
DPDispatcher

Deep Modeling

Jul 27, 2023

CONTENTS:

1	Install DPDispatcher	3
2	Getting Started	5
3	Supported contexts	9
3.1	LazyLocal	9
3.2	Local	9
3.3	SSH	9
3.4	Bohrium	9
3.5	HDFS	10
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
5	Machine parameters	13
6	Resources parameters	17
7	Task parameters	23
8	DPDispatcher API	25
8.1	dpdispatcher package	25
9	Running the DeePMD-kit on the Expanse cluster	91
10	Running Gaussian 16 with failure allowed	93
11	Running multiple MD tasks on a GPU workstation	95
12	Authors	97
13	Indices and tables	99

Python Module Index	101
Index	103

DPDispatcher is a Python package used to generate HPC (High Performance Computing) scheduler systems (Slurm/PBS/LSF/dpcloudserver) jobs input scripts and submit these scripts to HPC systems and poke until they finish.

DPDispatcher will monitor (poke) until these jobs finish and download the results files (if these jobs is running on remote systems connected by SSH).

INSTALL DPDISPATCHER

DPDispatcher can installed by pip:

```
pip install dpdispatcher
```

To add [Bohrium](#) support, execute

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


GETTING STARTED

DPDispatcher provides the following classes:

- *Task* class, which represents a command to be run on batch job system, as well as the essential files need by the command.
- *Submission* class, which represents a collection of jobs defined by the HPC system. And there may be common files to be uploaded by them. DPDispatcher will create and submit these jobs when a *submission* instance execute *run_submission* method. This method will poke until the jobs finish and return.
- *Job* class, a class used by *Submission* class, which represents a job on the HPC system. *Submission* will generate jobs' submitting scripts used by HPC systems automatically with the *Task* and *Resources*
- *Resources* class, which represents the computing resources for each job within a submission.

You can use DPDispatcher in a Python script to submit five tasks:

```
from dpdispatcher import Machine, Resources, Task, Submission

machine = Machine.load_from_json("machine.json")
resources = Resources.load_from_json("resources.json")

task0 = Task.load_from_json("task.json")

task1 = Task(
    command="cat example.txt",
    task_work_path="dir1/",
    forward_files=["example.txt"],
    backward_files=["out.txt"],
    outlog="out.txt",
)
task2 = Task(
    command="cat example.txt",
    task_work_path="dir2/",
    forward_files=["example.txt"],
    backward_files=["out.txt"],
    outlog="out.txt",
)
task3 = Task(
    command="cat example.txt",
    task_work_path="dir3/",
    forward_files=["example.txt"],
    backward_files=["out.txt"],
    outlog="out.txt",
```

(continues on next page)

(continued from previous page)

```

)
task4 = Task(
    command="cat example.txt",
    task_work_path="dir4/",
    forward_files=["example.txt"],
    backward_files=["out.txt"],
    outlog="out.txt",
)

task_list = [task0, task1, task2, task3, task4]

submission = Submission(
    work_base="lammps_md_300K_5GPa/",
    machine=machine,
    resources=resources,
    task_list=task_list,
    forward_common_files=["graph.pb"],
    backward_common_files=[],
)

submission.run_submission()

```

where machine.json is

```

{
    "batch_type": "Slurm",
    "context_type": "SSHContext",
    "local_root": "/home/user123/workplace/22_new_project/",
    "remote_root": "/home/user123/dpdispatcher_work_dir/",
    "remote_profile": {
        "hostname": "39.106.xx.xxx",
        "username": "user123",
        "port": 22,
        "timeout": 10
    }
}

```

resources.json is

```

{
    "number_node": 1,
    "cpu_per_node": 4,
    "gpu_per_node": 1,
    "queue_name": "GPUV100",
    "group_size": 5
}

```

and task.json is

```

{
    "command": "lmp -i input.lammps",
    "task_work_path": "bct-0/",
    "forward_files": [

```

(continues on next page)

(continued from previous page)

```

        "conf.lmp",
        "input.lammps"
    ],
    "backward_files": [
        "log.lammps"
    ],
    "outlog": "log",
    "errlog": "err",
}

```

You may also submit mutiple GPU jobs: complex resources example

```

resources = Resources(
    number_node=1,
    cpu_per_node=4,
    gpu_per_node=2,
    queue_name="GPU_2080Ti",
    group_size=4,
    custom_flags=[
        "#SBATCH --nice=100",
        "#SBATCH --time=24:00:00"
    ],
    strategy={
        # used when you want to add CUDA_VISIBLE_DEVICES automatically
        "if_cuda_multi_devices": True
    },
    para_deg=1,
    # will unload these modules before running tasks
    module_unload_list=["singularity"],
    # will load these modules before running tasks
    module_list=["singularity/3.0.0"],
    # will source the environment files before running tasks
    source_list=["./slurm_test.env"],
    # the envs option is used to export environment variables
    # And it will generate a line like below.
    # export DP_DISPATCHER_EXPORT=test_foo_bar_baz
    envs={"DP_DISPATCHER_EXPORT": "test_foo_bar_baz"},
)

```

The details of parameters can be found in *Machine Parameters*, *Resources Parameters*, and *Task Parameters*.

SUPPORTED CONTEXTS

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

3.1 LazyLocal

`context_type`: LazyLocal

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

3.2 Local

`context_type`: Local

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

3.3 SSH

`context_type`: SSH

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

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

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

3.4 Bohrium

`context_type`: Bohrium

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

3.5 HDFS

context_type: HDFS

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

3.6 OpenAPI

context_type: OpenAPI

OpenAPI is a new way to submit jobs to Bohrium. It uses [AccessKey](#) instead of username and password. Read [Bohrium documentation](#) for details. To use OpenAPI, one needs to provide necessary parameters in [remote_profile](#).

SUPPORTED BATCH JOB SYSTEMS

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

4.1 Bash

batch_type: Shell

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

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

4.2 Slurm

batch_type: Slurm, SlurmJobArray

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

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

4.3 OpenPBS or PBSPro

batch_type: PBS

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

Note that do not use PBS for Torque.

4.4 TORQUE

batch_type: Torque

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

4.5 LSF

batch_type: LSF

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

4.6 Bohrium

batch_type: Bohrium

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

4.7 DistributedShell

batch_type: DistributedShell

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

4.8 Fugaku

batch_type: Fugaku

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

Read Fujitsu cloud service documentation for details.

4.9 OpenAPI

batch_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.

MACHINE PARAMETERS

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

machine:

type: dict

argument path: machine

batch_type:

type: str

argument path: machine/batch_type

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

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

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

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

remote_profile:

type: dict, optional

argument path: `machine[LocalContext]/remote_profile`

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

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

remote_profile:

type: dict

argument path: `machine[SSHContext]/remote_profile`

The information used to maintain the connection with remote machine.

hostname:

type: str

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

hostname or ip of ssh connection.

username:

type: str

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

username of target linux system

password:

type: str, optional

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

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

port:

type: int, optional, default: 22

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

ssh connection port.

key_filename:

type: 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 `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 `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

keep_backup:

type: bool, optional

argument path:

machine[BohriumContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path:

machine[BohriumContext]/remote_profile/input_data

Configuration of job

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

remote_profile:

type: dict, optional

argument path: machine[LazyLocalContext]/remote_profile

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

When `context_type` is set to `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.

RESOURCES PARAMETERS

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

resources:

type: dict

argument path: resources

number_node:

type: int, optional, default: 1

argument path: resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1

argument path: resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0

argument path: resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

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

argument path: resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int

argument path: resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional
argument path: resources/custom_flags
The extra lines pass to job submitting script header

strategy:

type: dict, optional
argument path: resources/strategy
strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: bool, optional, default: False
argument path: resources/strategy/if_cuda_multi_devices
If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS. If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task. Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: float, optional, default: 0.0
argument path: resources/strategy/ratio_unfinished
The ratio of *tasks* that can be unfinished.

para_deg:

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

source_list:

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

module_purge:

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

module_unload_list:

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

module_list:

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

envs:

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

The environment variables to be exported on before submitting jobs

prepend_script:

type: list, optional, default: []
argument path: resources/prepend_script

Optional script run before jobs submitted.

append_script:

type: list, optional, default: []
argument path: resources/append_script

Optional script run after jobs submitted.

wait_time:

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

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

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

batch_type:

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

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 *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 *LSF* (or its alias *lsf*):

kwargs:

type: dict

argument path: `resources[LSF]/kwargs`

Extra arguments.

gpu_usage:

type: bool, optional, default: False

argument path: `resources[LSF]/kwargs/gpu_usage`

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False

argument path: `resources[LSF]/kwargs/gpu_new_syntax`

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

gpu_exclusive:

type: bool, optional, default: True

argument path: `resources[LSF]/kwargs/gpu_exclusive`

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

custom_gpu_line:

type: 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 `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 `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 `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 `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 `Shell` (or its alias `shell`):

kwargs:

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

This field is empty for this batch.

When `batch_type` is set to `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.

TASK PARAMETERS

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

task:

type: dict

argument path: task

command:

type: str

argument path: task/command

A command to be executed of this task. The expected return code is 0.

task_work_path:

type: str

argument path: task/task_work_path

The dir where the command to be executed.

forward_files:

type: list

argument path: task/forward_files

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

backward_files:

type: list

argument path: task/backward_files

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

outlog:

type: str | NoneType

argument path: task/outlog

The out log file name. redirect from stdout

errlog:

type: `str | NoneType`

argument path: `task/errlog`

The err log file name. redirect from stderr

DPDISPATCHER API

8.1 dpdispatcher package

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

Bases: *Machine*

Methods

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

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

`check_finish_tag(job)`

`check_status(job)`

do_submit(*job*)

Submit th job to yarn using distributed shell.

Parameters

job

[Job class instance] job to be submitted

Returns

job_id: string

submit process id

gen_script_end(*job*)

gen_script_env(*job*)

gen_script_header(*job*)

dpdispatcher.DpCloudServer

alias of *Bohrrium*

dpdispatcher.DpCloudServerContext

alias of *BohrriumContext*

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

Bases: *Machine*

Methods

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

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

`check_finish_tag(job)`

`check_status(job)`

`default_resources(resources)`

`do_submit(job)`

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

`gen_script(job)`

`gen_script_header(job)`

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

Bases: [BaseContext](#)

Methods

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

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*)**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

<i>deserialize</i> (<i>job_dict</i> [, <i>machine</i>])	Convert the <i>job_dict</i> to a Submission class object.
<i>get_job_state</i> ()	Get the jobs.
<i>serialize</i> ([<i>if_static</i>])	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.

handle_unexpected_job_state()

job_to_json()

register_job_id(*job_id*)

serialize(*if_static=False*)

Convert the Task class instance to a dictionary.

Parameters

if_static

[bool] whether dump the job runtime information (job_id, job_state, fail_count, job_uuid etc.) to the dictionary.

Returns

task_dict

[dict] the dictionary converted from the Task class instance

submit_job()

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

Bases: *Machine*

LSF batch.

Methods

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

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

check_finish_tag(*job*)

check_status(***kwargs*)

default_resources(*resources*)

do_submit(***kwargs*)

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

gen_script(*job*)

gen_script_header(*job*)

kill(*job*)

Kill the job.

