules

#### dpdispatcher.dpcloudserver.client module

**class** dpdispatcher.dpcloudserver.client.**Client**(*email=None, password=None, debug=False, base\_url='https://bohrium.dp.tech/'*)

Bases: `object`

## Methods

<b>download</b>	
<b>download_from_url</b>	
<b>get</b>	
<b>get_job_detail</b>	
<b>get_job_result_url</b>	
<b>get_log</b>	
<b>get_tasks_list</b>	
<b>job_create</b>	
<b>post</b>	
<b>refresh_token</b>	
<b>upload</b>	

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

**download\_from\_url**(*url*, *save\_file*)

**get**(*url*, *header*=None, *params*=None, *retry*=5)

**get\_job\_detail**(*job\_id*)

**get\_job\_result\_url**(*job\_id*)

**get\_log**(*job\_id*)

**get\_tasks\_list**(*group\_id*, *per\_page*=30)

**job\_create**(*job\_type*, *oss\_path*, *input\_data*, *program\_id*=None, *group\_id*=None)

**post**(*url*, *data*=None, *header*=None, *params*=None, *retry*=5)

**refresh\_token**(*retry*=3)

**upload**(*oss\_task\_zip*, *zip\_task\_file*, *endpoint*, *bucket\_name*)

**exception** dpdispatcher.dpcloudserver.client.RequestInfoException

Bases: [Exception](#)

**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=[])`

### **8.1.2 Submodules**

#### **8.1.3 dpdispatcher.JobStatus module**

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

Bases: `IntEnum`

An enumeration.

**completing** = 6

**finished** = 5

**running** = 3

**terminated** = 4

**unknown** = 100

**unsubmitted** = 1

**waiting** = 2

## 8.1.4 dpdispatcher.arginfo module

## 8.1.5 dpdispatcher.base\_context module

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

Bases: `object`

### Methods

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

<code>bind_submission</code>	
<code>check_finish</code>	
<code>clean</code>	
<code>download</code>	
<code>load_from_dict</code>	
<code>read_file</code>	
<code>upload</code>	
<code>write_file</code>	

```
alias: Tuple[str, ...] = ()
```

```
bind_submission(submission)
```

```
check_finish(proc)
```

```
abstract clean()
```

```
abstract download(submission, check_exists=False, mark_failure=True, back_error=False)
```

```
classmethod load_from_dict(context_dict)
```

```
classmethod machine_arginfo() → Argument
```

Generate the machine arginfo.

#### Returns

**Argument**

machine arginfo

```
classmethod machine_subfields() → List[Argument]
```

Generate the machine subfields.

#### Returns

**list[Argument]**

machine subfields

```
options = {'BohriumContext', 'HDFSContext', 'LazyLocalContext', 'LocalContext', 'SSHContext'}
```

```
abstract read_file(fname)
```



```

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

```

```
abstract upload(submission)
```

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

## 8.1.6 dpdispatcher.distributed\_shell module

```
class dpdispatcher.distributed_shell.DistributedShell(*args, **kwargs)
```

Bases: [Machine](#)

### Methods

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

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

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

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

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

Submit th job to yarn using distributed shell.

**Parameters**

**job**

[Job class instance] job to be submitted

**Returns**

**job\_id: string**

submit process id

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

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

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

### 8.1.7 dpdispatcher.dp\_cloud\_server module

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

Bases: *Machine*

## Methods

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

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

`alias: Tuple[str, ...] = ('Lebesgue', 'DpCloudServer')`

`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)`

`dpdispatcher.dp_cloud_server.DpCloudServer`

alias of *Bohrrium*

`dpdispatcher.dp_cloud_server.Lebesgue`

alias of *Bohrium*

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

**class** `dpdispatcher.dp_cloud_server_context.BohriumContext(*args, **kwargs)`

Bases: *BaseContext*

#### Methods

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

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

**alias:** `Tuple[str, ...] = ('DpCloudServerContext', 'LebesgueContext')`

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

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

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

**clean**()

**download**(*submission*)

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

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

Generate the machine subfields.

#### Returns

`list[Argument]`

machine subfields

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

`read_home_file(fname)`

`upload(submission)`

`upload_job(job, common_files=None)`

`write_file(fname, write_str)`

`write_home_file(fname, write_str)`

`write_local_file(fname, write_str)`

`dpdispatcher.dp_cloud_server_context.DpCloudServerContext`

alias of [\*BohrriumContext\*](#)

`dpdispatcher.dp_cloud_server_context.LebesgueContext`

alias of [\*BohrriumContext\*](#)

### 8.1.9 dpdispatcher.dpdisp module

`dpdispatcher.dpdisp.main()`

### 8.1.10 dpdispatcher.fugaku module

`class dpdispatcher.fugaku.Fugaku(*args, **kwargs)`

Bases: [\*Machine\*](#)

#### Methods

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

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>gen_command_env_cuda_devices</code>	
<code>gen_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>serialize</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)`

### 8.1.11 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.

<b>copy_to_local</b>	
<b>move</b>	
<b>read_hdfs_file</b>	

**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.

### 8.1.12 dpdispatcher.hdfs\_context module

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

Bases: *BaseContext*

#### Methods

<code>check_file_exists(fname)</code>	Check whether the given file exists, often used in checking whether the belonging job has finished.
<code>download(submission[, check_exists, ...])</code>	Download backward files from HDFS root dir.
<code>machine_arginfo()</code>	Generate the machine arginfo.
<code>machine_subfields()</code>	Generate the machine subfields.
<code>upload(submission[, dereference])</code>	Upload forward files and forward command files to HDFS root dir.

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

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

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

Check whether the given file exists, often used in checking whether the belonging job has finished.

**Parameters****fname**

[string] file name to be checked

**Returns**

**status: boolean**

**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

**check\_exists**

[bool] whether to check if the file exists

**mark\_failure**

[bool] whether to mark the task as failed if the file does not exist

**back\_error**

[bool] whether to download error files

**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

**dereference**

[bool] whether to dereference symbolic links

**Returns**

**none**

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



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

**class** dpdispatcher.lazy\_local\_context.LazyLocalContext(\*args, \*\*kwargs)

Bases: [BaseContext](#)

Run jobs in the local server and local directory.

#### Parameters

##### **local\_root**

[str] The local directory to store the jobs.

##### **remote\_root**

[str, optional] The argument takes no effect.

##### **remote\_profile**

[dict, optional] The remote profile. The default is {}.

##### **\*args**

The arguments.

##### **\*\*kwargs**

The keyword arguments.

#### Methods

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

<b>bind_submission</b>	
<b>block_call</b>	
<b>block_checkcall</b>	
<b>call</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>download</b>	
<b>get_job_root</b>	
<b>get_return</b>	
<b>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)

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()
```

## 8.1.14 dpdispatcher.local\_context module

```
class dpdispatcher.local_context.LocalContext(*args, **kwargs)
    Bases: BaseContext
```

Run jobs in the local server and remote directory.

### Parameters

**local\_root**  
[str] The local directory to store the jobs.

**remote\_root**  
[str] The remote directory to store the jobs.

**remote\_profile**  
[dict, optional] The remote profile. The default is {}.

**\*args**  
The arguments.

**\*\*kwargs**  
The keyword arguments.

## Methods

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

<b>bind_submission</b>	
<b>block_call</b>	
<b>block_checkcall</b>	
<b>call</b>	
<b>check_file_exists</b>	
<b>check_finish</b>	
<b>clean</b>	
<b>download</b>	
<b>get_job_root</b>	
<b>get_return</b>	
<b>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(submission, check_exists=False, mark_failure=True, back_error=False)

get_job_root()

get_return(proc)

classmethod load_from_dict(context_dict)

read_file(fname)

upload(submission)

write_file(fname, write_str)

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

```

## Methods

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

**read()**

**readlines()**

### 8.1.15 dpdispatcher.lsf module

**class** `dpdispatcher.lsf.LSF(*args, **kwargs)`

Bases: *Machine*

LSF batch.

## Methods

---

<i>default_resources</i> (resources)	
<i>kill</i> (job)	Kill the job.
<i>resources_arginfo</i> ()	Generate the resources arginfo.
<i>resources_subfields</i> ()	Generate the resources subfields.

---

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

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

**check\_status**(\*\*kwargs)

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

**do\_submit**(\*\**kwargs*)

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

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

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

**kill**(*job*)

Kill the job.

#### Parameters

**job**

[Job] job

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

Generate the resources subfields.

#### Returns

**list[Argument]**

resources subfields

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

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

## 8.1.16 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</code> ( <i>job</i> )	Submit a single job, assuming that no job is running there.
<code>kill</code> ( <i>job</i> )	Kill the job.
<code>resources_arginfo</code> ()	Generate the resources arginfo.
<code>resources_subfields</code> ()	Generate the resources subfields.

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

```
alias: Tuple[str, ...] = ()

classmethod arginfo()

bind_context(context)

abstract check_finish_tag(**kwargs)

check_if_recover(submission)

abstract check_status(job)

default_resources(res)

classmethod deserialize(machine_dict)

abstract 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)

abstract gen_script_header(job)

gen_script_wait(resources)
```

**kill**(*job*)

Kill the job.

If not implemented, pass and let the user manually kill it.

#### Parameters

**job**

[Job] job

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

**classmethod** **load\_from\_json**(*json\_path*)

**options** = {'Bohrium', 'DistributedShell', 'Fugaku', 'LSF', 'PBS', 'Shell', 'Slurm', 'SlurmJobArray', 'Torque'}

**classmethod** **resources\_arginfo**() → *Argument*

Generate the resources arginfo.

#### Returns

**Argument**

resources arginfo

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

Generate the resources subfields.

#### Returns

**list[Argument]**

resources subfields

**serialize**(*if\_empty\_remote\_profile=False*)

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

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

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

### 8.1.17 dpdispatcher.pbs module

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

Bases: *Machine*

#### Methods

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

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

**check\_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)

**kill**(job)

Kill the job.

#### Parameters



**job**  
[Job] job

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

Bases: *PBS*

### Methods

do_submit(job)	Submit a single job, assuming that no job is running there.
kill(job)	Kill the job.
resources_arginfo()	Generate the resources arginfo.
resources_subfields()	Generate the resources subfields.

