context

Release 0.1

bakezq

May 10, 2021

CONTENTS:

CHAPTER

ONE

QUICK START

1.1 Installation

- Install from PYPI or from source.

```
pip install flowsaber
```

Or

```
https://github.com/zhqu1148980644/flowsaber.git
cd flowsaber && pip install .
```

1.2 Main concepts

For users who are familiar with nextflow usages can skip this part, other's can also check this for similar descriptions that comes from the introduction section of the nextflow documentation.

Basically there are several components that are exposed to users for building and configuring workflows.

1.2.1 Task

`Task` is the object represents the execution unit in your workflow. Tasks runs in parallel for different inputs.

1.2.2 Flow

`Flow` is the object that combines and records dependencies of all tasks you have put in. After it's been instantiated, it should be called for the actual dependency building and then can be executed by call `flow.execute()`.

1.2.3 Channel

`Channel` represents a flow/queue of data, it's consumed by Tasks. Any user defined data should be enqueued into a channel and then as arguments of Task/Flow.

1.2.4 Operator

`Operator` is built-in tasks that helps to manipulate the control of channels.

- For example: `map, sum, count, collect, flatten....`

1.3 A Typical workflow

The workflow defined below covers most sets of features supported by pyflow.

- Note: the output of the workflow may be different in different runs, this is due to the parallelism of task's executions for different inputs.

CHAPTER

TWO

TASK

There are four types of `Task` in total:

- `BaseTask` is base class for all tasks, and all operators are subclassed from it. Codes in `BaseTask` will always berun in the main thread and event loop. Due to this limitation, you should not put computation-cost logic in it, otherwise the whole eventloop will be blocked and hinder other task's execution.
- `Task` for running python functions. Execution of `run_job()` method will be scheduled and further executed by executor configured.
- `ShellTask` for executing bash script. Support `conda` and `image` option compared to `Task`. When these options are specified, additional bash command for handling enviroment creation, activation will berun before the execution of user defined bash commnd.

2.1 EnvTask

2.2 Operators

All operators has three ways to used:

- Operator's class name, a task object needs to be created before using. Usage: `Map()(channel)`
- Operator's function name, all operator has been wraped into a function accepts the same argument as the original operator class. Usage: `merge(ch1, ch2, ch3)`
- `Channel`'s method name, all operator has been added as a method of `Channel`. Usage: `ch1.map(fn)`

All predefined operators are:

- `Merge()`
- `Flatten()`
- `Mix()`
- ....

2.2.1 add custom operator

- Since operators are all `Tasks`, you can define your own operator task and even add it as a `Channel` method.

CHAPTER

THREE

EXECUTOR

3.1 Local

All jobsrun in local event loop, mainly used for debugging

3.2 Process

Jobs arerun in different processes.

3.3 Ray

The same as `Process` executor, besidesrun in a single machines, it's possible torun in different machines or even cloud services by carefully configuring `Ray`. Check the offical documentation for details.

CHAPTER

FOUR

INTERNAL

- The second call will generate a new task instance depends on `(ch1, ch2, ch3)` and return a new `Channel`.

```
task = Task()
output_ch = task(ch1, ch2, ch3)
```

- For each task instance, will get a unique `task_key` equals to it's working directory, since a task object can be used multiple times, so there may be multiple tasks share the same `task_key`.

```
task.task_key = classname-hash(task.run_job.__code__ + task.run.__annotation__ +␣
↪other_info)
task.task_workdir = task.config_dict.task_workdir/task.task_key
```

- For each input of `task.run`, the run will have a `run_key` equals to it's working directory

```
run.task_workdir = run_key = task.task_workdir/hash(inputs, other_info)
```

- All task objects with the same `task_key` will share the same `lock pool` and `cache`, different runs will be scheduled in parallel but runs with the same `run_key` will compete for a sharing `run_key` lock to avoid conflictions.
- For task receives no input channels, the task's will only be ran once

```
output_ch = task()
```

CHAPTER

FIVE

API REFERENCE

5.1 cli

5.2 task

5.3 operators

5.4 shell

5.5 flow

5.6 base

5.7 context

5.8 channel

5.9 state

5.10 cache

5.11 target

5.12 executor

5.13 runner

5.14 task_runner

5.15 flow_runner

5.16 models

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search