Parameters

job

[Job] job

classmethod resources_subfields() → List[Argument]

Generate the resources subfields.

Returns

list[Argument]

resources subfields

sub_script_cmd(*res*)

sub_script_head(*res*)

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

Bases: [BaseContext](#)

Run jobs in the local server and local directory.

Parameters

local_root

[str] The local directory to store the jobs.

remote_root

[str, optional] The argument takes no effect.

remote_profile

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

***args**

The arguments.

****kwargs**

The keyword arguments.

Methods

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)

```

`dpdispatcher.Lebesgue`

alias of [*Bohrrium*](#)

`dpdispatcher.LebesgueContext`

alias of [*BohrriumContext*](#)

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

Bases: [*BaseContext*](#)

Run jobs in the local server and remote directory.

Parameters

local_root

[str] The local directory to store the jobs.

remote_root

[str] The remote directory to store the jobs.

remote_profile

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

***args**

The arguments.

****kwargs**

The keyword arguments.

Methods

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

bind_submission	
block_call	
block_checkcall	
call	
check_file_exists	
check_finish	
clean	
download	
get_job_root	
get_return	
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.**Machine**(*args, **kwargs)

Bases: `object`

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

Parameters

context

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

Methods

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

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

```

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

classmethod arginfo()

bind_context(context)

abstract check_finish_tag(**kwargs)

check_if_recover(submission)

abstract check_status(job)

default_resources(res)

classmethod deserialize(machine_dict)

abstract do_submit(job)
    Submit a single job, assuming that no job is running there.

gen_command_env_cuda_devices(resources)

gen_script(job)

gen_script_command(job)

```

gen_script_custom_flags_lines(*job*)

gen_script_end(*job*)

gen_script_env(*job*)

abstract gen_script_header(*job*)

gen_script_wait(*resources*)

kill(*job*)

Kill the job.

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

Parameters

job

[Job] job

classmethod load_from_dict(*machine_dict*)

classmethod load_from_json(*json_path*)

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

classmethod resources_arginfo() → [Argument](#)

Generate the resources arginfo.

Returns

Argument

resources arginfo

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

Generate the resources subfields.

Returns

list[Argument]

resources subfields

serialize(*if_empty_remote_profile=False*)

sub_script_cmd(*res*)

sub_script_head(*res*)


```

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

```

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

Bases: [Machine](#)

Methods

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

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

check_finish_tag(*job*)

check_if_recover(*submission*)

check_status(*job*)

do_submit(*job*)

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

gen_local_script(*job*)

gen_script(*job*)

gen_script_header(*job*)

static map_dp_job_state(*status*)

class dpdispatcher.**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_exists	
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_exists(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)

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

```

Methods

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

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

check_finish_tag(*job*)

check_status(*job*)

default_resources(*resources*)

do_submit(*job*)

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

gen_script(*job*)

gen_script_header(*job*)

kill(*job*)

Kill the job.

Parameters

job

[Job] job

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

Bases: `object`

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

Parameters

number_node

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

cpu_per_node
[int] cpu numbers of each node.

gpu_per_node
[int] gpu numbers of each node.

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

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

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

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

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

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

para_deg
[int] Decide how many tasks will be run in parallel. Usually run with *strategy*['if_cuda_multi_devices']

source_list
[list of Path] The env file to be sourced before the command execution.

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

Methods

arginfo	
deserialize	
load_from_dict	
load_from_json	
serialize	

static arginfo(*detail_kwargs=True*)

classmethod deserialize(*resources_dict*)

classmethod load_from_dict(*resources_dict*)

classmethod load_from_json(*json_file*)

serialize()

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

Bases: [BaseContext](#)

Attributes

sftp
ssh

Methods

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

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

classmethod load_from_dict(*context_dict*)

classmethod machine_subfields() → List[Argument]

Generate the machine subfields.

Returns

list[Argument]

machine subfields

read_file(*fname*)

property sftp

property ssh

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

write_file(*fname*, *write_str*)

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

Bases: [Machine](#)

Methods

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

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

`check_finish_tag(job)`

`check_status(job)`

`default_resources(resources)`

`do_submit(job)`

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

`gen_script(job)`

`gen_script_header(job)`

`kill(job)`

Kill the job.

Parameters

job

[Job] job

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

Bases: [Machine](#)

Methods

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

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

check_finish_tag(*job*)

check_status(***kwargs*)

default_resources(*resources*)

do_submit(***kwargs*)

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

gen_script(*job*)

gen_script_header(*job*)

kill(*job*)

Kill the job.

Parameters

job

[Job] job

classmethod resources_subfields() → List[Argument]

Generate the resources subfields.

Returns

list[Argument]

resources subfields

class dpdispatcher.**Submission**(*work_base*, *machine=None*, *resources=None*, *forward_common_files=[]*,
backward_common_files=[], *, *task_list=[]*)

Bases: `object`

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

Parameters

work_base

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

machine

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

resources

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

forward_common_files

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

backward_common_files

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

task_list

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

Methods

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

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

async `async_run_submission(**kwargs)`

Async interface of `run_submission`.

Examples

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

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

<code>arginfo</code>	
<code>get_hash</code>	
<code>load_from_dict</code>	
<code>load_from_json</code>	
<code>serialize</code>	

static arginfo()

classmethod `deserialize(task_dict)`

Convert the task_dict to a Task class object.

Parameters

task_dict

[dict] the dictionary which contains the task information

Returns

task

[Task] the Task class instance converted from the task_dict

get_hash()

get_task_state(context)

Get the task state by checking the tag file.

Parameters

context

[Context] the context of the task

classmethod `load_from_dict(task_dict: dict) → Task`

classmethod `load_from_json(json_file)`

serialize()

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

Bases: [PBS](#)

Methods

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

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

`check_status(job)`

`gen_script_header(job)`

`dpdispatcher.info()`

Show basic information about dpdispatcher, its location and version.

8.1.1 Subpackages

dpdispatcher.dpcloudserver package

`class dpdispatcher.dpcloudserver.Client(email=None, password=None, debug=False, 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	
post	
refresh_token	
upload	

```
download(oss_file, save_file, endpoint, bucket_name)

download_from_url(url, save_file)

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

get_job_detail(job_id)

get_job_result_url(job_id)

get_log(job_id)

get_tasks_list(group_id, per_page=30)

job_create(job_type, oss_path, input_data, program_id=None, group_id=None)

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

refresh_token(retry=3)

upload(oss_task_zip, zip_task_file, endpoint, bucket_name)
```

Submodules

dpdispatcher.dpcloudserver.client module

```
class dpdispatcher.dpcloudserver.client.Client(email=None, password=None, debug=False,
                                              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	
post	
refresh_token	
upload	

```
download(oss_file, save_file, endpoint, bucket_name)

download_from_url(url, save_file)

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

get_job_detail(job_id)

get_job_result_url(job_id)
```



```

get_log(job_id)

get_tasks_list(group_id, per_page=30)

job_create(job_type, oss_path, input_data, program_id=None, group_id=None)

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

refresh_token(retry=3)

upload(oss_task_zip, zip_task_file, endpoint, bucket_name)

exception dpdispatcher.dpcloudserver.client.RequestInfoException
    Bases: Exception

```

dpdispatcher.dpcloudserver.config module

dpdispatcher.dpcloudserver.retcode module

```

class dpdispatcher.dpcloudserver.retcode.RETCODE
    Bases: object

    DATAERR = '2002'

    DBERR = '2000'

    IOERR = '2003'

    NODATA = '2300'

    OK = '0000'

    PARAMERR = '2101'

    PWDERR = '2104'

    REQERR = '2200'

    ROLEERR = '2103'

    THIRDDERR = '2001'

    TOKENINVALID = '2100'

    UNDERDEBUG = '2301'

    UNKOWNERR = '2400'

    USERERR = '2102'

    VERIFYERR = '2105'

```

dpdispatcher.dpcloudserver.temp_test module

dpdispatcher.dpcloudserver.zip_file module

`dpdispatcher.dpcloudserver.zip_file.unzip_file(zip_file, out_dir='.')`

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

8.1.2 Submodules

8.1.3 dpdispatcher.JobStatus module

class `dpdispatcher.JobStatus.JobStatus(value)`

Bases: `IntEnum`

An enumeration.

completing = 6

finished = 5

running = 3

terminated = 4

unknown = 100

unsubmitted = 1

waiting = 2

8.1.4 dpdispatcher.arginfo module

8.1.5 dpdispatcher.base_context module

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

Bases: `object`

Methods

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

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

```

```
abstract upload(submission)
```

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

8.1.6 dpdispatcher.distributed_shell module

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

Bases: *Machine*

Methods

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

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

`check_finish_tag(job)`

`check_status(job)`

`do_submit(job)`

Submit th job to yarn using distributed shell.

Parameters

job

[Job class instance] job to be submitted

Returns

job_id: string

submit process id

`gen_script_end(job)`

`gen_script_env(job)`

`gen_script_header(job)`

8.1.7 dpdispatcher.dp_cloud_server module

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

Bases: *Machine*

Methods

<i>do_submit</i> (job)	Submit a single job, assuming that no job is running there.
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_wait	
load_from_dict	
load_from_json	
map_dp_job_state	
serialize	
sub_script_cmd	
sub_script_head	

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

check_finish_tag(job)

check_if_recover(submission)

check_status(job)

do_submit(job)

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

gen_local_script(job)

gen_script(job)

```
gen_script_header(job)
```

```
static map_dp_job_state(status)
```

```
dpdispatcher.dp_cloud_server.DpCloudServer
```

alias of *Bohrium*

```
dpdispatcher.dp_cloud_server.Lebesgue
```

alias of *Bohrium*

8.1.8 dpdispatcher.dp_cloud_server_context module

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

Bases: *BaseContext*

Methods

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

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

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

```
bind_submission(submission)
```

```
check_file_exists(fname)
```

```
check_home_file_exists(fname)
```

```
clean()
```

```
download(submission)
```

```
classmethod load_from_dict(context_dict)
```

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

Generate the machine subfields.

Returns

list[Argument]
machine subfields

read_file(*fname*)

read_home_file(*fname*)

upload(*submission*)

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

write_file(*fname*, *write_str*)

write_home_file(*fname*, *write_str*)

write_local_file(*fname*, *write_str*)

`dpdispatcher.dp_cloud_server_context.DpCloudServerContext`

alias of *BohrriumContext*

`dpdispatcher.dp_cloud_server_context.LebesgueContext`

alias of *BohrriumContext*

8.1.9 dpdispatcher.dpdisp module

`dpdispatcher.dpdisp.main()`

8.1.10 dpdispatcher.fugaku module

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

Bases: *Machine*

Methods

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

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

`check_finish_tag(job)`

`check_status(job)`

`default_resources(resources)`

`do_submit(job)`

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

`gen_script(job)`

`gen_script_header(job)`

8.1.11 dpdispatcher.hdfs_cli module

class `dpdispatcher.hdfs_cli.HDFS`

Bases: `object`

Fundamental class for HDFS basic manipulation.

Methods

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

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.

