DPDispatcher

Deep Modeling

Apr 18, 2024

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 . . . 10
- 3.5 HDFS . . . 10
- 3.6 OpenAPI . . . 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
- 4.8 Fugaku . . . 12
- 4.9 OpenAPI . . . 12
- 4.10 SGE . . . 13

5 Machine parameters **15**

6 Resources parameters **19**

7 Task parameters **25**

8 Command line interface **27**
- 8.1 Valid subcommands . . . 27
- 8.2 Sub-commands . . . 27

9 DPDispatcher API **29**
- 9.1 dpdispatcher package . . . 29

10 Running the DeePMD-kit on the Expanse cluster **85**

11 Running Gaussian 16 with failure allowed **87**

12 Running multiple MD tasks on a GPU workstation **89**

13 Customizing the submission script header 91

14 Authors 93

15 Indices and tables 95

Python Module Index 97

Index 99

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).

CHAPTER

ONE

INSTALL DPDISPATCHER

DPDispatcher can installed by `pip`:

```
pip install dpdispatcher
```

To add Bohrium support, execute

```
pip install dpdispatcher[bohrium]
```

CHAPTER

TWO

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": ["conf.lmp", "input.lammps"],
```

(continues on next page)

(continued from previous page)

```
  "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_DIVECES 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*.

CHAPTER

THREE

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.

Since `bash -l` is used in the shebang line of the submission scripts, the login shell startup files will be executed, potentially overriding the current environmental variables. Therefore, it's advisable to explicitly set the environmental variables using *envs* or *source_list*.

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.

Since `bash -l` is used in the shebang line of the submission scripts, the login shell startup files will be executed, potentially overriding the current environmental variables. Therefore, it's advisable to explicitly set the environmental variables using *envs* or *source_list*.

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 outside the submission script.

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.

3.6 OpenAPI

context_type: OpenAPI

OpenAPI is a new way to submit jobs to Bohrium. It using AccessKey instead of username and password. Read Bohrium documentation for details. To use OpenAPI, one needs to provide necessary parameters in *remote_profile*.

CHAPTER

FOUR

SUPPORTED BATCH JOB SYSTEMS

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

4.1 Bash

batch_type: `Shell`

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

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

4.2 Slurm

batch_type: `Slurm`, `SlurmJobArray`

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

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

4.3 OpenPBS or PBSPro

batch_type: `PBS`

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

Note that do not use `PBS` for Torque.

4.4 TORQUE

batch_type: `Torque`

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

4.5 LSF

batch_type: `LSF`

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

4.6 Bohrium

batch_type: `Bohrium`

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

4.7 DistributedShell

batch_type: `DistributedShell`

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

4.8 Fugaku

batch_type: `Fugaku`

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

Read Fujitsu cloud service documentation for details.

4.9 OpenAPI

batcy_type: `OpenAPI` OpenAPI is a new way to submit jobs to Bohrium. It using AccessKey instead of username and password. Read Bohrium documentation for details.

4.10 SGE

batch_type: SGE

The Sun Grid Engine (SGE) scheduler is a batch-queueing system distributed resource management. The commands and flags of SGE share a lot similarity with PBS except when checking job status. Use this argument if one is submitting job to SGE based batch system.

CHAPTER

FIVE

MACHINE PARAMETERS

Note: One can load, modify, and export the input file by using our effective web-based tool DP-GUI online or hosted using the *command line interface* `dpdisp 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: LSF, PBS, Torque, Slurm, Shell, Bohrium, Fugaku, OpenAPI, SGE, DistributedShell, SlurmJobArray

local_root:

type: `str` | `NoneType`

argument path: `machine/local_root`

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

remote_root:

type: `str` | `NoneType`, optional

argument path: `machine/remote_root`

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

clean_asynchronously:

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

argument path: `machine/clean_asynchronously`

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: `str` (flag key)

argument path: `machine/context_type`

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

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

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

remote_profile:

type: `dict`

argument path: `machine[SSHContext]/remote_profile`

The information used to maintain the connection with remote machine.

hostname:

type: `str`

argument path: `machine[SSHContext]/remote_profile/hostname`

hostname or ip of ssh connection.

username:

type: `str`

argument path: `machine[SSHContext]/remote_profile/username`

username of target linux system

password:

type: `str`, optional

argument path: `machine[SSHContext]/remote_profile/password`

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

port:

type: `int`, optional, default: `22`

argument path: `machine[SSHContext]/remote_profile/port`

ssh connection port.

key_filename:

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

argument path: `machine[SSHContext]/remote_profile/key_filename`

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

passphrase:

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

argument path: `machine[SSHContext]/remote_profile/passphrase`

passphrase of key used by ssh connection

timeout:

type: `int`, optional, default: `10`

argument path: `machine[SSHContext]/remote_profile/timeout`

timeout of ssh connection

totp_secret:

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

argument path: machine[SSHContext]/remote_profile/totp_secret

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

tar_compress:

type: bool, optional, default: True

argument path: machine[SSHContext]/remote_profile/tar_compress

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

look_for_keys:

type: bool, optional, default: True

argument path: machine[SSHContext]/remote_profile/look_for_keys

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

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

remote_profile:

type: dict, optional

argument path: machine[LocalContext]/remote_profile

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

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

remote_profile:

type: dict, optional

argument path: machine[HDFSContext]/remote_profile

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

When context_type is set to OpenAPIContext (or its aliases openapicontext, OpenAPI, openapi):

remote_profile:

type: dict, optional

argument path: machine[OpenAPIContext]/remote_profile

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

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

remote_profile:

type: dict, optional

argument path: machine[LazyLocalContext]/remote_profile

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

When context_type is set to `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`, optional
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

retry_count:

type: `NoneType | int`, optional, default: `2`
argument path:
`machine[BohriumContext]/remote_profile/retry_count`

The retry count when a job is terminated

ignore_exit_code:

type: `bool`, optional, default: `True`
argument path:
`machine[BohriumContext]/remote_profile/ignore_exit_code`

The job state will be marked as finished if the exit code is non-zero when set to True. Otherwise,
the job state will be designated as terminated.

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

CHAPTER

SIX

RESOURCES PARAMETERS

Note: One can load, modify, and export the input file by using our effective web-based tool DP-GUI online or hosted using the *command line interface* `dpdisp 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: typing.List[str], optional
argument path: resources/custom_flags

The extra lines pass to job submitting script header

strategy:

type: dict, optional
argument path: resources/strategy

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: bool, optional, default: False
argument path: resources/strategy/if_cuda_multi_devices

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

ratio_unfinished:

type: float, optional, default: 0.0
argument path: resources/strategy/ratio_unfinished

The ratio of *tasks* that can be unfinished.

customized_script_header_template_file:

type: str, optional
argument path: resources/strategy/
customized_script_header_template_file

The customized template file to generate job submitting script header, which overrides the default file.

para_deg:

type: int, optional, default: 1
argument path: resources/para_deg

Decide how many tasks will be run in parallel.

source_list:

type: typing.List[str], 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: typing.List[str], optional, default: []

argument path: `resources/module_unload_list`

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: `typing.List[str]`, 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: `typing.List[str]`, optional, default: `[]`
argument path: `resources/prepend_script`

Optional script run before jobs submitted.

append_script:

type: `typing.List[str]`, optional, default: `[]`
argument path: `resources/append_script`

Optional script run after jobs submitted.

wait_time:

type: `float` | `int`, optional, default: `0`
argument path: `resources/wait_time`

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

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

batch_type:

type: `str` (flag key)
argument path: `resources/batch_type`
possible choices: *Slurm*, *Bohrium*, *DistributedShell*, *OpenAPI*, *LSF*, *SGE*, *Torque*, *PBS*, *Fugaku*, *SlurmJobArray*, *Shell*

The batch job system type loaded from machine/batch_type.

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

kwargs:

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

Extra arguments.

custom_gpu_line:

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

Custom GPU configuration, starting with #SBATCH

When batch_type is set to `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 `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 `OpenAPI` (or its alias `openapi`):

kwargs:

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

This field is empty for this batch.

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

kwargs:

type: `dict`
argument path: `resources[LSF]/kwargs`

Extra arguments.

gpu_usage:

type: `bool`, optional, default: `False`
argument path: `resources[LSF]/kwargs/gpu_usage`

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: `bool`, optional, default: `False`
argument path: `resources[LSF]/kwargs/gpu_new_syntax`

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

gpu_exclusive:

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

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

custom_gpu_line:

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

Custom GPU configuration, starting with #BSUB

When batch_type is set to SGE (or its alias sge):

kwargs:

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

This field is empty for this batch.

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

kwargs:

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

This field is empty for this batch.

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

kwargs:

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

This field is empty for this batch.

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

kwargs:

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

Extra arguments.

custom_gpu_line:

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

Custom GPU configuration, starting with #SBATCH

slurm_job_size:

type: int, optional, default: 1
argument path:
resources[SlurmJobArray]/kwargs/slurm_job_size

Number of tasks in a Slurm job

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

kwargs:

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

This field is empty for this batch.

CHAPTER

SEVEN

TASK PARAMETERS

Note: One can load, modify, and export the input file by using our effective web-based tool DP-GUI online or hosted using the *command line interface* `dpdisp 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: `typing.List[str]`, optional, default: `[]`
argument path: `task/forward_files`

The files to be uploaded in task_work_path before the task exectued.

backward_files:

type: `typing.List[str]`, optional, default: `[]`
argument path: `task/backward_files`

The files to be download to local_root in task_work_path after the task finished

outlog:

type: `str` | `NoneType`, optional, default: `log`
argument path: `task/outlog`

The out log file name. redirect from stdout

errlog:

type: str | NoneType, optional, default: err
argument path: task/errlog

The err log file name. redirect from stderr

CHAPTER

EIGHT

COMMAND LINE INTERFACE

dpdispatcher: Generate HPC scheduler systems jobs input scripts, submit these scripts to HPC systems, and poke until they finish

```
usage: dpdisp [-h] {submission,gui} ...
```

8.1 Valid subcommands

command Possible choices: submission, gui

8.2 Sub-commands

8.2.1 submission

Handle terminated submission.

```
dpdisp submission [-h] [--download-terminated-log] [--download-finished-task]
                  [--clean] [--reset-fail-count]
                  SUBMISSION_HASH
```

Positional Arguments

SUBMISSION_HASH Submission hash to download.

Actions

One or more actions to take on submission.

--download-terminated-log Download log files of terminated tasks.

Default: False

--download-finished-task Download finished tasks.

Default: False

--clean Clean submission.

Default: False

--reset-fail-count Reset fail count of all jobs to zero.

Default: False

8.2.2 gui

Serve DP-GUI.

```
dpdisp gui [-h] [-p PORT] [--bind_all]
```

Named Arguments

-p, --port The port to serve DP-GUI on.

