
DPDispatcher

Deep Modeling

Dec 02, 2021

CONTENTS:

- 1 Install DPDispatcher** **3**
- 2 Getting Started** **5**
- 3 Machine parameters** **9**
- 4 Resources parameters** **13**
- 5 Task parameters** **17**
- 6 DPDispatcher API** **19**
 - 6.1 dpdispatcher package 19
- 7 Indices and tables** **49**
- Python Module Index** **51**
- Index** **53**

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

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

INSTALL DPDISPATCHER

DPDispatcher can installed by pip:

```
pip install dpdispatcher
```


GETTING STARTED

DPDispatcher provides the following classes:

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

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

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

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

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

task1 = Task(command='cat example.txt', task_work_path='dir1/', forward_files=['example.
↳txt'], backward_files=['out.txt'], outlog='out.txt')
task2 = Task(command='cat example.txt', task_work_path='dir2/', forward_files=['example.
↳txt'], backward_files=['out.txt'], outlog='out.txt')
task3 = Task(command='cat example.txt', task_work_path='dir3/', forward_files=['example.
↳txt'], backward_files=['out.txt'], outlog='out.txt')
task4 = Task(command='cat example.txt', task_work_path='dir4/', forward_files=['example.
↳txt'], backward_files=['out.txt'], outlog='out.txt')

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

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

submission.run_submission()
```

where `machine.json` is

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

`resources.json` is

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

and `task.json` is

```
{
  "command": "lmp -i input.lammps",
  "task_work_path": "bct-0/",
  "forward_files": [
    "conf.lmp",
    "input.lammps"
  ],
  "backward_files": [
    "log.lammps"
  ],
  "outlog": "log",
  "errlog": "err",
}
```

You may also submit mutiple GPU jobs: complex resources example

```
resources = Resources(
  number_node=1,
  cpu_per_node=4,
  gpu_per_node=2,
  queue_name="GPU_2080Ti",
  group_size=4,
  custom_flags=[
    "#SBATCH --nice=100",
    "#SBATCH --time=24:00:00"
  ],
  strategy={
```

(continues on next page)

(continued from previous page)