8.1.12 dpdispatcher.hdfs_context module

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

Bases: *BaseContext*

Methods

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

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*)

8.1.13 dpdispatcher.lazy_local_context module

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

Bases: [BaseContext](#)

Run jobs in the local server and local directory.

Parameters

local_root

[str] The local directory to store the jobs.

remote_root

[str, optional] The argument takes no effect.

remote_profile

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

***args**

The arguments.

****kwargs**

The keyword arguments.

Methods

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

<code>bind_submission</code>	
<code>block_call</code>	
<code>block_checkcall</code>	
<code>call</code>	
<code>check_file_exists</code>	
<code>check_finish</code>	
<code>clean</code>	
<code>download</code>	
<code>get_job_root</code>	
<code>get_return</code>	
<code>load_from_dict</code>	
<code>read_file</code>	
<code>upload</code>	
<code>write_file</code>	

`bind_submission(submission)`

`block_call(cmd)`

`block_checkcall(cmd)`

`call(cmd)`

`check_file_exists(fname)`

```

check_finish(proc)

clean()

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

get_job_root()

get_return(proc)

classmethod load_from_dict(context_dict)

read_file(fname)

upload(jobs, dereference=True)

write_file(fname, write_str)

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

```

Methods

read	
readlines	

```

read()

readlines()

```

8.1.14 dpdispatcher.local_context module

```

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

```

Run jobs in the local server and remote directory.

Parameters

local_root
[str] The local directory to store the jobs.

remote_root
[str] The remote directory to store the jobs.

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

***args**
The arguments.

****kwargs**
The keyword arguments.

Methods

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

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.local_context.SPRetObj(ret)
    Bases: object
```

Methods

read	
readlines	

read()

readlines()

8.1.15 dpdispatcher.lsf module

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

Bases: *Machine*

LSF batch.

Methods

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

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

check_finish_tag(job)

check_status(**kwargs)

default_resources(*resources*)

do_submit(***kwargs*)

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

gen_script(*job*)

gen_script_header(*job*)

kill(*job*)

Kill the job.

Parameters

job

[Job] job

classmethod resources_subfields() → List[Argument]

Generate the resources subfields.

Returns

list[Argument]

resources subfields

sub_script_cmd(*res*)

sub_script_head(*res*)

8.1.16 dpdispatcher.machine module

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

Bases: [object](#)

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

Parameters

context

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

Methods

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

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

```

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

classmethod arginfo()

bind_context(context)

abstract check_finish_tag(**kwargs)

check_if_recover(submission)

abstract check_status(job)

default_resources(res)

classmethod deserialize(machine_dict)

abstract do_submit(job)
    Submit a single job, assuming that no job is running there.

gen_command_env_cuda_devices(resources)

gen_script(job)

gen_script_command(job)

gen_script_custom_flags_lines(job)

gen_script_end(job)

gen_script_env(job)

abstract gen_script_header(job)

gen_script_wait(resources)

```

kill(*job*)

Kill the job.

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

Parameters**job**

[Job] job

classmethod **load_from_dict**(*machine_dict*)**classmethod** **load_from_json**(*json_path*)**options** = {'Bohrium', 'DistributedShell', 'Fugaku', 'LSF', 'OpenAPI', 'PBS', 'Shell', 'Slurm', 'SlurmJobArray', 'Torque'}**classmethod** **resources_arginfo**() → *Argument*

Generate the resources arginfo.

Returns**Argument**

resources arginfo

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

Generate the resources subfields.

Returns**list[Argument]**

resources subfields

serialize(*if_empty_remote_profile=False*)**sub_script_cmd**(*res*)**sub_script_head**(*res*)

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

8.1.17 dpdispatcher.openapi module

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

Bases: *Machine*

Methods

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

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

check_finish_tag(job)

check_if_recover(submission)

check_status(job)

do_submit(job)

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

gen_local_script(job)

gen_script(job)

gen_script_header(job)

static map_dp_job_state(status)

8.1.18 dpdispatcher.openapi_context module

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

Bases: `BaseContext`

Methods

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

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

`bind_submission(submission)`

`check_file_exists(fname)`

`check_home_file_exists(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)`

8.1.19 dpdispatcher.pbs module

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

Bases: *Machine*

Methods

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

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

check_finish_tag(job)

check_status(job)

default_resources(resources)

do_submit(job)

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

gen_script(job)

gen_script_header(job)

kill(job)

Kill the job.

Parameters

job
[Job] job

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

Bases: *PBS*

Methods

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

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

check_status(job)

gen_script_header(job)

8.1.20 dpdispatcher.shell module

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

Bases: *Machine*

Methods

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

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

`check_finish_tag(job)`

`check_status(job)`

`default_resources(resources)`

`do_submit(job)`

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

`gen_script(job)`

`gen_script_header(job)`

`kill(job)`

Kill the job.

Parameters

job

[Job] job

8.1.21 dpdispatcher.slurm module

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

Bases: *Machine*

Methods

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

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

check_finish_tag(job)

check_status(**kwargs)

default_resources(resources)

do_submit(**kwargs)

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

gen_script(job)

gen_script_header(job)

kill(job)

Kill the job.

Parameters

job
[Job] job

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

Generate the resources subfields.

Returns

`list[Argument]`

resources subfields

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

Bases: `Slurm`

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

Methods

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

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

`check_finish_tag(job)`

`check_status(**kwargs)`

`gen_script_command(job)`

`gen_script_end(job)`

`gen_script_header(job)`

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

Generate the resources subfields.

Returns

`list[Argument]`

resources subfields

8.1.22 dpdispatcher.ssh_context module

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

Bases: `BaseContext`

Attributes

`sftp`

`ssh`

Methods

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

<code>bind_submission</code>	
<code>block_call</code>	
<code>call</code>	
<code>check_file_exists</code>	
<code>check_finish</code>	
<code>clean</code>	
<code>close</code>	
<code>download</code>	
<code>get_job_root</code>	
<code>get_return</code>	
<code>load_from_dict</code>	
<code>read_file</code>	
<code>upload</code>	
<code>write_file</code>	

`bind_submission(submission)`

`block_call(cmd)`

`block_checkcall(cmd, asynchronously=False, stderr_whitelist=None)`

Run command with arguments. Wait for command to complete. If the return code was zero then return, otherwise raise `RuntimeError`.

Parameters

`cmd`

[str] The command to run.

asynchronously

[bool, optional, default=False] Run command asynchronously. If True, *nohup* will be used to run the command.

stderr_whitelist

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

call(*cmd*)

check_file_exists(*fname*)

check_finish(*cmd_pipes*)

clean()

close()

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

get_job_root()

get_return(*cmd_pipes*)

classmethod load_from_dict(*context_dict*)

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

Generate the machine subfields.

Returns

list[Argument]

machine subfields

read_file(*fname*)

property sftp

property ssh

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

write_file(*fname*, *write_str*)

class dpdispatcher.ssh_context.SSHSession(*hostname*, *username*, *password=None*, *port=22*,
key_filename=None, *passphrase=None*, *timeout=10*,
totp_secret=None, *tar_compress=True*, *look_for_keys=True*)

Bases: [object](#)

Attributes

remote

rsync_available

[sftp](#)

Returns sftp.

Methods

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

arginfo	
close	
ensure_alive	
exec_command	
get	
get_ssh_client	
put	

static `arginfo()`

close()

ensure_alive(max_check=10, sleep_time=10)

exec_command(**kwargs)

get(from_f, to_f)

get_ssh_client()

inter_handler(title, instructions, prompt_list)

inter_handler: the callback for paramiko.transport.auth_interactive.

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

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

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

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

put(from_f, to_f)

property remote: `str`

property rsync_available: `bool`

property sftp

Returns sftp. Open a new one if not existing.

8.1.23 dpdispatcher.submission module

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

Bases: `object`

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

Parameters

job_task_list

[list of Task] the tasks belonging to the job

resources

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

machine

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

Methods

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

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

classmethod `deserialize(job_dict, machine=None)`

Convert the job_dict to a Submission class object.

Parameters

job_dict

[dict] the dictionary which contains the job information

machine

[Machine] the machine object to execute the job

Returns

submission

[Job] the Job class instance converted from the job_dict

get_hash()

get_job_state()

Get the jobs. Usually, this method will query the database of slurm or pbs job scheduler system and get the results.

Notes

this method will not submit or resubmit the jobs if the job is unsubmitted.

handle_unexpected_job_state()

job_to_json()

register_job_id(*job_id*)

serialize(*if_static=False*)

Convert the Task class instance to a dictionary.

Parameters

if_static

[bool] whether dump the job runtime information (job_id, job_state, fail_count, job_uuid etc.) to the dictionary.

Returns

task_dict

[dict] the dictionary converted from the Task class instance

submit_job()

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

Bases: `object`

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

Parameters

number_node

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

cpu_per_node

[int] cpu numbers of each node.

gpu_per_node

[int] gpu numbers of each node.

queue_name

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

group_size

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

custom_flags

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

strategy

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

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

ratio_unfinished

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

para_deg

[int] Decide how many tasks will be run in parallel. Usually run with *strategy*['if_cuda_multi_devices']

source_list

[list of Path] The env file to be sourced before the command execution.

wait_time

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

Methods

arginfo	
deserialize	
load_from_dict	
load_from_json	
serialize	

static arginfo(*detail_kwargs=True*)

classmethod deserialize(*resources_dict*)

classmethod load_from_dict(*resources_dict*)

classmethod load_from_json(*json_file*)

serialize()

class dpdispatcher.submission.Submission(*work_base, machine=None, resources=None, forward_common_files=[], backward_common_files=[], *, task_list=[]*)

Bases: `object`

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

Parameters**work_base**

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

machine

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

resources

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

forward_common_files

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

backward_common_files

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

task_list

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

Methods

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

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

async async_run_submission(kwargs)**

Async interface of run_submission.

Examples

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

(continues on next page)

(continued from previous page)

