
DPDispatcher

Deep Modeling

Apr 13, 2023

CONTENTS:

1	Install DPDispatcher	3
2	Getting Started	5
3	Supported contexts	9
3.1	LazyLocal	9
3.2	Local	9
3.3	SSH	9
3.4	Bohrium	9
3.5	HDFS	10
4	Supported batch job systems	11
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
5	Machine parameters	13
6	Resources parameters	17
7	Task parameters	23
8	DPDispatcher API	25
8.1	dpdispatcher package	25
9	Running the DeePMD-kit on the Expanse cluster	81
10	Running Gaussian 16 with failure allowed	83
11	Running multiple MD tasks on a GPU workstation	85
12	Authors	87
13	Indices and tables	89
	Python Module Index	91
	Index	93

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, a dpdispatcher *task* maps 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* 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.

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: PBS, Bohrium, Torque, DistributedShell, LSF, SlurmJobArray, Shell, Slurm

local_root:

type: NoneType | str

argument path: machine/local_root

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

remote_root:

type: NoneType | str, optional

argument path: machine/remote_root

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

clean_asynchronously:

type: bool, optional, default: False

argument path: machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

Depending on the value of *context_type*, different sub args are accepted.

context_type:

type: str (flag key)

argument path: machine/context_type

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

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

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 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 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.

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

argument path:

machine[SSHContext]/remote_profile/key_filename

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

passphrase:

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

argument path: machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

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

argument path:

`machine[SSHContext]/remote_profile/totp_secret`

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

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 `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.

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 *jobs* 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: *DistributedShell*, *Bohrium*, *PBS*, *LSF*, *Torque*, *Shell*,
Slurm, *SlurmJobArray*

The batch job system type loaded from machine/*batch_type*.

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

gpu_exclusive:

type: bool, optional, default: True
argument path: `resources[LSF]/kwargs/gpu_exclusive`

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

custom_gpu_line:

type: `NoneType` | str, optional, default: None
argument path: `resources[LSF]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with `#BSUB`

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

kwargs:

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

This field is empty for this batch.

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

kwargs:

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

This field is empty for this batch.

When `batch_type` is set to Slurm (or its alias `slurm`):

kwargs:

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

Extra arguments.

custom_gpu_line:

type: `NoneType` | `str`, optional, default: `None`
argument path: `resources[Slurm]/kwargs/custom_gpu_line`
Custom GPU configuration, starting with `#SBATCH`

When `batch_type` is set to `SlurmJobArray` (or its alias `slurmjobarray`):

kwargs:

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

Extra arguments.

custom_gpu_line:

type: `NoneType` | `str`, optional, default: `None`
argument path:
`resources[SlurmJobArray]/kwargs/custom_gpu_line`
Custom GPU configuration, starting with `#SBATCH`

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: NoneType | str

argument path: task/outlog

The out log file name. redirect from stdout

errlog:

type: `NoneType | str`

argument path: `task/errlog`

The err log file name. redirect from stderr

DPDISPATCHER API

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>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 *Bohrium***dpdispatcher.DpCloudServerContext**alias of *BohriumContext***class dpdispatcher.HDFSContext**(*args, **kwargs)Bases: *BaseContext***Methods**

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

bind_submission	
check_finish	
clean	
get_job_root	
kill	
load_from_dict	
read_file	
write_file	

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()****kill**(*job_id*)**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 usually has many tasks and it may request computing resources from job scheduler systems. Each Job can generate a script file to be submitted to the job scheduler system or executed locally.

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.LSF(*args, **kwargs)

Bases: *Machine*

LSF batch.

Methods

default_resources(resources)

<i>resources_arginfo</i> ()	Generate the resources arginfo.
-----------------------------	---------------------------------

<i>resources_subfields</i> ()	Generate the resources subfields.
-------------------------------	-----------------------------------

arginfo	
bind_context	
check_finish_tag	
check_if_recover	
check_status	
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*)

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.

bind_submission	
block_call	
block_checkcall	
call	
check_file_exists	
check_finish	
clean	
download	
get_job_root	
get_return	
kill	
load_from_dict	
read_file	
upload	
write_file	

bind_submission(*submission*)

block_call(*cmd*)

block_checkcall(*cmd*)

call(*cmd*)

check_file_exists(*fname*)

check_finish(*proc*)

clean()

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

get_job_root()

get_return(*proc*)

kill(*job_id*)

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.

bind_submission	
block_call	
block_checkcall	
call	
check_file_exists	
check_finish	
clean	
download	
get_job_root	
get_return	
kill	
load_from_dict	
read_file	
upload	
write_file	

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)
kill(job_id)
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>resources_arginfo()</code>	Generate the resources arginfo.
<code>resources_subfields()</code>	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	

```
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)

classmethod load_from_dict(machine_dict)

classmethod load_from_json(json_path)

options = {'Bohrium', 'DistributedShell', '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'>, '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'>, '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

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

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*)

```
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 *jobs* 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

arginfo	
deserialize	
load_from_dict	
load_from_json	
serialize	

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

<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.

bind_submission	
block_call	
call	
check_file_exists	
check_finish	
clean	
close	
download	
get_job_root	
get_return	
kill	
load_from_dict	
read_file	
upload	
write_file	

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)
kill(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

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

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*)

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

Bases: [Machine](#)

Methods

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	
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*)

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

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

check_ratio_unfinished	
clean_jobs	
download_jobs	
get_hash	
register_task	
register_task_list	
remove_unfinished_jobs	
submission_from_json	
submission_to_json	
try_recover_from_json	
upload_jobs	

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*)

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 unexpectly), resubmit the job. If the job state is unknown, raise an error.

register_task(*task*)

register_task_list(*task_list*)

remove_unfinished_jobs()

run_submission(*, *dry_run=False*, *exit_on_submit=False*, *clean=True*)

Main method to execute the submission. First, check whether old Submission exists on the remote machine, and try to recover from it. Second, upload the local files to the remote machine where the tasks to be executed. Third, run the submission defined previously. Forth, wait until the tasks in the submission finished and download the result file to local directory. If *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 infomation (like *job_id*, *job_state*, *fail_count*) to the dictionary.

Returns

submission_dict

[dict] the dictionary converted from the Submission class instance

classmethod **submission_from_json**(*json_file_name='submission.json'*)

submission_to_json()

try_recover_from_json()

update_submission_state()

Check whether all the jobs in the submission.

Notes

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

upload_jobs()

class **dpdispatcher.Task**(*command, task_work_path, forward_files=[], backward_files=[], outlog='log', errlog='err'*)

Bases: **object**

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

Parameters**command**

[Str] the command to be executed.

task_work_path

[Path] the directory of each file where the files are dependent on.

forward_files

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

backward_files

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

outlog

[Str] the filename to which command redirect stdout

errlog

[Str] the filename to which command redirect stderr

Methods

deserialize(task_dict)

Convert the task_dict to a Task class object.

arginfo	
get_hash	
load_from_dict	
load_from_json	
serialize	

static **arginfo**()

classmethod `deserialize(task_dict)`

Convert the task_dict to a Task class object.