```
    # used when you want to add CUDA_VISIBLE_DEVICES automatically
    "if_cuda_multi_devices": True
},
para_deg=1,
# will unload these modules before running tasks
module_unload_list=["singularity"],
# will load these modules before running tasks
module_list=["singularity/3.0.0"],
# will source the environment files before running tasks
source_list=["./slurm_test.env"],
# the envs option is used to export environment variables
# And it will generate a line like below.
# export DP_DISPATCHER_EXPORT=test_foo_bar_baz
envs={"DP_DISPATCHER_EXPORT": "test_foo_bar_baz"},
)
```

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

MACHINE PARAMETERS

machine:

type: dict
argument path: machine

batch_type:

type: str
argument path: machine/batch_type

The batch job system type. Option: PBS, Shell, DpCloudServer, DistributedShell, Torque, LSF, Slurm

local_root:

type: str
argument path: machine/local_root

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

remote_root:

type: str, optional
argument path: machine/remote_root

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

clean_asynchronously:

type: bool, optional, default: False
argument path: machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)
argument path: machine/context_type
possible choices: SSHContext, DpCloudServerContext, HDFSContext, LazyLocalContext, LocalContext

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

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
password of linux system

port:

type: int, optional, default: 22
argument path: machine[SSHContext]/remote_profile/port
ssh connection port.

key_filename:

type: NoneType | str, optional, default: None
argument path: machine[SSHContext]/remote_profile/key_filename
key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None
argument path: machine[SSHContext]/remote_profile/passphrase
passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10
argument path: machine[SSHContext]/remote_profile/timeout
timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None
argument path: machine[SSHContext]/remote_profile/totp_secret
Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

When *context_type* is set to `DpCloudServerContext` (or its aliases `dpcloudservercontext`, `DpCloudServer`, `dpcloudserver`):

remote_profile:

type: dict
argument path: machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str
argument path: machine[DpCloudServerContext]/remote_profile/email
Email

password:

type: str
argument path: machine[DpCloudServerContext]/remote_profile/password
Password

program_id:

type: int
argument path: machine[DpCloudServerContext]/remote_profile/program_id
Program ID

input_data:

type: dict
argument path: machine[DpCloudServerContext]/remote_profile/input_data
Configuration of job

job_name:

type: str, optional
argument path:
machine[DpCloudServerContext]/remote_profile/input_data/job_name
Job name

image_name:

type: str, optional
argument path:
machine[DpCloudServerContext]/remote_profile/input_data/image_name
Name of the image which run the job, optional when platform is not ali/oss.

disk_size:

type: str, optional
argument path:
machine[DpCloudServerContext]/remote_profile/input_data/disk_size
disk size (GB), optional when platform is not ali/oss.

scass_type:

type: str
argument path:
machine[DpCloudServerContext]/remote_profile/input_data/scass_type
machine configuration.

platform:

type: str

argument path:
machine[DpCloudServerContext]/remote_profile/input_data/platform

Job run in which platform.

log_file:

type: str, optional
argument path:
machine[DpCloudServerContext]/remote_profile/input_data/log_file

location of log file.

checkpoint_files:

type: list | str, optional
argument path:
machine[DpCloudServerContext]/remote_profile/input_data/checkpoint_files

location of checkpoint files when it is list type. record file changes when it is string value 'sync_files'

checkpoint_time:

type: int, optional, default: 15
argument path:
machine[DpCloudServerContext]/remote_profile/input_data/checkpoint_time

interval of checkpoint data been stored minimum 15.

backward_files:

type: list, optional
argument path:
machine[DpCloudServerContext]/remote_profile/input_data/backward_files

which files to be uploaded to remote resources. Upload all the files when it is None or empty.

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

remote_profile:

type: dict, optional
argument path: machine[HDFSContext]/remote_profile

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

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

remote_profile:

type: dict, optional
argument path: machine[LazyLocalContext]/remote_profile

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

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

remote_profile:

type: dict, optional
argument path: machine[LocalContext]/remote_profile

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

RESOURCES PARAMETERS

resources:

type: dict

argument path: resources

number_node:

type: int

argument path: resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int

argument path: resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int

argument path: resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

type: str

argument path: resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int

argument path: resources/group_size

The number of *tasks* in a *job*.

custom_flags:

type: list, optional

argument path: resources/custom_flags

The extra lines pass to job submitting script header

strategy:

type: dict, optional

argument path: `resources/strategy`

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: `bool`, optional, default: `True`

argument path: `resources/strategy/if_cuda_multi_devices`

para_deg:

type: `int`, optional, default: `1`

argument path: `resources/para_deg`

Decide how many tasks will be run in parallel.

source_list:

type: `list`, optional, default: `[]`

argument path: `resources/source_list`

The env file to be sourced before the command execution.

module_unload_list:

type: `list`, optional, default: `[]`

argument path: `resources/module_unload_list`

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: `list`, optional, default: `[]`

argument path: `resources/module_list`

The modules to be loaded on HPC system before submitting jobs

envs:

type: `dict`, optional, default: `{}`

argument path: `resources/envs`

The environment variables to be exported on before submitting jobs

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, Torque, Shell, PBS, DpCloudServer, DistributedShell, LSF

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

argument path: `resources[Slurm]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with `#SBATCH`

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

kwargs:

type: dict, optional

argument path: `resources[Torque]/kwargs`

This field is empty for this batch.

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

kwargs:

type: dict, optional

argument path: `resources[Shell]/kwargs`

This field is empty for this batch.

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

kwargs:

type: dict, optional

argument path: `resources[PBS]/kwargs`

This field is empty for this batch.

When `batch_type` is set to DpCloudServer (or its alias `dpcloudserver`):

kwargs:

type: dict, optional

argument path: `resources[DpCloudServer]/kwargs`

This field is empty for this batch.

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

kwargs:

type: dict, optional

argument path: `resources[DistributedShell]/kwargs`

This field is empty for this batch.

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

kwargs:

type: dict

argument path: `resources[LSF]/kwargs`

Extra arguments.

gpu_usage:

type: bool, optional, default: False

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

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False

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

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

gpu_exclusive:

type: `bool`, optional, default: `True`

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

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

custom_gpu_line:

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

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

Custom GPU configuration, starting with `#BSUB`

TASK PARAMETERS

task:

type: dict

argument path: task

command:

type: str

argument path: task/command

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

task_work_path:

type: str

argument path: task/task_work_path

The dir where the command to be executed.

forward_files:

type: list

argument path: task/forward_files

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

backward_files:

type: list

argument path: task/backward_files

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

outlog:

type: NoneType | str

argument path: task/outlog

The out log file name. redirect from stdout

errlog:

type: NoneType | str

argument path: task/errlog

The err log file name. redirect from stderr

DPDISPATCHER API

6.1 dpdispatcher package

`dpdispatcher.info()`

6.1.1 Subpackages

`dpdispatcher.dpcloudserver` package

Submodules

`dpdispatcher.dpcloudserver.api` module

`class dpdispatcher.dpcloudserver.api.API(email, password)`
Bases: `object`

Methods

<code>download</code>	
<code>download_from_url</code>	
<code>get</code>	
<code>get_jobs</code>	
<code>get_tasks</code>	
<code>get_tasks_v2</code>	
<code>get_tasks_v2_list</code>	
<code>job_create</code>	
<code>job_create_v2</code>	
<code>post</code>	
<code>refresh_token</code>	
<code>upload</code>	

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

`download_from_url(url, save_file)`

`get(url, params, retry=0)`

`get_jobs(page=1, per_page=10)`

`get_tasks(job_id, page=1, per_page=10)`

`get_tasks_v2(job_id, group_id, page=1, per_page=10)`

`get_tasks_v2_list(group_id, per_page=30)`

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

`job_create_v2(job_type, oss_path, input_data, program_id=None, group_id=None)`

`post(url, params, retry=0)`

`refresh_token()`

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

dpdispatcher.dpcloudserver.config module

dpdispatcher.dpcloudserver.retcode module

class `dpdispatcher.dpcloudserver.retcode.RETCODE`

Bases: `object`

`DATAERR = '2002'`

`DBERR = '2000'`

`IOERR = '2003'`

`NODATA = '2300'`

`OK = '0000'`

`PARAMERR = '2101'`

`PWDERR = '2104'`

`REQERR = '2200'`

`ROLEERR = '2103'`

`THIRDERR = '2001'`

`TOKENINVALID = '2100'`

`UNDERDEBUG = '2301'`

`UNKOWNERR = '2400'`

`USERERR = '2102'`


```
VERIFYERR = '2105'
```

dpdispatcher.dpcloudserver.temp_test module

dpdispatcher.dpcloudserver.zip_file module

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

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

6.1.2 Submodules

6.1.3 dpdispatcher.JobStatus module

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

```
Bases: enum.IntEnum
```

```
An enumeration.
```

```
completing = 6
```

```
finished = 5
```

```
running = 3
```

```
terminated = 4
```

```
unknown = 100
```

```
unsubmitted = 1
```

```
waiting = 2
```

6.1.4 dpdispatcher.base_context module

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

```
Bases: object
```

Methods

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

bind_submission	
check_finish	
clean	
download	
kill	
load_from_dict	
read_file	
upload	
write_file	

bind_submission(*submission*)

check_finish(*proc*)

clean()

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

kill(*proc*)

classmethod load_from_dict(*context_dict*)

classmethod machine_arginfo() → dargs.dargs.Argument

Generate the machine arginfo.

Returns

Argument machine arginfo

classmethod machine_subfields() → List[dargs.dargs.Argument]

Generate the machine subfields.

Returns

list[Argument] machine subfields

options = {'DpCloudServerContext', 'HDFSContext', 'LazyLocalContext',
'LocalContext', 'SSHContext'}

read_file(*fname*)