Default: 6042

--bind_all Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

Default: False

CHAPTER

NINE

DPDISPATCHER API

9.1 dpdispatcher package

class dpdispatcher.**Job**(*job_task_list*, ***, *resources*, *machine=None*)

Bases: object

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

Parameters

job_task_list
[list of Task] the tasks belonging to the job

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

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

Methods

deserialize(job_dict[, machine])	Convert the job_dict to a Submission class object.
get_job_state()	Get the jobs.
get_last_error_message()	Get last error message when the job is terminated.
serialize([if_static])	Convert the Task class instance to a dictionary.

get_hash
handle_unexpected_job_state
job_to_json
register_job_id
submit_job

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.

`get_last_error_message()` → str | None

Get last error message when the job is terminated.

`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 infomation (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.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 mainatin the connection with remote machine.

Methods

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

arginfo
bind_context
check_finish_tag
check_if_recover
check_status
default_resources
deserialize
gen_command_env_cuda_devices
gen_script
gen_script_command
gen_script_custom_flags_lines
gen_script_end
gen_script_env
gen_script_header
gen_script_run_command
gen_script_wait
load_from_dict
load_from_json
load_from_yaml
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_run_command(*job*)

gen_script_wait(*resources*)

get_exit_code(*job*)

Get exit code of the job.

Parameters

job
[Job] job

kill(*job*)

Kill the job.

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

Parameters

job
[Job] job

classmethod load_from_dict(*machine_dict*)

classmethod load_from_json(*json_path*)

classmethod load_from_yaml(*yaml_path*)

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

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

Bases: object

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

Parameters

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

cpu_per_node
[int] cpu numbers of each node.

gpu_per_node
[int] gpu numbers of each node.

queue_name
[str] The queue name of batch job scheduler system.

group_size
[int] The number of *tasks* in a *job*.

custom_flags
[list of Str] The extra lines pass to job submitting script header

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

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

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

customized_script_header_template_file
[str] The customized template file to generate job submitting script header, which overrides the default file.

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 waitting time in second after a single task submitted. Default: 0.

Methods

arginfo
deserialize
load_from_dict
load_from_json
load_from_yaml
serialize

`static arginfo`(*detail_kwargs=True*)

`classmethod deserialize`(*resources_dict*)

`classmethod load_from_dict`(*resources_dict*)

`classmethod load_from_json`(*json_file*)

`classmethod load_from_yaml`(*yaml_file*)

`serialize`()

`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

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

clean_jobs
download_jobs
get_hash
register_task
register_task_list
remove_unfinished_tasks
submission_from_json
submission_to_json
try_download_result
try_recover_from_json
upload_jobs

async async_run_submission(***kwargs*)

Async interface of run_submission.

Examples

```
>>> import asyncio
>>> from dpdispacher import Machine, Resource, Submission
>>> async def run_jobs():
...     backgroud_task = set()
...     # task1
...     task1 = Task(...)
...     submission1 = Submission(..., task_list=[task1])
...     background_task = asyncio.create_task(
...         submission1.async_run_submission(check_interval=2, clean=False)
...     )
...     # task2
...     task2 = Task(...)
...     submission2 = Submission(..., task_list=[task1])
...     background_task = asyncio.create_task(
...         submission2.async_run_submission(check_interval=2, clean=False)
...     )
...     background_tasks.add(background_task)
...     result = await asyncio.gather(*background_tasks)
...     return result
>>> run_jobs()
```

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

bind_machine(*machine*)

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

Parameters

machine
[Machine] the machine to bind with

check_all_finished()

Check whether all the jobs in the submission.

Notes

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

check_ratio_unfinished(*ratio_unfinished: float*) → bool

Calculate the ratio of unfinished tasks in the submission.

Parameters

ratio_unfinished
[float] the ratio of unfinished tasks in the submission

Returns

bool
whether the ratio of unfinished tasks in the submission is larger than ratio_unfinished

clean_jobs()

classmethod deserialize(*submission_dict*, *machine=None*)

Convert the submission_dict to a Submission class object.

Parameters

submission_dict
[dict] path-like, the base directory of the local tasks

machine
[Machine] Machine class Object to execute the jobs

Returns

submission
[Submission] the Submission class instance converted from the submission_dict

download_jobs()

generate_jobs()

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

get_hash()

handle_unexpected_submission_state()

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

register_task(*task*)

register_task_list(*task_list*)

remove_unfinished_tasks()

run_submission(*, *dry_run=False*, *exit_on_submit=False*, *clean=True*, *check_interval=30*)

Main method to execute the submission. First, check whether old Submission exists on the remote machine, and try to recover from it. Second, upload the local files to the remote machine where the tasks to be executed. Third, run the submission defined previously. Forth, wait until the tasks in the submission finished and download the result file to local directory. If dry_run is True, submission will be uploaded but not be executed and exit. If exit_on_submit is True, submission will exit.

serialize(*if_static=False*)

Convert the Submission class instance to a dictionary.

Parameters

if_static
[bool] whether dump the job runtime 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_download_result()

try_recover_from_json()

update_submission_state()

Check whether all the jobs in the submission.

Notes

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

upload_jobs()

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

Bases: object

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

Parameters

command
[Str] the command to be executed.

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

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

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

outlog
[Str] the filename to which command redirect stdout

errlog
[Str] the filename to which command redirect stderr

Methods

deserialize(task_dict)	Convert the task_dict to a Task class object.
get_task_state(context)	Get the task state by checking the tag file.

arginfo
get_hash
load_from_dict
load_from_json
load_from_yaml
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()

get_task_state(*context*)

Get the task state by checking the tag file.

Parameters

context
[Context] the context of the task

classmethod load_from_dict(*task_dict: dict*) → *Task*

classmethod load_from_json(*json_file*)

classmethod load_from_yaml(*yaml_file*)

serialize()

9.1.1 Subpackages

dpdispatcher.contexts package

Contexts.

Submodules

dpdispatcher.contexts.dp_cloud_server_context module

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

Bases: *BaseContext*

Methods

machine_arginfo()	Generate the machine arginfo.
machine_subfields()	Generate the machine subfields.

bind_submission
check_file_exists
check_finish
check_home_file_exits
clean
download
load_from_dict
read_file
read_home_file
upload
upload_job
write_file
write_home_file
write_local_file

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

bind_submission(*submission*)

check_file_exists(*fname*)

check_home_file_exits(*fname*)

clean()

download(*submission*)

classmethod **load_from_dict**(*context_dict*)

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

Generate the machine subfields.

Returns

list[Argument]
machine subfields

read_file(*fname*)

read_home_file(*fname*)

upload(*submission*)

upload_job(*job*, *common_files=None*)

write_file(*fname*, *write_str*)

write_home_file(*fname*, *write_str*)

write_local_file(*fname*, *write_str*)

dpdispatcher.contexts.dp_cloud_server_context.**DpCloudServerContext**

alias of *BohriumContext*

dpdispatcher.contexts.dp_cloud_server_context.**LebesgueContext**

alias of *BohriumContext*

dpdispatcher.contexts.hdfs_context module

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

Bases: *BaseContext*

Methods

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

bind_submission
check_finish
clean
get_job_root
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`()

`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*)

dpdispatcher.contexts.lazy_local_context module

`class dpdispatcher.contexts.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

machine_arginfo()	Generate the machine arginfo.
machine_subfields()	Generate the machine subfields.

bind_submission
block_call
block_checkcall
call
check_file_exists
check_finish
clean
download
get_job_root
get_return
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*)

classmethod load_from_dict(*context_dict*)

read_file(*fname*)

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

write_file(*fname*, *write_str*)

class dpdispatcher.contexts.lazy_local_context.**SPRetObj**(*ret*)

Bases: object

Methods

read
readlines

read()

readlines()

dpdispatcher.contexts.local_context module

class dpdispatcher.contexts.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

machine_arginfo()	Generate the machine arginfo.
machine_subfields()	Generate the machine subfields.

bind_submission
block_call
block_checkcall
call
check_file_exists
check_finish
clean
download
get_job_root
get_return
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*)

classmethod load_from_dict(*context_dict*)

read_file(*fname*)

upload(*submission*)

write_file(*fname*, *write_str*)

class dpdispatcher.contexts.local_context.**SPRetObj**(*ret*)

Bases: `object`

Methods

read
readlines

read()

readlines()

dpdispatcher.contexts.openapi_context module

class dpdispatcher.contexts.openapi_context.**OpenAPIContext**(**args*, ***kwargs*)

Bases: *BaseContext*

Methods

machine_arginfo()	Generate the machine arginfo.
machine_subfields()	Generate the machine subfields.

bind_submission
check_file_exists
check_finish
check_home_file_exits
clean
download
load_from_dict
read_file
read_home_file
upload
upload_job
write_file
write_home_file
write_local_file

bind_submission(*submission*)

check_file_exists(*fname*)

check_home_file_exits(*fname*)

clean()

download(*submission*)

classmethod load_from_dict(*context_dict*)

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.contexts.ssh_context module

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

Bases: *BaseContext*

Attributes

sftp
ssh

Methods

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

bind_submission
block_call
call
check_file_exists
check_finish
clean
close
download
get_job_root
get_return
list_remote_dir
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*)

list_remote_dir(*sftp*, *remote_dir*, *ref_remote_root*, *result_list*)

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.contexts.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.

dpdispatcher.dpcloudserver package

Submodules

dpdispatcher.dpcloudserver.client module

Provide backward compatbility with dflow.

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

Bases: Exception

dpdispatcher.entrypoints package

Entry points.

Submodules

dpdispatcher.entrypoints.gui module

DP-GUI entrypoint.

dpdispatcher.entrypoints.gui.**start_dpgui**(*, *port: int*, *bind_all: bool*, ***kwargs*)

Host DP-GUI server.

Parameters

port
[int] The port to serve DP-GUI on.

bind_all
[bool] Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

****kwargs**
additional arguments

Raises

ModuleNotFoundError
The dpgui package is not installed

dpdispatcher.entrypoints.submission module

dpdispatcher.entrypoints.submission.**handle_submission**(*, *submission_hash: str*, *download_terminated_log: bool = False*, *download_finished_task: bool = False*, *clean: bool = False*, *reset_fail_count: bool = False*)

Handle terminated submission.

Parameters

submission_hash
[str] Submission hash to download.

download_terminated_log
[bool, optional] Download log files of terminated tasks.

download_finished_task
[bool, optional] Download finished tasks.

clean
[bool, optional] Clean submission.

reset_fail_count
[bool, optional] Reset fail count of all jobs to zero.

Raises

ValueError
At least one action should be specified.

dpdispatcher.machines package

Machines.

Submodules

dpdispatcher.machines.distributed_shell module

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

Bases: *Machine*

Methods

do_submit(job)	Submit th job to yarn using distributed shell.
get_exit_code(job)	Get exit code of the job.
kill(job)	Kill the job.
resources_arginfo()	Generate the resources arginfo.
resources_subfields()	Generate the resources subfields.