Parameters

task_dict

[dict] the dictionary which contains the task information

Returns

task

[Task] the Task class instance converted from the task_dict

get_hash()

classmethod `load_from_dict(task_dict: dict) → Task`

classmethod `load_from_json(json_file)`

serialize()

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

Bases: *PBS*

Methods

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

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)
```

```
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

download	
download_from_url	
get	
get_job_detail	
get_job_result_url	
get_log	
get_tasks_list	
job_create	
post	
refresh_token	
upload	

```
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

download	
download_from_url	
get	
get_job_detail	
get_job_result_url	
get_log	
get_tasks_list	
job_create	
post	
refresh_token	
upload	

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>kill</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)

kill(proc)

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

<code>do_submit(job)</code>	Submit th job to yarn using distributed shell.
<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)`

8.1.7 dpdispatcher.dp_cloud_server module

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

Bases: *Machine*

Methods

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

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

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 *Bohrium*

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

<code>bind_submission</code>	
<code>check_file_exists</code>	
<code>check_finish</code>	
<code>check_home_file_exists</code>	
<code>clean</code>	
<code>download</code>	
<code>kill</code>	
<code>load_from_dict</code>	
<code>read_file</code>	
<code>read_home_file</code>	
<code>upload</code>	
<code>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)
```

```
kill(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*)

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.hdfs_cli module

class `dpdispatcher.hdfs_cli.HDFS`

Bases: `object`

Fundamental class for HDFS basic manipulation.

Methods

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

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

```

static copy_from_local(local_path, to_uri)
    Returns: True on success Raises: on unexpected error.

static copy_to_local(from_uri, local_path)

static exists(uri)
    Check existence of hdfs uri Returns: True on exists Raises: RuntimeError.

static mkdir(uri)
    Make new hdfs directory Returns: True on success Raises: RuntimeError.

static move(from_uri, to_uri)

static read_hdfs_file(uri)

static remove(uri)
    Check existence of hdfs uri Returns: True on exists Raises: RuntimeError.

```

8.1.11 dpdispatcher.hdfs_context module

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

Bases: [*BaseContext*](#)

Methods

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

bind_submission	
check_finish	
clean	
get_job_root	
kill	
load_from_dict	
read_file	
write_file	

```
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()

kill(*job_id*)

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*)

8.1.12 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.

bind_submission	
block_call	
block_checkcall	
call	
check_file_exists	
check_finish	
clean	
download	
get_job_root	
get_return	
kill	
load_from_dict	
read_file	
upload	
write_file	

bind_submission(*submission*)

block_call(*cmd*)

block_checkcall(*cmd*)

call(*cmd*)

check_file_exists(*fname*)

check_finish(*proc*)

clean()

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

get_job_root()

get_return(*proc*)

kill(*job_id*)

```
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

read	
readlines	

```
read()

readlines()
```

8.1.13 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.

bind_submission	
block_call	
block_checkcall	
call	
check_file_exists	
check_finish	
clean	
download	
get_job_root	
get_return	
kill	
load_from_dict	
read_file	
upload	
write_file	

```

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)

kill(job_id)

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

read	
readlines	

read()

readlines()

8.1.14 dpdispatcher.lsf module

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

Bases: *Machine*

LSF batch.

Methods

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

arginfo	
bind_context	
check_finish_tag	
check_if_recover	
check_status	
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*)

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

Returns

list[Argument]
resources subfields

sub_script_cmd(*res*)

sub_script_head(*res*)

8.1.15 dpdispatcher.machine module

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

Bases: `object`

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

Parameters

context

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

Methods

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

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	

```
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)
```

```

classmethod load_from_dict(machine_dict)

classmethod load_from_json(json_path)

options = {'Bohrium', 'DistributedShell', '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'>, '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'>, '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'>}]

```

8.1.16 dpdispatcher.pbs module

```

class dpdispatcher.pbs.PBS(*args, **kwargs)
    Bases: Machine

```

Methods

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

<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.pbs.Torque(*args, **kwargs)`

Bases: [*PBS*](#)

Methods

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

<code>arginfo</code>	
<code>bind_context</code>	
<code>check_finish_tag</code>	
<code>check_if_recover</code>	
<code>check_status</code>	
<code>default_resources</code>	
<code>deserialize</code>	
<code>gen_command_env_cuda_devices</code>	
<code>gen_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_status(job)`

`gen_script_header(job)`

8.1.17 dpdispatcher.shell module

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

Bases: [*Machine*](#)

Methods

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

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

8.1.18 dpdispatcher.slurm module

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

Bases: *Machine*

Methods

<code>resources_arginfo()</code>	Generate the resources arginfo.
<code>resources_subfields()</code>	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*)

classmethod resources_subfields() → [List\[Argument\]](#)

Generate the resources subfields.

Returns

list[Argument]

resources subfields

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

Bases: [Slurm](#)

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

Methods

<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)`

8.1.19 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.

bind_submission	
block_call	
call	
check_file_exists	
check_finish	
clean	
close	
download	
get_job_root	
get_return	
kill	
load_from_dict	
read_file	
upload	
write_file	

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)

kill(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

inter_handler (title, instructions, prompt_list)	inter_handler:	the	callback	for
			paramiko.transport.auth_interactive.	

arginfo	
close	
ensure_alive	
exec_command	
get	
get_ssh_client	
put	

```

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.

8.1.20 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 *jobs* 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

arginfo	
deserialize	
load_from_dict	
load_from_json	
serialize	

```
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>bind_machine(machine)</code>	Bind this submission to a machine.
<code>check_all_finished()</code>	Check whether all the jobs in the submission.
<code>deserialize(submission_dict[, machine])</code>	Convert the submission_dict to a Submission class object.
<code>generate_jobs()</code>	After tasks register to the self.belonging_tasks, This method generate the jobs and add these jobs to self.belonging_jobs.
<code>handle_unexpected_submission_state()</code>	Handle unexpected job state of the submission.
<code>run_submission(*[, 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>check_ratio_unfinished</code>	
<code>clean_jobs</code>	
<code>download_jobs</code>	
<code>get_hash</code>	
<code>register_task</code>	
<code>register_task_list</code>	
<code>remove_unfinished_jobs</code>	
<code>submission_from_json</code>	
<code>submission_to_json</code>	
<code>try_recover_from_json</code>	
<code>upload_jobs</code>	

`bind_machine(machine)`

Bind this submission to a machine. update the machine's context remote_root and local_root.

Parameters

machine

[Machine] the machine to bind with

`check_all_finished()`

Check whether all the jobs in the submission.

Notes

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

`check_ratio_unfinished(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 unexpectly), resubmit the job. If the job state is unknown, raise an error.

register_task(task)**register_task_list(task_list)****remove_unfinished_jobs()****run_submission(*, dry_run=False, exit_on_submit=False, clean=True)**

Main method to execute the submission. First, check whether old Submission exists on the remote machine, and try to recover from it. Second, upload the local files to the remote machine where the tasks to be executed. Third, run the submission defined previously. Forth, wait until the tasks in the submission finished and download the result file to local directory. If 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 infomation (like job_id, job_state, fail_count) to the dictionary.

Returns**submission_dict**

[dict] the dictionary converted from the Submission class instance

classmethod submission_from_json(json_file_name='submission.json')**submission_to_json()****try_recover_from_json()****update_submission_state()**

Check whether all the jobs in the submission.

Notes

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

upload_jobs()

```
class dpdispatcher.submission.Task(command, task_work_path, forward_files=[], backward_files=[],  
                                outlog='log', errlog='err')
```

Bases: `object`

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

Parameters

command

[Str] the command to be executed.

task_work_path

[Path] the directory of each file where the files are dependent on.

forward_files

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

backward_files

[list of Path] the files to be transmitted from remote machine after the comand finished.

outlog

[Str] the filename to which command redirect stdout

errlog

[Str] the filename to which command redirect stderr

Methods

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

arginfo	
get_hash	
load_from_dict	
load_from_json	
serialize	

static arginfo()

classmethod deserialize(task_dict)

Convert the task_dict to a Task class object.