```

subclasses_dict = {'DpCloudServer': <class
'dpdispatcher.dp_cloud_server_context.DpCloudServerContext'>,
'DpCloudServerContext': <class
'dpdispatcher.dp_cloud_server_context.DpCloudServerContext'>, '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'>, 'Local': <class
'dpdispatcher.local_context.LocalContext'>, 'LocalContext': <class
'dpdispatcher.local_context.LocalContext'>, 'SSH': <class
'dpdispatcher.ssh_context.SSHContext'>, 'SSHContext': <class
'dpdispatcher.ssh_context.SSHContext'>, 'dpcloudserver': <class
'dpdispatcher.dp_cloud_server_context.DpCloudServerContext'>,
'dpcloudservercontext': <class
'dpdispatcher.dp_cloud_server_context.DpCloudServerContext'>, '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'>, 'local': <class
'dpdispatcher.local_context.LocalContext'>, 'localcontext': <class
'dpdispatcher.local_context.LocalContext'>, 'ssh': <class
'dpdispatcher.ssh_context.SSHContext'>, 'sshcontext': <class
'dpdispatcher.ssh_context.SSHContext'>}]

```

```
upload(submission)
```

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

6.1.5 dpdispatcher.distributed_shell module

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

Bases: *dpdispatcher.machine.Machine*

Methods

<i>do_submit</i> (job)	submit th job to yarn using distributed shell
<i>resources_arginfo</i> ()	Generate the resources arginfo.
<i>resources_subfields</i> ()	Generate the resources subfields.

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

check_finish_tag(*job*)

check_status(*job*)

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*)

6.1.6 dpdispatcher.dp_cloud_server module

class `dpdispatcher.dp_cloud_server.DpCloudServer(*args, **kwargs)`
 Bases: `dpdispatcher.machine.Machine`

Methods

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

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

`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)`

6.1.7 dpdispatcher.dp_cloud_server_context module

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

Bases: `dpdispatcher.base_context.BaseContext`

Methods

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

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

`kill(cmd_pipes)`

`classmethod load_from_dict(context_dict)`

`classmethod machine_subfields() → List[dargs.dargs.Argument]`
 Generate the machine subfields.

Returns

list[Argument] machine subfields

read_file(*fname*)

read_home_file(*fname*)

upload(*submission*)

write_file(*fname, write_str*)

write_home_file(*fname, write_str*)

write_local_file(*fname, write_str*)

6.1.8 dpdispatcher.dpdisp module

`dpdispatcher.dpdisp.main()`

6.1.9 dpdispatcher.hdfs_cli module

class `dpdispatcher.hdfs_cli.HDFS`

Bases: `object`

Fundamental class for HDFS basic manipulation

Methods

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

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

6.1.10 dpdispatcher.hdfs_context module

class dpdispatcher.hdfs_context.HDFSContext(*args, **kwargs)
 Bases: *dpdispatcher.base_context.BaseContext*

Methods

<i>download</i> (submission[, check_exists, ...])	download backward files from HDFS root dir
<i>machine_arginfo</i> ()	Generate the machine arginfo.
<i>machine_subfields</i> ()	Generate the machine subfields.
<i>upload</i> (submission[, dereference])	upload forward files and forward command files to HDFS root dir

bind_submission	
check_file_exists	
check_finish	
clean	
get_job_root	
kill	
load_from_dict	
read_file	
write_file	

bind_submission(*submission*)

check_file_exists(*fname*)

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

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**Returns****none****write_file**(*fname, write_str*)

6.1.11 dpdispatcher.lazy_local_context module

class dpdispatcher.lazy_local_context.**LazyLocalContext**(*args, **kwargs)Bases: *dpdispatcher.base_context.BaseContext***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	
kill	
load_from_dict	
read_file	
upload	
write_file	

`bind_submission(submission)`

`block_call(cmd)`

`block_checkcall(cmd)`

`call(cmd)`

`check_file_exists(fname)`

`check_finish(proc)`

`clean()`

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

`get_job_root()`

`get_return(proc)`

`kill(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

<code>read</code>	
<code>readlines</code>	

`read()`

`readlines()`

6.1.12 dpdispatcher.local_context module

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

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	
download_	
get_job_root	
get_return	
kill	
load_from_dict	
read_file	
upload	
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*)

download_(*job_dirs*, *remote_down_files*, *check_exists=False*, *mark_failure=True*, *back_error=False*)

`get_job_root()`

`get_return(proc)`

`kill(proc)`

`classmethod load_from_dict(context_dict)`

`read_file(fname)`

`upload(submission)`

`upload_(job_dirs, local_up_files, dereference=True)`

`write_file(fname, write_str)`

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

Methods

<code>read</code>	
<code>readlines</code>	

`read()`

`readlines()`

6.1.13 dpdispatcher.lsf module

`class dpdispatcher.lsf.LSF(*args, **kwargs)`
Bases: `dpdispatcher.machine.Machine`
LSF batch

Methods

`default_resources(resources)`

`do_submit(job)` submit a single job, assuming that no job is running there.

`resources_arginfo()` Generate the resources arginfo.

`resources_subfields()` Generate the resources subfields.

arginfo	
bind_context	
check_finish_tag	
check_if_recover	
check_status	
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*)

classmethod resources_subfields() → List[dargs.dargs.Argument]

Generate the resources subfields.

Returns

list[Argument] resources subfields

sub_script_cmd(*res*)

sub_script_head(*res*)

6.1.14 dpdispatcher.machine module

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

Bases: object

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

Parameters

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

Methods

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

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

classmethod arginfo()

bind_context(context)

check_finish_tag(**kwargs)

check_if_recover(submission)

check_status(job)

```

default_resources(res)

classmethod deserialize(machine_dict)

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

gen_command_env_cuda_devices(resources)

gen_script(job)

gen_script_command(job)

gen_script_custom_flags_lines(job)

gen_script_end(job)

gen_script_env(job)

gen_script_header(job)

gen_script_wait(resources)

classmethod load_from_dict(machine_dict)

classmethod load_from_json(json_path)

options = {'DistributedShell', 'DpCloudServer', 'LSF', 'PBS', 'Shell', 'Slurm',
            'Torque'}

classmethod resources_arginfo() → dargs.dargs.Argument
    Generate the resources arginfo.