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

check_finish_tag(*job*)

check_status(*job*)

do_submit(*job*)

Submit th job to yarn using distributed shell.

Parameters

job
[Job class instance] job to be submitted

Returns

job_id: string
submit process id

`gen_script_end`(*job*)

`gen_script_env`(*job*)

`gen_script_header`(*job*)

dpdispatcher.machines.dp_cloud_server module

`class` dpdispatcher.machines.dp_cloud_server.**Bohrium**(**args*, ***kwargs*)
Bases: *Machine*

Methods

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

arginfo
bind_context
check_finish_tag
check_if_recover
check_status
default_resources
deserialize
gen_command_env_cuda_devices
gen_local_script
gen_script
gen_script_command
gen_script_custom_flags_lines
gen_script_end
gen_script_env
gen_script_header
gen_script_run_command
gen_script_wait
load_from_dict
load_from_json
load_from_yaml
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*)

get_exit_code(*job*) → int

Get exit code of the job.

Parameters

job
[Job] job

kill(*job*)

Kill the job.

Parameters

job
[Job] job

static map_dp_job_state(*status*, *exit_code*, *ignore_exit_code=True*)

dpdispatcher.machines.dp_cloud_server.**DpCloudServer**

alias of *Bohrium*

dpdispatcher.machines.dp_cloud_server.**Lebesgue**

alias of *Bohrium*

dpdispatcher.machines.fugaku module

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

Bases: *Machine*

Methods

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

arginfo
bind_context
check_finish_tag
check_if_recover
check_status
default_resources
deserialize
gen_command_env_cuda_devices
gen_script
gen_script_command
gen_script_custom_flags_lines
gen_script_end
gen_script_env
gen_script_header
gen_script_run_command
gen_script_wait
load_from_dict
load_from_json
load_from_yaml
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*)

dpdispatcher.machines.lsf module

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

Bases: *Machine*

LSF batch.

Methods

default_resources(resources)	
get_exit_code(job)	Get exit code of the job.
kill(job)	Kill the job.
resources_arginfo()	Generate the resources arginfo.
resources_subfields()	Generate the resources subfields.

arginfo
bind_context
check_finish_tag
check_if_recover
check_status
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_run_command
gen_script_wait
load_from_dict
load_from_json
load_from_yaml
serialize
sub_script_cmd
sub_script_head

check_finish_tag(*job*)

check_status(***kwargs*)

default_resources(*resources*)

do_submit(***kwargs*)

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

gen_script(*job*)

gen_script_header(*job*)

kill(*job*)

Kill the job.

Parameters

job

[Job] job

classmethod resources_subfields() → List[Argument]

Generate the resources subfields.

Returns

list[Argument]
resources subfields

sub_script_cmd(*res*)

sub_script_head(*res*)

dpdispatcher.machines.openapi module

class dpdispatcher.machines.openapi.**OpenAPI**(**args*, ***kwargs*)

Bases: *Machine*

Methods

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

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

check_finish_tag(*job*)

check_if_recover(*submission*)

check_status(*job*)

do_submit(*job*)

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

gen_local_script(*job*)

gen_script(*job*)

gen_script_header(*job*)

get_exit_code(*job*)

Get exit code of the job.

Parameters

job
[Job] job

Returns

int
exit code

kill(*job*)

Kill the job.

Parameters

job
[Job] job

static map_dp_job_state(*status*, *exit_code*, *ignore_exit_code=True*)

dpdispatcher.machines.pbs module

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

Bases: *Machine*

Methods

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

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

check_finish_tag(*job*)

check_status(*job*)

default_resources(*resources*)

do_submit(*job*)

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

gen_script(*job*)

gen_script_header(*job*)

kill(*job*)

Kill the job.

Parameters

job

[Job] job

class dpdispatcher.machines.pbs.**SGE**(**args*, ***kwargs*)

Bases: *PBS*

Methods

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

arginfo
bind_context
check_finish_tag
check_if_recover
check_status
default_resources
deserialize
gen_command_env_cuda_devices
gen_script
gen_script_command
gen_script_custom_flags_lines
gen_script_end
gen_script_env
gen_script_header
gen_script_run_command
gen_script_wait
load_from_dict
load_from_json
load_from_yaml
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_header(*job*)

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

Bases: *PBS*

Methods

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

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

check_status(*job*)

gen_script_header(*job*)

dpdispatcher.machines.shell module

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

Bases: *Machine*

Methods

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

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

check_finish_tag(*job*)

check_status(*job*)

default_resources(*resources*)

do_submit(*job*)

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

gen_script(*job*)

gen_script_header(*job*)

kill(*job*)

Kill the job.

Parameters

job
[Job] job

dpdispatcher.machines.slurm module

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

Bases: *Machine*

Methods

get_exit_code(job)	Get exit code of the job.
kill(job)	Kill the job.
resources_arginfo()	Generate the resources arginfo.
resources_subfields()	Generate the resources subfields.

arginfo
bind_context
check_finish_tag
check_if_recover
check_status
default_resources
deserialize
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_run_command
gen_script_wait
load_from_dict
load_from_json
load_from_yaml
serialize
sub_script_cmd
sub_script_head

check_finish_tag(*job*)

check_status(***kwargs*)

default_resources(*resources*)

do_submit(***kwargs*)

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

gen_script(*job*)

gen_script_header(*job*)

kill(*job*)

Kill the job.

Parameters

job
[Job] job

classmethod resources_subfields() → List[Argument]

Generate the resources subfields.

Returns

list[Argument]
resources subfields

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

Bases: *Slurm*

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

Methods

get_exit_code(job)	Get exit code of the job.
kill(job)	Kill the job.
resources_arginfo()	Generate the resources arginfo.
resources_subfields()	Generate the resources subfields.

arginfo
bind_context
check_finish_tag
check_if_recover
check_status
default_resources
deserialize
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_run_command
gen_script_wait
load_from_dict
load_from_json
load_from_yaml
serialize
sub_script_cmd
sub_script_head

`check_finish_tag`(*job*)

`check_status`(***kwargs*)

`gen_script_command`(*job*)

`gen_script_end`(*job*)

`gen_script_header`(*job*)

`classmethod resources_subfields`() → List[Argument]

Generate the resources subfields.

Returns

list[Argument]
resources subfields

dpdispatcher.utils package

Utils.

Subpackages

dpdispatcher.utils.dpcloudserver package

class dpdispatcher.utils.dpcloudserver.**Client**(*email=None*, *password=None*, *debug=False*, *ticket=None*, *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
kill
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*)

kill(*job_id*)

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.utils.dpcloudserver.client module

class dpdispatcher.utils.dpcloudserver.client.**Client**(*email=None*, *password=None*, *debug=False*, *ticket=None*, *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
kill
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*)

kill(*job_id*)

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.utils.dpcloudserver.client.**RequestInfoException**

Bases: `Exception`

dpdispatcher.utils.dpcloudserver.config module

dpdispatcher.utils.dpcloudserver.retcode module

class dpdispatcher.utils.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.utils.dpcloudserver.zip_file module

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

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

Submodules

dpdispatcher.utils.hdfs_cli module

class dpdispatcher.utils.hdfs_cli.**HDFS**

Bases: object

Fundamental class for HDFS basic manipulation.

Methods

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

copy_to_local
move
read_hdfs_file

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.

dpdispatcher.utils.job_status module

class dpdispatcher.utils.job_status.**JobStatus**(*value*)

Bases: IntEnum

An enumeration.

completing = 6

finished = 5

running = 3

terminated = 4

unknown = 100

unsubmitted = 1

waiting = 2

dpdispatcher.utils.record module

dpdispatcher.utils.utils module

exception dpdispatcher.utils.utils.**RetrySignal**

Bases: Exception

Exception to give a signal to retry the function.

dpdispatcher.utils.utils.**customized_script_header_template**(*filename: PathLike*, *resources:* Resources) → str

dpdispatcher.utils.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.utils.**get_sha256**(*filename*)

Get sha256 of a file.

Parameters

filename
[str] The filename.

Returns

sha256: str
The sha256.

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

dpdispatcher.utils.utils.**retry**(*max_retry: int = 3*, *sleep: int | float = 60*, *catch_exception: ~typing.Type[BaseException] = <class 'dpdispatcher.utils.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.utils.**rsync**(*from_file: str*, *to_file: str*, *port: int = 22*, *key_filename: str | None = None*, *timeout: 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.utils.**run_cmd_with_all_output**(*cmd*, *shell=True*)

9.1.2 Submodules

9.1.3 dpdispatcher.arginfo module

9.1.4 dpdispatcher.base_context module

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

Bases: object

Methods

machine_arginfo()	Generate the machine arginfo.
machine_subfields()	Generate the machine subfields.

bind_submission
check_finish
clean
download
load_from_dict
read_file
upload
write_file

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

bind_submission(*submission*)

check_finish(*proc*)

abstract clean()

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

classmethod load_from_dict(*context_dict*)

classmethod machine_arginfo() → Argument

Generate the machine arginfo.

Returns

Argument
machine arginfo

classmethod machine_subfields() → List[Argument]

Generate the machine subfields.

Returns

list[Argument]
machine subfields

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

abstract read_file(*fname*)

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

abstract upload(*submission*)

abstract write_file(*fname*, *write_str*)

9.1.5 dpdispatcher.dlog module

9.1.6 dpdispatcher.dpdisp module

dpdispatcher.dpdisp.**main**()

dpdispatcher.dpdisp.**main_parser**() → ArgumentParser

Dpdispatcher commandline options argument parser.

Returns

argparse.ArgumentParser
the argument parser

Notes

This function is used by documentation.

dpdispatcher.dpdisp.**parse_args**(*args: List[str] | None = None*)

Dpdispatcher commandline options argument parsing.

Parameters

args
[List[str]] list of command line arguments, main purpose is testing default option None takes arguments from sys.argv

9.1.7 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 mainatin the connection with remote machine.

Methods

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

arginfo
bind_context
check_finish_tag
check_if_recover
check_status
default_resources
deserialize
gen_command_env_cuda_devices
gen_script
gen_script_command
gen_script_custom_flags_lines
gen_script_end
gen_script_env
gen_script_header
gen_script_run_command
gen_script_wait
load_from_dict
load_from_json
load_from_yaml
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_run_command(*job*)

gen_script_wait(*resources*)

get_exit_code(*job*)

Get exit code of the job.

Parameters

job
[Job] job

kill(*job*)

Kill the job.