Parameters

task_dict

[dict] the dictionary which contains the task information

Returns

task

[Task] the Task class instance converted from the task_dict

```
get_hash()

classmethod load_from_dict(task_dict: dict) → Task

classmethod load_from_json(json_file)

serialize()
```

8.1.21 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
- Xuanyan Chen
- Yixiao Chen
- Yuan Fengbo
- Yuan Fengbo ()
- Yunpei Liu
- Zhengju Sha
- Zhiwei Zhang
- chenglab
- ck
- dingzhaohan

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

INDICES AND TABLES

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

PYTHON MODULE INDEX

d

- `dpdispatcher`, 25
- `dpdispatcher.arginfo`, 49
- `dpdispatcher.base_context`, 49
- `dpdispatcher.distributed_shell`, 51
- `dpdispatcher.dp_cloud_server`, 52
- `dpdispatcher.dp_cloud_server_context`, 53
- `dpdispatcher.dpcloudserver`, 46
- `dpdispatcher.dpcloudserver.client`, 47
- `dpdispatcher.dpcloudserver.config`, 48
- `dpdispatcher.dpcloudserver.retcode`, 48
- `dpdispatcher.dpcloudserver.zip_file`, 48
- `dpdispatcher.dpdisp`, 54
- `dpdispatcher.hdfs_cli`, 54
- `dpdispatcher.hdfs_context`, 55
- `dpdispatcher.JobStatus`, 48
- `dpdispatcher.lazy_local_context`, 56
- `dpdispatcher.local_context`, 58
- `dpdispatcher.lsf`, 60
- `dpdispatcher.machine`, 61
- `dpdispatcher.pbs`, 63
- `dpdispatcher.shell`, 65
- `dpdispatcher.slurm`, 66
- `dpdispatcher.ssh_context`, 68
- `dpdispatcher.submission`, 71
- `dpdispatcher.utils`, 78

INDEX

A

alias (*dpdispatcher.base_context.BaseContext* attribute), 49

alias (*dpdispatcher.dp_cloud_server.Bohrium* attribute), 52

alias (*dpdispatcher.dp_cloud_server_context.BohriumContext* attribute), 53

alias (*dpdispatcher.Machine* attribute), 34

alias (*dpdispatcher.machine.Machine* attribute), 62

append_script:
resources/append_script (*Argument*), 19

arginfo() (*dpdispatcher.Machine* class method), 34

arginfo() (*dpdispatcher.machine.Machine* class method), 62

arginfo() (*dpdispatcher.Resources* static method), 37

arginfo() (*dpdispatcher.ssh_context.SSHSession* static method), 70

arginfo() (*dpdispatcher.submission.Resources* static method), 74

arginfo() (*dpdispatcher.submission.Task* static method), 77

arginfo() (*dpdispatcher.Task* static method), 44

B

backward_files:
task/backward_files (*Argument*), 23

BaseContext (class in *dpdispatcher.base_context*), 49

batch_type:
machine/batch_type (*Argument*), 13

resources/batch_type (*Argument*), 19

bind_context() (*dpdispatcher.Machine* method), 34

bind_context() (*dpdispatcher.machine.Machine* method), 62

bind_machine() (*dpdispatcher.Submission* method), 42

bind_machine() (*dpdispatcher.submission.Submission* method), 75

bind_submission() (*dpdispatcher.base_context.BaseContext* method), 49

bind_submission() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 53

bind_submission() (*dpdispatcher.patcher.hdfs_context.HDFSContext* method), 55

bind_submission() (*dpdispatcher.HDFSContext* method), 26

bind_submission() (*dpdispatcher.patcher.lazy_local_context.LazyLocalContext* method), 57

bind_submission() (*dpdispatcher.LazyLocalContext* method), 31

bind_submission() (*dpdispatcher.patcher.local_context.LocalContext* method), 59

bind_submission() (*dpdispatcher.LocalContext* method), 32

bind_submission() (*dpdispatcher.patcher.ssh_context.SSHContext* method), 69

bind_submission() (*dpdispatcher.SSHContext* method), 38

block_call() (*dpdispatcher.patcher.lazy_local_context.LazyLocalContext* method), 57

block_call() (*dpdispatcher.LazyLocalContext* method), 31

block_call() (*dpdispatcher.local_context.LocalContext* method), 59

block_call() (*dpdispatcher.LocalContext* method), 32

block_call() (*dpdispatcher.ssh_context.SSHContext* method), 69

block_call() (*dpdispatcher.SSHContext* method), 38

block_checkcall() (*dpdispatcher.patcher.lazy_local_context.LazyLocalContext* method), 57

block_checkcall() (*dpdispatcher.LazyLocalContext* method), 31

block_checkcall() (*dpdispatcher.patcher.local_context.LocalContext* method), 59

block_checkcall() (*dpdispatcher.LocalContext* method), 32

block_checkcall() (*dpdispatcher.patcher.ssh_context.SSHContext* method), 69

69
 block_checkcall() (dpdispatcher.SSHContext method), 38
 Bohrium (class in dpdispatcher.dp_cloud_server), 52
 BohriumContext (class in dpdispatcher.dp_cloud_server_context), 53

C
 call() (dpdispatcher.lazy_local_context.LazyLocalContext method), 57
 call() (dpdispatcher.LazyLocalContext method), 31
 call() (dpdispatcher.local_context.LocalContext method), 59
 call() (dpdispatcher.LocalContext method), 32
 call() (dpdispatcher.ssh_context.SSHContext method), 69
 call() (dpdispatcher.SSHContext method), 38
 check_all_finished() (dpdispatcher.Submission method), 42
 check_all_finished() (dpdispatcher.submission.Submission method), 75
 check_file_exists() (dpdispatcher.dp_cloud_server_context.BohriumContext method), 53
 check_file_exists() (dpdispatcher.hdfs_context.HDFSContext method), 55
 check_file_exists() (dpdispatcher.HDFSContext method), 26
 check_file_exists() (dpdispatcher.lazy_local_context.LazyLocalContext method), 57
 check_file_exists() (dpdispatcher.LazyLocalContext method), 31
 check_file_exists() (dpdispatcher.local_context.LocalContext method), 59
 check_file_exists() (dpdispatcher.LocalContext method), 32
 check_file_exists() (dpdispatcher.ssh_context.SSHContext method), 69
 check_file_exists() (dpdispatcher.SSHContext method), 38
 check_finish() (dpdispatcher.base_context.BaseContext method), 49
 check_finish() (dpdispatcher.lazy_local_context.LazyLocalContext method), 57
 check_finish() (dpdispatcher.LazyLocalContext method), 31
 check_finish() (dpdispatcher.local_context.LocalContext method), 59
 check_finish() (dpdispatcher.LocalContext method), 32
 check_finish() (dpdispatcher.ssh_context.SSHContext method), 69
 check_finish() (dpdispatcher.SSHContext method), 38
 check_finish_tag() (dpdispatcher.distributed_shell.DistributedShell method), 51
 check_finish_tag() (dpdispatcher.DistributedShell method), 25
 check_finish_tag() (dpdispatcher.dp_cloud_server.Bohrium method), 52
 check_finish_tag() (dpdispatcher.LSF method), 29
 check_finish_tag() (dpdispatcher.lsf.LSF method), 60
 check_finish_tag() (dpdispatcher.Machine method), 34
 check_finish_tag() (dpdispatcher.machine.Machine method), 62
 check_finish_tag() (dpdispatcher.PBS method), 36
 check_finish_tag() (dpdispatcher.pbs.PBS method), 64
 check_finish_tag() (dpdispatcher.Shell method), 40
 check_finish_tag() (dpdispatcher.shell.Shell method), 66
 check_finish_tag() (dpdispatcher.Slurm method), 41
 check_finish_tag() (dpdispatcher.slurm.Slurm method), 67
 check_finish_tag() (dpdispatcher.slurm.SlurmJobArray method), 68
 check_home_file_exists() (dpdispatcher.dp_cloud_server_context.BohriumContext method), 53
 check_if_recover() (dpdispatcher.dp_cloud_server.Bohrium method), 52
 check_if_recover() (dpdispatcher.Machine method), 34
 check_if_recover() (dpdispatcher.machine.Machine method), 62
 check_ratio_unfinished() (dpdispatcher.Submission method), 43
 check_ratio_unfinished() (dpdispatcher.submission.Submission method), 75
 check_status() (dpdispatcher.distributed_shell.DistributedShell method), 51
 check_status() (dpdispatcher.DistributedShell method), 25