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

check\_status(job)

gen\_script\_header(job)

### 8.1.18 dpdispatcher.shell module

**class** dpdispatcher.shell.Shell(\*args, \*\*kwargs)

Bases: *Machine*

## Methods

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

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>gen_command_env_cuda_devices</code>	
<code>gen_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>serialize</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)`

`kill(job)`

Kill the job.

### Parameters

**job**

[Job] job

## 8.1.19 dpdispatcher.slurm module

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

Bases: *Machine*

### Methods

<i>kill</i> (job)	Kill the job.
resources_arginfo()	Generate the resources arginfo.
<i>resources_subfields</i> ()	Generate the resources subfields.

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

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

**check\_status**(\*\*kwargs)

**default\_resources**(resources)

**do\_submit**(\*\*kwargs)

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

**gen\_script**(job)

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

**kill**(job)

Kill the job.

### Parameters

**job**

[Job] job

**classmethod** `resources_subfields()` → `List[Argument]`

Generate the resources subfields.

**Returns**

`list[Argument]`

resources subfields

**class** `dpdispatcher.slurm.SlurmJobArray(*args, **kwargs)`

Bases: `Slurm`

Slurm with job array enabled for multiple tasks in a job.

**Methods**

<code>kill(job)</code>	Kill the job.
<code>resources_arginfo()</code>	Generate the resources arginfo.
<code>resources_subfields()</code>	Generate the resources subfields.

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>do_submit</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>serialize</code>	
<code>sub_script_cmd</code>	
<code>sub_script_head</code>	

`check_finish_tag(job)`

`check_status(**kwargs)`

`gen_script_command(job)`

`gen_script_end(job)`

`gen_script_header(job)`

**classmethod** `resources_subfields()` → `List[Argument]`

Generate the resources subfields.

#### Returns

**list[Argument]**  
resources subfields

## 8.1.20 dpdispatcher.ssh\_context module

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

Bases: `BaseContext`

#### Attributes

**sftp**  
**ssh**

#### Methods

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

<b>bind_submission</b>	
<b>block_call</b>	
<b>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>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.

**stderr\_whitelist**

[list of str, optional, default=None] If not None, the stderr will be checked against the whitelist. If the stderr contains any of the strings in the whitelist, the command will be considered successful.

**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*)

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

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

Generate the machine subfields.

**Returns**

**list[Argument]**

machine subfields

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

**property sftp**

**property ssh**

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

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

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

Bases: [object](#)

**Attributes**

**remote**

**rsync\_available**

[sftp](#)

Returns sftp.

## Methods

---

<code>inter_handler</code> (title, instructions, prompt_list)	inter_handler: the callback for paramiko.transport.auth_interactive.
---	--

---

<b>arginfo</b>	
<b>close</b>	
<b>ensure_alive</b>	
<b>exec_command</b>	
<b>get</b>	
<b>get_ssh_client</b>	
<b>put</b>	

**static** `arginfo()`

**close()**

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

**exec\_command**(\*\*kwargs)

**get**(from\_f, to\_f)

**get\_ssh\_client()**

**inter\_handler**(title, instructions, prompt\_list)

inter\_handler: the callback for paramiko.transport.auth\_interactive.

The prototype for this function is defined by Paramiko, so all of the arguments need to be there, even though we don't use 'title' or 'instructions'.

The function is expected to return a tuple of data containing the responses to the provided prompts. Experimental results suggests that there will be one call of this function per prompt, but the mechanism allows for multiple prompts to be sent at once, so it's best to assume that that can happen.

Since tuples can't really be built on the fly, the responses are collected in a list which is then converted to a tuple when it's time to return a value.

Experiments suggest that the username prompt never happens. This makes sense, but the Username prompt is included here just in case.

**put**(from\_f, to\_f)

**property remote:** `str`

**property rsync\_available:** `bool`

**property sftp**

Returns sftp. Open a new one if not existing.

### 8.1.21 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

##### **job\_dict**

[dict] the dictionary which contains the job information

##### **machine**

[Machine] the machine object to execute the job

#### 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, 'ratio_unfinished':
                                       0.0}, para_deg=1, module_unload_list=[],
                                       module_purge=False, module_list=[], source_list=[], envs={},
                                       prepend_script=[], append_script=[], wait_time=0, **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.

**ratio\_unfinished**

[float] The ratio of *task* that can be unfinished.

**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.

**wait\_time**

[int] The waiting time in second after a single task submitted. Default: 0.

**Methods**

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

**static** **arginfo**(*detail\_kwargs=True*)

**classmethod** **deserialize**(*resources\_dict*)

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

**classmethod** **load\_from\_json**(*json\_file*)

**serialize**()

**class** dpdispatcher.submission.**Submission**(*work\_base, machine=None, resources=None, forward\_common\_files=[], backward\_common\_files=[], \*, task\_list=[]*)

Bases: `object`

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

**Parameters****work\_base**

[Path] the base directory of the local tasks. It is usually the dir name of project .

**machine**

[Machine] machine class object (for example, PBS, Slurm, Shell) to execute the jobs. The machine can still be bound after the instantiation with the `bind_submission` method.

**resources**

[Resources] the machine resources (cpu or gpu) used to generate the slurm/pbs script

**forward\_common\_files**

[list] the common files to be uploaded to other computers before the jobs begin

**backward\_common\_files**

[list] the common files to be downloaded from other computers after the jobs finish

**task\_list**

[list of Task] a list of tasks to be run.

## Methods

<code>async_run_submission(**kwargs)</code>	Async interface of run_submission.
<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>check_ratio_unfinished(ratio_unfinished)</code>	Calculate the ratio of unfinished tasks in the submission.
<code>deserialize(submission_dict[, machine])</code>	Convert the submission_dict to a Submission class object.
<code>generate_jobs()</code>	After tasks register to the self.belonging_tasks, This method generate the jobs and add these jobs to self.belonging_jobs.
<code>handle_unexpected_submission_state()</code>	Handle unexpected job state of the submission.
<code>run_submission(*[, dry_run, exit_on_submit, ...])</code>	Main method to execute the submission.
<code>serialize([if_static])</code>	Convert the Submission class instance to a dictionary.
<code>update_submission_state()</code>	Check whether all the jobs in the submission.

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

**async async\_run\_submission(\*\*kwargs)**

Async interface of run\_submission.

## Examples

```
>>> import asyncio
>>> from dpdispatcher import Machine, Resource, Submission
>>> async def run_jobs():
...     background_task = set()
...     # task1
...     task1 = Task(...)
...     submission1 = Submission(..., task_list=[task1])
...     background_task = asyncio.create_task(
...         submission1.async_run_submission(check_interval=2, clean=False)
...     )
...     # task2
...     task2 = Task(...)
...     submission2 = Submission(..., task_list=[task1])
...     background_task = asyncio.create_task(
...         submission2.async_run_submission(check_interval=2, clean=False)
...     )
```

(continues on next page)

(continued from previous page)

```
...     )
...     background_tasks.add(background_task)
...     result = await asyncio.gather(*background_tasks)
...     return result
>>> run_jobs()
```

May raise Error if pass *clean=True* explicitly when submit to pbs or slurm.

**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.

**check\_ratio\_unfinished**(*ratio\_unfinished: float*) → bool

Calculate the ratio of unfinished tasks in the submission.

**Parameters**

**ratio\_unfinished**

[float] the ratio of unfinished tasks in the submission

**Returns**

**bool**

whether the ratio of unfinished tasks in the submission is larger than ratio\_unfinished

**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

**machine**

[Machine] Machine class Object to execute the jobs

**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)****remove\_unfinished\_tasks()****run\_submission(\*, dry\_run=False, exit\_on\_submit=False, clean=True, check\_interval=30)**

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 dry\_run is True, submission will be uploaded but not be executed and exit. If exit\_on\_submit is True, submission will exit.

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

Convert the Submission class instance to a dictionary.

**Parameters****if\_static**

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

**Returns****submission\_dict**

[dict] the dictionary converted from the Submission class instance

**classmethod submission\_from\_json(json\_file\_name='submission.json')****submission\_to\_json()****try\_download\_result()****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.
<code>get_task_state(context)</code>	Get the task state by checking the tag file.

<b>arginfo</b>	
<b>get_hash</b>	
<b>load_from_dict</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()**

**get\_task\_state(context)**

Get the task state by checking the tag file.

#### Parameters

**context**

[Context] the context of the task

**classmethod load\_from\_dict(task\_dict: dict) → Task**

**classmethod load\_from\_json(json\_file)**

**serialize()**

## 8.1.22 dpdispatcher.utils module

**exception dpdispatcher.utils.RetrySignal**

Bases: [Exception](#)

Exception to give a signal to retry the function.

**dpdispatcher.utils.generate\_totp(secret: str, period: int = 30, token\_length: int = 6) → str**

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: str**

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.hotp(key: str, period: int, token_length: int = 6, digest='sha1')`

`dpdispatcher.utils.retry(max_retry: int = 3, sleep: ~typing.Union[int, float] = 60, catch_exception: ~typing.Type[BaseException] = <class 'dpdispatcher.utils.RetrySignal'>) → Callable`

Retry the function until it succeeds or fails for certain times.

#### Parameters

**max\_retry**

[int, default=3] The maximum retry times. If None, it will retry forever.

**sleep**

[int or float, default=60] The sleep time in seconds.

**catch\_exception**

[Exception, default=Exception] The exception to catch.

#### Returns

**decorator: Callable**

The decorator.

### Examples

```
>>> @retry(max_retry=3, sleep=60, catch_exception=RetrySignal)
... def func():
...     raise RetrySignal("Failed")
```

`dpdispatcher.utils.rsync(from_file: str, to_file: str, port: int = 22, key_filename: Optional[str] = None, timeout: Union[int, float] = 10)`

Call rsync to transfer files.

#### Parameters

**from\_file**

[str] SRC

**to\_file**

[str] DEST

**port**

[int, default=22] port for ssh

**key\_filename**

[str, optional] identity file name

**timeout**

[int, default=10] timeout for ssh

#### Raises

**RuntimeError**

when return code is not 0

`dpdispatcher.utils.run_cmd_with_all_output(cmd, shell=True)`



## RUNNING THE DEEPM-D-KIT ON THE EXPANSE CLUSTER

Expanse is a cluster operated by the San Diego Supercomputer Center. Here we provide an example to run jobs on the expanse.