    Returns
        Argument resources arginfo

classmethod resources_subfields() → List[dargs.dargs.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 = {'DistributedShell': <class
'dpdispatcher.distributed_shell.DistributedShell'>, 'DpCloudServer': <class
'dpdispatcher.dp_cloud_server.DpCloudServer'>, 'LSF': <class
'dpdispatcher.lsf.LSF'>, 'PBS': <class 'dpdispatcher.pbs.PBS'>, 'Shell': <class
'dpdispatcher.shell.Shell'>, 'Slurm': <class 'dpdispatcher.slurm.Slurm'>, 'Torque':
<class 'dpdispatcher.pbs.Torque'>, 'distributedshell': <class
'dpdispatcher.distributed_shell.DistributedShell'>, 'dpcloudserver': <class
'dpdispatcher.dp_cloud_server.DpCloudServer'>, 'lsf': <class
'dpdispatcher.lsf.LSF'>, 'pbs': <class 'dpdispatcher.pbs.PBS'>, 'shell': <class
'dpdispatcher.shell.Shell'>, 'slurm': <class 'dpdispatcher.slurm.Slurm'>, 'torque':
<class 'dpdispatcher.pbs.Torque'>}
```

6.1.15 dpdispatcher.pbs module

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

Methods

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

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

`check_finish_tag(job)`

`check_status(job)`

default_resources(resources)

do_submit(job)

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

gen_script(job)

gen_script_header(job)

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

Bases: *dpdispatcher.pbs.PBS*

Methods

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

6.1.16 dpdispatcher.shell module

class `dpdispatcher.shell.Shell(*args, **kwargs)`
 Bases: `dpdispatcher.machine.Machine`

Methods

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

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

check_finish_tag(*job*)

check_status(*job*)

default_resources(*resources*)

do_submit(*job*)
 submit a single job, assuming that no job is running there.

gen_script(*job*)

gen_script_header(*job*)

6.1.17 dpdispatcher.slurm module

class dpdispatcher.slurm.Slurm(*args, **kwargs)
 Bases: dpdispatcher.machine.Machine

Methods

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

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

check_finish_tag(job)

check_status(job, retry=0, max_retry=3)

default_resources(resources)

do_submit(job, retry=0, max_retry=3)
 submit a single job, assuming that no job is running there.

gen_script(job)

gen_script_header(job)

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

Returns

list[Argument] resources subfields

6.1.18 dpdispatcher.ssh_context module

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

Bases: `dpdispatcher.base_context.BaseContext`

Attributes

sftp

ssh

Methods

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

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

bind_submission(*submission*)

block_call(*cmd*)

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

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

Parameters

cmd: **str** The command to run.

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

call(*cmd*)

check_file_exists(*fname*)

check_finish(*cmd_pipes*)

clean()

close()

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

get_job_root()

get_return(*cmd_pipes*)

kill(*cmd_pipes*)

classmethod load_from_dict(*context_dict*)

classmethod machine_subfields() → List[dargs.dargs.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)

Bases: `object`

Attributes

sftp Returns sftp.

Methods

<code>exec_command(cmd[, retry])</code>	Calling self.ssh.exec_command but has an exception check.
---	---

arginfo	
close	
ensure_alive	
get_ssh_client	

static arginfo()

close()

ensure_alive(*max_check=10, sleep_time=10*)

exec_command(*cmd, retry=0*)

Calling self.ssh.exec_command but has an exception check.

get_ssh_client()

property sftp

Returns sftp. Open a new one if not existing.

6.1.19 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 <code>job_dict</code> 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

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

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}, para_deg=1,
                                       module_unload_list=[], module_list=[], source_list=[],
                                       envs={}, **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.
- 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.

Methods

arginfo	
deserialize	
load_from_dict	
load_from_json	
serialize	

static `arginfo()`

classmethod `deserialize(resources_dict)`

classmethod `load_from_dict(resources_dict)`

classmethod `load_from_json(json_file)`

serialize()


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

Bases: `object`

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

Parameters

- work_base** [Path] the base directory of the local tasks. It is usually the dir name of project .
- machine** [Machine] machine class object (for example, PBS, Slurm, Shell) to execute the jobs.
The machine can still be bound after the instantiation with the `bind_submission` method.
- resources** [Resources] the machine resources (cpu or gpu) used to generate the slurm/pbs script
- forward_common_files** [list] the common files to be uploaded to other computers before the jobs begin
- backward_common_files** [list] the common files to be downloaded from other computers after the jobs finish
- task_list** [list of Task] a list of tasks to be run.

Methods

<code>bind_machine(machine)</code>	bind this submission to a machine.
<code>check_all_finished()</code>	check whether all the jobs in the submission.
<code>deserialize(submission_dict[, machine])</code>	convert the submission_dict to a Submission class object
<code>generate_jobs()</code>	After tasks register to the self.belonging_tasks, This method generate the jobs and add these jobs to self.belonging_jobs.