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

Parameters

job
[Job] job

classmethod load_from_dict(*machine_dict*)

classmethod load_from_json(*json_path*)

classmethod load_from_yaml(*yaml_path*)

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

9.1.8 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

deserialize(job_dict[, machine])	Convert the job_dict to a Submission class object.
get_job_state()	Get the jobs.
get_last_error_message()	Get last error message when the job is terminated.
serialize([if_static])	Convert the Task class instance to a dictionary.

get_hash
handle_unexpected_job_state
job_to_json
register_job_id
submit_job

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.

get_last_error_message() → str | None

Get last error message when the job is terminated.

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 infomation (job_id, job_state, fail_count, job_uuid etc.) to the dictionary.

Returns

task_dict
[dict] the dictionary converted from the Task class instance

`submit_job()`

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

Bases: `object`

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

Parameters

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

cpu_per_node
[int] cpu numbers of each node.

gpu_per_node
[int] gpu numbers of each node.

queue_name
[str] The queue name of batch job scheduler system.

group_size
[int] The number of *tasks* in a *job*.

custom_flags
[list of Str] The extra lines pass to job submitting script header

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

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

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

customized_script_header_template_file
[str] The customized template file to generate job submitting script header, which overrides the default file.

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 waitting time in second after a single task submitted. Default: 0.

Methods

arginfo
deserialize
load_from_dict
load_from_json
load_from_yaml
serialize

static arginfo(*detail_kwargs=True*)

classmethod deserialize(*resources_dict*)

classmethod load_from_dict(*resources_dict*)

classmethod load_from_json(*json_file*)

classmethod load_from_yaml(*yaml_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

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

clean_jobs
download_jobs
get_hash
register_task
register_task_list
remove_unfinished_tasks
submission_from_json
submission_to_json
try_download_result
try_recover_from_json
upload_jobs

async async_run_submission(***kwargs*)

Async interface of run_submission.

Examples

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

(continues on next page)

(continued from previous page)

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

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

bind_machine(*machine*)

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

Parameters

machine
[Machine] the machine to bind with

check_all_finished()

Check whether all the jobs in the submission.

Notes

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

check_ratio_unfinished(*ratio_unfinished: float*) → bool

Calculate the ratio of unfinished tasks in the submission.

Parameters

ratio_unfinished
[float] the ratio of unfinished tasks in the submission

Returns

bool
whether the ratio of unfinished tasks in the submission is larger than ratio_unfinished

clean_jobs()

classmethod deserialize(*submission_dict*, *machine=None*)

Convert the submission_dict to a Submission class object.

Parameters

submission_dict
[dict] path-like, the base directory of the local tasks

machine
[Machine] Machine class Object to execute the jobs

Returns

submission
[Submission] the Submission class instance converted from the submission_dict

download_jobs()

generate_jobs()

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

get_hash()

handle_unexpected_submission_state()

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

register_task(*task*)

register_task_list(*task_list*)

remove_unfinished_tasks()

run_submission(*, *dry_run=False*, *exit_on_submit=False*, *clean=True*, *check_interval=30*)

Main method to execute the submission. First, check whether old Submission exists on the remote machine, and try to recover from it. Second, upload the local files to the remote machine where the tasks to be executed. Third, run the submission defined previously. Forth, wait until the tasks in the submission finished and download the result file to local directory. If dry_run is True, submission will be uploaded but not be executed and exit. If exit_on_submit is True, submission will exit.

serialize(*if_static=False*)

Convert the Submission class instance to a dictionary.

Parameters

if_static
[bool] whether dump the job runtime 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_download_result()

try_recover_from_json()

update_submission_state()

Check whether all the jobs in the submission.

Notes

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

`upload_jobs()`

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

Bases: `object`

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

Parameters

command
[Str] the command to be executed.

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

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

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

outlog
[Str] the filename to which command redirect stdout

errlog
[Str] the filename to which command redirect stderr

Methods

deserialize(task_dict)	Convert the task_dict to a Task class object.
get_task_state(context)	Get the task state by checking the tag file.

arginfo
get_hash
load_from_dict
load_from_json
load_from_yaml
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`()

`get_task_state`(*context*)

Get the task state by checking the tag file.

Parameters

context
[Context] the context of the task

`classmethod load_from_dict`(*task_dict: dict*) → *Task*

`classmethod load_from_json`(*json_file*)

`classmethod load_from_yaml`(*yaml_file*)

`serialize`()

CHAPTER

TEN

RUNNING THE DEEPMD-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 metioned that we do not recommend to use the password, so SSH keys are used instead to improve security.

```
{
  "batch_type": "Slurm",
  "local_root": "./",
  "remote_root": "/expanse/lustre/scratch/njzjz/temp_project/dpgen_workdir",
  "clean_asynchronously": true,
  "context_type": "SSHContext",
  "remote_profile": {
    "hostname": "login.expanse.sdsc.edu",
    "username": "njzjz",
    "port": 22
  }
}
```

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.

```
{
  "number_node": 1,
  "cpu_per_node": 1,
  "gpu_per_node": 0,
  "queue_name": "shared",
  "group_size": 1,
  "custom_flags": [
    "#SBATCH -c 32",
    "#SBATCH --mem=16G",
    "#SBATCH --time=48:00:00",
    "#SBATCH --account=rut149",
    "#SBATCH --requeue"
  ],
  "source_list": [
    "activate /home/njzjz/deepmd-kit"
  ],
  "envs": {
```

(continues on next page)

(continued from previous page)

```
    "OMP_NUM_THREADS": 4,
    "TF_INTRA_OP_PARALLELISM_THREADS": 4,
    "TF_INTER_OP_PARALLELISM_THREADS": 8,
    "DP_AUTO_PARALLELIZATION": 1
  },
  "batch_type": "Slurm",
  "kwargs": {
    "custom_gpu_line": "#SBATCH --gpus=0"
  }
}
```

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.

```
{
    "command": "dp train input.json && dp freeze && dp compress",
    "task_work_path": "model1/",
    "forward_files": [
      "input.json"
    ],
    "backward_files": [
      "frozen_model.pb",
      "frozen_model_compressed.pb"
    ],
    "outlog": "log",
    "errlog": "err"
}
```

CHAPTER

ELEVEN

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:

```
{
    "command": "g16 < input > output ||:",
    "task_work_path": "p1/",
    "forward_files": [
      "input"
    ],
    "backward_files": [
      "output"
    ]
}
```

This command ensures the task will always provide zero code.

CHAPTER

TWELVE

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*.

```
{
  "batch_type": "Shell",
  "local_root": "./",
  "remote_root": "/data2/jinzhe/dpgen_workdir",
  "clean_asynchronously": true,
  "context_type": "SSHContext",
  "remote_profile": {
    "hostname": "mandu.iqb.rutgers.edu",
    "username": "jz748",
    "port": 22
  }
}
```

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.

```
{
  "number_node": 1,
  "cpu_per_node": 48,
  "gpu_per_node": 8,
  "queue_name": "shell",
  "group_size": 0,
  "strategy": {
    "if_cuda_multi_devices": true
  },
  "source_list": [
    "activate /home/jz748/deepmd-kit"
  ],
  "envs": {
    "OMP_NUM_THREADS": 1,
    "TF_INTRA_OP_PARALLELISM_THREADS": 1,
    "TF_INTER_OP_PARALLELISM_THREADS": 1
  },
  "para_deg": 6
}
```

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.

CHAPTER

THIRTEEN

CUSTOMIZING THE SUBMISSION SCRIPT HEADER

When submitting jobs to some clusters, such as the Tiger Cluster at Princeton University, the Slurm header is quite different from the standard one. In this case, DPDispatcher allows users to customize the templates by setting *strategy/customized_script_header_template_file* to a template file:

```
{
  "number_node": 1,
  "cpu_per_node": 32,
  "kwargs":{
    "qos": "tiger-vshort"
  },
  "source_list": ["activate abacus_env"],
  "strategy": {
    "customized_script_header_template_file": "./template.slurm"
  },
  "group_size": 2000
}
```

`template.slurm` is the template file, where `str.format()` is used to format the template with *Resources Parameters*:

```
#!/bin/bash -l
#SBATCH --parsable
#SBATCH --nodes={number_node}
#SBATCH --ntasks-per-node={cpu_per_node}
#SBATCH --qos={kwargs[qos]}
#SBATCH --time=01:02:00
#SBATCH --mem-per-cpu=4G
```

See Python Format String Syntax for how to insert parameters inside the template.

CHAPTER

FOURTEEN

AUTHORS

- A bot of @njzjz
- AnguseZhang
- Byron
- Cloudac7
- Feifei Tian
- Feiyang472
- Franklalalala
- Futaki Haduki
- Futaki Hatsuki
- GravityPHY
- Han Wang
- Han Y.B
- HuangJiameng
- Jinzhe Zeng
- KZHIWEI
- Levi Zhou
- PKUfjh
- Pengchao Zhang
- Tongqi Wen
- TongqiWen
- Xiaoshan Luo
- Xuanyan Chen
- Yifan Li
- Yixiao Chen
- Yongbin Zhuang
- Yuan Fengbo
- Yuan Fengbo ()

- Yunpei Liu
- Zhang Yaotang
- Zhengju Sha
- Zhiwei Zhang
- chenglab
- ck
- dependabot[bot]
- dingzhaohan
- dinngzhaohan
- felix5572
- haidi
- likefallwind
- luobangkui
- pre-commit-ci[bot]
- robinzyb
- saltball
- shazj99
- tuoping
- unknown
- wangxiangfei
- yuzhi
- zhangbei07
- zhaohan
- zjgemi

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

dpdispatcher, 29
dpdispatcher.arginfo, 70
dpdispatcher.base_context, 70
dpdispatcher.contexts, 39
dpdispatcher.contexts.dp_cloud_server_context, 39
dpdispatcher.contexts.hdfs_context, 41
dpdispatcher.contexts.lazy_local_context, 42
dpdispatcher.contexts.local_context, 44
dpdispatcher.contexts.openapi_context, 45
dpdispatcher.contexts.ssh_context, 46
dpdispatcher.dlog, 73
dpdispatcher.dpcloudserver, 49
dpdispatcher.dpcloudserver.client, 49
dpdispatcher.dpdisp, 73
dpdispatcher.entrypoints, 50
dpdispatcher.entrypoints.gui, 50
dpdispatcher.entrypoints.submission, 50
dpdispatcher.machine, 73
dpdispatcher.machines, 51
dpdispatcher.machines.distributed_shell, 51
dpdispatcher.machines.dp_cloud_server, 52
dpdispatcher.machines.fugaku, 53
dpdispatcher.machines.lsf, 54
dpdispatcher.machines.openapi, 56
dpdispatcher.machines.pbs, 57
dpdispatcher.machines.shell, 61
dpdispatcher.machines.slurm, 62
dpdispatcher.submission, 76
dpdispatcher.utils, 64
dpdispatcher.utils.dpcloudserver, 65
dpdispatcher.utils.dpcloudserver.client, 66
dpdispatcher.utils.dpcloudserver.config, 67
dpdispatcher.utils.dpcloudserver.retcode, 67
dpdispatcher.utils.dpcloudserver.zip_file, 67
dpdispatcher.utils.hdfs_cli, 67
dpdispatcher.utils.job_status, 68
dpdispatcher.utils.record, 69
dpdispatcher.utils.utils, 69