`check_status()` (*dpdispatcher.dp_cloud_server.Bohrium method*), 52
`check_status()` (*dpdispatcher.LSF method*), 29
`check_status()` (*dpdispatcher.lsf.LSF method*), 60
`check_status()` (*dpdispatcher.Machine method*), 34
`check_status()` (*dpdispatcher.machine.Machine method*), 62
`check_status()` (*dpdispatcher.PBS method*), 36
`check_status()` (*dpdispatcher.pbs.PBS method*), 64
`check_status()` (*dpdispatcher.pbs.Torque method*), 65
`check_status()` (*dpdispatcher.Shell method*), 40
`check_status()` (*dpdispatcher.shell.Shell method*), 66
`check_status()` (*dpdispatcher.Slurm method*), 41
`check_status()` (*dpdispatcher.slurm.Slurm method*), 67
`check_status()` (*dpdispatcher.slurm.SlurmJobArray method*), 68
`check_status()` (*dpdispatcher.Torque method*), 45
`clean()` (*dpdispatcher.base_context.BaseContext method*), 49
`clean()` (*dpdispatcher.dp_cloud_server_context.BohriumContext method*), 53
`clean()` (*dpdispatcher.hdfs_context.HDFSContext method*), 56
`clean()` (*dpdispatcher.HDFSContext method*), 27
`clean()` (*dpdispatcher.lazy_local_context.LazyLocalContext method*), 57
`clean()` (*dpdispatcher.LazyLocalContext method*), 31
`clean()` (*dpdispatcher.local_context.LocalContext method*), 59
`clean()` (*dpdispatcher.LocalContext method*), 33
`clean()` (*dpdispatcher.ssh_context.SSHContext method*), 69
`clean()` (*dpdispatcher.SSHContext method*), 38
`clean_asynchronously:`
 machine/clean_asynchronously (*Argument*), 13
`clean_jobs()` (*dpdispatcher.Submission method*), 43
`clean_jobs()` (*dpdispatcher.submission.Submission method*), 75
`Client` (*class in dpdispatcher.dpcloudserver*), 46
`Client` (*class in dpdispatcher.dpcloudserver.client*), 47
`close()` (*dpdispatcher.ssh_context.SSHContext method*), 69
`close()` (*dpdispatcher.ssh_context.SSHSession method*), 70
`close()` (*dpdispatcher.SSHContext method*), 39
`command:`
 task/command (*Argument*), 23
`completing` (*dpdispatcher.JobStatus.JobStatus attribute*), 48
`context_type:`
 machine/context_type (*Argument*), 13
`copy_from_local()` (*dpdispatcher.hdfs_cli.HDFS static method*), 54
`copy_to_local()` (*dpdispatcher.hdfs_cli.HDFS static method*), 55
`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*), 21
 resources[Slurm]/kwargs/custom_gpu_line (*Argument*), 21

D

`DATAERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE attribute*), 48
`DBERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE attribute*), 48
`default_resources()` (*dpdispatcher.LSF method*), 29
`default_resources()` (*dpdispatcher.lsf.LSF method*), 60
`default_resources()` (*dpdispatcher.Machine method*), 34
`default_resources()` (*dpdispatcher.machine.Machine method*), 62
`default_resources()` (*dpdispatcher.PBS method*), 36
`default_resources()` (*dpdispatcher.pbs.PBS method*), 64
`default_resources()` (*dpdispatcher.Shell method*), 40
`default_resources()` (*dpdispatcher.shell.Shell method*), 66
`default_resources()` (*dpdispatcher.Slurm method*), 41
`default_resources()` (*dpdispatcher.slurm.Slurm method*), 67
`deserialize()` (*dpdispatcher.Job class method*), 28
`deserialize()` (*dpdispatcher.Machine class method*), 34
`deserialize()` (*dpdispatcher.machine.Machine class method*), 62
`deserialize()` (*dpdispatcher.Resources class method*), 37
`deserialize()` (*dpdispatcher.Submission class method*), 43
`deserialize()` (*dpdispatcher.submission.Job class method*), 72
`deserialize()` (*dpdispatcher.submission.Resources class method*), 74
`deserialize()` (*dpdispatcher.submission.Submission class method*), 75
`deserialize()` (*dpdispatcher.submission.Task class method*), 77