```

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

```

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

bind_machine(*machine*)

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

Parameters

machine

[Machine] the machine to bind with

check_all_finished()

Check whether all the jobs in the submission.

Notes

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

check_ratio_unfinished(*ratio_unfinished: float*) → bool

Calculate the ratio of unfinished tasks in the submission.

Parameters

ratio_unfinished

[float] the ratio of unfinished tasks in the submission

Returns

bool

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

clean_jobs()

classmethod deserialize(*submission_dict, machine=None*)

Convert the submission_dict to a Submission class object.

Parameters

submission_dict

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

machine

[Machine] Machine class Object to execute the jobs

Returns

submission

[Submission] the Submission class instance converted from the submission_dict

download_jobs()

generate_jobs()

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

get_hash()**handle_unexpected_submission_state()**

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

register_task(task)**register_task_list(task_list)****remove_unfinished_tasks()****run_submission(*, dry_run=False, exit_on_submit=False, clean=True, check_interval=30)**

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

serialize(if_static=False)

Convert the Submission class instance to a dictionary.

Parameters**if_static**

[bool] whether dump the job runtime information (like job_id, job_state, fail_count) to the dictionary.

Returns**submission_dict**

[dict] the dictionary converted from the Submission class instance

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

Check whether all the jobs in the submission.

Notes

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

upload_jobs()

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

Bases: `object`

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

Parameters

command

[Str] the command to be executed.

task_work_path

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

forward_files

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

backward_files

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

outlog

[Str] the filename to which command redirect stdout

errlog

[Str] the filename to which command redirect stderr

Methods

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

arginfo	
get_hash	
load_from_dict	
load_from_json	
serialize	

static arginfo()

classmethod deserialize(task_dict)

Convert the task_dict to a Task class object.

Parameters

task_dict

[dict] the dictionary which contains the task information

Returns

task

[Task] the Task class instance converted from the task_dict

get_hash()

get_task_state(context)

Get the task state by checking the tag file.

Parameters

context

[Context] the context of the task

classmethod load_from_dict(task_dict: dict) → Task

classmethod load_from_json(json_file)

serialize()

8.1.24 dpdispatcher.utils module

exception dpdispatcher.utils.RetrySignal

Bases: [Exception](#)

Exception to give a signal to retry the function.

dpdispatcher.utils.generate_totp(secret: str, period: int = 30, token_length: int = 6) → str

Generate time-based one time password (TOTP) from the secret.

Some HPCs use TOTP for two-factor authentication for safety.

Parameters

secret

[str] The encoded secret provided by the HPC. It's usually extracted from a 2D code and base32 encoded.

period

[int, default=30] Time period where the code is valid in seconds.

token_length

[int, default=6] The token length.

Returns

token: str

The generated token.

References

<https://github.com/lepture/otpauth/blob/49914d83d36dbcd33c9e26f65002b21ce09a6303/otpauth.py#L143-L160>

dpdispatcher.utils.get_sha256(filename)

Get sha256 of a file.

Parameters

filename

[str] The filename.

Returns

sha256: str

The sha256.

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

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

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

Parameters

max_retry

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

sleep

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

catch_exception

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

Returns

decorator: Callable

The decorator.

Examples

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

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

Call rsync to transfer files.

Parameters

from_file

[str] SRC

to_file

[str] DEST

port

[int, default=22] port for ssh

key_filename

[str, optional] identity file name

timeout

[int, default=10] timeout for ssh

Raises

RuntimeError

when return code is not 0

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

RUNNING THE DEEPM-D-KIT ON THE EXPANSE CLUSTER

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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


RUNNING GAUSSIAN 16 WITH FAILURE ALLOWED

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

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

This command ensures the task will always provide zero code.

RUNNING MULTIPLE MD TASKS ON A GPU WORKSTATION

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

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

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

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

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

at the same time.

AUTHORS

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

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

INDICES AND TABLES

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

PYTHON MODULE INDEX

d

- `dpdispatcher`, 25
- `dpdispatcher.arginfo`, 54
- `dpdispatcher.base_context`, 54
- `dpdispatcher.distributed_shell`, 56
- `dpdispatcher.dp_cloud_server`, 58
- `dpdispatcher.dp_cloud_server_context`, 59
- `dpdispatcher.dpcloudserver`, 51
- `dpdispatcher.dpcloudserver.client`, 52
- `dpdispatcher.dpcloudserver.config`, 53
- `dpdispatcher.dpcloudserver.retcode`, 53
- `dpdispatcher.dpcloudserver.zip_file`, 54
- `dpdispatcher.dpdisp`, 60
- `dpdispatcher.fugaku`, 60
- `dpdispatcher.hdfs_cli`, 61
- `dpdispatcher.hdfs_context`, 62
- `dpdispatcher.JobStatus`, 54
- `dpdispatcher.lazy_local_context`, 64
- `dpdispatcher.local_context`, 65
- `dpdispatcher.lsf`, 67
- `dpdispatcher.machine`, 68
- `dpdispatcher.openapi`, 71
- `dpdispatcher.openapi_context`, 72
- `dpdispatcher.pbs`, 73
- `dpdispatcher.shell`, 74
- `dpdispatcher.slurm`, 76
- `dpdispatcher.ssh_context`, 78
- `dpdispatcher.submission`, 81
- `dpdispatcher.utils`, 88

INDEX

A

alias (*dpdispatcher.base_context.BaseContext* attribute), 55

alias (*dpdispatcher.dp_cloud_server.Bohrium* attribute), 58

alias (*dpdispatcher.dp_cloud_server_context.BohriumContext* attribute), 59

alias (*dpdispatcher.Machine* attribute), 35

alias (*dpdispatcher.machine.Machine* attribute), 69

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

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

arginfo() (*dpdispatcher.machine.Machine* class method), 69

arginfo() (*dpdispatcher.Resources* static method), 41

arginfo() (*dpdispatcher.ssh_context.SSHSession* static method), 80

arginfo() (*dpdispatcher.submission.Resources* static method), 83

arginfo() (*dpdispatcher.submission.Task* static method), 87

arginfo() (*dpdispatcher.Task* static method), 49

async_run_submission() (*dpdispatcher.Submission* method), 46

async_run_submission() (*dpdispatcher.submission.Submission* method), 84

B

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

BaseContext (class in *dpdispatcher.base_context*), 54

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

resources/batch_type (*Argument*), 19

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

bind_context() (*dpdispatcher.machine.Machine* method), 69

bind_machine() (*dpdispatcher.Submission* method), 47

bind_machine() (*dpdispatcher.submission.Submission* method), 85

bind_submission() (*dpdispatcher.base_context.BaseContext* method), 55

bind_submission() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 59

bind_submission() (*dpdispatcher.hdfs_context.HDFSContext* method), 62

bind_submission() (*dpdispatcher.HDFSContext* method), 28

bind_submission() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 64

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

bind_submission() (*dpdispatcher.local_context.LocalContext* method), 66

bind_submission() (*dpdispatcher.LocalContext* method), 34

bind_submission() (*dpdispatcher.openapi_context.OpenAPIContext* method), 72

bind_submission() (*dpdispatcher.OpenAPIContext* method), 39

bind_submission() (*dpdispatcher.ssh_context.SSHContext* method), 78

bind_submission() (*dpdispatcher.SSHContext* method), 42

block_call() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 64

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

block_call() (*dpdispatcher.local_context.LocalContext* method), 66

block_call() (*dpdispatcher.LocalContext* method), 34

block_call() (*dpdispatcher.ssh_context.SSHContext* method), 78

block_call() (*dpdispatcher.SSHContext* method), 42

block_checkcall() (*dpdispatcher.SSHContext* method), 42

patcher.lazy_local_context.LazyLocalContext
method), 64
 block_checkcall() (*dpdispatcher.LazyLocalContext*
method), 32
 block_checkcall() (*dpdis-*
patcher.local_context.LocalContext *method*),
 66
 block_checkcall() (*dpdispatcher.LocalContext*
method), 34
 block_checkcall() (*dpdis-*
patcher.ssh_context.SSHContext *method*),
 78
 block_checkcall() (*dpdispatcher.SSHContext*
method), 42
 Bohrium (class in *dpdispatcher.dp_cloud_server*), 58
 BohriumContext (class in *dpdis-*
patcher.dp_cloud_server_context), 59

C

call() (*dpdispatcher.lazy_local_context.LazyLocalContext*
method), 64
 call() (*dpdispatcher.LazyLocalContext* *method*), 32
 call() (*dpdispatcher.local_context.LocalContext*
method), 66
 call() (*dpdispatcher.LocalContext* *method*), 34
 call() (*dpdispatcher.ssh_context.SSHContext* *method*),
 79
 call() (*dpdispatcher.SSHContext* *method*), 42
 check_all_finished() (*dpdispatcher.Submission*
method), 47
 check_all_finished() (*dpdis-*
patcher.submission.Submission *method*),
 85
 check_file_exists() (*dpdis-*
patcher.dp_cloud_server_context.BohriumContext
method), 59
 check_file_exists() (*dpdis-*
patcher.hdfs_context.HDFSContext *method*),
 62
 check_file_exists() (*dpdispatcher.HDFSContext*
method), 28
 check_file_exists() (*dpdis-*
patcher.lazy_local_context.LazyLocalContext
method), 64
 check_file_exists() (*dpdis-*
patcher.LazyLocalContext *method*), 32
 check_file_exists() (*dpdis-*
patcher.local_context.LocalContext *method*),
 66
 check_file_exists() (*dpdispatcher.LocalContext*
method), 34
 check_file_exists() (*dpdis-*
patcher.openapi_context.OpenAPIContext
method), 72