INDEX

A

alias (*dpdispatcher.base_context.BaseContext attribute*), 71
alias (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext attribute*), 40
alias (*dpdispatcher.Machine attribute*), 31
alias (*dpdispatcher.machine.Machine attribute*), 74
alias (*dpdispatcher.machines.dp_cloud_server.Bohrium attribute*), 53
append_script:
    resources/append_script (*Argument*), **21**
arginfo() (*dpdispatcher.contexts.ssh_context.SSHSession static method*), 48
arginfo() (*dpdispatcher.Machine class method*), 31
arginfo() (*dpdispatcher.machine.Machine class method*), 74
arginfo() (*dpdispatcher.Resources static method*), 34
arginfo() (*dpdispatcher.submission.Resources static method*), 79
arginfo() (*dpdispatcher.submission.Task static method*), 83
arginfo() (*dpdispatcher.Task static method*), 38
async_run_submission() (*dpdispatcher.Submission method*), 35
async_run_submission() (*dpdispatcher.submission.Submission method*), 80

B

backward_files:
    task/backward_files (*Argument*), **25**
BaseContext (*class in dpdispatcher.base_context*), 70
batch_type:
    machine/batch_type (*Argument*), **15**
    resources/batch_type (*Argument*), **21**
bind_context() (*dpdispatcher.Machine method*), 31
bind_context() (*dpdispatcher.machine.Machine method*), 74
bind_machine() (*dpdispatcher.Submission method*), 36
bind_machine() (*dpdispatcher.submission.Submission method*), 81
bind_submission() (*dpdispatcher.base_context.BaseContext method*), 71
bind_submission() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
bind_submission() (*dpdispatcher.contexts.hdfs_context.HDFSContext method*), 41
bind_submission() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
bind_submission() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
bind_submission() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
bind_submission() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 47
block_call() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContex method*), 43
block_call() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
block_call() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 47
block_checkcall() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
block_checkcall() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
block_checkcall() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 47
Bohrium (*class in dpdispatcher.machines.dp_cloud_server*), 52
BohriumContext (*class in dpdispatcher.contexts.dp_cloud_server_context*), 39

C

call() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
call() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
call() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 47
check_all_finished() (*dpdispatcher.Submission method*), 36
check_all_finished() (*dpdispatcher.submission.Submission method*), 81
check_file_exists() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
check_file_exists() (*dpdispatcher.contexts.hdfs_context.HDFSContext method*), 41
check_file_exists() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
check_file_exists() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
check_file_exists() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
check_file_exists() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 47
check_finish() (*dpdispatcher.base_context.BaseContext method*), 71
check_finish() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
check_finish() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
check_finish() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 47
check_finish_tag() (*dpdispatcher.Machine method*), 31
check_finish_tag() (*dpdispatcher.machine.Machine method*), 74
check_finish_tag() (*dpdispatcher.machines.distributed_shell.DistributedShell method*), 51
check_finish_tag() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
check_finish_tag() (*dpdispatcher.machines.fugaku.Fugaku method*), 54
check_finish_tag() (*dpdispatcher.machines.lsf.LSF method*), 55
check_finish_tag() (*dpdispatcher.machines.openapi.OpenAPI method*), 56
check_finish_tag() (*dpdispatcher.machines.pbs.PBS method*), 58
check_finish_tag() (*dpdispatcher.machines.pbs.SGE method*), 59
check_finish_tag() (*dpdispatcher.machines.shell.Shell method*), 61
check_finish_tag() (*dpdispatcher.machines.slurm.Slurm method*), 62
check_finish_tag() (*dpdispatcher.machines.slurm.SlurmJobArray method*), 64
check_home_file_exits() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
check_home_file_exits() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
check_if_recover() (*dpdispatcher.Machine method*), 31
check_if_recover() (*dpdispatcher.machine.Machine method*), 74
check_if_recover() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
check_if_recover() (*dpdispatcher.machines.openapi.OpenAPI method*), 57
check_ratio_unfinished() (*dpdispatcher.Submission method*), 36
check_ratio_unfinished() (*dpdispatcher.submission.Submission method*), 81
check_status() (*dpdispatcher.Machine method*), 31
check_status() (*dpdispatcher.machine.Machine method*), 74
check_status() (*dpdispatcher.machines.distributed_shell.DistributedShell method*), 51
check_status() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
check_status() (*dpdispatcher.machines.fugaku.Fugaku method*), 54
check_status() (*dpdispatcher.machines.lsf.LSF method*), 55
check_status() (*dpdis-*

patcher.machines.openapi.OpenAPI method), 57
check_status() (dpdispatcher.machines.pbs.PBS method), 58
check_status() (dpdispatcher.machines.pbs.SGE method), 59
check_status() (dpdispatcher.machines.pbs.Torque method), 60
check_status() (dpdispatcher.machines.shell.Shell method), 61
check_status() (dpdispatcher.machines.slurm.Slurm method), 62
check_status() (dpdispatcher.machines.slurm.SlurmJobArray method), 64
clean() (dpdispatcher.base_context.BaseContext method), 71
clean() (dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method), 40
clean() (dpdispatcher.contexts.hdfs_context.HDFSContext method), 41
clean() (dpdispatcher.contexts.lazy_local_context.LazyLocalContext method), 43
clean() (dpdispatcher.contexts.local_context.LocalContext method), 45
clean() (dpdispatcher.contexts.openapi_context.OpenAPIContext method), 46
clean() (dpdispatcher.contexts.ssh_context.SSHContext method), 47
clean_asynchronously:
machine/clean_asynchronously (Argument), 15
clean_jobs() (dpdispatcher.Submission method), 36
clean_jobs() (dpdispatcher.submission.Submission method), 81
Client (class in dpdispatcher.utils.dpcloudserver), 65
Client (class in dpdispatcher.utils.dpcloudserver.client), 66
close() (dpdispatcher.contexts.ssh_context.SSHContext method), 47
close() (dpdispatcher.contexts.ssh_context.SSHSession method), 49
command:
task/command (Argument), 25
completing (dpdispatcher.utils.job_status.JobStatus attribute), 68
context_type:
machine/context_type (Argument), 15
copy_from_local() (dpdispatcher.utils.hdfs_cli.HDFS static method), 68
copy_to_local() (dpdispatcher.utils.hdfs_cli.HDFS static method), 68
cpu_per_node:
resources/cpu_per_node (Argument), 19
custom_flags:
resources/custom_flags (Argument), 19
custom_gpu_line:
resources[LSF]/kwargs/custom_gpu_line (Argument), 22
resources[SlurmJobArray]/kwargs/custom_gpu_line (Argument), 23
resources[Slurm]/kwargs/custom_gpu_line (Argument), 21
customized_script_header_template() (in module dpdispatcher.utils.utils), 69
customized_script_header_template_file:
resources/strategy/customized_script_header_template_f (Argument), 20

D

DATAERR (dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute), 67
DBERR (dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute), 67
default_resources() (dpdispatcher.Machine method), 31
default_resources() (dpdispatcher.machine.Machine method), 74
default_resources() (dpdispatcher.machines.fugaku.Fugaku method), 54
default_resources() (dpdispatcher.machines.lsf.LSF method), 55
default_resources() (dpdispatcher.machines.pbs.PBS method), 58
default_resources() (dpdispatcher.machines.pbs.SGE method), 59
default_resources() (dpdispatcher.machines.shell.Shell method), 61
default_resources() (dpdispatcher.machines.slurm.Slurm method), 62
deserialize() (dpdispatcher.Job class method), 29
deserialize() (dpdispatcher.Machine class method), 31
deserialize() (dpdispatcher.machine.Machine class method), 74
deserialize() (dpdispatcher.Resources class method), 34
deserialize() (dpdispatcher.Submission class method), 36
deserialize() (dpdispatcher.submission.Job class method), 77
deserialize() (dpdispatcher.submission.Resources class method), 79
deserialize() (dpdispatcher.submission.Submission class method), 81
deserialize() (dpdispatcher.submission.Task class method), 83

deserialize() (*dpdispatcher.Task class method*), 38
DistributedShell (*class in dpdispatcher.machines.distributed_shell*), 51
do_submit() (*dpdispatcher.Machine method*), 31
do_submit() (*dpdispatcher.machine.Machine method*), 74
do_submit() (*dpdispatcher.machines.distributed_shell.DistributedShell method*), 51
do_submit() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
do_submit() (*dpdispatcher.machines.fugaku.Fugaku method*), 54
do_submit() (*dpdispatcher.machines.lsf.LSF method*), 55
do_submit() (*dpdispatcher.machines.openapi.OpenAPI method*), 57
do_submit() (*dpdispatcher.machines.pbs.PBS method*), 58
do_submit() (*dpdispatcher.machines.pbs.SGE method*), 59
do_submit() (*dpdispatcher.machines.shell.Shell method*), 61
do_submit() (*dpdispatcher.machines.slurm.Slurm method*), 62
download() (*dpdispatcher.base_context.BaseContext method*), 71
download() (*dpdispatcher.contexts.dp_cloud_server_context.DpCloudServerContext method*), 40
download() (*dpdispatcher.contexts.hdfs_context.HDFSContext method*), 41
download() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
download() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
download() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
download() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 47
download() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
download() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
download_from_url() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
download_from_url() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
download_jobs() (*dpdispatcher.Submission method*), 37
download_jobs() (*dpdispatcher.submission.Submission method*), 81
DpCloudServer (*in module dpdispatcher.machines.dp_cloud_server*), 53
DpCloudServerContext (*in module dpdispatcher.contexts.dp_cloud_server_context*), 40
dpdispatcher
    module, 29
dpdispatcher.arginfo
    module, 70
dpdispatcher.base_context
    module, 70
dpdispatcher.contexts
    module, 39
dpdispatcher.contexts.dp_cloud_server_context
    module, 39
dpdispatcher.contexts.hdfs_context
    module, 41
dpdispatcher.contexts.lazy_local_context
    module, 42
dpdispatcher.contexts.local_context
    module, 44
dpdispatcher.contexts.openapi_context
    module, 45
dpdispatcher.contexts.ssh_context
    module, 46
dpdispatcher.dlog
    module, 73
dpdispatcher.dpcloudserver
    module, 49
dpdispatcher.dpcloudserver.client
    module, 49
dpdispatcher.dpdisp
    module, 73
dpdispatcher.entrypoints
    module, 50
dpdispatcher.entrypoints.gui
    module, 50
dpdispatcher.entrypoints.submission
    module, 50
dpdispatcher.machine
    module, 73
dpdispatcher.machines
    module, 51
dpdispatcher.machines.distributed_shell
    module, 51
dpdispatcher.machines.dp_cloud_server
    module, 52
dpdispatcher.machines.fugaku
    module, 53
dpdispatcher.machines.lsf
    module, 54
dpdispatcher.machines.openapi
    module, 56
dpdispatcher.machines.pbs
    module, 57

dpdispatcher.machines.shell
    module, 61
dpdispatcher.machines.slurm
    module, 62
dpdispatcher.submission
    module, 76
dpdispatcher.utils
    module, 64
dpdispatcher.utils.dpcloudserver
    module, 65
dpdispatcher.utils.dpcloudserver.client
    module, 66
dpdispatcher.utils.dpcloudserver.config
    module, 67
dpdispatcher.utils.dpcloudserver.retcode
    module, 67
dpdispatcher.utils.dpcloudserver.zip_file
    module, 67
dpdispatcher.utils.hdfs_cli
    module, 67
dpdispatcher.utils.job_status
    module, 68
dpdispatcher.utils.record
    module, 69
dpdispatcher.utils.utils
    module, 69