The machine parameters are provided below. Expanse uses the SLURM workload manager for job scheduling. *remote\_root* has been created in advance. It's worth mentioned that we do not recommend to use the password, so *SSH keys* are used instead to improve security.

```
1 {
2   "batch_type": "Slurm",
3   "local_root": "./",
4   "remote_root": "/expanse/lustre/scratch/njzjz/temp_project/dpgen_workdir",
5   "clean_asynchronously": true,
6   "context_type": "SSHContext",
7   "remote_profile": {
8     "hostname": "login.expanse.sdsc.edu",
9     "username": "njzjz",
10    "port": 22
11  }
12 }
```

Expanse's standard compute nodes are each powered by two 64-core AMD EPYC 7742 processors and contain 256 GB of DDR4 memory. Here, we request one node with 32 cores and 16 GB memory from the shared partition. Expanse does not support `--gres=gpu:0` command, so we use *custom\_gpu\_line* to customize the statement.

```
1 {
2   "number_node": 1,
3   "cpu_per_node": 1,
4   "gpu_per_node": 0,
5   "queue_name": "shared",
6   "group_size": 1,
7   "custom_flags": [
8     "#SBATCH -c 32",
9     "#SBATCH --mem=16G",
10    "#SBATCH --time=48:00:00",
11    "#SBATCH --account=rut149",
12    "#SBATCH --requeue"
13  ],
14   "source_list": [
15     "activate /home/njzjz/deepmd-kit"
16  ],
17   "envs": {
```

(continues on next page)

(continued from previous page)

```
18     "OMP_NUM_THREADS": 4,  
19     "TF_INTRA_OP_PARALLELISM_THREADS": 4,  
20     "TF_INTER_OP_PARALLELISM_THREADS": 8,  
21     "DP_AUTO_PARALLELIZATION": 1  
22 },  
23 "batch_type": "Slurm",  
24 "kwargs": {  
25     "custom_gpu_line": "#SBATCH --gpus=0"  
26 }  
27 }
```

The following task parameter runs a DeePMD-kit task, forwarding an input file and backwarding graph files. Here, the data set will be used among all the tasks, so it is not included in the *forward\_files*. Instead, it should be included in the submission's *forward\_common\_files*.

```
1 {  
2     "command": "dp train input.json && dp freeze && dp compress",  
3     "task_work_path": "model1/",  
4     "forward_files": [  
5         "input.json"  
6     ],  
7     "backward_files": [  
8         "frozen_model.pb",  
9         "frozen_model_compressed.pb"  
10    ],  
11     "outlog": "log",  
12     "errlog": "err"  
13 }
```

## RUNNING GAUSSIAN 16 WITH FAILURE ALLOWED

Typically, a task will retry three times if the exit code is not zero. Sometimes, one may allow non-zero code. For example, when running large amounts of Gaussian 16 single-point calculation tasks, some of the Gaussian 16 tasks may throw SCF errors and return a non-zero code. One can append `||:` to the command:

```
1 {  
2   "command": "g16 < input > output ||:",  
3   "task_work_path": "p1/",  
4   "forward_files": [  
5     "input"  
6   ],  
7   "backward_files": [  
8     "output"  
9   ]  
10 }
```

This command ensures the task will always provide zero code.



## RUNNING MULTIPLE MD TASKS ON A GPU WORKSTATION

In this example, we are going to show how to run multiple MD tasks on a GPU workstation. This workstation does not install any job scheduling packages installed, so we will use Shell as *batch\_type*.

```
1 {
2   "batch_type": "Shell",
3   "local_root": "./",
4   "remote_root": "/data2/jinzhe/dpgen_workdir",
5   "clean_asynchronously": true,
6   "context_type": "SSHContext",
7   "remote_profile": {
8     "hostname": "mandu.iqb.rutgers.edu",
9     "username": "jz748",
10    "port": 22
11  }
12 }
```

The workstation has 48 cores of CPUs and 8 RTX3090 cards. Here we hope each card runs 6 tasks at the same time, as each task does not consume too many GPU resources. Thus, *strategy/if\_cuda\_multi\_devices* is set to `true` and *para\_deg* is set to 6.

```
1 {
2   "number_node": 1,
3   "cpu_per_node": 48,
4   "gpu_per_node": 8,
5   "queue_name": "shell",
6   "group_size": 0,
7   "strategy": {
8     "if_cuda_multi_devices": true
9   },
10  "source_list": [
11    "activate /home/jz748/deepmd-kit"
12  ],
13  "envs": {
14    "OMP_NUM_THREADS": 1,
15    "TF_INTRA_OP_PARALLELISM_THREADS": 1,
16    "TF_INTER_OP_PARALLELISM_THREADS": 1
17  },
18  "para_deg": 6
19 }
```

Note that *group\_size* should be set to 0 (means infinity) to ensure there is only one job and avoid running multiple jobs

at the same time.

## AUTHORS

- AnguseZhang
- Byron
- Cloudac7
- Feifei Tian
- Feiyang472
- Franklalalala
- Futaki Haduki
- Futaki Hatsuki
- Han Wang
- Han Y.B
- HuangJiameng
- Jinzhe Zeng
- KZHIWEI
- PKUfjh
- Pengchao Zhang
- Tongqi Wen
- TongqiWen
- Xiaoshan Luo
- Xuanyan Chen
- Yixiao Chen
- Yuan Fengbo
- Yuan Fengbo ()
- Yunpei Liu
- Zhang Yaotang
- Zhengju Sha
- Zhiwei Zhang
- chenglab

- ck
- dingzhaohan
- dinngzhaohan
- felix5572
- haidi
- likefallwind
- luobangkui
- pre-commit-ci[bot]
- robinzyb
- saltball
- shazj99
- tuoping
- unknown
- wangxiangfei
- yuzhi
- zhangbei07
- zhaohan
- zjgemi



## INDICES AND TABLES

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



## PYTHON MODULE INDEX

### d

- `dpdispatcher`, 25
- `dpdispatcher.arginfo`, 52
- `dpdispatcher.base_context`, 52
- `dpdispatcher.distributed_shell`, 53
- `dpdispatcher.dp_cloud_server`, 54
- `dpdispatcher.dp_cloud_server_context`, 56
- `dpdispatcher.dpcloudserver`, 49
- `dpdispatcher.dpcloudserver.client`, 49
- `dpdispatcher.dpcloudserver.config`, 50
- `dpdispatcher.dpcloudserver.retcode`, 50
- `dpdispatcher.dpcloudserver.zip_file`, 51
- `dpdispatcher.dpdisp`, 57
- `dpdispatcher.fugaku`, 57
- `dpdispatcher.hdfs_cli`, 58
- `dpdispatcher.hdfs_context`, 59
- `dpdispatcher.JobStatus`, 51
- `dpdispatcher.lazy_local_context`, 61
- `dpdispatcher.local_context`, 62
- `dpdispatcher.lsf`, 64
- `dpdispatcher.machine`, 65
- `dpdispatcher.pbs`, 68
- `dpdispatcher.shell`, 69
- `dpdispatcher.slurm`, 71
- `dpdispatcher.ssh_context`, 73
- `dpdispatcher.submission`, 76
- `dpdispatcher.utils`, 83



## INDEX

### A

alias (*dpdispatcher.base\_context.BaseContext* attribute), 52

alias (*dpdispatcher.dp\_cloud\_server.Bohrium* attribute), 55

alias (*dpdispatcher.dp\_cloud\_server\_context.BohriumContext* attribute), 56

alias (*dpdispatcher.Machine* attribute), 35

alias (*dpdispatcher.machine.Machine* attribute), 66

append\_script:  
    resources/append\_script (*Argument*), 19

arginfo() (*dpdispatcher.Machine* class method), 35

arginfo() (*dpdispatcher.machine.Machine* class method), 66

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

arginfo() (*dpdispatcher.ssh\_context.SSHSession* static method), 75

arginfo() (*dpdispatcher.submission.Resources* static method), 78

arginfo() (*dpdispatcher.submission.Task* static method), 82

arginfo() (*dpdispatcher.Task* static method), 47

async\_run\_submission() (*dpdispatcher.Submission* method), 44

async\_run\_submission() (*dpdispatcher.submission.Submission* method), 79

### B

backward\_files:  
    task/backward\_files (*Argument*), 23

BaseContext (class in *dpdispatcher.base\_context*), 52

batch\_type:  
    machine/batch\_type (*Argument*), 13

    resources/batch\_type (*Argument*), 19

bind\_context() (*dpdispatcher.Machine* method), 35

bind\_context() (*dpdispatcher.machine.Machine* method), 66

bind\_machine() (*dpdispatcher.Submission* method), 45

bind\_machine() (*dpdispatcher.submission.Submission* method), 80

bind\_submission() (*dpdispatcher.base\_context.BaseContext* method), 52

bind\_submission() (*dpdispatcher.dp\_cloud\_server\_context.BohriumContext* method), 56

bind\_submission() (*dpdispatcher.hdfs\_context.HDFSContext* method), 59

bind\_submission() (*dpdispatcher.HDFSContext* method), 28

bind\_submission() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 61

bind\_submission() (*dpdispatcher.LazyLocalContext* method), 32

bind\_submission() (*dpdispatcher.local\_context.LocalContext* method), 63

bind\_submission() (*dpdispatcher.LocalContext* method), 34

bind\_submission() (*dpdispatcher.ssh\_context.SSHContext* method), 73

bind\_submission() (*dpdispatcher.SSHContext* method), 40

block\_call() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 61

block\_call() (*dpdispatcher.LazyLocalContext* method), 32

block\_call() (*dpdispatcher.local\_context.LocalContext* method), 63

block\_call() (*dpdispatcher.LocalContext* method), 34

block\_call() (*dpdispatcher.ssh\_context.SSHContext* method), 73

block\_call() (*dpdispatcher.SSHContext* method), 40

block\_checkcall() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 61

block\_checkcall() (*dpdispatcher.LazyLocalContext* method), 32

block\_checkcall() (*dpdispatcher* method), 32

*patcher.local\_context.LocalContext* method), 63  
 block\_checkcall() (*dpdispatcher.LocalContext* method), 34  
 block\_checkcall() (*dpdispatcher.ssh\_context.SSHContext* method), 73  
 block\_checkcall() (*dpdispatcher.SSHContext* method), 40  
 Bohrium (class in *dpdispatcher.dp\_cloud\_server*), 54  
 BohriumContext (class in *dpdispatcher.dp\_cloud\_server\_context*), 56