 check_file_exists() (*dpdispatcher.OpenAPIContext*
method), 39
 check_file_exists() (*dpdis-*
patcher.ssh_context.SSHContext *method*),
 79
 check_file_exists() (*dpdispatcher.SSHContext*
method), 42
 check_finish() (*dpdis-*
patcher.base_context.BaseContext *method*),
 55
 check_finish() (*dpdis-*
patcher.lazy_local_context.LazyLocalContext
method), 64
 check_finish() (*dpdispatcher.LazyLocalContext*
method), 32
 check_finish() (*dpdis-*
patcher.local_context.LocalContext *method*),
 66
 check_finish() (*dpdispatcher.LocalContext* *method*),
 34
 check_finish() (*dpdispatcher.ssh_context.SSHContext*
method), 79
 check_finish() (*dpdispatcher.SSHContext* *method*), 42
 check_finish_tag() (*dpdis-*
patcher.distributed_shell.DistributedShell
method), 57
 check_finish_tag() (*dpdispatcher.DistributedShell*
method), 25
 check_finish_tag() (*dpdis-*
patcher.dp_cloud_server.Bohrium *method*),
 58
 check_finish_tag() (*dpdispatcher.Fugaku* *method*),
 27
 check_finish_tag() (*dpdispatcher.fugaku.Fugaku*
method), 61
 check_finish_tag() (*dpdispatcher.LSF* *method*), 31
 check_finish_tag() (*dpdispatcher.lsf.LSF* *method*),
 67
 check_finish_tag() (*dpdispatcher.Machine* *method*),
 35
 check_finish_tag() (*dpdispatcher.machine.Machine*
method), 69
 check_finish_tag() (*dpdispatcher.OpenAPI* *method*),
 38
 check_finish_tag() (*dpdispatcher.openapi.OpenAPI*
method), 71
 check_finish_tag() (*dpdispatcher.PBS* *method*), 40
 check_finish_tag() (*dpdispatcher.pbs.PBS* *method*),
 73
 check_finish_tag() (*dpdispatcher.Shell* *method*), 44
 check_finish_tag() (*dpdispatcher.shell.Shell*
method), 75
 check_finish_tag() (*dpdispatcher.Slurm* *method*), 45
 check_finish_tag() (*dpdispatcher.slurm.Slurm*

`method`), 76
`check_finish_tag()` (`dpdispatcher.slurm.SlurmJobArray` method), 77
`check_home_file_exits()` (`dpdispatcher.dp_cloud_server_context.BohriumContext` method), 59
`check_home_file_exits()` (`dpdispatcher.openapi_context.OpenAPIContext` method), 72
`check_home_file_exits()` (`dpdispatcher.OpenAPIContext` method), 39
`check_if_recover()` (`dpdispatcher.dp_cloud_server.Bohrium` method), 58
`check_if_recover()` (`dpdispatcher.Machine` method), 35
`check_if_recover()` (`dpdispatcher.machine.Machine` method), 69
`check_if_recover()` (`dpdispatcher.OpenAPI` method), 38
`check_if_recover()` (`dpdispatcher.openapi.OpenAPI` method), 71
`check_ratio_unfinished()` (`dpdispatcher.Submission` method), 47
`check_ratio_unfinished()` (`dpdispatcher.submission.Submission` method), 85
`check_status()` (`dpdispatcher.distributed_shell.DistributedShell` method), 57
`check_status()` (`dpdispatcher.DistributedShell` method), 25
`check_status()` (`dpdispatcher.dp_cloud_server.Bohrium` method), 58
`check_status()` (`dpdispatcher.Fugaku` method), 27
`check_status()` (`dpdispatcher.fugaku.Fugaku` method), 61
`check_status()` (`dpdispatcher.LSF` method), 31
`check_status()` (`dpdispatcher.lsf.LSF` method), 67
`check_status()` (`dpdispatcher.Machine` method), 35
`check_status()` (`dpdispatcher.machine.Machine` method), 69
`check_status()` (`dpdispatcher.OpenAPI` method), 38
`check_status()` (`dpdispatcher.openapi.OpenAPI` method), 71
`check_status()` (`dpdispatcher.PBS` method), 40
`check_status()` (`dpdispatcher.pbs.PBS` method), 73
`check_status()` (`dpdispatcher.pbs.Torque` method), 74
`check_status()` (`dpdispatcher.Shell` method), 44
`check_status()` (`dpdispatcher.shell.Shell` method), 75
`check_status()` (`dpdispatcher.Slurm` method), 45
`check_status()` (`dpdispatcher.slurm.Slurm` method), 76
`check_status()` (`dpdispatcher.slurm.SlurmJobArray` method), 77
`check_status()` (`dpdispatcher.Torque` method), 51
`clean()` (`dpdispatcher.base_context.BaseContext` method), 55
`clean()` (`dpdispatcher.dp_cloud_server_context.BohriumContext` method), 59
`clean()` (`dpdispatcher.hdfs_context.HDFSContext` method), 63
`clean()` (`dpdispatcher.HDFSContext` method), 28
`clean()` (`dpdispatcher.lazy_local_context.LazyLocalContext` method), 65
`clean()` (`dpdispatcher.LazyLocalContext` method), 32
`clean()` (`dpdispatcher.local_context.LocalContext` method), 66
`clean()` (`dpdispatcher.LocalContext` method), 34
`clean()` (`dpdispatcher.openapi_context.OpenAPIContext` method), 72
`clean()` (`dpdispatcher.OpenAPIContext` method), 39
`clean()` (`dpdispatcher.ssh_context.SSHContext` method), 79
`clean()` (`dpdispatcher.SSHContext` method), 42
`clean_asynchronously:`
 `machine/clean_asynchronously (Argument)`, 13
`clean_jobs()` (`dpdispatcher.Submission` method), 47
`clean_jobs()` (`dpdispatcher.submission.Submission` method), 85
`Client` (class in `dpdispatcher.dpcloudserver`), 51
`Client` (class in `dpdispatcher.dpcloudserver.client`), 52
`close()` (`dpdispatcher.ssh_context.SSHContext` method), 79
`close()` (`dpdispatcher.ssh_context.SSHSession` method), 80
`close()` (`dpdispatcher.SSHContext` method), 42
`command:`
 `task/command (Argument)`, 23
`completing` (`dpdispatcher.JobStatus.JobStatus` attribute), 54
`context_type:`
 `machine/context_type (Argument)`, 13
`copy_from_local()` (`dpdispatcher.hdfs_cli.HDFS` static method), 62
`copy_to_local()` (`dpdispatcher.hdfs_cli.HDFS` static method), 62
`cpu_per_node:`
 `resources/cpu_per_node (Argument)`, 17
`custom_flags:`
 `resources/custom_flags (Argument)`, 17
`custom_gpu_line:`
 `resources[LSF]/kwargs/custom_gpu_line (Argument)`, 20
 `resources[SlurmJobArray]/kwargs/custom_gpu_line (Argument)`, 20
 `resources[Slurm]/kwargs/custom_gpu_line`

(Argument), 19

D

DATAERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53

DBERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53

default_resources() (*dpdispatcher.Fugaku* method), 27

default_resources() (*dpdispatcher.fugaku.Fugaku* method), 61

default_resources() (*dpdispatcher.LSF* method), 31

default_resources() (*dpdispatcher.lsf.LSF* method), 67

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

default_resources() (*dpdispatcher.machine.Machine* method), 69

default_resources() (*dpdispatcher.PBS* method), 40

default_resources() (*dpdispatcher.pbs.PBS* method), 73

default_resources() (*dpdispatcher.Shell* method), 44

default_resources() (*dpdispatcher.shell.Shell* method), 75

default_resources() (*dpdispatcher.Slurm* method), 45

default_resources() (*dpdispatcher.slurm.Slurm* method), 76

deserialize() (*dpdispatcher.Job* class method), 29

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

deserialize() (*dpdispatcher.machine.Machine* class method), 69

deserialize() (*dpdispatcher.Resources* class method), 41

deserialize() (*dpdispatcher.Submission* class method), 47

deserialize() (*dpdispatcher.submission.Job* class method), 81

deserialize() (*dpdispatcher.submission.Resources* class method), 83

deserialize() (*dpdispatcher.submission.Submission* class method), 85

deserialize() (*dpdispatcher.submission.Task* class method), 87

deserialize() (*dpdispatcher.Task* class method), 49

DistributedShell (class in *dpdispatcher*), 25

DistributedShell (class in *dpdispatcher.distributed_shell*), 56

do_submit() (*dpdispatcher.distributed_shell.DistributedShell* method), 57

do_submit() (*dpdispatcher.DistributedShell* method), 25

do_submit() (*dpdispatcher.dp_cloud_server.Bohrium* method), 58

do_submit() (*dpdispatcher.Fugaku* method), 27

do_submit() (*dpdispatcher.fugaku.Fugaku* method), 61

do_submit() (*dpdispatcher.LSF* method), 31

do_submit() (*dpdispatcher.lsf.LSF* method), 68

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

do_submit() (*dpdispatcher.machine.Machine* method), 69

do_submit() (*dpdispatcher.OpenAPI* method), 38

do_submit() (*dpdispatcher.openapi.OpenAPI* method), 71

do_submit() (*dpdispatcher.PBS* method), 40

do_submit() (*dpdispatcher.pbs.PBS* method), 73

do_submit() (*dpdispatcher.Shell* method), 44

do_submit() (*dpdispatcher.shell.Shell* method), 75

do_submit() (*dpdispatcher.Slurm* method), 45

do_submit() (*dpdispatcher.slurm.Slurm* method), 76

download() (*dpdispatcher.base_context.BaseContext* method), 55

download() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 59

download() (*dpdispatcher.dpcloudserver.Client* method), 51

download() (*dpdispatcher.dpcloudserver.client.Client* method), 52

download() (*dpdispatcher.hdfs_context.HDFSContext* method), 63

download() (*dpdispatcher.HDFSContext* method), 28

download() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 65

download() (*dpdispatcher.LazyLocalContext* method), 32

download() (*dpdispatcher.local_context.LocalContext* method), 66

download() (*dpdispatcher.LocalContext* method), 34

download() (*dpdispatcher.openapi_context.OpenAPIContext* method), 72

download() (*dpdispatcher.OpenAPIContext* method), 39

download() (*dpdispatcher.ssh_context.SSHContext* method), 79

download() (*dpdispatcher.SSHContext* method), 43

download_from_url() (*dpdispatcher.dpcloudserver.Client* method), 52

download_from_url() (*dpdispatcher.dpcloudserver.client.Client* method), 52

download_jobs() (*dpdispatcher.Submission* method), 48

download_jobs() (*dpdispatcher.submission.Submission* method), 85

DpCloudServer (in module *dpdispatcher*), 26

DpCloudServer (in module *dpdis-*

patcher.dp_cloud_server), 59
 DpCloudServerContext (in module *dpdispatcher*), 26
 DpCloudServerContext (in module *dpdispatcher.dp_cloud_server_context*), 60
 dpdispatcher
 module, 25
 dpdispatcher.arginfo
 module, 54
 dpdispatcher.base_context
 module, 54
 dpdispatcher.distributed_shell
 module, 56
 dpdispatcher.dp_cloud_server
 module, 58
 dpdispatcher.dp_cloud_server_context
 module, 59
 dpdispatcher.dpcloudserver
 module, 51
 dpdispatcher.dpcloudserver.client
 module, 52
 dpdispatcher.dpcloudserver.config
 module, 53
 dpdispatcher.dpcloudserver.retcode
 module, 53
 dpdispatcher.dpcloudserver.zip_file
 module, 54
 dpdispatcher.dpdisp
 module, 60
 dpdispatcher.fugaku
 module, 60
 dpdispatcher.hdfs_cli
 module, 61
 dpdispatcher.hdfs_context
 module, 62
 dpdispatcher.JobStatus
 module, 54
 dpdispatcher.lazy_local_context
 module, 64
 dpdispatcher.local_context
 module, 65
 dpdispatcher.lsf
 module, 67
 dpdispatcher.machine
 module, 68
 dpdispatcher.openapi
 module, 71
 dpdispatcher.openapi_context
 module, 72
 dpdispatcher.pbs
 module, 73
 dpdispatcher.shell
 module, 74
 dpdispatcher.slurm
 module, 76

dpdispatcher.ssh_context
 module, 78
 dpdispatcher.submission
 module, 81
 dpdispatcher.utils
 module, 88

E

email:
 machine[BohriumContext]/remote_profile/email
 (*Argument*), 15
 ensure_alive() (*dpdispatcher.ssh_context.SSHSession*
 method), 80
 envs:
 resources/envs (*Argument*), 19
 errlog:
 task/errlog (*Argument*), 23
 exec_command() (*dpdispatcher.ssh_context.SSHSession*
 method), 80
 exists() (*dpdispatcher.hdfs_cli.HDFS static method*),
 62

F

finished (*dpdispatcher.JobStatus.JobStatus attribute*),
 54
 forward_files:
 task/forward_files (*Argument*), 23
 Fugaku (class in *dpdispatcher*), 26
 Fugaku (class in *dpdispatcher.fugaku*), 60

G

gen_command_env_cuda_devices() (*dpdis-*
 patcher.Machine method), 35
 gen_command_env_cuda_devices() (*dpdis-*