E

email:
    machine[BohriumContext]/remote_profile/email (*Argument*), **18**
ensure_alive() (*dpdispatcher.contexts.ssh_context.SSHSession method*), 49
envs:
    resources/envs (*Argument*), **21**
errlog:
    task/errlog (*Argument*), **25**
exec_command() (*dpdispatcher.contexts.ssh_context.SSHSession method*), 49
exists() (*dpdispatcher.utils.hdfs_cli.HDFS static method*), 68

F

finished (*dpdispatcher.utils.job_status.JobStatus attribute*), 68
forward_files:
    task/forward_files (*Argument*), **25**
Fugaku (*class in dpdispatcher.machines.fugaku*), 53

G

gen_command_env_cuda_devices() (*dpdispatcher.Machine method*), 31
gen_command_env_cuda_devices() (*dpdispatcher.machine.Machine method*), 74
gen_local_script() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
gen_local_script() (*dpdispatcher.machines.openapi.OpenAPI method*), 57
gen_script() (*dpdispatcher.Machine method*), 31
gen_script() (*dpdispatcher.machine.Machine method*), 74
gen_script() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
gen_script() (*dpdispatcher.machines.fugaku.Fugaku method*), 54
gen_script() (*dpdispatcher.machines.lsf.LSF method*), 55
gen_script() (*dpdispatcher.machines.openapi.OpenAPI method*), 57
gen_script() (*dpdispatcher.machines.pbs.PBS method*), 58
gen_script() (*dpdispatcher.machines.shell.Shell method*), 61
gen_script() (*dpdispatcher.machines.slurm.Slurm method*), 63
gen_script_command() (*dpdispatcher.Machine method*), 32
gen_script_command() (*dpdispatcher.machine.Machine method*), 74
gen_script_command() (*dpdispatcher.machines.slurm.SlurmJobArray method*), 64
gen_script_custom_flags_lines() (*dpdispatcher.Machine method*), 32
gen_script_custom_flags_lines() (*dpdispatcher.machine.Machine method*), 74
gen_script_end() (*dpdispatcher.Machine method*), 32
gen_script_end() (*dpdispatcher.machine.Machine method*), 74
gen_script_end() (*dpdispatcher.machines.distributed_shell.DistributedShell method*), 52
gen_script_end() (*dpdispatcher.machines.slurm.SlurmJobArray method*), 64
gen_script_env() (*dpdispatcher.Machine method*), 32
gen_script_env() (*dpdispatcher.machine.Machine method*), 74
gen_script_env() (*dpdispatcher.machines.distributed_shell.DistributedShell method*), 52
gen_script_header() (*dpdispatcher.Machine method*), 32
gen_script_header() (*dpdispatcher.machine.Machine*

method), 74
gen_script_header() (*dpdispatcher.machines.distributed_shell.DistributedShell method*), 52
gen_script_header() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
gen_script_header() (*dpdispatcher.machines.fugaku.Fugaku method*), 54
gen_script_header() (*dpdispatcher.machines.lsf.LSF method*), 55
gen_script_header() (*dpdispatcher.machines.openapi.OpenAPI method*), 57
gen_script_header() (*dpdispatcher.machines.pbs.PBS method*), 58
gen_script_header() (*dpdispatcher.machines.pbs.SGE method*), 59
gen_script_header() (*dpdispatcher.machines.pbs.Torque method*), 60
gen_script_header() (*dpdispatcher.machines.shell.Shell method*), 61
gen_script_header() (*dpdispatcher.machines.slurm.Slurm method*), 63
gen_script_header() (*dpdispatcher.machines.slurm.SlurmJobArray method*), 64
gen_script_run_command() (*dpdispatcher.Machine method*), 32
gen_script_run_command() (*dpdispatcher.machine.Machine method*), 74
gen_script_wait() (*dpdispatcher.Machine method*), 32
gen_script_wait() (*dpdispatcher.machine.Machine method*), 75
generate_jobs() (*dpdispatcher.Submission method*), 37
generate_jobs() (*dpdispatcher.submission.Submission method*), 81
generate_totp() (*in module dpdispatcher.utils.utils*), 69
get() (*dpdispatcher.contexts.ssh_context.SSHSession method*), 49
get() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
get() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
get_exit_code() (*dpdispatcher.Machine method*), 32
get_exit_code() (*dpdispatcher.machine.Machine method*), 75
get_exit_code() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
get_exit_code() (*dpdispatcher.machines.openapi.OpenAPI method*), 57
get_hash() (*dpdispatcher.Job method*), 30
get_hash() (*dpdispatcher.Submission method*), 37
get_hash() (*dpdispatcher.submission.Job method*), 77
get_hash() (*dpdispatcher.submission.Submission method*), 82
get_hash() (*dpdispatcher.submission.Task method*), 84
get_hash() (*dpdispatcher.Task method*), 39
get_job_detail() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
get_job_detail() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
get_job_result_url() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
get_job_result_url() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
get_job_root() (*dpdispatcher.contexts.hdfs_context.HDFSContext method*), 42
get_job_root() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
get_job_root() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
get_job_root() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 48
get_job_state() (*dpdispatcher.Job method*), 30
get_job_state() (*dpdispatcher.submission.Job method*), 77
get_last_error_message() (*dpdispatcher.Job method*), 30
get_last_error_message() (*dpdispatcher.submission.Job method*), 77
get_log() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
get_log() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
get_return() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
get_return() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
get_return() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 48
get_sha256() (*in module dpdispatcher.utils.utils*), 69

get_ssh_client() (*dpdispatcher.contexts.ssh_context.SSHSession method*), 49
get_task_state() (*dpdispatcher.submission.Task method*), 84
get_task_state() (*dpdispatcher.Task method*), 39
get_tasks_list() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
get_tasks_list() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
gpu_exclusive:
    resources[LSF]/kwargs/gpu_exclusive (*Argument*), **22**
gpu_new_syntax:
    resources[LSF]/kwargs/gpu_new_syntax (*Argument*), **22**
gpu_per_node:
    resources/gpu_per_node (*Argument*), **19**
gpu_usage:
    resources[LSF]/kwargs/gpu_usage (*Argument*), **22**
group_size:
    resources/group_size (*Argument*), **19**

H

handle_submission() (*in module dpdispatcher.entrypoints.submission*), 50
handle_unexpected_job_state() (*dpdispatcher.Job method*), 30
handle_unexpected_job_state() (*dpdispatcher.submission.Job method*), 77
handle_unexpected_submission_state() (*dpdispatcher.Submission method*), 37
handle_unexpected_submission_state() (*dpdispatcher.submission.Submission method*), 82
HDFS (*class in dpdispatcher.utils.hdfs_cli*), 67
HDFSContext (*class in dpdispatcher.contexts.hdfs_context*), 41
hostname:
    machine[SSHContext]/remote_profile/hostname (*Argument*), **16**
hotp() (*in module dpdispatcher.utils.utils*), 69

I

if_cuda_multi_devices:
    resources/strategy/if_cuda_multi_devices (*Argument*), **20**
ignore_exit_code:
    machine[BohriumContext]/remote_profile/ignore_exit_code (*Argument*), **18**
input_data:
    machine[BohriumContext]/remote_profile/input_data (*Argument*), **18**
inter_handler() (*dpdispatcher.contexts.ssh_context.SSHSession method*), 49
IOERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67

J

Job (*class in dpdispatcher*), 29
Job (*class in dpdispatcher.submission*), 76
job_create() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
job_create() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
job_to_json() (*dpdispatcher.Job method*), 30
job_to_json() (*dpdispatcher.submission.Job method*), 77
JobStatus (*class in dpdispatcher.utils.job_status*), 68

K

keep_backup:
    machine[BohriumContext]/remote_profile/keep_backup (*Argument*), **18**
key_filename:
    machine[SSHContext]/remote_profile/key_filename (*Argument*), **16**
kill() (*dpdispatcher.Machine method*), 32
kill() (*dpdispatcher.machine.Machine method*), 75
kill() (*dpdispatcher.machines.dp_cloud_server.Bohrium method*), 53
kill() (*dpdispatcher.machines.lsf.LSF method*), 55
kill() (*dpdispatcher.machines.openapi.OpenAPI method*), 57
kill() (*dpdispatcher.machines.pbs.PBS method*), 58
kill() (*dpdispatcher.machines.shell.Shell method*), 61
kill() (*dpdispatcher.machines.slurm.Slurm method*), 63
kill() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
kill() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
kwargs:
    resources[Bohrium]/kwargs (*Argument*), **22**
    resources[DistributedShell]/kwargs (*Argument*), **22**
    resources[Fugaku]/kwargs (*Argument*), **23**
    resources[LSF]/kwargs (*Argument*), **22**
    resources[OpenAPI]/kwargs (*Argument*), **22**
    resources[PBS]/kwargs (*Argument*), **23**
    resources[SGE]/kwargs (*Argument*), **23**
    resources[Shell]/kwargs (*Argument*), **23**
    resources[SlurmJobArray]/kwargs (*Argument*), **23**
    resources[Slurm]/kwargs (*Argument*), **21**