## C

call() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 61  
 call() (*dpdispatcher.LazyLocalContext* method), 32  
 call() (*dpdispatcher.local\_context.LocalContext* method), 63  
 call() (*dpdispatcher.LocalContext* method), 34  
 call() (*dpdispatcher.ssh\_context.SSHContext* method), 74  
 call() (*dpdispatcher.SSHContext* method), 40  
 check\_all\_finished() (*dpdispatcher.Submission* method), 45  
 check\_all\_finished() (*dpdispatcher.submission.Submission* method), 80  
 check\_file\_exists() (*dpdispatcher.dp\_cloud\_server\_context.BohriumContext* method), 56  
 check\_file\_exists() (*dpdispatcher.hdfs\_context.HDFSContext* method), 59  
 check\_file\_exists() (*dpdispatcher.HDFSContext* method), 28  
 check\_file\_exists() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 61  
 check\_file\_exists() (*dpdispatcher.LazyLocalContext* method), 32  
 check\_file\_exists() (*dpdispatcher.local\_context.LocalContext* method), 63  
 check\_file\_exists() (*dpdispatcher.LocalContext* method), 34  
 check\_file\_exists() (*dpdispatcher.ssh\_context.SSHContext* method), 74  
 check\_file\_exists() (*dpdispatcher.SSHContext* method), 40  
 check\_finish() (*dpdispatcher.base\_context.BaseContext* method), 52  
 check\_finish() (*dpdispatcher.patcher.local\_context.LocalContext* method), 61  
 check\_finish() (*dpdispatcher.LazyLocalContext* method), 32  
 check\_finish() (*dpdispatcher.local\_context.LocalContext* method), 63  
 check\_finish() (*dpdispatcher.LocalContext* method), 34  
 check\_finish() (*dpdispatcher.ssh\_context.SSHContext* method), 74  
 check\_finish() (*dpdispatcher.SSHContext* method), 40  
 check\_finish\_tag() (*dpdispatcher.patcher.distributed\_shell.DistributedShell* method), 54  
 check\_finish\_tag() (*dpdispatcher.DistributedShell* method), 25  
 check\_finish\_tag() (*dpdispatcher.patcher.dp\_cloud\_server.Bohrium* method), 55  
 check\_finish\_tag() (*dpdispatcher.Fugaku* method), 27  
 check\_finish\_tag() (*dpdispatcher.fugaku.Fugaku* method), 58  
 check\_finish\_tag() (*dpdispatcher.LSF* method), 31  
 check\_finish\_tag() (*dpdispatcher.lsf.LSF* method), 64  
 check\_finish\_tag() (*dpdispatcher.Machine* method), 35  
 check\_finish\_tag() (*dpdispatcher.machine.Machine* method), 66  
 check\_finish\_tag() (*dpdispatcher.PBS* method), 37  
 check\_finish\_tag() (*dpdispatcher.pbs.PBS* method), 68  
 check\_finish\_tag() (*dpdispatcher.Shell* method), 41  
 check\_finish\_tag() (*dpdispatcher.shell.Shell* method), 70  
 check\_finish\_tag() (*dpdispatcher.Slurm* method), 42  
 check\_finish\_tag() (*dpdispatcher.slurm.Slurm* method), 71  
 check\_finish\_tag() (*dpdispatcher.slurm.SlurmJobArray* method), 72  
 check\_home\_file\_exits() (*dpdispatcher.dp\_cloud\_server\_context.BohriumContext* method), 56  
 check\_if\_recover() (*dpdispatcher.patcher.dp\_cloud\_server.Bohrium* method), 55  
 check\_if\_recover() (*dpdispatcher.Machine* method), 35  
 check\_if\_recover() (*dpdispatcher.machine.Machine* method), 66  
 check\_ratio\_unfinished()

*patcher.Submission method*), 45  
 check\_ratio\_unfinished() (*dpdispatcher.submission.Submission method*), 80  
 check\_status() (*dpdispatcher.distributed\_shell.DistributedShell method*), 54  
 check\_status() (*dpdispatcher.DistributedShell method*), 25  
 check\_status() (*dpdispatcher.dp\_cloud\_server.Bohrium method*), 55  
 check\_status() (*dpdispatcher.Fugaku method*), 27  
 check\_status() (*dpdispatcher.fugaku.Fugaku method*), 58  
 check\_status() (*dpdispatcher.LSF method*), 31  
 check\_status() (*dpdispatcher.lsf.LSF method*), 64  
 check\_status() (*dpdispatcher.Machine method*), 35  
 check\_status() (*dpdispatcher.machine.Machine method*), 66  
 check\_status() (*dpdispatcher.PBS method*), 38  
 check\_status() (*dpdispatcher.pbs.PBS method*), 68  
 check\_status() (*dpdispatcher.pbs.Torque method*), 69  
 check\_status() (*dpdispatcher.Shell method*), 41  
 check\_status() (*dpdispatcher.shell.Shell method*), 70  
 check\_status() (*dpdispatcher.Slurm method*), 42  
 check\_status() (*dpdispatcher.slurm.Slurm method*), 71  
 check\_status() (*dpdispatcher.slurm.SlurmJobArray method*), 72  
 check\_status() (*dpdispatcher.Torque method*), 48  
 clean() (*dpdispatcher.base\_context.BaseContext method*), 52  
 clean() (*dpdispatcher.dp\_cloud\_server\_context.BohriumContext method*), 56  
 clean() (*dpdispatcher.hdfs\_context.HDFSContext method*), 60  
 clean() (*dpdispatcher.HDFSContext method*), 28  
 clean() (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 62  
 clean() (*dpdispatcher.LazyLocalContext method*), 32  
 clean() (*dpdispatcher.local\_context.LocalContext method*), 63  
 clean() (*dpdispatcher.LocalContext method*), 34  
 clean() (*dpdispatcher.ssh\_context.SSHContext method*), 74  
 clean() (*dpdispatcher.SSHContext method*), 40  
 clean\_asynchronously:  
     machine/clean\_asynchronously (*Argument*), 13  
 clean\_jobs() (*dpdispatcher.Submission method*), 45  
 clean\_jobs() (*dpdispatcher.submission.Submission method*), 80  
 Client (*class in dpdispatcher.dpcloudserver*), 49  
 Client (*class in dpdispatcher.dpcloudserver.client*), 49  
 close() (*dpdispatcher.ssh\_context.SSHContext method*), 74  
 close() (*dpdispatcher.ssh\_context.SSHSession method*), 75  
 close() (*dpdispatcher.SSHContext method*), 40  
 command:  
     task/command (*Argument*), 23  
 completing (*dpdispatcher.JobStatus.JobStatus attribute*), 51  
 context\_type:  
     machine/context\_type (*Argument*), 13  
 copy\_from\_local() (*dpdispatcher.hdfs\_cli.HDFS static method*), 59  
 copy\_to\_local() (*dpdispatcher.hdfs\_cli.HDFS static method*), 59  
 cpu\_per\_node:  
     resources/cpu\_per\_node (*Argument*), 17  
 custom\_flags:  
     resources/custom\_flags (*Argument*), 17  
 custom\_gpu\_line:  
     resources[LSF]/kwargs/custom\_gpu\_line (*Argument*), 20  
     resources[SlurmJobArray]/kwargs/custom\_gpu\_line (*Argument*), 19  
     resources[Slurm]/kwargs/custom\_gpu\_line (*Argument*), 19  
**D**  
 DATAERR (*dpdispatcher.dpcloudserver.retcode.RETCODE attribute*), 50  
 DBERR (*dpdispatcher.dpcloudserver.retcode.RETCODE attribute*), 50  
 default\_resources() (*dpdispatcher.Fugaku method*), 27  
 default\_resources() (*dpdispatcher.fugaku.Fugaku method*), 58  
 default\_resources() (*dpdispatcher.LSF method*), 31  
 default\_resources() (*dpdispatcher.lsf.LSF method*), 64  
 default\_resources() (*dpdispatcher.Machine method*), 35  
 default\_resources() (*dpdispatcher.machine.Machine method*), 66  
 default\_resources() (*dpdispatcher.PBS method*), 38  
 default\_resources() (*dpdispatcher.pbs.PBS method*), 68  
 default\_resources() (*dpdispatcher.Shell method*), 42  
 default\_resources() (*dpdispatcher.shell.Shell method*), 70  
 default\_resources() (*dpdispatcher.Slurm method*), 43  
 default\_resources() (*dpdispatcher.slurm.Slurm method*), 71  
 deserialize() (*dpdispatcher.Job class method*), 29