<code>handle_unexpected_submission_state()</code>	handle unexpected job state of the submission.
<code>run_submission(*[, exit_on_submit, clean])</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.

clean_jobs	
download_jobs	
get_hash	
register_task	
register_task_list	
submission_from_json	
submission_to_json	
try_recover_from_json	
upload_jobs	

bind_machine(*machine*)

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

Parameters

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

check_all_finished()

check whether all the jobs in the submission.

Notes

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

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

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

main method to execute the submission. First, check whether old Submission exists on the remote machine, and try to recover from it. Second, upload the local files to the remote machine where the tasks to be executed. Third, run the submission defined previously. Forth, wait until the tasks in the submission finished and download the result file to local directory. if *exit_on_submit* is True, submission will exit.

serialize(*if_static=False*)

convert the Submission class instance to a dictionary.

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_recover_from_json()`

`update_submission_state()`

check whether all the jobs in the submission.

Notes

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

`upload_jobs()`

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

Bases: `object`

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

Parameters

command [Str] the command to be executed.

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

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

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

outlog [Str] the filename to which command redirect stdout

errlog [Str] the filename to which command redirect stderr

Methods

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

arginfo	
get_hash	
load_from_json	
serialize	

`static arginfo()`

`classmethod deserialize(task_dict)`

convert the task_dict to a Task class object

Parameters

task_dict [dict] the dictionary which contains the task information

Returns

task [Task] the Task class instance converted from the task_dict

get_hash()

classmethod load_from_json(json_file)

serialize()

6.1.20 dpdispatcher.utils module

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

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

- dpdispatcher, 19
- dpdispatcher.base_context, 21
- dpdispatcher.distributed_shell, 23
- dpdispatcher.dp_cloud_server, 25
- dpdispatcher.dp_cloud_server_context, 26
- dpdispatcher.dpcloudserver, 19
- dpdispatcher.dpcloudserver.api, 19
- dpdispatcher.dpcloudserver.config, 20
- dpdispatcher.dpcloudserver.retcode, 20
- dpdispatcher.dpcloudserver.zip_file, 21
- dpdispatcher.dpdisp, 27
- dpdispatcher.hdfs_cli, 27
- dpdispatcher.hdfs_context, 28
- dpdispatcher.JobStatus, 21
- dpdispatcher.lazy_local_context, 29
- dpdispatcher.local_context, 31
- dpdispatcher.lsf, 32
- dpdispatcher.machine, 34
- dpdispatcher.pbs, 36
- dpdispatcher.shell, 38
- dpdispatcher.slurm, 39
- dpdispatcher.ssh_context, 40
- dpdispatcher.submission, 42
- dpdispatcher.utils, 48

A

API (class in *dpdispatcher.dpcloudserver.api*), 19
 arginfo() (*dpdispatcher.machine.Machine* class method), 34
 arginfo() (*dpdispatcher.ssh_context.SSHSession* static method), 42
 arginfo() (*dpdispatcher.submission.Resources* static method), 44
 arginfo() (*dpdispatcher.submission.Task* static method), 47

B

BaseContext (class in *dpdispatcher.base_context*), 21
 bind_context() (*dpdispatcher.machine.Machine* method), 34
 bind_machine() (*dpdispatcher.submission.Submission* method), 45
 bind_submission() (*dpdispatcher.base_context.BaseContext* method), 22
 bind_submission() (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 26
 bind_submission() (*dpdispatcher.hdfs_context.HDFSContext* method), 28
 bind_submission() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 29
 bind_submission() (*dpdispatcher.local_context.LocalContext* method), 31
 bind_submission() (*dpdispatcher.ssh_context.SSHContext* method), 40
 block_call() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 30
 block_call() (*dpdispatcher.local_context.LocalContext* method), 31
 block_call() (*dpdispatcher.ssh_context.SSHContext* method), 40
 block_checkcall() (*dpdis-*

patcher.lazy_local_context.LazyLocalContext method), 30
 block_checkcall() (*dpdispatcher.local_context.LocalContext* method), 31
 block_checkcall() (*dpdispatcher.ssh_context.SSHContext* method), 40

C

call() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 30
 call() (*dpdispatcher.local_context.LocalContext* method), 31
 call() (*dpdispatcher.ssh_context.SSHContext* method), 40
 check_all_finished() (*dpdispatcher.submission.Submission* method), 45
 check_file_exists() (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 26
 check_file_exists() (*dpdispatcher.hdfs_context.HDFSContext* method), 28
 check_file_exists() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 30
 check_file_exists() (*dpdispatcher.local_context.LocalContext* method), 31
 check_file_exists() (*dpdispatcher.ssh_context.SSHContext* method), 41
 check_finish() (*dpdispatcher.base_context.BaseContext* method), 22
 check_finish() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 30
 check_finish() (*dpdispatcher.local_context.LocalContext* method), 30

- 31
- `check_finish()` (*dpdispatcher.ssh_context.SSHContext* method), 41
- `check_finish_tag()` (*dpdispatcher.distributed_shell.DistributedShell* method), 24
- `check_finish_tag()` (*dpdispatcher.dp_cloud_server.DpCloudServer* method), 25
- `check_finish_tag()` (*dpdispatcher.lsf.LSF* method), 33