resources[Torque]/kwargs (*Argument*), 23

L

LazyLocalContext (*class in dpdispatcher.contexts.lazy_local_context*), 42
Lebesgue (*in module dpdispatcher.machines.dp_cloud_server*), 53
LebesgueContext (*in module dpdispatcher.contexts.dp_cloud_server_context*), 40
list_remote_dir() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 48
load_from_dict() (*dpdispatcher.base_context.BaseContext class method*), 71
load_from_dict() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext class method*), 40
load_from_dict() (*dpdispatcher.contexts.hdfs_context.HDFSContext class method*), 42
load_from_dict() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext class method*), 43
load_from_dict() (*dpdispatcher.contexts.local_context.LocalContext class method*), 45
load_from_dict() (*dpdispatcher.contexts.openapi_context.OpenAPIContext class method*), 46
load_from_dict() (*dpdispatcher.contexts.ssh_context.SSHContext class method*), 48
load_from_dict() (*dpdispatcher.Machine class method*), 32
load_from_dict() (*dpdispatcher.machine.Machine class method*), 75
load_from_dict() (*dpdispatcher.Resources class method*), 34
load_from_dict() (*dpdispatcher.submission.Resources class method*), 79
load_from_dict() (*dpdispatcher.submission.Task class method*), 84
load_from_dict() (*dpdispatcher.Task class method*), 39
load_from_json() (*dpdispatcher.Machine class method*), 32
load_from_json() (*dpdispatcher.machine.Machine class method*), 75
load_from_json() (*dpdispatcher.Resources class method*), 34
load_from_json() (*dpdispatcher.submission.Resources class method*), 79
load_from_json() (*dpdispatcher.submission.Task class method*), 84
load_from_json() (*dpdispatcher.Task class method*), 39
load_from_yaml() (*dpdispatcher.Machine class method*), 32
load_from_yaml() (*dpdispatcher.machine.Machine class method*), 75
load_from_yaml() (*dpdispatcher.Resources class method*), 34
load_from_yaml() (*dpdispatcher.submission.Resources class method*), 79
load_from_yaml() (*dpdispatcher.submission.Task class method*), 84
load_from_yaml() (*dpdispatcher.Task class method*), 39
local_root:
    machine/local_root (*Argument*), 15
LocalContext (*class in dpdispatcher.contexts.local_context*), 44
look_for_keys:
    machine[SSHContext]/remote_profile/look_for_keys (*Argument*), 17
LSF (*class in dpdispatcher.machines.lsf*), 54

M

machine (*Argument*)
    machine:, 15
Machine (*class in dpdispatcher*), 30
Machine (*class in dpdispatcher.machine*), 73
machine/batch_type (*Argument*)
    batch_type:, 15
machine/clean_asynchronously (*Argument*)
    clean_asynchronously:, 15
machine/context_type (*Argument*)
    context_type:, 15
machine/local_root (*Argument*)
    local_root:, 15
machine/remote_root (*Argument*)
    remote_root:, 15
machine:
    machine (*Argument*), 15
machine_arginfo() (*dpdispatcher.base_context.BaseContext class method*), 71
machine_subfields() (*dpdispatcher.base_context.BaseContext class method*), 71
machine_subfields() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext*

class method), 40
machine_subfields() (*dpdispatcher.contexts.ssh_context.SSHContext class method*), 48
machine[BohriumContext]/remote_profile (*Argument*)
    remote_profile:, 18
machine[BohriumContext]/remote_profile/email (*Argument*)
    email:, 18
machine[BohriumContext]/remote_profile/ignore_exit_code (*Argument*)
    ignore_exit_code:, 18
machine[BohriumContext]/remote_profile/input_data (*Argument*)
    input_data:, 18
machine[BohriumContext]/remote_profile/keep_backup (*Argument*)
    keep_backup:, 18
machine[BohriumContext]/remote_profile/password (*Argument*)
    password:, 18
machine[BohriumContext]/remote_profile/program_id (*Argument*)
    program_id:, 18
machine[BohriumContext]/remote_profile/retry_count (*Argument*)
    retry_count:, 18
machine[HDFSContext]/remote_profile (*Argument*)
    remote_profile:, 17
machine[LazyLocalContext]/remote_profile (*Argument*)
    remote_profile:, 17
machine[LocalContext]/remote_profile (*Argument*)
    remote_profile:, 17
machine[OpenAPIContext]/remote_profile (*Argument*)
    remote_profile:, 17
machine[SSHContext]/remote_profile (*Argument*)
    remote_profile:, 16
machine[SSHContext]/remote_profile/hostname (*Argument*)
    hostname:, 16
machine[SSHContext]/remote_profile/key_filename (*Argument*)
    key_filename:, 16
machine[SSHContext]/remote_profile/look_for_keys (*Argument*)
    look_for_keys:, 17
machine[SSHContext]/remote_profile/passphrase (*Argument*)
    passphrase:, 16
machine[SSHContext]/remote_profile/password (*Argument*)
    password:, 16
machine[SSHContext]/remote_profile/port (*Argument*)
    port:, 16
machine[SSHContext]/remote_profile/tar_compress (*Argument*)
    tar_compress:, 17
machine[SSHContext]/remote_profile/timeout (*Argument*)
    timeout:, 16
machine[SSHContext]/remote_profile/totp_secret (*Argument*)
    totp_secret:, 16
machine[SSHContext]/remote_profile/username (*Argument*)
    username:, 16
main() (*in module dpdispatcher.dpdisp*), 73
main_parser() (*in module dpdispatcher.dpdisp*), 73
map_dp_job_state() (*dpdispatcher.machines.dp_cloud_server.Bohrium static method*), 53
map_dp_job_state() (*dpdispatcher.machines.openapi.OpenAPI static method*), 57
mkdir() (*dpdispatcher.utils.hdfs_cli.HDFS static method*), 68
module
    dpdispatcher, 29
    dpdispatcher.arginfo, 70
    dpdispatcher.base_context, 70
    dpdispatcher.contexts, 39
    dpdispatcher.contexts.dp_cloud_server_context, 39
    dpdispatcher.contexts.hdfs_context, 41
    dpdispatcher.contexts.lazy_local_context, 42
    dpdispatcher.contexts.local_context, 44
    dpdispatcher.contexts.openapi_context, 45
    dpdispatcher.contexts.ssh_context, 46
    dpdispatcher.dlog, 73
    dpdispatcher.dpcloudserver, 49
    dpdispatcher.dpcloudserver.client, 49
    dpdispatcher.dpdisp, 73
    dpdispatcher.entrypoints, 50
    dpdispatcher.entrypoints.gui, 50
    dpdispatcher.entrypoints.submission, 50
    dpdispatcher.machine, 73
    dpdispatcher.machines, 51
    dpdispatcher.machines.distributed_shell, 51
    dpdispatcher.machines.dp_cloud_server, 52
    dpdispatcher.machines.fugaku, 53

dpdispatcher.machines.lsf, 54
dpdispatcher.machines.openapi, 56
dpdispatcher.machines.pbs, 57
dpdispatcher.machines.shell, 61
dpdispatcher.machines.slurm, 62
dpdispatcher.submission, 76
dpdispatcher.utils, 64
dpdispatcher.utils.dpcloudserver, 65
dpdispatcher.utils.dpcloudserver.client, 66
dpdispatcher.utils.dpcloudserver.config, 67
dpdispatcher.utils.dpcloudserver.retcode, 67
dpdispatcher.utils.dpcloudserver.zip_file, 67
dpdispatcher.utils.hdfs_cli, 67
dpdispatcher.utils.job_status, 68
dpdispatcher.utils.record, 69
dpdispatcher.utils.utils, 69
module_list:
resources/module_list (*Argument*), **21**
module_purge:
resources/module_purge (*Argument*), **20**
module_unload_list:
resources/module_unload_list (*Argument*), **20**
move() (*dpdispatcher.utils.hdfs_cli.HDFS static method*), 68

N

NODATA (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
number_node:
resources/number_node (*Argument*), **19**

O

OK (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
OpenAPI (*class in dpdispatcher.machines.openapi*), 56
OpenAPIContext (*class in dpdispatcher.contexts.openapi_context*), 45
options (*dpdispatcher.base_context.BaseContext attribute*), 71
options (*dpdispatcher.Machine attribute*), 32
options (*dpdispatcher.machine.Machine attribute*), 75
outlog:
task/outlog (*Argument*), **25**

P

para_deg:
resources/para_deg (*Argument*), **20**
PARAMERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
parse_args() (*in module dpdispatcher.dpdisp*), 73
passphrase:
machine[SSHContext]/remote_profile/passphrase (*Argument*), **16**
password:
machine[BohriumContext]/remote_profile/password (*Argument*), **18**
machine[SSHContext]/remote_profile/password (*Argument*), **16**
PBS (*class in dpdispatcher.machines.pbs*), 57
port:
machine[SSHContext]/remote_profile/port (*Argument*), **16**
post() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
post() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
prepend_script:
resources/prepend_script (*Argument*), **21**
program_id:
machine[BohriumContext]/remote_profile/program_id (*Argument*), **18**
put() (*dpdispatcher.contexts.ssh_context.SSHSession method*), 49
PWDERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67

Q

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

R

ratio_unfinished:
resources/strategy/ratio_unfinished (*Argument*), **20**
read() (*dpdispatcher.contexts.lazy_local_context.SPRetObj method*), 44
read() (*dpdispatcher.contexts.local_context.SPRetObj method*), 45
read_file() (*dpdispatcher.base_context.BaseContext method*), 71
read_file() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumCo method*), 40
read_file() (*dpdispatcher.contexts.hdfs_context.HDFSContext method*), 42
read_file() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
read_file() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
read_file() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
read_file() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 48
read_hdfs_file() (*dpdispatcher.utils.hdfs_cli.HDFS static method*), 68

read_home_file() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
read_home_file() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
readlines() (*dpdispatcher.contexts.lazy_local_context.SPRetObj method*), 44
readlines() (*dpdispatcher.contexts.local_context.SPRetObj method*), 45
refresh_token() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
refresh_token() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
register_job_id() (*dpdispatcher.Job method*), 30
register_job_id() (*dpdispatcher.submission.Job method*), 77
register_task() (*dpdispatcher.Submission method*), 37
register_task() (*dpdispatcher.submission.Submission method*), 82
register_task_list() (*dpdispatcher.Submission method*), 37
register_task_list() (*dpdispatcher.submission.Submission method*), 82
remote (*dpdispatcher.contexts.ssh_context.SSHSession property*), 49
remote_profile:
    machine[BohriumContext]/remote_profile (*Argument*), **18**
    machine[HDFSContext]/remote_profile (*Argument*), **17**
    machine[LazyLocalContext]/remote_profile (*Argument*), **17**
    machine[LocalContext]/remote_profile (*Argument*), **17**
    machine[OpenAPIContext]/remote_profile (*Argument*), **17**
    machine[SSHContext]/remote_profile (*Argument*), **16**
remote_root:
    machine/remote_root (*Argument*), **15**
remove() (*dpdispatcher.utils.hdfs_cli.HDFS static method*), 68
remove_unfinished_tasks() (*dpdispatcher.Submission method*), 37
remove_unfinished_tasks() (*dpdispatcher.submission.Submission method*), 82
REQERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
RequestInfoException, 49, 66
resources (*Argument*)
    resources:, **19**
Resources (*class in dpdispatcher*), 33
Resources (*class in dpdispatcher.submission*), 78
resources/append_script (*Argument*)
    append_script:, **21**
resources/batch_type (*Argument*)
    batch_type:, **21**
resources/cpu_per_node (*Argument*)
    cpu_per_node:, **19**
resources/custom_flags (*Argument*)
    custom_flags:, **19**
resources/envs (*Argument*)
    envs:, **21**
resources/gpu_per_node (*Argument*)
    gpu_per_node:, **19**
resources/group_size (*Argument*)
    group_size:, **19**
resources/module_list (*Argument*)
    module_list:, **21**
resources/module_purge (*Argument*)
    module_purge:, **20**
resources/module_unload_list (*Argument*)
    module_unload_list:, **20**
resources/number_node (*Argument*)
    number_node:, **19**
resources/para_deg (*Argument*)
    para_deg:, **20**
resources/prepend_script (*Argument*)
    prepend_script:, **21**
resources/queue_name (*Argument*)
    queue_name:, **19**
resources/source_list (*Argument*)
    source_list:, **20**
resources/strategy (*Argument*)
    strategy:, **20**
resources/strategy/customized_script_header_template_file (*Argument*)
    customized_script_header_template_file:, **20**
resources/strategy/if_cuda_multi_devices (*Argument*)
    if_cuda_multi_devices:, **20**
resources/strategy/ratio_unfinished (*Argument*)
    ratio_unfinished:, **20**
resources/wait_time (*Argument*)
    wait_time:, **21**
resources:
    resources (*Argument*), **19**
resources_arginfo() (*dpdispatcher.Machine class method*), 32