[deserialize\(\) \(dpdispatcher.Machine class method\), 35](#)  
[deserialize\(\) \(dpdispatcher.machine.Machine class method\), 66](#)  
[deserialize\(\) \(dpdispatcher.Resources class method\), 39](#)  
[deserialize\(\) \(dpdispatcher.Submission class method\), 45](#)  
[deserialize\(\) \(dpdispatcher.submission.Job class method\), 76](#)  
[deserialize\(\) \(dpdispatcher.submission.Resources class method\), 78](#)  
[deserialize\(\) \(dpdispatcher.submission.Submission class method\), 80](#)  
[deserialize\(\) \(dpdispatcher.submission.Task class method\), 82](#)  
[deserialize\(\) \(dpdispatcher.Task class method\), 47](#)  
[DistributedShell \(class in dpdispatcher\), 25](#)  
[DistributedShell \(class in dpdispatcher.distributed\\_shell\), 53](#)  
[do\\_submit\(\) \(dpdispatcher.distributed\\_shell.DistributedShell method\), 54](#)  
[do\\_submit\(\) \(dpdispatcher.DistributedShell method\), 25](#)  
[do\\_submit\(\) \(dpdispatcher.dp\\_cloud\\_server.Bohrium method\), 55](#)  
[do\\_submit\(\) \(dpdispatcher.Fugaku method\), 27](#)  
[do\\_submit\(\) \(dpdispatcher.fugaku.Fugaku method\), 58](#)  
[do\\_submit\(\) \(dpdispatcher.LSF method\), 31](#)  
[do\\_submit\(\) \(dpdispatcher.lsf.LSF method\), 65](#)  
[do\\_submit\(\) \(dpdispatcher.Machine method\), 35](#)  
[do\\_submit\(\) \(dpdispatcher.machine.Machine method\), 66](#)  
[do\\_submit\(\) \(dpdispatcher.PBS method\), 38](#)  
[do\\_submit\(\) \(dpdispatcher.pbs.PBS method\), 68](#)  
[do\\_submit\(\) \(dpdispatcher.Shell method\), 42](#)  
[do\\_submit\(\) \(dpdispatcher.shell.Shell method\), 70](#)  
[do\\_submit\(\) \(dpdispatcher.Slurm method\), 43](#)  
[do\\_submit\(\) \(dpdispatcher.slurm.Slurm method\), 71](#)  
[download\(\) \(dpdispatcher.base\\_context.BaseContext method\), 52](#)  
[download\(\) \(dpdispatcher.dp\\_cloud\\_server\\_context.BohriumContext method\), 56](#)  
[download\(\) \(dpdispatcher.dpcloudserver.Client method\), 49](#)  
[download\(\) \(dpdispatcher.dpcloudserver.client.Client method\), 50](#)  
[download\(\) \(dpdispatcher.hdfs\\_context.HDFSContext method\), 60](#)  
[download\(\) \(dpdispatcher.HDFSContext method\), 28](#)  
[download\(\) \(dpdispatcher.lazy\\_local\\_context.LazyLocalContext method\), 62](#)  
[download\(\) \(dpdispatcher.LazyLocalContext method\), 32](#)  
[download\(\) \(dpdispatcher.local\\_context.LocalContext method\), 63](#)  
[download\(\) \(dpdispatcher.LocalContext method\), 34](#)  
[download\(\) \(dpdispatcher.ssh\\_context.SSHContext method\), 74](#)  
[download\(\) \(dpdispatcher.SSHContext method\), 40](#)  
[download\\_from\\_url\(\) \(dpdispatcher.dpcloudserver.Client method\), 49](#)  
[download\\_from\\_url\(\) \(dpdispatcher.dpcloudserver.client.Client method\), 50](#)  
[download\\_jobs\(\) \(dpdispatcher.Submission method\), 45](#)  
[download\\_jobs\(\) \(dpdispatcher.submission.Submission method\), 80](#)  
[DpCloudServer \(in module dpdispatcher\), 26](#)  
[DpCloudServer \(in module dpdispatcher.dp\\_cloud\\_server\), 55](#)  
[DpCloudServerContext \(in module dpdispatcher\), 26](#)  
[DpCloudServerContext \(in module dpdispatcher.dp\\_cloud\\_server\\_context\), 57](#)  
[dpdispatcher module, 25](#)  
[dpdispatcher.arginfo module, 52](#)  
[dpdispatcher.base\\_context module, 52](#)  
[dpdispatcher.distributed\\_shell module, 53](#)  
[dpdispatcher.dp\\_cloud\\_server module, 54](#)  
[dpdispatcher.dp\\_cloud\\_server\\_context module, 56](#)  
[dpdispatcher.dpcloudserver module, 49](#)  
[dpdispatcher.dpcloudserver.client module, 49](#)  
[dpdispatcher.dpcloudserver.config module, 50](#)  
[dpdispatcher.dpcloudserver.retcode module, 50](#)  
[dpdispatcher.dpcloudserver.zip\\_file module, 51](#)  
[dpdispatcher.dpdisp module, 57](#)  
[dpdispatcher.fugaku module, 57](#)  
[dpdispatcher.hdfs\\_cli module, 58](#)  
[dpdispatcher.hdfs\\_context module, 59](#)  
[dpdispatcher.JobStatus module, 51](#)



dpdispatcher.lazy\_local\_context  
     module, 61  
 dpdispatcher.local\_context  
     module, 62  
 dpdispatcher.lsf  
     module, 64  
 dpdispatcher.machine  
     module, 65  
 dpdispatcher.pbs  
     module, 68  
 dpdispatcher.shell  
     module, 69  
 dpdispatcher.slurm  
     module, 71  
 dpdispatcher.ssh\_context  
     module, 73  
 dpdispatcher.submission  
     module, 76  
 dpdispatcher.utils  
     module, 83

## E

email:  
     machine[BohriumContext]/remote\_profile/email/  
         (Argument), 14  
 ensure\_alive() (dpdispatcher.ssh\_context.SSHSession  
     method), 75  
 envs:  
     resources/envs (Argument), 19  
 errlog:  
     task/errlog (Argument), 23  
 exec\_command() (dpdispatcher.ssh\_context.SSHSession  
     method), 75  
 exists() (dpdispatcher.hdfs\_cli.HDFS static method),  
     59

## F

finished (dpdispatcher.JobStatus.JobStatus attribute),  
     51  
 forward\_files:  
     task/forward\_files (Argument), 23  
 Fugaku (class in dpdispatcher), 26  
 Fugaku (class in dpdispatcher.fugaku), 57

## G

gen\_command\_env\_cuda\_devices() (dpdis-  
     patcher.Machine method), 35  
 gen\_command\_env\_cuda\_devices() (dpdis-  
     patcher.machine.Machine method), 66  
 gen\_local\_script() (dpdis-  
     patcher.dp\_cloud\_server.Bohrium method),  
     55  
 gen\_script() (dpdispatcher.dp\_cloud\_server.Bohrium  
     method), 55  
 gen\_script() (dpdispatcher.Fugaku method), 27  
 gen\_script() (dpdispatcher.fugaku.Fugaku method), 58  
 gen\_script() (dpdispatcher.LSF method), 31  
 gen\_script() (dpdispatcher.lsf.LSF method), 65  
 gen\_script() (dpdispatcher.Machine method), 35  
 gen\_script() (dpdispatcher.machine.Machine method),  
     66  
 gen\_script() (dpdispatcher.PBS method), 38  
 gen\_script() (dpdispatcher.pbs.PBS method), 68  
 gen\_script() (dpdispatcher.Shell method), 42  
 gen\_script() (dpdispatcher.shell.Shell method), 70  
 gen\_script() (dpdispatcher.Slurm method), 43  
 gen\_script() (dpdispatcher.slurm.Slurm method), 71  
 gen\_script\_command() (dpdispatcher.Machine  
     method), 35  
 gen\_script\_command() (dpdis-  
     patcher.machine.Machine method), 66  
 gen\_script\_command() (dpdis-  
     patcher.slurm.Slurm.JobArray method), 72  
 gen\_script\_custom\_flags\_lines() (dpdis-  
     patcher.Machine method), 35  
 gen\_script\_custom\_flags\_lines() (dpdis-  
     patcher.machine.Machine method), 66  
 gen\_script\_end() (dpdis-  
     patcher.distributed\_shell.DistributedShell  
     method), 54  
 gen\_script\_end() (dpdispatcher.DistributedShell  
     method), 26  
 gen\_script\_end() (dpdispatcher.Machine method), 36  
 gen\_script\_end() (dpdispatcher.machine.Machine  
     method), 66  
 gen\_script\_end() (dpdispatcher.slurm.Slurm.JobArray  
     method), 72  
 gen\_script\_env() (dpdis-  
     patcher.distributed\_shell.DistributedShell  
     method), 54  
 gen\_script\_env() (dpdispatcher.DistributedShell  
     method), 26  
 gen\_script\_env() (dpdispatcher.Machine method), 36  
 gen\_script\_env() (dpdispatcher.machine.Machine  
     method), 66  
 gen\_script\_header() (dpdis-  
     patcher.distributed\_shell.DistributedShell  
     method), 54  
 gen\_script\_header() (dpdispatcher.DistributedShell  
     method), 26  
 gen\_script\_header() (dpdis-  
     patcher.dp\_cloud\_server.Bohrium method),  
     55  
 gen\_script\_header() (dpdispatcher.Fugaku method),  
     27  
 gen\_script\_header() (dpdispatcher.fugaku.Fugaku  
     method), 58  
 gen\_script\_header() (dpdispatcher.LSF method), 31