- `check_finish_tag()` (*dpdispatcher.machine.Machine* method), 34
- `check_finish_tag()` (*dpdispatcher.pbs.PBS* method), 36
- `check_finish_tag()` (*dpdispatcher.shell.Shell* method), 38
- `check_finish_tag()` (*dpdispatcher.slurm.Slurm* method), 39
- `check_home_file_exists()` (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 26
- `check_if_recover()` (*dpdispatcher.dp_cloud_server.DpCloudServer* method), 25
- `check_if_recover()` (*dpdispatcher.machine.Machine* method), 34
- `check_status()` (*dpdispatcher.distributed_shell.DistributedShell* method), 24
- `check_status()` (*dpdispatcher.dp_cloud_server.DpCloudServer* method), 25
- `check_status()` (*dpdispatcher.lsf.LSF* method), 33
- `check_status()` (*dpdispatcher.machine.Machine* method), 34
- `check_status()` (*dpdispatcher.pbs.PBS* method), 36
- `check_status()` (*dpdispatcher.pbs.Torque* method), 37
- `check_status()` (*dpdispatcher.shell.Shell* method), 38
- `check_status()` (*dpdispatcher.slurm.Slurm* method), 39
- `clean()` (*dpdispatcher.base_context.BaseContext* method), 22
- `clean()` (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 26
- `clean()` (*dpdispatcher.hdfs_context.HDFSContext* method), 28
- `clean()` (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 30
- `clean()` (*dpdispatcher.local_context.LocalContext* method), 31
- `clean()` (*dpdispatcher.ssh_context.SSHContext* method), 41
- `clean_jobs()` (*dpdispatcher.submission.Submission* method), 46
- `close()` (*dpdispatcher.ssh_context.SSHContext* method), 41
- `close()` (*dpdispatcher.ssh_context.SSHSession* method), 42
- `completing` (*dpdispatcher.JobStatus.JobStatus* attribute), 21
- `copy_from_local()` (*dpdispatcher.hdfs_cli.HDFS* static method), 27
- `copy_to_local()` (*dpdispatcher.hdfs_cli.HDFS* static method), 27
- ## D
- `DATAERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
- `DBERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
- `default_resources()` (*dpdispatcher.lsf.LSF* method), 33
- `default_resources()` (*dpdispatcher.machine.MachineContext* method), 34
- `default_resources()` (*dpdispatcher.pbs.PBS* method), 36
- `default_resources()` (*dpdispatcher.shell.Shell* method), 38
- `default_resources()` (*dpdispatcher.slurm.Slurm* method), 39
- `deserialize()` (*dpdispatcher.machine.Machine* class method), 35
- `deserialize()` (*dpdispatcher.submission.Job* class method), 43
- `deserialize()` (*dpdispatcher.submission.Resources* class method), 44
- `deserialize()` (*dpdispatcher.submission.Submission* class method), 46
- `deserialize()` (*dpdispatcher.submission.Task* class method), 47
- `DistributedShell` (class in *dpdispatcher.distributed_shell*), 23
- `do_submit()` (*dpdispatcher.distributed_shell.DistributedShell* method), 24
- `do_submit()` (*dpdispatcher.dp_cloud_server.DpCloudServer* method), 25
- `do_submit()` (*dpdispatcher.lsf.LSF* method), 33
- `do_submit()` (*dpdispatcher.machine.Machine* method), 35
- `do_submit()` (*dpdispatcher.pbs.PBS* method), 37
- `do_submit()` (*dpdispatcher.shell.Shell* method), 38
- `do_submit()` (*dpdispatcher.slurm.Slurm* method), 39
- `download()` (*dpdispatcher.base_context.BaseContext* method), 22
- `download()` (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 26

`download()` (*dpdispatcher.dpcloudserver.api.API method*), 19
`download()` (*dpdispatcher.hdfs_context.HDFSContext method*), 28
`download()` (*dpdispatcher.lazy_local_context.LazyLocalContext method*), 30
`download()` (*dpdispatcher.local_context.LocalContext method*), 31
`download()` (*dpdispatcher.ssh_context.SSHContext method*), 41
`download_()` (*dpdispatcher.local_context.LocalContext method*), 31
`download_from_url()` (*dpdispatcher.dpcloudserver.api.API method*), 19
`download_jobs()` (*dpdispatcher.submission.Submission method*), 46
`DpCloudServer` (class in *dpdispatcher.dp_cloud_server*), 25
`DpCloudServerContext` (class in *dpdispatcher.dp_cloud_server_context*), 26
`dpdispatcher` module, 19
`dpdispatcher.base_context` module, 21
`dpdispatcher.distributed_shell` module, 23
`dpdispatcher.dp_cloud_server` module, 25
`dpdispatcher.dp_cloud_server_context` module, 26
`dpdispatcher.dpcloudserver` module, 19
`dpdispatcher.dpcloudserver.api` module, 19
`dpdispatcher.dpcloudserver.config` module, 20
`dpdispatcher.dpcloudserver.retcode` module, 20
`dpdispatcher.dpcloudserver.zip_file` module, 21
`dpdispatcher.dpdisp` module, 27
`dpdispatcher.hdfs_cli` module, 27
`dpdispatcher.hdfs_context` module, 28
`dpdispatcher.JobStatus` module, 21
`dpdispatcher.lazy_local_context` module, 29
`dpdispatcher.local_context` module, 31
`dpdispatcher.lsf` module, 32
`dpdispatcher.machine` module, 34
`dpdispatcher.pbs` module, 36
`dpdispatcher.shell` module, 38
`dpdispatcher.slurm` module, 39
`dpdispatcher.ssh_context` module, 40
`dpdispatcher.submission` module, 42
`dpdispatcher.utils` module, 48
E
`ensure_alive()` (*dpdispatcher.ssh_context.SSHSession method*), 42
`exec_command()` (*dpdispatcher.ssh_context.SSHSession method*), 42
`exists()` (*dpdispatcher.hdfs_cli.HDFS static method*), 27
F
`finished` (*dpdispatcher.JobStatus.JobStatus attribute*), 21
G
`gen_command_env_cuda_devices()` (*dpdispatcher.machine.Machine method*), 35
`gen_local_script()` (*dpdispatcher.dp_cloud_server.DpCloudServer method*), 25
`gen_script()` (*dpdispatcher.dp_cloud_server.DpCloudServer method*), 25
`gen_script()` (*dpdispatcher.lsf.LSF method*), 33
`gen_script()` (*dpdispatcher.machine.Machine method*), 35
`gen_script()` (*dpdispatcher.pbs.PBS method*), 37
`gen_script()` (*dpdispatcher.shell.Shell method*), 38
`gen_script()` (*dpdispatcher.slurm.Slurm method*), 39
`gen_script_command()` (*dpdispatcher.machine.Machine method*), 35
`gen_script_custom_flags_lines()` (*dpdispatcher.machine.Machine method*), 35