 patcher.machine.Machine method), 69
 gen_local_script() (*dpdis-*
 patcher.dp_cloud_server.Bohrium method),
 58
 gen_local_script() (*dpdispatcher.OpenAPI method*),
 38
 gen_local_script() (*dpdispatcher.openapi.OpenAPI*
 method), 71
 gen_script() (*dpdispatcher.dp_cloud_server.Bohrium*
 method), 58
 gen_script() (*dpdispatcher.Fugaku method*), 27
 gen_script() (*dpdispatcher.fugaku.Fugaku method*), 61
 gen_script() (*dpdispatcher.LSF method*), 31
 gen_script() (*dpdispatcher.lsf.LSF method*), 68
 gen_script() (*dpdispatcher.Machine method*), 35
 gen_script() (*dpdispatcher.machine.Machine method*),
 69
 gen_script() (*dpdispatcher.OpenAPI method*), 38
 gen_script() (*dpdispatcher.openapi.OpenAPI*
 method), 71

`gen_script()` (*dpdispatcher.PBS method*), 40
`gen_script()` (*dpdispatcher.pbs.PBS method*), 73
`gen_script()` (*dpdispatcher.Shell method*), 44
`gen_script()` (*dpdispatcher.shell.Shell method*), 75
`gen_script()` (*dpdispatcher.Slurm method*), 45
`gen_script()` (*dpdispatcher.slurm.Slurm method*), 76
`gen_script_command()` (*dpdispatcher.Machine method*), 35
`gen_script_command()` (*dpdispatcher.machine.Machine method*), 69
`gen_script_command()` (*dpdispatcher.slurm.SlurmJobArray method*), 77
`gen_script_custom_flags_lines()` (*dpdispatcher.Machine method*), 35
`gen_script_custom_flags_lines()` (*dpdispatcher.machine.Machine method*), 69
`gen_script_end()` (*dpdispatcher.distributed_shell.DistributedShell method*), 57
`gen_script_end()` (*dpdispatcher.DistributedShell method*), 26
`gen_script_end()` (*dpdispatcher.Machine method*), 36
`gen_script_end()` (*dpdispatcher.machine.Machine method*), 69
`gen_script_end()` (*dpdispatcher.slurm.SlurmJobArray method*), 77
`gen_script_env()` (*dpdispatcher.distributed_shell.DistributedShell method*), 57
`gen_script_env()` (*dpdispatcher.DistributedShell method*), 26
`gen_script_env()` (*dpdispatcher.Machine method*), 36
`gen_script_env()` (*dpdispatcher.machine.Machine method*), 69
`gen_script_header()` (*dpdispatcher.distributed_shell.DistributedShell method*), 57
`gen_script_header()` (*dpdispatcher.DistributedShell method*), 26
`gen_script_header()` (*dpdispatcher.dp_cloud_server.Bohrium method*), 58
`gen_script_header()` (*dpdispatcher.Fugaku method*), 27
`gen_script_header()` (*dpdispatcher.fugaku.Fugaku method*), 61
`gen_script_header()` (*dpdispatcher.LSF method*), 31
`gen_script_header()` (*dpdispatcher.lsf.LSF method*), 68
`gen_script_header()` (*dpdispatcher.Machine method*), 36
`gen_script_header()` (*dpdispatcher.machine.Machine method*), 69
`gen_script_header()` (*dpdispatcher.OpenAPI method*), 38
`gen_script_header()` (*dpdispatcher.openapi.OpenAPI method*), 71
`gen_script_header()` (*dpdispatcher.PBS method*), 40
`gen_script_header()` (*dpdispatcher.pbs.PBS method*), 73
`gen_script_header()` (*dpdispatcher.pbs.Torque method*), 74
`gen_script_header()` (*dpdispatcher.Shell method*), 44
`gen_script_header()` (*dpdispatcher.shell.Shell method*), 75
`gen_script_header()` (*dpdispatcher.Slurm method*), 45
`gen_script_header()` (*dpdispatcher.slurm.Slurm method*), 76
`gen_script_header()` (*dpdispatcher.slurm.SlurmJobArray method*), 77
`gen_script_header()` (*dpdispatcher.Torque method*), 51
`gen_script_wait()` (*dpdispatcher.Machine method*), 36
`gen_script_wait()` (*dpdispatcher.machine.Machine method*), 69
`generate_jobs()` (*dpdispatcher.Submission method*), 48
`generate_jobs()` (*dpdispatcher.submission.Submission method*), 85
`generate_totp()` (*in module dpdispatcher.utils*), 88
`get()` (*dpdispatcher.dpcloudserver.Client method*), 52
`get()` (*dpdispatcher.dpcloudserver.client.Client method*), 52
`get()` (*dpdispatcher.ssh_context.SSHSession method*), 80
`get_hash()` (*dpdispatcher.Job method*), 29
`get_hash()` (*dpdispatcher.Submission method*), 48
`get_hash()` (*dpdispatcher.submission.Job method*), 81
`get_hash()` (*dpdispatcher.submission.Submission method*), 86
`get_hash()` (*dpdispatcher.submission.Task method*), 87
`get_hash()` (*dpdispatcher.Task method*), 50
`get_job_detail()` (*dpdispatcher.dpcloudserver.Client method*), 52
`get_job_detail()` (*dpdispatcher.dpcloudserver.client.Client method*), 52
`get_job_result_url()` (*dpdispatcher.dpcloudserver.Client method*), 52
`get_job_result_url()` (*dpdispatcher.dpcloudserver.client.Client method*), 52
`get_job_root()` (*dpdispatcher.hdfs_context.HDFSContext method*), 63

`get_job_root()` (*dpdispatcher.HDFSContext* method), 28
`get_job_root()` (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 65
`get_job_root()` (*dpdispatcher.LazyLocalContext* method), 33
`get_job_root()` (*dpdispatcher.local_context.LocalContext* method), 66
`get_job_root()` (*dpdispatcher.LocalContext* method), 34
`get_job_root()` (*dpdispatcher.ssh_context.SSHContext* method), 79
`get_job_root()` (*dpdispatcher.SSHContext* method), 43
`get_job_state()` (*dpdispatcher.Job* method), 29
`get_job_state()` (*dpdispatcher.submission.Job* method), 81
`get_log()` (*dpdispatcher.dpcloudserver.Client* method), 52
`get_log()` (*dpdispatcher.dpcloudserver.client.Client* method), 52
`get_return()` (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 65
`get_return()` (*dpdispatcher.LazyLocalContext* method), 33
`get_return()` (*dpdispatcher.local_context.LocalContext* method), 66
`get_return()` (*dpdispatcher.LocalContext* method), 34
`get_return()` (*dpdispatcher.ssh_context.SSHContext* method), 79
`get_return()` (*dpdispatcher.SSHContext* method), 43
`get_sha256()` (in module *dpdispatcher.utils*), 88
`get_ssh_client()` (*dpdispatcher.ssh_context.SSHSession* method), 80
`get_task_state()` (*dpdispatcher.submission.Task* method), 88
`get_task_state()` (*dpdispatcher.Task* method), 50
`get_tasks_list()` (*dpdispatcher.dpcloudserver.Client* method), 52
`get_tasks_list()` (*dpdispatcher.dpcloudserver.client.Client* method), 53
`gpu_exclusive:`
 resources[LSF]/kwargs/gpu_exclusive (Argument), 20
`gpu_new_syntax:`
 resources[LSF]/kwargs/gpu_new_syntax (Argument), 20
`gpu_per_node:`
 resources/gpu_per_node (Argument), 17
`gpu_usage:`
 resources[LSF]/kwargs/gpu_usage (Argument), 20
`group_size:`
 resources/group_size (Argument), 17
H
`handle_unexpected_job_state()` (*dpdispatcher.Job* method), 30
`handle_unexpected_job_state()` (*dpdispatcher.submission.Job* method), 82
`handle_unexpected_submission_state()` (*dpdispatcher.Submission* method), 48
`handle_unexpected_submission_state()` (*dpdispatcher.submission.Submission* method), 86
HDFS (class in *dpdispatcher.hdfs_cli*), 61
HDFSContext (class in *dpdispatcher*), 27
HDFSContext (class in *dpdispatcher.hdfs_context*), 62
hostname:
 machine[SSHContext]/remote_profile/hostname (Argument), 14
`hotp()` (in module *dpdispatcher.utils*), 89
I
`if_cuda_multi_devices:`
 resources/strategy/if_cuda_multi_devices (Argument), 18
`info()` (in module *dpdispatcher*), 51
`input_data:`
 machine[BohriumContext]/remote_profile/input_data (Argument), 16
`inter_handler()` (*dpdispatcher.ssh_context.SSHSession* method), 80
IOERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53
J
Job (class in *dpdispatcher*), 29
Job (class in *dpdispatcher.submission*), 81
`job_create()` (*dpdispatcher.dpcloudserver.Client* method), 52
`job_create()` (*dpdispatcher.dpcloudserver.client.Client* method), 53
`job_to_json()` (*dpdispatcher.Job* method), 30
`job_to_json()` (*dpdispatcher.submission.Job* method), 82
JobStatus (class in *dpdispatcher.JobStatus*), 54
K
`keep_backup:`
 machine[BohriumContext]/remote_profile/keep_backup (Argument), 16
`key_filename:`
 machine[SSHContext]/remote_profile/key_filename (Argument), 14

kill() (*dpdispatcher.LSF method*), 31
 kill() (*dpdispatcher.lsf.LSF method*), 68
 kill() (*dpdispatcher.Machine method*), 36
 kill() (*dpdispatcher.machine.Machine method*), 69
 kill() (*dpdispatcher.PBS method*), 40
 kill() (*dpdispatcher.pbs.PBS method*), 73
 kill() (*dpdispatcher.Shell method*), 44
 kill() (*dpdispatcher.shell.Shell method*), 75
 kill() (*dpdispatcher.Slurm method*), 45
 kill() (*dpdispatcher.slurm.Slurm method*), 76

kwargs:

resources[Bohrium]/kwargs (*Argument*), 21
 resources[DistributedShell]/kwargs (*Argument*), 19
 resources[Fugaku]/kwargs (*Argument*), 20
 resources[LSF]/kwargs (*Argument*), 19
 resources[OpenAPI]/kwargs (*Argument*), 21
 resources[PBS]/kwargs (*Argument*), 21
 resources[Shell]/kwargs (*Argument*), 21
 resources[SlurmJobArray]/kwargs (*Argument*), 20
 resources[Slurm]/kwargs (*Argument*), 19
 resources[Torque]/kwargs (*Argument*), 21

L

LazyLocalContext (*class in dpdispatcher*), 31

LazyLocalContext (*class in dpdispatcher.lazy_local_context*), 64

Lebesgue (*in module dpdispatcher*), 33

Lebesgue (*in module dpdispatcher.dp_cloud_server*), 59

LebesgueContext (*in module dpdispatcher*), 33

LebesgueContext (*in module dpdispatcher.dp_cloud_server_context*), 60

load_from_dict() (*dpdispatcher.base_context.BaseContext class method*), 55

load_from_dict() (*dpdispatcher.dp_cloud_server_context.BohriumContext class method*), 59

load_from_dict() (*dpdispatcher.hdfs_context.HDFSContext class method*), 63

load_from_dict() (*dpdispatcher.HDFSContext class method*), 28

load_from_dict() (*dpdispatcher.lazy_local_context.LazyLocalContext class method*), 65

load_from_dict() (*dpdispatcher.LazyLocalContext class method*), 33

load_from_dict() (*dpdispatcher.local_context.LocalContext class method*), 66

load_from_dict() (*dpdispatcher.LocalContext class method*), 34

load_from_dict() (*dpdispatcher.Machine class method*), 36

load_from_dict() (*dpdispatcher.machine.Machine class method*), 70

load_from_dict() (*dpdispatcher.openapi_context.OpenAPIContext class method*), 72

load_from_dict() (*dpdispatcher.OpenAPIContext class method*), 39

load_from_dict() (*dpdispatcher.Resources class method*), 41

load_from_dict() (*dpdispatcher.ssh_context.SSHContext class method*), 79

load_from_dict() (*dpdispatcher.SSHContext class method*), 43

load_from_dict() (*dpdispatcher.submission.Resources class method*), 83

load_from_dict() (*dpdispatcher.submission.Task class method*), 88

load_from_dict() (*dpdispatcher.Task class method*), 50

load_from_json() (*dpdispatcher.Machine class method*), 36

load_from_json() (*dpdispatcher.machine.Machine class method*), 70

load_from_json() (*dpdispatcher.Resources class method*), 41

load_from_json() (*dpdispatcher.submission.Resources class method*), 83

load_from_json() (*dpdispatcher.submission.Task class method*), 88

load_from_json() (*dpdispatcher.Task class method*), 50

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

LocalContext (*class in dpdispatcher*), 33

LocalContext (*class in dpdispatcher.local_context*), 65

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

LSF (*class in dpdispatcher*), 30

LSF (*class in dpdispatcher.lsf*), 67

M

machine (*Argument*)
 machine:, 13

Machine (*class in dpdispatcher*), 34

Machine (*class in dpdispatcher.machine*), 68

machine/batch_type (*Argument*)

batch_type:, 13

machine/clean_asynchronously (*Argument*)