<code>deserialize()</code> (<i>dpdispatcher.Task</i> class method), 44	<code>patcher.dp_cloud_server</code>), 53
<code>DistributedShell</code> (class in <i>dpdispatcher</i>), 25	<code>DpCloudServerContext</code> (in module <i>dpdispatcher</i>), 26
<code>DistributedShell</code> (class in <i>dpdispatcher.distributed_shell</i>), 51	<code>DpCloudServerContext</code> (in module <i>dpdispatcher.dp_cloud_server_context</i>), 54
<code>do_submit()</code> (<i>dpdispatcher.distributed_shell.DistributedShell</i> method), 51	<i>dpdispatcher</i>
<code>do_submit()</code> (<i>dpdispatcher.DistributedShell</i> method), 25	module, 25
<code>do_submit()</code> (<i>dpdispatcher.dp_cloud_server.Bohrium</i> method), 52	<code>dpdispatcher.arginfo</code>
<code>do_submit()</code> (<i>dpdispatcher.LSF</i> method), 29	module, 49
<code>do_submit()</code> (<i>dpdispatcher.lsf.LSF</i> method), 61	<code>dpdispatcher.base_context</code>
<code>do_submit()</code> (<i>dpdispatcher.Machine</i> method), 34	module, 49
<code>do_submit()</code> (<i>dpdispatcher.machine.Machine</i> method), 62	<code>dpdispatcher.distributed_shell</code>
<code>do_submit()</code> (<i>dpdispatcher.PBS</i> method), 36	module, 51
<code>do_submit()</code> (<i>dpdispatcher.pbs.PBS</i> method), 64	<code>dpdispatcher.dp_cloud_server</code>
<code>do_submit()</code> (<i>dpdispatcher.Shell</i> method), 40	module, 52
<code>do_submit()</code> (<i>dpdispatcher.shell.Shell</i> method), 66	<code>dpdispatcher.dp_cloud_server_context</code>
<code>do_submit()</code> (<i>dpdispatcher.Slurm</i> method), 41	module, 53
<code>do_submit()</code> (<i>dpdispatcher.slurm.Slurm</i> method), 67	<code>dpdispatcher.dpcloudserver</code>
<code>download()</code> (<i>dpdispatcher.base_context.BaseContext</i> method), 49	module, 46
<code>download()</code> (<i>dpdispatcher.dp_cloud_server_context.BohriumContext</i> method), 53	<code>dpdispatcher.dpcloudserver.client</code>
<code>download()</code> (<i>dpdispatcher.dpcloudserver.Client</i> method), 46	module, 47
<code>download()</code> (<i>dpdispatcher.dpcloudserver.client.Client</i> method), 47	<code>dpdispatcher.dpcloudserver.config</code>
<code>download()</code> (<i>dpdispatcher.hdfs_context.HDFSContext</i> method), 56	module, 48
<code>download()</code> (<i>dpdispatcher.HDFSContext</i> method), 27	<code>dpdispatcher.dpcloudserver.retcode</code>
<code>download()</code> (<i>dpdispatcher.lazy_local_context.LazyLocalContext</i> method), 57	module, 48
<code>download()</code> (<i>dpdispatcher.LazyLocalContext</i> method), 31	<code>dpdispatcher.dpcloudserver.zip_file</code>
<code>download()</code> (<i>dpdispatcher.local_context.LocalContext</i> method), 59	module, 48
<code>download()</code> (<i>dpdispatcher.LocalContext</i> method), 33	<code>dpdispatcher.dpdisp</code>
<code>download()</code> (<i>dpdispatcher.ssh_context.SSHContext</i> method), 69	module, 54
<code>download()</code> (<i>dpdispatcher.SSHContext</i> method), 39	<code>dpdispatcher.hdfs_cli</code>
<code>download_from_url()</code> (<i>dpdispatcher.dpcloudserver.Client</i> method), 46	module, 54
<code>download_from_url()</code> (<i>dpdispatcher.dpcloudserver.client.Client</i> method), 47	<code>dpdispatcher.hdfs_context</code>
<code>download_jobs()</code> (<i>dpdispatcher.Submission</i> method), 43	module, 55
<code>download_jobs()</code> (<i>dpdispatcher.submission.Submission</i> method), 76	<code>dpdispatcher.JobStatus</code>
<code>DpCloudServer</code> (in module <i>dpdispatcher</i>), 26	module, 48
<code>DpCloudServer</code> (in module <i>dpdispatcher</i>), 26	<code>dpdispatcher.lazy_local_context</code>
	module, 56
	<code>dpdispatcher.local_context</code>
	module, 58
	<code>dpdispatcher.lsf</code>
	module, 60
	<code>dpdispatcher.machine</code>
	module, 61
	<code>dpdispatcher.pbs</code>
	module, 63
	<code>dpdispatcher.shell</code>
	module, 65
	<code>dpdispatcher.slurm</code>
	module, 66
	<code>dpdispatcher.ssh_context</code>
	module, 68
	<code>dpdispatcher.submission</code>
	module, 71
	<code>dpdispatcher.utils</code>
	module, 78

E

email:
 machine[BohriumContext]/remote_profile/email
 (Argument), 14
 ensure_alive() (dpdispatcher.ssh_context.SSHSession
 method), 70
 envs:
 resources/envs (Argument), 19
 errlog:
 task/errlog (Argument), 23
 exec_command() (dpdispatcher.ssh_context.SSHSession
 method), 71
 exists() (dpdispatcher.hdfs_cli.HDFS static method),
 55

F

finished (dpdispatcher.JobStatus.JobStatus attribute),
 49
 forward_files:
 task/forward_files (Argument), 23

G

gen_command_env_cuda_devices() (dpdis-
 patcher.Machine method), 34
 gen_command_env_cuda_devices() (dpdis-
 patcher.machine.Machine method), 62
 gen_local_script() (dpdis-
 patcher.dp_cloud_server.Bohrium method),
 52
 gen_script() (dpdispatcher.dp_cloud_server.Bohrium
 method), 52
 gen_script() (dpdispatcher.LSF method), 29
 gen_script() (dpdispatcher.lsf.LSF method), 61
 gen_script() (dpdispatcher.Machine method), 34
 gen_script() (dpdispatcher.machine.Machine method),
 62
 gen_script() (dpdispatcher.PBS method), 36
 gen_script() (dpdispatcher.pbs.PBS method), 64
 gen_script() (dpdispatcher.Shell method), 40
 gen_script() (dpdispatcher.shell.Shell method), 66
 gen_script() (dpdispatcher.Slurm method), 41
 gen_script() (dpdispatcher.slurm.Slurm method), 67
 gen_script_command() (dpdispatcher.Machine
 method), 34
 gen_script_command() (dpdis-
 patcher.machine.Machine method), 62
 gen_script_command() (dpdis-
 patcher.slurm.SlurmJobArray method), 68
 gen_script_custom_flags_lines() (dpdis-
 patcher.Machine method), 34
 gen_script_custom_flags_lines() (dpdis-
 patcher.machine.Machine method), 62

gen_script_end() (dpdis-
 patcher.distributed_shell.DistributedShell
 method), 51
 gen_script_end() (dpdispatcher.DistributedShell
 method), 26
 gen_script_end() (dpdispatcher.Machine method), 35
 gen_script_end() (dpdispatcher.machine.Machine
 method), 62
 gen_script_end() (dpdispatcher.slurm.SlurmJobArray
 method), 68
 gen_script_env() (dpdis-
 patcher.distributed_shell.DistributedShell
 method), 51
 gen_script_env() (dpdispatcher.DistributedShell
 method), 26
 gen_script_env() (dpdispatcher.Machine method), 35
 gen_script_env() (dpdispatcher.machine.Machine
 method), 62
 gen_script_header() (dpdis-
 patcher.distributed_shell.DistributedShell
 method), 51
 gen_script_header() (dpdispatcher.DistributedShell
 method), 26
 gen_script_header() (dpdispatcher.dp_cloud_server.Bohrium
 method),
 52
 gen_script_header() (dpdispatcher.LSF method), 30
 gen_script_header() (dpdispatcher.lsf.LSF method),
 61
 gen_script_header() (dpdispatcher.Machine
 method), 35
 gen_script_header() (dpdispatcher.machine.Machine
 method), 62
 gen_script_header() (dpdispatcher.PBS method), 36
 gen_script_header() (dpdispatcher.pbs.PBS method),
 64
 gen_script_header() (dpdispatcher.pbs.Torque
 method), 65
 gen_script_header() (dpdispatcher.Shell method), 40
 gen_script_header() (dpdispatcher.shell.Shell
 method), 66
 gen_script_header() (dpdispatcher.Slurm method),
 41
 gen_script_header() (dpdispatcher.slurm.Slurm
 method), 67
 gen_script_header() (dpdis-
 patcher.slurm.SlurmJobArray method), 68
 gen_script_header() (dpdispatcher.Torque method),
 45
 gen_script_wait() (dpdispatcher.Machine method),
 35
 gen_script_wait() (dpdispatcher.machine.Machine
 method), 62
 generate_jobs() (dpdispatcher.Submission method),