`gen_script_end()` (*dpdispatcher.distributed_shell.DistributedShell method*), 24
`gen_script_end()` (*dpdispatcher.machine.Machine method*), 35

`gen_script_env()` (*dpdispatcher.distributed_shell.DistributedShell method*), 24
`gen_script_env()` (*dpdispatcher.machine.Machine method*), 35
`gen_script_header()` (*dpdispatcher.distributed_shell.DistributedShell method*), 24
`gen_script_header()` (*dpdispatcher.dp_cloud_server.DpCloudServer method*), 25
`gen_script_header()` (*dpdispatcher.lsf.LSF method*), 33
`gen_script_header()` (*dpdispatcher.machine.Machine method*), 35
`gen_script_header()` (*dpdispatcher.pbs.PBS method*), 37
`gen_script_header()` (*dpdispatcher.shell.Shell method*), 38
`gen_script_header()` (*dpdispatcher.slurm.Slurm method*), 39
`gen_script_wait()` (*dpdispatcher.machine.Machine method*), 35
`generate_jobs()` (*dpdispatcher.submission.Submission method*), 46
`generate_totp()` (*in module dpdispatcher.utils*), 48
`get()` (*dpdispatcher.dpcloudserver.api.API method*), 19
`get_hash()` (*dpdispatcher.submission.Job method*), 43
`get_hash()` (*dpdispatcher.submission.Submission method*), 46
`get_hash()` (*dpdispatcher.submission.Task method*), 48
`get_job_root()` (*dpdispatcher.hdfs_context.HDFSContext method*), 29
`get_job_root()` (*dpdispatcher.lazy_local_context.LazyLocalContext method*), 30
`get_job_root()` (*dpdispatcher.local_context.LocalContext method*), 31
`get_job_root()` (*dpdispatcher.ssh_context.SSHContext method*), 41
`get_job_state()` (*dpdispatcher.submission.Job method*), 43
`get_jobs()` (*dpdispatcher.dpcloudserver.api.API method*), 20
`get_return()` (*dpdispatcher.lazy_local_context.LazyLocalContext method*), 30
`get_return()` (*dpdispatcher.local_context.LocalContext method*), 32
`get_return()` (*dpdispatcher.ssh_context.SSHContext method*), 41
`get_sha256()` (*in module dpdispatcher.utils*), 48
`get_ssh_client()` (*dpdispatcher.ssh_context.SSHSession method*), 42
`get_tasks()` (*dpdispatcher.dpcloudserver.api.API method*), 20
`get_tasks_v2()` (*dpdispatcher.dpcloudserver.api.API method*), 20
`get_tasks_v2_list()` (*dpdispatcher.dpcloudserver.api.API method*), 20
H
`handle_unexpected_job_state()` (*dpdispatcher.submission.Job method*), 43
`handle_unexpected_submission_state()` (*dpdispatcher.submission.Submission method*), 46
HDFS (*class in dpdispatcher.hdfs_cli*), 27
HDFSContext (*class in dpdispatcher.hdfs_context*), 28
I
`info()` (*in module dpdispatcher*), 19
IOERR (*dpdispatcher.dpcloudserver.retcode.RETCODE attribute*), 20
J
Job (*class in dpdispatcher.submission*), 42
`job_create()` (*dpdispatcher.dpcloudserver.api.API method*), 20
`job_create_v2()` (*dpdispatcher.dpcloudserver.api.API method*), 20
`job_to_json()` (*dpdispatcher.submission.Job method*), 43
JobStatus (*class in dpdispatcher.JobStatus*), 21
K
`kill()` (*dpdispatcher.base_context.BaseContext method*), 22
`kill()` (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext method*), 26
`kill()` (*dpdispatcher.lazy_local_context.LazyLocalContext method*), 30
`kill()` (*dpdispatcher.local_context.LocalContext method*), 32
`kill()` (*dpdispatcher.ssh_context.SSHContext method*), 41
L
LazyLocalContext (*class in dpdispatcher.lazy_local_context*), 29
`load_from_dict()` (*dpdispatcher.base_context.BaseContext class method*), 22

`load_from_dict()` (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* class method), 26
`load_from_dict()` (*dpdispatcher.hdfs_context.HDFSContext* class method), 29
`load_from_dict()` (*dpdispatcher.lazy_local_context.LazyLocalContext* class method), 30
`load_from_dict()` (*dpdispatcher.local_context.LocalContext* class method), 32
`load_from_dict()` (*dpdispatcher.machine.Machine* class method), 35
`load_from_dict()` (*dpdispatcher.ssh_context.SSHContext* class method), 41
`load_from_dict()` (*dpdispatcher.submission.Resources* class method), 44
`load_from_json()` (*dpdispatcher.machine.Machine* class method), 35
`load_from_json()` (*dpdispatcher.submission.Resources* class method), 44
`load_from_json()` (*dpdispatcher.submission.Task* class method), 48
`LocalContext` (class in *dpdispatcher.local_context*), 31
`LSF` (class in *dpdispatcher.lsf*), 32

M

`Machine` (class in *dpdispatcher.machine*), 34
`machine_arginfo()` (*dpdispatcher.base_context.BaseContext* class method), 22
`machine_subfields()` (*dpdispatcher.base_context.BaseContext* class method), 22
`machine_subfields()` (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* class method), 26
`machine_subfields()` (*dpdispatcher.ssh_context.SSHContext* class method), 41
`main()` (in module *dpdispatcher.dpdisp*), 27
`map_dp_job_state()` (*dpdispatcher.dp_cloud_server.DpCloudServer* static method), 26
`mkdir()` (*dpdispatcher.hdfs_cli.HDFS* static method), 28
module
 dpdispatcher, 19
 dpdispatcher.base_context, 21
 dpdispatcher.distributed_shell, 23
 dpdispatcher.dp_cloud_server, 25
 dpdispatcher.dp_cloud_server_context, 26
 dpdispatcher.dpcloudserver, 19
 dpdispatcher.dpcloudserver.api, 19
 dpdispatcher.dpcloudserver.config, 20
 dpdispatcher.dpcloudserver.retcode, 20
 dpdispatcher.dpcloudserver.zip_file, 21
 dpdispatcher.dpdisp, 27
 dpdispatcher.hdfs_cli, 27
 dpdispatcher.hdfs_context, 28
 dpdispatcher.JobStatus, 21
 dpdispatcher.lazy_local_context, 29
 dpdispatcher.local_context, 31
 dpdispatcher.lsf, 32
 dpdispatcher.machine, 34
 dpdispatcher.pbs, 36
 dpdispatcher.shell, 38
 dpdispatcher.slurm, 39
 dpdispatcher.ssh_context, 40
 dpdispatcher.submission, 42
 dpdispatcher.utils, 48
`move()` (*dpdispatcher.hdfs_cli.HDFS* static method), 28

N

`NODATA` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20

O

`OK` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
`options` (*dpdispatcher.base_context.BaseContext* attribute), 22
`options` (*dpdispatcher.machine.Machine* attribute), 35