`gen_script_header()` (*dpdispatcher.lsf.LSF method*), 65  
`gen_script_header()` (*dpdispatcher.Machine method*), 36  
`gen_script_header()` (*dpdispatcher.machine.Machine method*), 66  
`gen_script_header()` (*dpdispatcher.PBS method*), 38  
`gen_script_header()` (*dpdispatcher.pbs.PBS method*), 68  
`gen_script_header()` (*dpdispatcher.pbs.Torque method*), 69  
`gen_script_header()` (*dpdispatcher.Shell method*), 42  
`gen_script_header()` (*dpdispatcher.shell.Shell method*), 70  
`gen_script_header()` (*dpdispatcher.Slurm method*), 43  
`gen_script_header()` (*dpdispatcher.slurm.Slurm method*), 71  
`gen_script_header()` (*dpdispatcher.slurm.SlurmJobArray method*), 72  
`gen_script_header()` (*dpdispatcher.Torque method*), 48  
`gen_script_wait()` (*dpdispatcher.Machine method*), 36  
`gen_script_wait()` (*dpdispatcher.machine.Machine method*), 66  
`generate_jobs()` (*dpdispatcher.Submission method*), 45  
`generate_jobs()` (*dpdispatcher.submission.Submission method*), 80  
`generate_totp()` (*in module dpdispatcher.utils*), 83  
`get()` (*dpdispatcher.dpcloudserver.Client method*), 49  
`get()` (*dpdispatcher.dpcloudserver.client.Client method*), 50  
`get()` (*dpdispatcher.ssh\_context.SSHSession method*), 75  
`get_hash()` (*dpdispatcher.Job method*), 29  
`get_hash()` (*dpdispatcher.Submission method*), 46  
`get_hash()` (*dpdispatcher.submission.Job method*), 76  
`get_hash()` (*dpdispatcher.submission.Submission method*), 81  
`get_hash()` (*dpdispatcher.submission.Task method*), 82  
`get_hash()` (*dpdispatcher.Task method*), 47  
`get_job_detail()` (*dpdispatcher.dpcloudserver.Client method*), 49  
`get_job_detail()` (*dpdispatcher.dpcloudserver.client.Client method*), 50  
`get_job_result_url()` (*dpdispatcher.dpcloudserver.Client method*), 49  
`get_job_result_url()` (*dpdispatcher.dpcloudserver.client.Client method*), 50  
`get_job_root()` (*dpdispatcher.hdfs\_context.HDFSContext method*), 60  
`get_job_root()` (*dpdispatcher.HDFSContext method*), 28  
`get_job_root()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 62  
`get_job_root()` (*dpdispatcher.LazyLocalContext method*), 33  
`get_job_root()` (*dpdispatcher.local\_context.LocalContext method*), 63  
`get_job_root()` (*dpdispatcher.LocalContext method*), 34  
`get_job_root()` (*dpdispatcher.ssh\_context.SSHContext method*), 74  
`get_job_root()` (*dpdispatcher.SSHContext method*), 40  
`get_job_state()` (*dpdispatcher.Job method*), 29  
`get_job_state()` (*dpdispatcher.submission.Job method*), 76  
`get_log()` (*dpdispatcher.dpcloudserver.Client method*), 49  
`get_log()` (*dpdispatcher.dpcloudserver.client.Client method*), 50  
`get_return()` (*dpdispatcher.lazy\_local\_context.LazyLocalContext method*), 62  
`get_return()` (*dpdispatcher.LazyLocalContext method*), 33  
`get_return()` (*dpdispatcher.local\_context.LocalContext method*), 63  
`get_return()` (*dpdispatcher.LocalContext method*), 34  
`get_return()` (*dpdispatcher.ssh\_context.SSHContext method*), 74  
`get_return()` (*dpdispatcher.SSHContext method*), 40  
`get_sha256()` (*in module dpdispatcher.utils*), 83  
`get_ssh_client()` (*dpdispatcher.ssh\_context.SSHSession method*), 75  
`get_task_state()` (*dpdispatcher.submission.Task method*), 83  
`get_task_state()` (*dpdispatcher.Task method*), 48  
`get_tasks_list()` (*dpdispatcher.dpcloudserver.Client method*), 49  
`get_tasks_list()` (*dpdispatcher.dpcloudserver.client.Client method*), 50  
`gpu_exclusive:`  
    *resources[LSF]/kwargs/gpu\_exclusive* (*Argument*), 20  
`gpu_new_syntax:`  
    *resources[LSF]/kwargs/gpu\_new\_syntax* (*Argument*), 20  
`gpu_per_node:`

resources/gpu\_per\_node (Argument), 17  
 gpu\_usage:  
   resources[LSF]/kwargs/gpu\_usage (Argument), 20  
 group\_size:  
   resources/group\_size (Argument), 17

## H

handle\_unexpected\_job\_state() (dpdispatcher.Job method), 30  
 handle\_unexpected\_job\_state() (dpdispatcher.submission.Job method), 77  
 handle\_unexpected\_submission\_state() (dpdispatcher.Submission method), 46  
 handle\_unexpected\_submission\_state() (dpdispatcher.submission.Submission method), 81  
 HDFS (class in dpdispatcher.hdfs\_cli), 58  
 HDFSContext (class in dpdispatcher), 27  
 HDFSContext (class in dpdispatcher.hdfs\_context), 59  
 hostname:  
   machine[SSHContext]/remote\_profile/hostname (Argument), 15  
 hotp() (in module dpdispatcher.utils), 84

## I

if\_cuda\_multi\_devices:  
   resources/strategy/if\_cuda\_multi\_devices (Argument), 18  
 info() (in module dpdispatcher), 48  
 input\_data:  
   machine[BohriumContext]/remote\_profile/input\_data (Argument), 14  
 inter\_handler() (dpdispatcher.ssh\_context.SSHSession method), 75  
 IOERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 50

## J

Job (class in dpdispatcher), 29  
 Job (class in dpdispatcher.submission), 76  
 job\_create() (dpdispatcher.dpcloudserver.Client method), 49  
 job\_create() (dpdispatcher.dpcloudserver.client.Client method), 50  
 job\_to\_json() (dpdispatcher.Job method), 30  
 job\_to\_json() (dpdispatcher.submission.Job method), 77  
 JobStatus (class in dpdispatcher.JobStatus), 51

## K

keep\_backup:  
   machine[BohriumContext]/remote\_profile/keep\_backup (Argument), 14

key\_filename:  
   machine[SSHContext]/remote\_profile/key\_filename (Argument), 15  
 kill() (dpdispatcher.LSF method), 31  
 kill() (dpdispatcher.lsf.LSF method), 65  
 kill() (dpdispatcher.Machine method), 36  
 kill() (dpdispatcher.machine.Machine method), 66  
 kill() (dpdispatcher.PBS method), 38  
 kill() (dpdispatcher.pbs.PBS method), 68  
 kill() (dpdispatcher.Shell method), 42  
 kill() (dpdispatcher.shell.Shell method), 70  
 kill() (dpdispatcher.Slurm method), 43  
 kill() (dpdispatcher.slurm.Slurm method), 71  
 kwargs:  
   resources[Bohrium]/kwargs (Argument), 21  
   resources[DistributedShell]/kwargs (Argument), 20  
   resources[Fugaku]/kwargs (Argument), 21  
   resources[LSF]/kwargs (Argument), 20  
   resources[PBS]/kwargs (Argument), 20  
   resources[Shell]/kwargs (Argument), 21  
   resources[SlurmJobArray]/kwargs (Argument), 19  
   resources[Slurm]/kwargs (Argument), 19  
   resources[Torque]/kwargs (Argument), 21

## L

LazyLocalContext (class in dpdispatcher), 31  
 LazyLocalContext (class in dpdispatcher.lazy\_local\_context), 61  
 Lebesgue (in module dpdispatcher), 33  
 Lebesgue (in module dpdispatcher.dp\_cloud\_server), 55  
 LebesgueContext (in module dpdispatcher), 33  
 LebesgueContext (in module dpdispatcher.dp\_cloud\_server\_context), 57  
 load\_from\_dict() (dpdispatcher.base\_context.BaseContext class method), 52  
 load\_from\_dict() (dpdispatcher.dp\_cloud\_server\_context.BohriumContext class method), 56  
 load\_from\_dict() (dpdispatcher.hdfs\_context.HDFSContext class method), 60  
 load\_from\_dict() (dpdispatcher.HDFSContext class method), 28  
 load\_from\_dict() (dpdispatcher.lazy\_local\_context.LazyLocalContext class method), 62  
 load\_from\_dict() (dpdispatcher.LazyLocalContext class method), 33  
 load\_from\_dict() (dpdispatcher.local\_context.LocalContext class method), 63

`load_from_dict()` (*dpdispatcher.LocalContext class method*), 34  
`load_from_dict()` (*dpdispatcher.Machine class method*), 36  
`load_from_dict()` (*dpdispatcher.machine.Machine class method*), 67  
`load_from_dict()` (*dpdispatcher.Resources class method*), 39  
`load_from_dict()` (*dpdispatcher.ssh\_context.SSHContext class method*), 74  
`load_from_dict()` (*dpdispatcher.SSHContext class method*), 40  
`load_from_dict()` (*dpdispatcher.submission.Resources class method*), 78  
`load_from_dict()` (*dpdispatcher.submission.Task class method*), 83  
`load_from_dict()` (*dpdispatcher.Task class method*), 48  
`load_from_json()` (*dpdispatcher.Machine class method*), 36  
`load_from_json()` (*dpdispatcher.machine.Machine class method*), 67  
`load_from_json()` (*dpdispatcher.Resources class method*), 39  
`load_from_json()` (*dpdispatcher.submission.Resources class method*), 78  
`load_from_json()` (*dpdispatcher.submission.Task class method*), 83  
`load_from_json()` (*dpdispatcher.Task class method*), 48  
`local_root:`  
    *machine/local\_root (Argument)*, 13  
*LocalContext (class in dpdispatcher)*, 33  
*LocalContext (class in dpdispatcher.local\_context)*, 62  
`look_for_keys:`  
    *machine[SSHContext]/remote\_profile/look\_for\_keys (Argument)*, 16  
*LSF (class in dpdispatcher)*, 30  
*LSF (class in dpdispatcher.lsf)*, 64

## M

*machine (Argument)*  
    *machine:*, 13  
*Machine (class in dpdispatcher)*, 34  
*Machine (class in dpdispatcher.machine)*, 65  
*machine/batch\_type (Argument)*  
    *batch\_type:*, 13  
*machine/clean\_asynchronously (Argument)*  
    *clean\_asynchronously:*, 13  
*machine/context\_type (Argument)*  
    *context\_type:*, 13  
*machine/local\_root (Argument)*  
    *local\_root:*, 13  
*machine/remote\_root (Argument)*  
    *remote\_root:*, 13  
*machine:*  
    *machine (Argument)*, 13  
*machine\_arginfo()* (*dpdispatcher.base\_context.BaseContext class method*), 52  
*machine\_subfields()* (*dpdispatcher.base\_context.BaseContext class method*), 52  
*machine\_subfields()* (*dpdispatcher.dp\_cloud\_server\_context.BohriumContext class method*), 56  
*machine\_subfields()* (*dpdispatcher.ssh\_context.SSHContext class method*), 74  
*machine\_subfields()* (*dpdispatcher.SSHContext class method*), 40  
*machine[BohriumContext]/remote\_profile (Argument)*  
    *remote\_profile:*, 14  
*machine[BohriumContext]/remote\_profile/email (Argument)*  
    *email:*, 14  
*machine[BohriumContext]/remote\_profile/input\_data (Argument)*  
    *input\_data:*, 14  
*machine[BohriumContext]/remote\_profile/keep\_backup (Argument)*  
    *keep\_backup:*, 14  
*machine[BohriumContext]/remote\_profile/password (Argument)*  
    *password:*, 14  
*machine[BohriumContext]/remote\_profile/program\_id (Argument)*  
    *program\_id:*, 14  
*machine[HDfSContext]/remote\_profile (Argument)*  
    *remote\_profile:*, 14  
*machine[LazyLocalContext]/remote\_profile (Argument)*  
    *remote\_profile:*, 14  
*machine[LocalContext]/remote\_profile (Argument)*  
    *remote\_profile:*, 15  
*machine[SSHContext]/remote\_profile/hostname (Argument)*  
    *hostname:*, 15  
*machine[SSHContext]/remote\_profile/key\_filename (Argument)*