- 43
generate_jobs() (dpdispatcher.submission.Submission method), 76
generate_totp() (in module dpdispatcher.utils), 78
get() (dpdispatcher.dpcloudserver.Client method), 46
get() (dpdispatcher.dpcloudserver.client.Client method), 47
get() (dpdispatcher.ssh_context.SSHSession method), 71
get_hash() (dpdispatcher.Job method), 28
get_hash() (dpdispatcher.Submission method), 43
get_hash() (dpdispatcher.submission.Job method), 72
get_hash() (dpdispatcher.submission.Submission method), 76
get_hash() (dpdispatcher.submission.Task method), 77
get_hash() (dpdispatcher.Task method), 45
get_job_detail() (dpdispatcher.dpcloudserver.Client method), 46
get_job_detail() (dpdispatcher.dpcloudserver.client.Client method), 47
get_job_result_url() (dpdispatcher.dpcloudserver.Client method), 46
get_job_result_url() (dpdispatcher.dpcloudserver.client.Client method), 47
get_job_root() (dpdispatcher.hdfs_context.HDFSContext method), 56
get_job_root() (dpdispatcher.HDFSContext method), 27
get_job_root() (dpdispatcher.lazy_local_context.LazyLocalContext method), 57
get_job_root() (dpdispatcher.LazyLocalContext method), 31
get_job_root() (dpdispatcher.local_context.LocalContext method), 59
get_job_root() (dpdispatcher.LocalContext method), 33
get_job_root() (dpdispatcher.ssh_context.SSHContext method), 69
get_job_root() (dpdispatcher.SSHContext method), 39
get_job_state() (dpdispatcher.Job method), 28
get_job_state() (dpdispatcher.submission.Job method), 72
get_log() (dpdispatcher.dpcloudserver.Client method), 46
get_log() (dpdispatcher.dpcloudserver.client.Client method), 47
get_return() (dpdispatcher.lazy_local_context.LazyLocalContext method), 57
get_return() (dpdispatcher.LazyLocalContext method), 31
get_return() (dpdispatcher.local_context.LocalContext method), 59
get_return() (dpdispatcher.LocalContext method), 33
get_return() (dpdispatcher.ssh_context.SSHContext method), 70
get_return() (dpdispatcher.SSHContext method), 39
get_sha256() (in module dpdispatcher.utils), 78
get_ssh_client() (dpdispatcher.ssh_context.SSHSession method), 71
get_tasks_list() (dpdispatcher.dpcloudserver.Client method), 46
get_tasks_list() (dpdispatcher.dpcloudserver.client.Client method), 47
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), 28
handle_unexpected_job_state() (dpdispatcher.submission.Job method), 72
handle_unexpected_submission_state() (dpdispatcher.Submission method), 43
handle_unexpected_submission_state() (dpdispatcher.submission.Submission method), 76
HDFS (class in dpdispatcher.hdfs_cli), 54
HDFSContext (class in dpdispatcher), 26
HDFSContext (class in dpdispatcher.hdfs_context), 55
hostname: machine[SSHContext]/remote_profile/hostname (Argument), 15
http() (in module dpdispatcher.utils), 78
I
if_cuda_multi_devices: resources/strategy/if_cuda_multi_devices (Argument), 18
info() (in module dpdispatcher), 46
input_data:

machine[BohriumContext]/remote_profile/inplut_data
 (Argument), 14
 inter_handler() (dpdispatcher.ssh_context.SSHSession method), 71
 IOERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 48
J
 Job (class in dpdispatcher), 27
 Job (class in dpdispatcher.submission), 71
 job_create() (dpdispatcher.dpcloudserver.Client method), 46
 job_create() (dpdispatcher.dpcloudserver.client.Client method), 47
 job_to_json() (dpdispatcher.Job method), 28
 job_to_json() (dpdispatcher.submission.Job method), 72
 JobStatus (class in dpdispatcher.JobStatus), 48
K
 keep_backup:
 machine[BohriumContext]/remote_profile/keep_backup
 (Argument), 14
 key_filename:
 machine[SSHContext]/remote_profile/key_filename
 (Argument), 15
 kill() (dpdispatcher.base_context.BaseContext method), 49
 kill() (dpdispatcher.dp_cloud_server_context.BohriumContext method), 53
 kill() (dpdispatcher.hdfs_context.HDFSContext method), 56
 kill() (dpdispatcher.HDFSContext method), 27
 kill() (dpdispatcher.lazy_local_context.LazyLocalContext method), 57
 kill() (dpdispatcher.LazyLocalContext method), 31
 kill() (dpdispatcher.local_context.LocalContext method), 59
 kill() (dpdispatcher.LocalContext method), 33
 kill() (dpdispatcher.ssh_context.SSHContext method), 70
 kill() (dpdispatcher.SSHContext method), 39
 kwargs:
 resources[Bohrium]/kwargs (Argument), 19
 resources[DistributedShell]/kwargs (Argument), 19
 resources[LSF]/kwargs (Argument), 20
 resources[PBS]/kwargs (Argument), 19
 resources[Shell]/kwargs (Argument), 20
 resources[SlurmJobArray]/kwargs (Argument), 21
 resources[Slurm]/kwargs (Argument), 20
 resources[Torque]/kwargs (Argument), 20
 LazyLocalContext (class in dpdispatcher), 30
 LazyLocalContext (class in dpdispatcher.lazy_local_context), 56
 Lebesgue (in module dpdispatcher), 31
 Lebesgue (in module dpdispatcher.dp_cloud_server), 53
 LebesgueContext (in module dpdispatcher), 31
 LebesgueContext (in module dpdispatcher.dp_cloud_server_context), 54
 load_from_dict() (dpdispatcher.base_context.BaseContext class method), 49
 load_from_dict() (dpdispatcher.dp_cloud_server_context.BohriumContext class method), 53
 load_from_dict() (dpdispatcher.hdfs_context.HDFSContext class method), 56
 load_from_dict() (dpdispatcher.HDFSContext class method), 27
 load_from_dict() (dpdispatcher.lazy_local_context.LazyLocalContext class method), 57
 load_from_dict() (dpdispatcher.LazyLocalContext class method), 31
 load_from_dict() (dpdispatcher.local_context.LocalContext class method), 59
 load_from_dict() (dpdispatcher.LocalContext class method), 33
 load_from_dict() (dpdispatcher.Machine class method), 35
 load_from_dict() (dpdispatcher.machine.Machine class method), 62
 load_from_dict() (dpdispatcher.Resources class method), 37
 load_from_dict() (dpdispatcher.ssh_context.SSHContext class method), 70
 load_from_dict() (dpdispatcher.SSHContext class method), 39
 load_from_dict() (dpdispatcher.submission.Resources class method), 74
 load_from_dict() (dpdispatcher.submission.Task class method), 78
 load_from_dict() (dpdispatcher.Task class method), 45
 load_from_json() (dpdispatcher.Machine class method), 35
 load_from_json() (dpdispatcher.machine.Machine class method), 63
 load_from_json() (dpdispatcher.Resources class method), 37

[load_from_json\(\)](#) (*dpdispatcher.submission.Resources class method*), [74](#)
[load_from_json\(\)](#) (*dpdispatcher.submission.Task class method*), [78](#)
[load_from_json\(\)](#) (*dpdispatcher.Task class method*), [45](#)
[local_root:](#)
 [machine/local_root \(Argument\)](#), [13](#)
[LocalContext](#) (*class in dpdispatcher*), [32](#)
[LocalContext](#) (*class in dpdispatcher.local_context*), [58](#)
[look_for_keys:](#)
 [machine\[SSHContext\]/remote_profile/look_for_keys](#) (*Argument*), [16](#)
[LSF](#) (*class in dpdispatcher*), [29](#)
[LSF](#) (*class in dpdispatcher.lsf*), [60](#)