P

`PARAMERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
`PBS` (class in *dpdispatcher.pbs*), 36
`post()` (*dpdispatcher.dpcloudserver.api.API* method), 20
`PWDERR` (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20

R

`read()` (*dpdispatcher.lazy_local_context.SPRetObj* method), 30
`read()` (*dpdispatcher.local_context.SPRetObj* method), 32
`read_file()` (*dpdispatcher.base_context.BaseContext* method), 22
`read_file()` (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 27
`read_file()` (*dpdispatcher.hdfs_context.HDFSContext* method), 29
`read_file()` (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 30

- read_file() (*dpdispatcher.local_context.LocalContext* method), 32
- read_file() (*dpdispatcher.ssh_context.SSHContext* method), 41
- read_hdfs_file() (*dpdispatcher.hdfs_cli.HDFS* static method), 28
- read_home_file() (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 27
- readlines() (*dpdispatcher.lazy_local_context.SPRetObj* method), 30
- readlines() (*dpdispatcher.local_context.SPRetObj* method), 32
- refresh_token() (*dpdispatcher.dpcloudserver.api.API* method), 20
- register_job_id() (*dpdispatcher.submission.Job* method), 43
- register_task() (*dpdispatcher.submission.Submission* method), 46
- register_task_list() (*dpdispatcher.submission.Submission* method), 46
- remove() (*dpdispatcher.hdfs_cli.HDFS* static method), 28
- REQERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
- Resources (class in *dpdispatcher.submission*), 43
- resources_arginfo() (*dpdispatcher.machine.Machine* class method), 35
- resources_subfields() (*dpdispatcher.lsf.LSF* class method), 33
- resources_subfields() (*dpdispatcher.machine.Machine* class method), 35
- resources_subfields() (*dpdispatcher.slurm.Slurm* class method), 39
- RETCODE (class in *dpdispatcher.dpcloudserver.retcode*), 20
- ROLEERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
- run_cmd_with_all_output() (in module *dpdispatcher.utils*), 48
- run_submission() (*dpdispatcher.submission.Submission* method), 46
- running (*dpdispatcher.JobStatus.JobStatus* attribute), 21
- S**
- serialize() (*dpdispatcher.machine.Machine* method), 35
- serialize() (*dpdispatcher.submission.Job* method), 43
- serialize() (*dpdispatcher.submission.Resources* method), 44
- serialize() (*dpdispatcher.submission.Submission* method), 46
- serialize() (*dpdispatcher.submission.Task* method), 48
- sftp (*dpdispatcher.ssh_context.SSHContext* property), 41
- sftp (*dpdispatcher.ssh_context.SSHSession* property), 42
- Shell (class in *dpdispatcher.shell*), 38
- Slurm (class in *dpdispatcher.slurm*), 39
- SPRetObj (class in *dpdispatcher.lazy_local_context*), 30
- SPRetObj (class in *dpdispatcher.local_context*), 32
- ssh (*dpdispatcher.ssh_context.SSHContext* property), 41
- SSHContext (class in *dpdispatcher.ssh_context*), 40
- SSHSession (class in *dpdispatcher.ssh_context*), 41
- sub_script_cmd() (*dpdispatcher.lsf.LSF* method), 33
- sub_script_cmd() (*dpdispatcher.machine.Machine* method), 35
- sub_script_head() (*dpdispatcher.lsf.LSF* method), 33
- sub_script_head() (*dpdispatcher.machine.Machine* method), 35
- subclasses_dict (*dpdispatcher.base_context.BaseContext* attribute), 22
- subclasses_dict (*dpdispatcher.machine.Machine* attribute), 35
- Submission (class in *dpdispatcher.submission*), 44
- submission_from_json() (*dpdispatcher.submission.Submission* class method), 46
- submission_to_json() (*dpdispatcher.submission.Submission* method), 46
- submit_job() (*dpdispatcher.submission.Job* method), 43
- T**
- Task (class in *dpdispatcher.submission*), 47
- terminated (*dpdispatcher.JobStatus.JobStatus* attribute), 21
- THIRDERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
- TOKENINVALID (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
- Torque (class in *dpdispatcher.pbs*), 37
- try_recover_from_json() (*dpdispatcher.submission.Submission* method), 47
- U**
- UNDERDEBUG (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20
- unknown (*dpdispatcher.JobStatus.JobStatus* attribute), 21
- UNKOWNERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20

unsubmitted (*dpdispatcher.JobStatus.JobStatus* attribute), 21

unzip_file() (in module *dpdispatcher.dpcloudserver.zip_file*), 21

update_submission_state() (*dpdispatcher.submission.Submission* method), 47

upload() (*dpdispatcher.base_context.BaseContext* method), 23

upload() (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 27

upload() (*dpdispatcher.dpcloudserver.api.API* method), 20

upload() (*dpdispatcher.hdfs_context.HDFSContext* method), 29

upload() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 30

upload() (*dpdispatcher.local_context.LocalContext* method), 32

upload() (*dpdispatcher.ssh_context.SSHContext* method), 41

upload_() (*dpdispatcher.local_context.LocalContext* method), 32

upload_jobs() (*dpdispatcher.submission.Submission* method), 47

USERERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20

V

VERIFYERR (*dpdispatcher.dpcloudserver.retcode.RETCODE* attribute), 20

W

waiting (*dpdispatcher.JobStatus.JobStatus* attribute), 21

write_file() (*dpdispatcher.base_context.BaseContext* method), 23

write_file() (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 27

write_file() (*dpdispatcher.hdfs_context.HDFSContext* method), 29

write_file() (*dpdispatcher.lazy_local_context.LazyLocalContext* method), 30

write_file() (*dpdispatcher.local_context.LocalContext* method), 32

write_file() (*dpdispatcher.ssh_context.SSHContext* method), 41

write_home_file() (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 27

write_local_file() (*dpdispatcher.dp_cloud_server_context.DpCloudServerContext* method), 27

Z

zip_file_list() (in module *dpdispatcher.dpcloudserver.zip_file*), 21