key\_filename:, 15  
 machine[SSHContext]/remote\_profile/look\_for\_keys  
     (Argument)  
     look\_for\_keys:, 16  
 machine[SSHContext]/remote\_profile/passphrase  
     (Argument)  
     passphrase:, 15  
 machine[SSHContext]/remote\_profile/password  
     (Argument)  
     password:, 15  
 machine[SSHContext]/remote\_profile/port  
     (Argument)  
     port:, 15  
 machine[SSHContext]/remote\_profile/tar\_compress  
     (Argument)  
     tar\_compress:, 16  
 machine[SSHContext]/remote\_profile/timeout  
     (Argument)  
     timeout:, 15  
 machine[SSHContext]/remote\_profile/totp\_secret  
     (Argument)  
     totp\_secret:, 15  
 machine[SSHContext]/remote\_profile/username  
     (Argument)  
     username:, 15  
 main() (in module dpdispatcher.dpdisp), 57  
 map\_dp\_job\_state() (dpdispatcher.dp\_cloud\_server.Bohrium static method), 55  
 mkdir() (dpdispatcher.hdfs\_cli.HDFS static method), 59  
 module  
     dpdispatcher, 25  
     dpdispatcher.argo, 52  
     dpdispatcher.base\_context, 52  
     dpdispatcher.distributed\_shell, 53  
     dpdispatcher.dp\_cloud\_server, 54  
     dpdispatcher.dp\_cloud\_server\_context, 56  
     dpdispatcher.dpcloudserver, 49  
     dpdispatcher.dpcloudserver.client, 49  
     dpdispatcher.dpcloudserver.config, 50  
     dpdispatcher.dpcloudserver.retcode, 50  
     dpdispatcher.dpcloudserver.zip\_file, 51  
     dpdispatcher.dpdisp, 57  
     dpdispatcher.fugaku, 57  
     dpdispatcher.hdfs\_cli, 58  
     dpdispatcher.hdfs\_context, 59  
     dpdispatcher.JobStatus, 51  
     dpdispatcher.lazy\_local\_context, 61  
     dpdispatcher.local\_context, 62  
     dpdispatcher.lsf, 64  
     dpdispatcher.machine, 65  
     dpdispatcher.pbs, 68  
     dpdispatcher.shell, 69  
     dpdispatcher.slurm, 71  
     dpdispatcher.ssh\_context, 73  
     dpdispatcher.submission, 76  
     dpdispatcher.utils, 83  
     module\_list:  
         resources/module\_list (Argument), 18  
     module\_purge:  
         resources/module\_purge (Argument), 18  
     module\_unload\_list:  
         resources/module\_unload\_list (Argument), 18  
     move() (dpdispatcher.hdfs\_cli.HDFS static method), 59  
**N**  
 NODATA (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 50  
 number\_node:  
     resources/number\_node (Argument), 17  
**O**  
 OK (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 50  
 options (dpdispatcher.base\_context.BaseContext attribute), 52  
 options (dpdispatcher.Machine attribute), 36  
 options (dpdispatcher.machine.Machine attribute), 67  
 outlog:  
     task/outlog (Argument), 23  
**P**  
 para\_deg:  
     resources/para\_deg (Argument), 18  
 PARAMERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 51  
 passphrase:  
     machine[SSHContext]/remote\_profile/passphrase  
         (Argument), 15  
 password:  
     machine[BohriumContext]/remote\_profile/password  
         (Argument), 14  
     machine[SSHContext]/remote\_profile/password  
         (Argument), 15  
 PBS (class in dpdispatcher), 37  
 PBS (class in dpdispatcher.pbs), 68  
 port:  
     machine[SSHContext]/remote\_profile/port  
         (Argument), 15  
 post() (dpdispatcher.dpcloudserver.Client method), 49  
 post() (dpdispatcher.dpcloudserver.client.Client method), 50  
 prepend\_script:  
     resources/prepend\_script (Argument), 19  
 program\_id:  
     machine[BohriumContext]/remote\_profile/program\_id  
         (Argument), 14



put() (*dpdispatcher.ssh\_context.SSHSession* method), 75

PWDERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 51

## Q

queue\_name:  
resources/queue\_name (Argument), 17

## R

ratio\_unfinished:  
resources/strategy/ratio\_unfinished (Argument), 18

read() (*dpdispatcher.lazy\_local\_context.SPRetObj* method), 62

read() (*dpdispatcher.local\_context.SPRetObj* method), 64

read\_file() (*dpdispatcher.base\_context.BaseContext* method), 52

read\_file() (*dpdispatcher.dp\_cloud\_server\_context.BohriumContext* method), 56

read\_file() (*dpdispatcher.hdfs\_context.HDFSContext* method), 60

read\_file() (*dpdispatcher.HDFSContext* method), 28

read\_file() (*dpdispatcher.lazy\_local\_context.LazyLocalContext* method), 62

read\_file() (*dpdispatcher.LazyLocalContext* method), 33

read\_file() (*dpdispatcher.local\_context.LocalContext* method), 63

read\_file() (*dpdispatcher.LocalContext* method), 34

read\_file() (*dpdispatcher.ssh\_context.SSHContext* method), 74

read\_file() (*dpdispatcher.SSHContext* method), 41

read\_hdfs\_file() (*dpdispatcher.hdfs\_cli.HDFS* static method), 59

read\_home\_file() (*dpdispatcher.dp\_cloud\_server\_context.BohriumContext* method), 56

readlines() (*dpdispatcher.lazy\_local\_context.SPRetObj* method), 62

readlines() (*dpdispatcher.local\_context.SPRetObj* method), 64

refresh\_token() (*dpdispatcher.dpcloudserver.Client* method), 49

refresh\_token() (*dpdispatcher.dpcloudserver.client.Client* method), 50

register\_job\_id() (*dpdispatcher.Job* method), 30

register\_job\_id() (*dpdispatcher.submission.Job* method), 77

register\_task() (*dpdispatcher.Submission* method), 46

register\_task() (*dpdispatcher.submission.Submission* method), 81

register\_task\_list() (*dpdispatcher.Submission* method), 46

register\_task\_list() (*dpdispatcher.submission.Submission* method), 81

remote (*dpdispatcher.ssh\_context.SSHSession* property), 75

remote\_profile:  
machine[BohriumContext]/remote\_profile (Argument), 14  
machine[HDFSContext]/remote\_profile (Argument), 14  
machine[LazyLocalContext]/remote\_profile (Argument), 14  
machine[LocalContext]/remote\_profile (Argument), 16  
machine[SSHContext]/remote\_profile (Argument), 15

remote\_root:  
machine/remote\_root (Argument), 13

remove() (*dpdispatcher.hdfs\_cli.HDFS* static method), 59

remove\_unfinished\_tasks() (*dpdispatcher.Submission* method), 46

remove\_unfinished\_tasks() (*dpdispatcher.submission.Submission* method), 81

REQERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 51

RequestInfoException, 50

resources (Argument)  
resources:, 17

Resources (class in *dpdispatcher*), 38

Resources (class in *dpdispatcher.submission*), 77

resources/append\_script (Argument)  
append\_script:, 19

resources/batch\_type (Argument)  
batch\_type:, 19

resources/cpu\_per\_node (Argument)  
cpu\_per\_node:, 17

resources/custom\_flags (Argument)  
custom\_flags:, 17

resources/envs (Argument)  
envs:, 19

resources/gpu\_per\_node (Argument)  
gpu\_per\_node:, 17

resources/group\_size (Argument)  
group\_size:, 17

resources/module\_list (Argument)  
module\_list:, 18

resources/module\_purge (Argument)

**module\_purge:**, 18  
**resources/module\_unload\_list** (*Argument*)  
**module\_unload\_list:**, 18  
**resources/number\_node** (*Argument*)  
**number\_node:**, 17  
**resources/para\_deg** (*Argument*)  
**para\_deg:**, 18  
**resources/prepend\_script** (*Argument*)  
**prepend\_script:**, 19  
**resources/queue\_name** (*Argument*)  
**queue\_name:**, 17  
**resources/source\_list** (*Argument*)  
**source\_list:**, 18  
**resources/strategy** (*Argument*)  
**strategy:**, 18  
**resources/strategy/if\_cuda\_multi\_devices** (*Argument*)  
**if\_cuda\_multi\_devices:**, 18  
**resources/strategy/ratio\_unfinished** (*Argument*)  
**ratio\_unfinished:**, 18  
**resources/wait\_time** (*Argument*)  
**wait\_time:**, 19  
**resources:**  
**resources** (*Argument*), 17  
**resources\_arginfo**() (*dpdispatcher.Machine class method*), 36  
**resources\_arginfo**() (*dpdispatcher.machine.Machine class method*), 67  
**resources\_subfields**() (*dpdispatcher.LSF class method*), 31  
**resources\_subfields**() (*dpdispatcher.lsf.LSF class method*), 65  
**resources\_subfields**() (*dpdispatcher.Machine class method*), 36  
**resources\_subfields**() (*dpdispatcher.machine.Machine class method*), 67  
**resources\_subfields**() (*dpdispatcher.Slurm class method*), 43  
**resources\_subfields**() (*dpdispatcher.slurm.Slurm class method*), 72  
**resources\_subfields**() (*dpdispatcher.slurm.SlurmJobArray class method*), 72  
**resources[Bohrium]/kwargs** (*Argument*)  
**kwargs:**, 21  
**resources[DistributedShell]/kwargs** (*Argument*)  
**kwargs:**, 20  
**resources[Fugaku]/kwargs** (*Argument*)  
**kwargs:**, 21  
**resources[LSF]/kwargs** (*Argument*)  
**kwargs:**, 20  
**resources[LSF]/kwargs/custom\_gpu\_line** (*Argument*)  
**custom\_gpu\_line:**, 20  
**resources[LSF]/kwargs/gpu\_exclusive** (*Argument*)  
**gpu\_exclusive:**, 20  
**resources[LSF]/kwargs/gpu\_new\_syntax** (*Argument*)  
**gpu\_new\_syntax:**, 20  
**resources[LSF]/kwargs/gpu\_usage** (*Argument*)  
**gpu\_usage:**, 20  
**resources[PBS]/kwargs** (*Argument*)  
**kwargs:**, 20  
**resources[Shell]/kwargs** (*Argument*)  
**kwargs:**, 21  
**resources[SlurmJobArray]/kwargs** (*Argument*)  
**kwargs:**, 19  
**resources[SlurmJobArray]/kwargs/custom\_gpu\_line** (*Argument*)  
**custom\_gpu\_line:**, 19  
**resources[SlurmJobArray]/kwargs/slurm\_job\_size** (*Argument*)  
**slurm\_job\_size:**, 20  
**resources[Slurm]/kwargs** (*Argument*)  
**kwargs:**, 19  
**resources[Slurm]/kwargs/custom\_gpu\_line** (*Argument*)  
**custom\_gpu\_line:**, 19  
**resources[Torque]/kwargs** (*Argument*)  
**kwargs:**, 21  
**RETCODE** (*class in dpdispatcher.dpcloudserver.retcodes*), 50  
**retry**() (*in module dpdispatcher.utils*), 84  
**RetrySignal**, 83  
**ROLEERR** (*dpdispatcher.dpcloudserver.retcodes.RETCODE attribute*), 51  
**rsync**() (*in module dpdispatcher.utils*), 84  
**rsync\_available** (*dpdispatcher.ssh\_context.SSHSession property*), 75  
**run\_cmd\_with\_all\_output**() (*in module dpdispatcher.utils*), 84  
**run\_submission**() (*dpdispatcher.Submission method*), 46  
**run\_submission**() (*dpdispatcher.submission.Submission method*), 81  
**running** (*dpdispatcher.JobStatus.JobStatus attribute*), 51