M

[machine \(Argument\)](#)
 [machine:](#), [13](#)
[Machine](#) (*class in dpdispatcher*), [33](#)
[Machine](#) (*class in dpdispatcher.machine*), [61](#)
[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*), [49](#)
[machine_subfields\(\)](#) (*dpdispatcher.base_context.BaseContext class method*), [50](#)
[machine_subfields\(\)](#) (*dpdispatcher.dp_cloud_server_context.BohriumContext class method*), [53](#)
[machine_subfields\(\)](#) (*dpdispatcher.ssh_context.SSHContext class method*), [70](#)
[machine_subfields\(\)](#) (*dpdispatcher.SSHContext class method*), [39](#)
[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:](#), [16](#)
[machine\[LazyLocalContext\]/remote_profile \(Argument\)](#)
 [remote_profile:](#), [14](#)
[machine\[LocalContext\]/remote_profile \(Argument\)](#)
 [remote_profile:](#), [14](#)
[machine\[SSHContext\]/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/otp_secret \(Argument\)](#)
 [otp_secret:](#), [15](#)
[machine\[SSHContext\]/remote_profile/username \(Argument\)](#)
 [username:](#), [15](#)
[main\(\)](#) (*in module dpdispatcher.dpdisp*), [54](#)

map_dp_job_state() (dpdispatcher.dp_cloud_server.Bohrium static method), 52

mkdir() (dpdispatcher.hdfs_cli.HDFS static method), 55

module

- dpdispatcher, 25
- dpdispatcher.arginfo, 49
- dpdispatcher.base_context, 49
- dpdispatcher.distributed_shell, 51
- dpdispatcher.dp_cloud_server, 52
- dpdispatcher.dp_cloud_server_context, 53
- dpdispatcher.dpcloudserver, 46
- dpdispatcher.dpcloudserver.client, 47
- dpdispatcher.dpcloudserver.config, 48
- dpdispatcher.dpcloudserver.retcode, 48
- dpdispatcher.dpcloudserver.zip_file, 48
- dpdispatcher.dpdisp, 54
- dpdispatcher.hdfs_cli, 54
- dpdispatcher.hdfs_context, 55
- dpdispatcher.JobStatus, 48
- dpdispatcher.lazy_local_context, 56
- dpdispatcher.local_context, 58
- dpdispatcher.lsf, 60
- dpdispatcher.machine, 61
- dpdispatcher.pbs, 63
- dpdispatcher.shell, 65
- dpdispatcher.slurm, 66
- dpdispatcher.ssh_context, 68
- dpdispatcher.submission, 71
- dpdispatcher.utils, 78

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), 55

N

NODATA (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 48

number_node:

- resources/number_node (Argument), 17

O

OK (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 48

options (dpdispatcher.base_context.BaseContext attribute), 50

options (dpdispatcher.Machine attribute), 35

options (dpdispatcher.machine.Machine attribute), 63

outlog:

- task/outlog (Argument), 23

P

para_deg:

- resources/para_deg (Argument), 18

PARAMERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 48

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), 35

PBS (class in dpdispatcher.pbs), 63

port:

- machine[SSHContext]/remote_profile/port (Argument), 15

post() (dpdispatcher.dpcloudserver.Client method), 46

post() (dpdispatcher.dpcloudserver.client.Client method), 47

prepend_script:

- resources/prepend_script (Argument), 19

program_id:

- machine[BohriumContext]/remote_profile/program_id (Argument), 14

put() (dpdispatcher.ssh_context.SSHSession method), 71

PWDERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 48

Q

queue_name:

- resources/queue_name (Argument), 17

R

ratio_unfinished:

- resources/strategy/ratio_unfinished (Argument), 18

read() (dpdispatcher.lazy_local_context.SPRetObj method), 58

read() (dpdispatcher.local_context.SPRetObj method), 60

read_file() (dpdispatcher.base_context.BaseContext method), 50

read_file() (dpdispatcher.dp_cloud_server_context.BohriumContext method), 54

read_file() (dpdispatcher.hdfs_context.HDFSContext method), 56

read_file() (dpdispatcher.HDFSContext method), 27

read_file() (dpdispatcher.lazy_local_context.LazyLocalContext method), 58

read_file() (dpdispatcher.LazyLocalContext method), 31

`read_file()` (*dpdispatcher.local_context.LocalContext* method), 59
`read_file()` (*dpdispatcher.LocalContext* method), 33
`read_file()` (*dpdispatcher.ssh_context.SSHContext* method), 70
`read_file()` (*dpdispatcher.SSHContext* method), 39
`read_hdfs_file()` (*dpdispatcher.hdfs_cli.HDFS* static method), 55
`read_home_file()` (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 54
`readlines()` (*dpdispatcher.lazy_local_context.SPRetObj* method), 58
`readlines()` (*dpdispatcher.local_context.SPRetObj* method), 60
`refresh_token()` (*dpdispatcher.dpcloudserver.Client* method), 46
`refresh_token()` (*dpdispatcher.dpcloudserver.client.Client* method), 47
`register_job_id()` (*dpdispatcher.Job* method), 28
`register_job_id()` (*dpdispatcher.submission.Job* method), 72
`register_task()` (*dpdispatcher.Submission* method), 43
`register_task()` (*dpdispatcher.submission.Submission* method), 76
`register_task_list()` (*dpdispatcher.Submission* method), 43
`register_task_list()` (*dpdispatcher.submission.Submission* method), 76
`remote` (*dpdispatcher.ssh_context.SSHSession* property), 71
`remote_profile:`
 `machine[BohriumContext]/remote_profile` (Argument), 14
 `machine[HDFSContext]/remote_profile` (Argument), 16
 `machine[LazyLocalContext]/remote_profile` (Argument), 14
 `machine[LocalContext]/remote_profile` (Argument), 14
 `machine[SSHContext]/remote_profile` (Argument), 15
`remote_root:`
 `machine/remote_root` (Argument), 13
`remove()` (*dpdispatcher.hdfs_cli.HDFS* static method), 55
`remove_unfinished_jobs()` (*dpdispatcher.Submission* method), 43
`remove_unfinished_jobs()` (*dpdispatcher.submission.Submission* method), 76
`REQERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 48
`RequestInfoException`, 47
`resources` (Argument)
 `resources:`, 17
`Resources` (class in *dpdispatcher*), 36
`Resources` (class in *dpdispatcher.submission*), 73
`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), 35
`resources_arginfo()` (*dpdispatcher.machine.Machine* class method), 63

resources_subfields() (*dpdispatcher.LSF class method*), 30

resources_subfields() (*dpdispatcher.lsf.LSF class method*), 61

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

resources_subfields() (*dpdispatcher.machine.Machine class method*), 63

resources_subfields() (*dpdispatcher.Slurm class method*), 41

resources_subfields() (*dpdispatcher.slurm.Slurm class method*), 67

resources[Bohrium]/kwargs (*Argument*)
kwargs:, 19

resources[DistributedShell]/kwargs (*Argument*)
kwargs:, 19

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:, 19

resources[Shell]/kwargs (*Argument*)
kwargs:, 20

resources[SlurmJobArray]/kwargs (*Argument*)
kwargs:, 21

resources[SlurmJobArray]/kwargs/custom_gpu_line (*Argument*)
custom_gpu_line:, 21