```

    clean_asynchronously:, 13
machine/context_type (Argument)
    context_type:, 13
machine/local_root (Argument)
    local_root:, 13
machine/remote_root (Argument)
    remote_root:, 13
machine:
    machine (Argument), 13
machine_arginfo() (dpdispatcher.base_context.BaseContext class method), 55
machine_subfields() (dpdispatcher.base_context.BaseContext class method), 55
machine_subfields() (dpdispatcher.dp_cloud_server_context.BohriumContext class method), 59
machine_subfields() (dpdispatcher.ssh_context.SSHContext class method), 79
machine_subfields() (dpdispatcher.SSHContext class method), 43
machine[BohriumContext]/remote_profile (Argument)
    remote_profile:, 15
machine[BohriumContext]/remote_profile/email (Argument)
    email:, 15
machine[BohriumContext]/remote_profile/input_data (Argument)
    input_data:, 16
machine[BohriumContext]/remote_profile/keep_backup (Argument)
    keep_backup:, 16
machine[BohriumContext]/remote_profile/password (Argument)
    password:, 15
machine[BohriumContext]/remote_profile/program_id (Argument)
    program_id:, 15
machine[HDFSContext]/remote_profile (Argument)
    remote_profile:, 15
machine[LazyLocalContext]/remote_profile (Argument)
    remote_profile:, 16
machine[LocalContext]/remote_profile (Argument)
    remote_profile:, 14
machine[OpenAPIContext]/remote_profile (Argument)
    remote_profile:, 16
machine[SSHContext]/remote_profile (Argument)
    remote_profile:, 14
machine[SSHContext]/remote_profile/hostname (Argument)
    hostname:, 14
machine[SSHContext]/remote_profile/key_filename (Argument)
    key_filename:, 14
machine[SSHContext]/remote_profile/look_for_keys (Argument)
    look_for_keys:, 15
machine[SSHContext]/remote_profile/passphrase (Argument)
    passphrase:, 14
machine[SSHContext]/remote_profile/password (Argument)
    password:, 14
machine[SSHContext]/remote_profile/port (Argument)
    port:, 14
machine[SSHContext]/remote_profile/tar_compress (Argument)
    tar_compress:, 15
machine[SSHContext]/remote_profile/timeout (Argument)
    timeout:, 14
machine[SSHContext]/remote_profile/totp_secret (Argument)
    totp_secret:, 15
machine[SSHContext]/remote_profile/username (Argument)
    username:, 14
main() (in module dpdispatcher.dpdisp), 60
map_dp_job_state() (dpdispatcher.dp_cloud_server.Bohrium static method), 59
map_dp_job_state() (dpdispatcher.OpenAPI static method), 38
map_dp_job_state() (dpdispatcher.openapi.OpenAPI static method), 71
mkdir() (dpdispatcher.hdfs_cli.HDFS static method), 62
module
    dpdispatcher, 25
    dpdispatcher_arginfo, 54
    dpdispatcher_base_context, 54
    dpdispatcher_distributed_shell, 56
    dpdispatcher_dp_cloud_server, 58
    dpdispatcher_dp_cloud_server_context, 59
    dpdispatcher_dpcloudserver, 51
    dpdispatcher_dpcloudserver_client, 52
    dpdispatcher_dpcloudserver_config, 53
    dpdispatcher_dpcloudserver_retcode, 53
    dpdispatcher_dpcloudserver_zip_file, 54
    dpdispatcher_dpdisp, 60
    dpdispatcher_fugaku, 60

```

dpdispatcher.hdfs_cli, 61
 dpdispatcher.hdfs_context, 62
 dpdispatcher.JobStatus, 54
 dpdispatcher.lazy_local_context, 64
 dpdispatcher.local_context, 65
 dpdispatcher.lsf, 67
 dpdispatcher.machine, 68
 dpdispatcher.openapi, 71
 dpdispatcher.openapi_context, 72
 dpdispatcher.pbs, 73
 dpdispatcher.shell, 74
 dpdispatcher.slurm, 76
 dpdispatcher.ssh_context, 78
 dpdispatcher.submission, 81
 dpdispatcher.utils, 88
 module_list:
 resources/module_list (Argument), 18
 module_purge:
 resources/module_purge (Argument), 18
 module_unload_list:
 resources/module_unload_list (Argument), 18
 move() (dpdispatcher.hdfs_cli.HDFS static method), 62

N

NODATA (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 53
 number_node:
 resources/number_node (Argument), 17

O

OK (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 53
 OpenAPI (class in dpdispatcher), 37
 OpenAPI (class in dpdispatcher.openapi), 71
 OpenAPIContext (class in dpdispatcher), 38
 OpenAPIContext (class in dpdispatcher.openapi_context), 72
 options (dpdispatcher.base_context.BaseContext attribute), 55
 options (dpdispatcher.Machine attribute), 36
 options (dpdispatcher.machine.Machine attribute), 70
 outlog:
 task/outlog (Argument), 23

P

para_deg:
 resources/para_deg (Argument), 18
 PARAMERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 53
 passphrase:
 machine[SSHContext]/remote_profile/passphrase (Argument), 14
 password:

 machine[BohriumContext]/remote_profile/password (Argument), 15
 machine[SSHContext]/remote_profile/password (Argument), 14
 PBS (class in dpdispatcher), 39
 PBS (class in dpdispatcher.pbs), 73
 port:
 machine[SSHContext]/remote_profile/port (Argument), 14
 post() (dpdispatcher.dpcloudserver.Client method), 52
 post() (dpdispatcher.dpcloudserver.client.Client method), 53
 prepend_script:
 resources/prepend_script (Argument), 19
 program_id:
 machine[BohriumContext]/remote_profile/program_id (Argument), 15
 put() (dpdispatcher.ssh_context.SSHSession method), 80
 PWDERR (dpdispatcher.dpcloudserver.retcode.RETCODE attribute), 53

Q

queue_name:
 resources/queue_name (Argument), 17

R

ratio_unfinished:
 resources/strategy/ratio_unfinished (Argument), 18
 read() (dpdispatcher.lazy_local_context.SPRetObj method), 65
 read() (dpdispatcher.local_context.SPRetObj method), 67
 read_file() (dpdispatcher.base_context.BaseContext method), 55
 read_file() (dpdispatcher.dp_cloud_server_context.BohriumContext method), 60
 read_file() (dpdispatcher.hdfs_context.HDFSContext method), 63
 read_file() (dpdispatcher.HDFSContext method), 28
 read_file() (dpdispatcher.lazy_local_context.LazyLocalContext method), 65
 read_file() (dpdispatcher.LazyLocalContext method), 33
 read_file() (dpdispatcher.local_context.LocalContext method), 66
 read_file() (dpdispatcher.LocalContext method), 34
 read_file() (dpdispatcher.openapi_context.OpenAPIContext method), 72
 read_file() (dpdispatcher.OpenAPIContext method), 39
 read_file() (dpdispatcher.ssh_context.SSHContext method), 79

`read_file()` (*dpdispatcher.SSHContext* method), 43
`read_hdfs_file()` (*dpdispatcher.hdfs_cli.HDFS* static method), 62
`read_home_file()` (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 60
`read_home_file()` (*dpdispatcher.openapi_context.OpenAPIContext* method), 72
`read_home_file()` (*dpdispatcher.OpenAPIContext* method), 39
`readlines()` (*dpdispatcher.lazy_local_context.SPRetObj* method), 65
`readlines()` (*dpdispatcher.local_context.SPRetObj* method), 67
`refresh_token()` (*dpdispatcher.dpcloudserver.Client* method), 52
`refresh_token()` (*dpdispatcher.dpcloudserver.client.Client* method), 53
`register_job_id()` (*dpdispatcher.Job* method), 30
`register_job_id()` (*dpdispatcher.submission.Job* method), 82
`register_task()` (*dpdispatcher.Submission* method), 48
`register_task()` (*dpdispatcher.submission.Submission* method), 86
`register_task_list()` (*dpdispatcher.Submission* method), 48
`register_task_list()` (*dpdispatcher.submission.Submission* method), 86
`remote` (*dpdispatcher.ssh_context.SSHSession* property), 80
`remote_profile:`
 `machine[BohriumContext]/remote_profile` (*Argument*), 15
 `machine[HDFSContext]/remote_profile` (*Argument*), 15
 `machine[LazyLocalContext]/remote_profile` (*Argument*), 16
 `machine[LocalContext]/remote_profile` (*Argument*), 14
 `machine[OpenAPIContext]/remote_profile` (*Argument*), 16
 `machine[SSHContext]/remote_profile` (*Argument*), 14
`remote_root:`
 `machine/remote_root` (*Argument*), 13
`remove()` (*dpdispatcher.hdfs_cli.HDFS* static method), 62
`remove_unfinished_tasks()` (*dpdispatcher.Submission* method), 48
`remove_unfinished_tasks()` (*dpdispatcher.submission.Submission* method), 86
`REQERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53
`RequestInfoException`, 53
`resources` (*Argument*)
 `resources:`, 17
`Resources` (*class in dpdispatcher*), 40
`Resources` (*class in dpdispatcher.submission*), 82
`resources/append_script` (*Argument*)
 `append_script:`, 19
`resources/batch_type` (*Argument*)
 `batch_type:`, 19
`resources/cpu_per_node` (*Argument*)
 `cpu_per_node:`, 17
`resources/custom_flags` (*Argument*)
 `custom_flags:`, 17
`resources/envs` (*Argument*)
 `envs:`, 19
`resources/gpu_per_node` (*Argument*)
 `gpu_per_node:`, 17
`resources/group_size` (*Argument*)
 `group_size:`, 17
`resources/module_list` (*Argument*)
 `module_list:`, 18
`resources/module_purge` (*Argument*)
 `module_purge:`, 18
`resources/module_unload_list` (*Argument*)
 `module_unload_list:`, 18
`resources/number_node` (*Argument*)
 `number_node:`, 17
`resources/para_deg` (*Argument*)
 `para_deg:`, 18
`resources/prepend_script` (*Argument*)
 `prepend_script:`, 19
`resources/queue_name` (*Argument*)
 `queue_name:`, 17
`resources/source_list` (*Argument*)
 `source_list:`, 18
`resources/strategy` (*Argument*)
 `strategy:`, 18
`resources/strategy/if_cuda_multi_devices` (*Argument*)
 `if_cuda_multi_devices:`, 18
`resources/strategy/ratio_unfinished` (*Argument*)
 `ratio_unfinished:`, 18
`resources/wait_time` (*Argument*)
 `wait_time:`, 19
`resources:`
 `resources` (*Argument*), 17
`resources_arginfo()` (*dpdispatcher.Machine* class method), 36