## S

**serialize**() (*dpdispatcher.Job method*), 30  
**serialize**() (*dpdispatcher.Machine method*), 36  
**serialize**() (*dpdispatcher.machine.Machine method*), 67

- `serialize()` (*dpdispatcher.Resources* method), 39
  - `serialize()` (*dpdispatcher.Submission* method), 46
  - `serialize()` (*dpdispatcher.submission.Job* method), 77
  - `serialize()` (*dpdispatcher.submission.Resources* method), 78
  - `serialize()` (*dpdispatcher.submission.Submission* method), 81
  - `serialize()` (*dpdispatcher.submission.Task* method), 83
  - `serialize()` (*dpdispatcher.Task* method), 48
  - `sftp` (*dpdispatcher.ssh\_context.SSHContext* property), 74
  - `sftp` (*dpdispatcher.ssh\_context.SSHSession* property), 75
  - `sftp` (*dpdispatcher.SSHContext* property), 41
  - `Shell` (class in *dpdispatcher*), 41
  - `Shell` (class in *dpdispatcher.shell*), 69
  - `Slurm` (class in *dpdispatcher*), 42
  - `Slurm` (class in *dpdispatcher.slurm*), 71
  - `slurm_job_size:`
    - `resources[SlurmJobArray]/kwargs/slurm_job_size` (Argument), 20
  - `SlurmJobArray` (class in *dpdispatcher.slurm*), 72
  - `source_list:`
    - `resources/source_list` (Argument), 18
  - `SPRetObj` (class in *dpdispatcher.lazy\_local\_context*), 62
  - `SPRetObj` (class in *dpdispatcher.local\_context*), 63
  - `ssh` (*dpdispatcher.ssh\_context.SSHContext* property), 74
  - `ssh` (*dpdispatcher.SSHContext* property), 41
  - `SSHContext` (class in *dpdispatcher*), 39
  - `SSHContext` (class in *dpdispatcher.ssh\_context*), 73
  - `SSHSession` (class in *dpdispatcher.ssh\_context*), 74
  - `strategy:`
    - `resources/strategy` (Argument), 18
  - `sub_script_cmd()` (*dpdispatcher.LSF* method), 31
  - `sub_script_cmd()` (*dpdispatcher.lsf.LSF* method), 65
  - `sub_script_cmd()` (*dpdispatcher.Machine* method), 36
  - `sub_script_cmd()` (*dpdispatcher.machine.Machine* method), 67
  - `sub_script_head()` (*dpdispatcher.LSF* method), 31
  - `sub_script_head()` (*dpdispatcher.lsf.LSF* method), 65
  - `sub_script_head()` (*dpdispatcher.Machine* method), 36
  - `sub_script_head()` (*dpdispatcher.machine.Machine* method), 67
  - `subclasses_dict` (*dpdispatcher.base\_context.BaseContext* attribute), 52
  - `subclasses_dict` (*dpdispatcher.Machine* attribute), 36
  - `subclasses_dict` (*dpdispatcher.machine.Machine* attribute), 67
  - `Submission` (class in *dpdispatcher*), 43
  - `Submission` (class in *dpdispatcher.submission*), 78
  - `submission_from_json()` (*dpdispatcher.Submission* class method), 46
  - `submission_from_json()` (*dpdispatcher.submission.Submission* class method), 81
  - `submission_to_json()` (*dpdispatcher.Submission* method), 46
  - `submission_to_json()` (*dpdispatcher.submission.Submission* method), 81
  - `submit_job()` (*dpdispatcher.Job* method), 30
  - `submit_job()` (*dpdispatcher.submission.Job* method), 77
- T**
- `tar_compress:`
    - `machine[SSHContext]/remote_profile/tar_compress` (Argument), 16
  - `task` (Argument)
    - `task:`, 23
  - `Task` (class in *dpdispatcher*), 47
  - `Task` (class in *dpdispatcher.submission*), 82
  - `task/backward_files` (Argument)
    - `backward_files:`, 23
  - `task/command` (Argument)
    - `command:`, 23
  - `task/errlog` (Argument)
    - `errlog:`, 23
  - `task/forward_files` (Argument)
    - `forward_files:`, 23
  - `task/outlog` (Argument)
    - `outlog:`, 23
  - `task/task_work_path` (Argument)
    - `task_work_path:`, 23
  - `task:`
    - `task` (Argument), 23
  - `task_work_path:`
    - `task/task_work_path` (Argument), 23
  - `terminated` (*dpdispatcher.JobStatus.JobStatus* attribute), 51
  - `THIRDERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 51
  - `timeout:`
    - `machine[SSHContext]/remote_profile/timeout` (Argument), 15
  - `TOKENINVALID` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 51
  - `Torque` (class in *dpdispatcher*), 48
  - `Torque` (class in *dpdispatcher.pbs*), 69
  - `totp_secret:`
    - `machine[SSHContext]/remote_profile/totp_secret` (Argument), 15
  - `try_download_result()` (*dpdispatcher.Submission* method), 46
  - `try_download_result()` (*dpdispatcher.submission.Submission* method),



81  
 try\_recover\_from\_json() (dpdispatcher.Submission method), 46  
 try\_recover\_from\_json() (dpdispatcher.submission.Submission method), 81

**U**  
 UNDERDEBUG (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 51  
 unknown (dpdispatcher.JobStatus.JobStatus attribute), 51  
 UNKOWNERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 51  
 unsubmitted (dpdispatcher.JobStatus.JobStatus attribute), 51  
 unzip\_file() (in module dpdispatcher.dpcloudserver.zip\_file), 51  
 update\_submission\_state() (dpdispatcher.Submission method), 46  
 update\_submission\_state() (dpdispatcher.submission.Submission method), 81  
 upload() (dpdispatcher.base\_context.BaseContext method), 53  
 upload() (dpdispatcher.dp\_cloud\_server\_context.BohriumContext method), 57  
 upload() (dpdispatcher.dpcloudserver.Client method), 49  
 upload() (dpdispatcher.dpcloudserver.client.Client method), 50  
 upload() (dpdispatcher.hdfs\_context.HDFSContext method), 60  
 upload() (dpdispatcher.HDFSContext method), 28  
 upload() (dpdispatcher.lazy\_local\_context.LazyLocalContext method), 62  
 upload() (dpdispatcher.LazyLocalContext method), 33  
 upload() (dpdispatcher.local\_context.LocalContext method), 63  
 upload() (dpdispatcher.LocalContext method), 34  
 upload() (dpdispatcher.ssh\_context.SSHContext method), 74  
 upload() (dpdispatcher.SSHContext method), 41  
 upload\_job() (dpdispatcher.dp\_cloud\_server\_context.BohriumContext method), 57  
 upload\_jobs() (dpdispatcher.Submission method), 47  
 upload\_jobs() (dpdispatcher.submission.Submission method), 82  
 USERERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 51  
 username:  
   machine[SSHContext]/remote\_profile/username (Argument), 15

**V**  
 VERIFYERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 51

**W**  
 wait\_time:  
   resources/wait\_time (Argument), 19  
 waiting (dpdispatcher.JobStatus.JobStatus attribute), 51  
 write\_file() (dpdispatcher.base\_context.BaseContext method), 53  
 write\_file() (dpdispatcher.dp\_cloud\_server\_context.BohriumContext method), 57  
 write\_file() (dpdispatcher.hdfs\_context.HDFSContext method), 60  
 write\_file() (dpdispatcher.HDFSContext method), 29  
 write\_file() (dpdispatcher.lazy\_local\_context.LazyLocalContext method), 62  
 write\_file() (dpdispatcher.LazyLocalContext method), 33  
 write\_file() (dpdispatcher.local\_context.LocalContext method), 63  
 write\_file() (dpdispatcher.LocalContext method), 34  
 write\_file() (dpdispatcher.ssh\_context.SSHContext method), 74  
 write\_file() (dpdispatcher.SSHContext method), 41  
 write\_home\_file() (dpdispatcher.dp\_cloud\_server\_context.BohriumContext method), 57  
 write\_local\_file() (dpdispatcher.dp\_cloud\_server\_context.BohriumContext method), 57

**Z**  
 zip\_file\_list() (in module dpdispatcher.dpcloudserver.zip\_file), 51