resources[Slurm]/kwargs (*Argument*)
kwargs:, 20

resources[Slurm]/kwargs/custom_gpu_line (*Argument*)
custom_gpu_line:, 21

resources[Torque]/kwargs (*Argument*)
kwargs:, 20

RETCODE (*class in dpdispatcher.dpcloudserver.retcodes*), 48

retry() (*in module dpdispatcher.utils*), 78

RetrySignal, 78

ROLEERR (*dpdispatcher.dpcloudserver.retcodes.RETCODE attribute*), 48

rsync() (*in module dpdispatcher.utils*), 79

rsync_available (*dpdis-*

patcher.ssh_context.SSHSession *property*), 71

run_cmd_with_all_output() (*in module dpdispatcher.utils*), 79

run_submission() (*dpdispatcher.Submission method*), 43

run_submission() (*dpdispatcher.submission.Submission method*), 76

running (*dpdispatcher.JobStatus.JobStatus attribute*), 49

S

serialize() (*dpdispatcher.Job method*), 28

serialize() (*dpdispatcher.Machine method*), 35

serialize() (*dpdispatcher.machine.Machine method*), 63

serialize() (*dpdispatcher.Resources method*), 37

serialize() (*dpdispatcher.Submission method*), 43

serialize() (*dpdispatcher.submission.Job method*), 72

serialize() (*dpdispatcher.submission.Resources method*), 74

serialize() (*dpdispatcher.submission.Submission method*), 76

serialize() (*dpdispatcher.submission.Task method*), 78

serialize() (*dpdispatcher.Task method*), 45

sftp (*dpdispatcher.ssh_context.SSHContext property*), 70

sftp (*dpdispatcher.ssh_context.SSHSession property*), 71

sftp (*dpdispatcher.SSHContext property*), 39

Shell (*class in dpdispatcher*), 39

Shell (*class in dpdispatcher.shell*), 65

Slurm (*class in dpdispatcher*), 40

Slurm (*class in dpdispatcher.slurm*), 66

SlurmJobArray (*class in dpdispatcher.slurm*), 67

source_list:
resources/source_list (*Argument*), 18

SPRetObj (*class in dpdispatcher.lazy_local_context*), 58

SPRetObj (*class in dpdispatcher.local_context*), 59

ssh (*dpdispatcher.ssh_context.SSHContext property*), 70

ssh (*dpdispatcher.SSHContext property*), 39

SSHContext (*class in dpdispatcher*), 37

SSHContext (*class in dpdispatcher.ssh_context*), 68

SSHSession (*class in dpdispatcher.ssh_context*), 70

strategy:
resources/strategy (*Argument*), 18

sub_script_cmd() (*dpdispatcher.LSF method*), 30

sub_script_cmd() (*dpdispatcher.lsf.LSF method*), 61

sub_script_cmd() (*dpdispatcher.Machine method*), 35

sub_script_cmd() (*dpdispatcher.machine.Machine method*), 63

sub_script_head() (*dpdispatcher.LSF method*), 30

sub_script_head() (*dpdispatcher.lsf.LSF method*), 61

sub_script_head() (*dpdispatcher.Machine* method), 35

sub_script_head() (*dpdispatcher.machine.Machine* method), 63

subclasses_dict (*dpdispatcher.base_context.BaseContext* attribute), 50

subclasses_dict (*dpdispatcher.Machine* attribute), 35

subclasses_dict (*dpdispatcher.machine.Machine* attribute), 63

Submission (class in *dpdispatcher*), 41

Submission (class in *dpdispatcher.submission*), 74

submission_from_json() (*dpdispatcher.Submission* class method), 44

submission_from_json() (*dpdispatcher.submission.Submission* class method), 76

submission_to_json() (*dpdispatcher.Submission* method), 44

submission_to_json() (*dpdispatcher.submission.Submission* method), 76

submit_job() (*dpdispatcher.Job* method), 29

submit_job() (*dpdispatcher.submission.Job* method), 72

T

tar_compress:
machine[SSHContext]/remote_profile/tar_compress (Argument), 16

task (Argument)
task:, 23

Task (class in *dpdispatcher*), 44

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

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), 49

THIRDERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 48

timeout:
machine[SSHContext]/remote_profile/timeout (Argument), 15

TOKENINVALID (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 48

Torque (class in *dpdispatcher*), 45

Torque (class in *dpdispatcher.pbs*), 64

totp_secret:
machine[SSHContext]/remote_profile/totp_secret (Argument), 15

try_recover_from_json() (*dpdispatcher.Submission* method), 44

try_recover_from_json() (*dpdispatcher.submission.Submission* method), 76

U

UNDERDEBUG (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 48

unknown (*dpdispatcher.JobStatus.JobStatus* attribute), 49

UNKOWNERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 48

unsubmitted (*dpdispatcher.JobStatus.JobStatus* attribute), 49

unzip_file() (in module *dpdispatcher.dpcloudserver.zip_file*), 48

update_submission_state() (*dpdispatcher.Submission* method), 44

update_submission_state() (*dpdispatcher.submission.Submission* method), 76

upload() (*dpdispatcher.base_context.BaseContext* method), 50

upload() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 54

upload() (*dpdispatcher.dpcloudserver.Client* method), 46

upload() (*dpdispatcher.dpcloudserver.client.Client* method), 47

upload() (*dpdispatcher.hdfs_context.HDFSContext* method), 56

upload() (*dpdispatcher.HDFSContext* method), 27

upload() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 58

upload() (*dpdispatcher.LazyLocalContext* method), 31

upload() (*dpdispatcher.local_context.LocalContext* method), 59

upload() (*dpdispatcher.LocalContext* method), 33

upload() (*dpdispatcher.ssh_context.SSHContext* method), 70

upload() (*dpdispatcher.SSHContext* method), 39

upload_job() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 54

upload_jobs() (*dpdispatcher.Submission* method), 44

`upload_jobs()` (*dpdispatcher.submission.Submission*
method), 77
`USERERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE*
attribute), 48
`username:`
 `machine[SSHContext]/remote_profile/username`
 (*Argument*), 15

V

`VERIFYERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE*
attribute), 48

W

`wait_time:`
 `resources/wait_time` (*Argument*), 19
`waiting` (*dpdispatcher.JobStatus.JobStatus attribute*), 49
`write_file()` (*dpdispatcher.base_context.BaseContext*
method), 50
`write_file()` (*dpdispatcher.dp_cloud_server_context.BohriumContext*
method), 54
`write_file()` (*dpdispatcher.hdfs_context.HDFSContext*
method), 56
`write_file()` (*dpdispatcher.HDFSContext method*), 27
`write_file()` (*dpdispatcher.lazy_local_context.LazyLocalContext*
method), 58
`write_file()` (*dpdispatcher.LazyLocalContext*
method), 31
`write_file()` (*dpdispatcher.local_context.LocalContext*
method), 59
`write_file()` (*dpdispatcher.LocalContext method*), 33
`write_file()` (*dpdispatcher.ssh_context.SSHContext*
method), 70
`write_file()` (*dpdispatcher.SSHContext method*), 39
`write_home_file()` (*dpdis-*
 patcher.dp_cloud_server_context.BohriumContext
 method), 54
`write_local_file()` (*dpdis-*
 patcher.dp_cloud_server_context.BohriumContext
 method), 54

Z

`zip_file_list()` (*in module dpdis-*
 patcher.dpcloudserver.zip_file), 48