resources_arginfo() (*dpdispatcher.machine.Machine* class method), 70

resources_subfields() (*dpdispatcher.LSF* class method), 31

resources_subfields() (*dpdispatcher.lsf.LSF* class method), 68

resources_subfields() (*dpdispatcher.Machine* class method), 36

resources_subfields() (*dpdispatcher.machine.Machine* class method), 70

resources_subfields() (*dpdispatcher.Slurm* class method), 45

resources_subfields() (*dpdispatcher.slurm.Slurm* class method), 77

resources_subfields() (*dpdispatcher.slurm.SlurmJobArray* class method), 77

resources[Bohrium]/kwargs (Argument) kwargs:, 21

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

resources[Fugaku]/kwargs (Argument) kwargs:, 20

resources[LSF]/kwargs (Argument) kwargs:, 19

resources[LSF]/kwargs/custom_gpu_line (Argument) custom_gpu_line:, 20

resources[LSF]/kwargs/gpu_exclusive (Argument) gpu_exclusive:, 20

resources[LSF]/kwargs/gpu_new_syntax (Argument) gpu_new_syntax:, 20

resources[LSF]/kwargs/gpu_usage (Argument) gpu_usage:, 20

resources[OpenAPI]/kwargs (Argument) kwargs:, 21

resources[PBS]/kwargs (Argument) kwargs:, 21

resources[Shell]/kwargs (Argument) kwargs:, 21

resources[SlurmJobArray]/kwargs (Argument) kwargs:, 20

resources[SlurmJobArray]/kwargs/custom_gpu_line (Argument) custom_gpu_line:, 20

resources[SlurmJobArray]/kwargs/slurm_job_size (Argument) slurm_job_size:, 20

resources[Slurm]/kwargs (Argument) kwargs:, 19

resources[Slurm]/kwargs/custom_gpu_line (Argument) custom_gpu_line:, 19

resources[Torque]/kwargs (Argument) kwargs:, 21

RETCODE (class in *dpdispatcher.dpcloudserver.retcode*), 53

retry() (in module *dpdispatcher.utils*), 89

RetrySignal, 88

ROLEERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53

rsync() (in module *dpdispatcher.utils*), 89

rsync_available (*dpdispatcher.ssh_context.SSHSession* property), 80

run_cmd_with_all_output() (in module *dpdispatcher.utils*), 89

run_submission() (*dpdispatcher.Submission* method), 48

run_submission() (*dpdispatcher.submission.Submission* method), 86

running (*dpdispatcher.JobStatus.JobStatus* attribute), 54

S

serialize() (*dpdispatcher.Job* method), 30

serialize() (*dpdispatcher.Machine* method), 36

serialize() (*dpdispatcher.machine.Machine* method), 70

serialize() (*dpdispatcher.Resources* method), 41

serialize() (*dpdispatcher.Submission* method), 48

serialize() (*dpdispatcher.submission.Job* method), 82

serialize() (*dpdispatcher.submission.Resources* method), 83

serialize() (*dpdispatcher.submission.Submission* method), 86

serialize() (*dpdispatcher.submission.Task* method), 88

serialize() (*dpdispatcher.Task* method), 50

sftp (*dpdispatcher.ssh_context.SSHContext* property), 79

sftp (*dpdispatcher.ssh_context.SSHSession* property), 80

sftp (*dpdispatcher.SSHContext* property), 43

Shell (class in *dpdispatcher*), 43

Shell (class in *dpdispatcher.shell*), 74

Slurm (class in *dpdispatcher*), 44

Slurm (class in *dpdispatcher.slurm*), 76

slurm_job_size: resources[SlurmJobArray]/kwargs/slurm_job_size (Argument), 20

SlurmJobArray (class in *dpdispatcher.slurm*), 77

source_list: resources/source_list (Argument), 18

SPRetObj (class in *dpdispatcher.lazy_local_context*), 65

SPRetObj (class in *dpdispatcher.local_context*), 66

ssh (*dpdispatcher.ssh_context.SSHContext* property), 79
 ssh (*dpdispatcher.SSHContext* property), 43
 SSHContext (class in *dpdispatcher*), 41
 SSHContext (class in *dpdispatcher.ssh_context*), 78
 SSHSession (class in *dpdispatcher.ssh_context*), 79
 strategy:
 resources/strategy (Argument), 18
 sub_script_cmd() (*dpdispatcher.LSF* method), 31
 sub_script_cmd() (*dpdispatcher.lsf.LSF* method), 68
 sub_script_cmd() (*dpdispatcher.Machine* method), 36
 sub_script_cmd() (*dpdispatcher.machine.Machine* method), 70
 sub_script_head() (*dpdispatcher.LSF* method), 31
 sub_script_head() (*dpdispatcher.lsf.LSF* method), 68
 sub_script_head() (*dpdispatcher.Machine* method), 36
 sub_script_head() (*dpdispatcher.machine.Machine* method), 70
 subclasses_dict (*dpdispatcher.base_context.BaseContext* attribute), 55
 subclasses_dict (*dpdispatcher.Machine* attribute), 36
 subclasses_dict (*dpdispatcher.machine.Machine* attribute), 70
 Submission (class in *dpdispatcher*), 45
 Submission (class in *dpdispatcher.submission*), 83
 submission_from_json() (*dpdispatcher.Submission* class method), 48
 submission_from_json() (*dpdispatcher.submission.Submission* class method), 86
 submission_to_json() (*dpdispatcher.Submission* method), 48
 submission_to_json() (*dpdispatcher.submission.Submission* method), 86
 submit_job() (*dpdispatcher.Job* method), 30
 submit_job() (*dpdispatcher.submission.Job* method), 82

T

tar_compress:
 machine[SSHContext]/remote_profile/tar_compress (Argument), 15
 task (Argument)
 task:, 23
 Task (class in *dpdispatcher*), 49
 Task (class in *dpdispatcher.submission*), 87
 task/backward_files (Argument)
 backward_files:, 23
 task/command (Argument)
 command:, 23
 task/errlog (Argument)
 errlog:, 23
 task/forward_files (Argument)
 forward_files:, 23
 task/outlog (Argument)
 outlog:, 23
 task/task_work_path (Argument)
 task_work_path:, 23
 task:
 task (Argument), 23
 task_work_path:
 task/task_work_path (Argument), 23
 terminated (*dpdispatcher.JobStatus.JobStatus* attribute), 54
 THIRDDERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53
 timeout:
 machine[SSHContext]/remote_profile/timeout (Argument), 14
 TOKENINVALID (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53
 Torque (class in *dpdispatcher*), 50
 Torque (class in *dpdispatcher.pbs*), 74
 totp_secret:
 machine[SSHContext]/remote_profile/totp_secret (Argument), 15
 try_download_result() (*dpdispatcher.Submission* method), 48
 try_download_result() (*dpdispatcher.submission.Submission* method), 86
 try_recover_from_json() (*dpdispatcher.Submission* method), 49
 try_recover_from_json() (*dpdispatcher.submission.Submission* method), 86

U

UNDERDEBUG (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53
 unknown (*dpdispatcher.JobStatus.JobStatus* attribute), 54
 UNKOWNERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53
 unsubmitted (*dpdispatcher.JobStatus.JobStatus* attribute), 54
 unzip_file() (in module *dpdispatcher.dpcloudserver.zip_file*), 54
 update_submission_state() (*dpdispatcher.Submission* method), 49
 update_submission_state() (*dpdispatcher.submission.Submission* method), 86
 upload() (*dpdispatcher.base_context.BaseContext* method), 56
 upload() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 60

upload() (*dpdispatcher.dpcloudserver.Client* method), 52
 upload() (*dpdispatcher.dpcloudserver.client.Client* method), 53
 upload() (*dpdispatcher.hdfs_context.HDFSContext* method), 63
 upload() (*dpdispatcher.HDFSContext* method), 28
 upload() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 65
 upload() (*dpdispatcher.LazyLocalContext* method), 33
 upload() (*dpdispatcher.local_context.LocalContext* method), 66
 upload() (*dpdispatcher.LocalContext* method), 34
 upload() (*dpdispatcher.openapi_context.OpenAPIContext* method), 72
 upload() (*dpdispatcher.OpenAPIContext* method), 39
 upload() (*dpdispatcher.ssh_context.SSHContext* method), 79
 upload() (*dpdispatcher.SSHContext* method), 43
 upload_job() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 60
 upload_job() (*dpdispatcher.openapi_context.OpenAPIContext* method), 72
 upload_job() (*dpdispatcher.OpenAPIContext* method), 39
 upload_jobs() (*dpdispatcher.Submission* method), 49
 upload_jobs() (*dpdispatcher.submission.Submission* method), 87
 USERERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53
 username:
 machine[SSHContext]/remote_profile/username
 (Argument), 14

V

VERIFYERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 53

W

wait_time:
 resources/wait_time (Argument), 19
 waiting (*dpdispatcher.JobStatus.JobStatus* attribute), 54
 write_file() (*dpdispatcher.base_context.BaseContext* method), 56
 write_file() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 60
 write_file() (*dpdispatcher.hdfs_context.HDFSContext* method), 63
 write_file() (*dpdispatcher.HDFSContext* method), 29
 write_file() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 65
 write_file() (*dpdispatcher.LazyLocalContext* method), 33

write_file() (*dpdispatcher.local_context.LocalContext* method), 66
 write_file() (*dpdispatcher.LocalContext* method), 34
 write_file() (*dpdispatcher.openapi_context.OpenAPIContext* method), 72
 write_file() (*dpdispatcher.OpenAPIContext* method), 39
 write_file() (*dpdispatcher.ssh_context.SSHContext* method), 79
 write_file() (*dpdispatcher.SSHContext* method), 43
 write_home_file() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 60
 write_home_file() (*dpdispatcher.openapi_context.OpenAPIContext* method), 72
 write_home_file() (*dpdispatcher.OpenAPIContext* method), 39
 write_local_file() (*dpdispatcher.dp_cloud_server_context.BohriumContext* method), 60
 write_local_file() (*dpdispatcher.openapi_context.OpenAPIContext* method), 72
 write_local_file() (*dpdispatcher.OpenAPIContext* method), 39

Z

zip_file_list() (in module *dpdispatcher.dpcloudserver.zip_file*), 54