resources_arginfo() (*dpdispatcher.machine.Machine class method*), 75
resources_subfields() (*dpdispatcher.Machine class method*), 32
resources_subfields() (*dpdispatcher.machine.Machine class method*), 75
resources_subfields() (*dpdispatcher.machines.lsf.LSF class method*), 55
resources_subfields() (*dpdispatcher.machines.slurm.Slurm class method*), 63
resources_subfields() (*dpdispatcher.machines.slurm.SlurmJobArray class method*), 64
resources[Bohrium]/kwargs (*Argument*)
    kwargs:, **22**
resources[DistributedShell]/kwargs (*Argument*)
    kwargs:, **22**
resources[Fugaku]/kwargs (*Argument*)
    kwargs:, **23**
resources[LSF]/kwargs (*Argument*)
    kwargs:, **22**
resources[LSF]/kwargs/custom_gpu_line (*Argument*)
    custom_gpu_line:, **22**
resources[LSF]/kwargs/gpu_exclusive (*Argument*)
    gpu_exclusive:, **22**
resources[LSF]/kwargs/gpu_new_syntax (*Argument*)
    gpu_new_syntax:, **22**
resources[LSF]/kwargs/gpu_usage (*Argument*)
    gpu_usage:, **22**
resources[OpenAPI]/kwargs (*Argument*)
    kwargs:, **22**
resources[PBS]/kwargs (*Argument*)
    kwargs:, **23**
resources[SGE]/kwargs (*Argument*)
    kwargs:, **23**
resources[Shell]/kwargs (*Argument*)
    kwargs:, **23**
resources[SlurmJobArray]/kwargs (*Argument*)
    kwargs:, **23**
resources[SlurmJobArray]/kwargs/custom_gpu_line (*Argument*)
    custom_gpu_line:, **23**
resources[SlurmJobArray]/kwargs/slurm_job_size (*Argument*)
    slurm_job_size:, **23**
resources[Slurm]/kwargs (*Argument*)
    kwargs:, **21**
resources[Slurm]/kwargs/custom_gpu_line (*Argument*)
    custom_gpu_line:, **21**
resources[Torque]/kwargs (*Argument*)
    kwargs:, **23**
RETCODE (*class in dpdispatcher.utils.dpcloudserver.retcode*), 67
retry() (*in module dpdispatcher.utils.utils*), 69
retry_count:
    machine[BohriumContext]/remote_profile/retry_count (*Argument*), **18**
RetrySignal, 69
ROLEERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
rsync() (*in module dpdispatcher.utils.utils*), 70
rsync_available (*dpdispatcher.contexts.ssh_context.SSHSession property*), 49
run_cmd_with_all_output() (*in module dpdispatcher.utils.utils*), 70
run_submission() (*dpdispatcher.Submission method*), 37
run_submission() (*dpdispatcher.submission.Submission method*), 82
running (*dpdispatcher.utils.job_status.JobStatus attribute*), 68

S

serialize() (*dpdispatcher.Job method*), 30
serialize() (*dpdispatcher.Machine method*), 32
serialize() (*dpdispatcher.machine.Machine method*), 75
serialize() (*dpdispatcher.Resources method*), 34
serialize() (*dpdispatcher.Submission method*), 37
serialize() (*dpdispatcher.submission.Job method*), 77
serialize() (*dpdispatcher.submission.Resources method*), 79
serialize() (*dpdispatcher.submission.Submission method*), 82
serialize() (*dpdispatcher.submission.Task method*), 84
serialize() (*dpdispatcher.Task method*), 39
sftp (*dpdispatcher.contexts.ssh_context.SSHContext property*), 48
sftp (*dpdispatcher.contexts.ssh_context.SSHSession property*), 49
SGE (*class in dpdispatcher.machines.pbs*), 58
Shell (*class in dpdispatcher.machines.shell*), 61
Slurm (*class in dpdispatcher.machines.slurm*), 62
slurm_job_size:
    resources[SlurmJobArray]/kwargs/slurm_job_size (*Argument*), **23**
SlurmJobArray (*class in dpdispatcher.machines.slurm*), 63

source_list:
    resources/source_list (*Argument*), **20**
SPRetObj (*class in dpdispatcher.contexts.lazy_local_context*), 43
SPRetObj (*class in dpdispatcher.contexts.local_context*), 45
ssh (*dpdispatcher.contexts.ssh_context.SSHContext property*), 48
SSHContext (*class in dpdispatcher.contexts.ssh_context*), 46
SSHSession (*class in dpdispatcher.contexts.ssh_context*), 48
start_dpgui() (*in module dpdispatcher.entrypoints.gui*), 50
strategy:
    resources/strategy (*Argument*), **20**
sub_script_cmd() (*dpdispatcher.Machine method*), 32
sub_script_cmd() (*dpdispatcher.machine.Machine method*), 75
sub_script_cmd() (*dpdispatcher.machines.lsf.LSF method*), 56
sub_script_head() (*dpdispatcher.Machine method*), 32
sub_script_head() (*dpdispatcher.machine.Machine method*), 75
sub_script_head() (*dpdispatcher.machines.lsf.LSF method*), 56
subclasses_dict (*dpdispatcher.base_context.BaseContext attribute*), 71
subclasses_dict (*dpdispatcher.Machine attribute*), 33
subclasses_dict (*dpdispatcher.machine.Machine attribute*), 75
Submission (*class in dpdispatcher*), 34
Submission (*class in dpdispatcher.submission*), 79
submission_from_json() (*dpdispatcher.Submission class method*), 37
submission_from_json() (*dpdispatcher.submission.Submission class method*), 82
submission_to_json() (*dpdispatcher.Submission method*), 37
submission_to_json() (*dpdispatcher.submission.Submission method*), 82
submit_job() (*dpdispatcher.Job method*), 30
submit_job() (*dpdispatcher.submission.Job method*), 78

T

tar_compress:
    machine[SSHContext]/remote_profile/tar_compress (*Argument*), **17**
task (*Argument*)
    task:, **25**
Task (*class in dpdispatcher*), 38
Task (*class in dpdispatcher.submission*), 83
task/backward_files (*Argument*)
    backward_files:, **25**
task/command (*Argument*)
    command:, **25**
task/errlog (*Argument*)
    errlog:, **25**
task/forward_files (*Argument*)
    forward_files:, **25**
task/outlog (*Argument*)
    outlog:, **25**
task/task_work_path (*Argument*)
    task_work_path:, **25**
task:
    task (*Argument*), **25**
task_work_path:
    task/task_work_path (*Argument*), **25**
terminated (*dpdispatcher.utils.job_status.JobStatus attribute*), 68
THIRDERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
timeout:
    machine[SSHContext]/remote_profile/timeout (*Argument*), **16**
TOKENINVALID (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
Torque (*class in dpdispatcher.machines.pbs*), 59
totp_secret:
    machine[SSHContext]/remote_profile/totp_secret (*Argument*), **16**
try_download_result() (*dpdispatcher.Submission method*), 37
try_download_result() (*dpdispatcher.submission.Submission method*), 82
try_recover_from_json() (*dpdispatcher.Submission method*), 38
try_recover_from_json() (*dpdispatcher.submission.Submission method*), 82

U

UNDERDEBUG (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
unknown (*dpdispatcher.utils.job_status.JobStatus attribute*), 68
UNKOWNERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
unsubmitted (*dpdispatcher.utils.job_status.JobStatus attribute*), 68
unzip_file() (*in module dpdispatcher.utils.dpcloudserver.zip_file*), 67

update_submission_state() (*dpdispatcher.Submission method*), 38
update_submission_state() (*dpdispatcher.submission.Submission method*), 82
upload() (*dpdispatcher.base_context.BaseContext method*), 72
upload() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
upload() (*dpdispatcher.contexts.hdfs_context.HDFSContext method*), 42
upload() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
upload() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
upload() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
upload() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 48
upload() (*dpdispatcher.utils.dpcloudserver.Client method*), 65
upload() (*dpdispatcher.utils.dpcloudserver.client.Client method*), 66
upload_job() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
upload_job() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
upload_jobs() (*dpdispatcher.Submission method*), 38
upload_jobs() (*dpdispatcher.submission.Submission method*), 83
USERERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67
username:
machine[SSHContext]/remote_profile/username (*Argument*), **16**

V

VERIFYERR (*dpdispatcher.utils.dpcloudserver.retcode.RETCODE attribute*), 67

W

wait_time:
resources/wait_time (*Argument*), **21**
waiting (*dpdispatcher.utils.job_status.JobStatus attribute*), 68
write_file() (*dpdispatcher.base_context.BaseContext method*), 72
write_file() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
write_file() (*dpdispatcher.contexts.hdfs_context.HDFSContext method*), 42
write_file() (*dpdispatcher.contexts.lazy_local_context.LazyLocalContext method*), 43
write_file() (*dpdispatcher.contexts.local_context.LocalContext method*), 45
write_file() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
write_file() (*dpdispatcher.contexts.ssh_context.SSHContext method*), 48
write_home_file() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
write_home_file() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46
write_local_file() (*dpdispatcher.contexts.dp_cloud_server_context.BohriumContext method*), 40
write_local_file() (*dpdispatcher.contexts.openapi_context.OpenAPIContext method*), 46

Z

zip_file_list() (*in module dpdispatcher.utils.dpcloudserver.zip_file*), 67