parallel-execute Documentation

Release latest

Jul 15, 2022

Contents

Python wrappers for easy multiprocessing and threading

Run multiple functions in parallel using threading or multiprocessing

CHAPTER 1

Installation

```
pip install parallel-execute
```

CHAPTER 2

Usage Example

2.1 1. Create a loom:

This takes a number of tasks and executes them using a pool of threads/process.

- To use threading

```
from pexecute.thread import ThreadLoom
loom = ThreadLoom(max_runner_cap=10)
```

- To use multiprocessing

```
from pexecute.process import ProcessLoom
loom = ProcessLoom(max_runner_cap=10)
```

max_runner_cap: is the number of maximum threads/processes to run at a time. You can add as many as functions you want, but only `n` functions will run at a time in parallel, `n` is the max_runner_cap

2.2 2. Add tasks in loom

- Add a function in loom using **add_function**

```
loom.add_function(f1, args1, kw1)
loom.add_function(f2, args2, kw2)
loom.add_function(f3, args3, kw3)
```

- Add multiple functions together using **add_work** method

```
work = [(f1, args1, kwargs1), (f2, args2, kwargs2), (f3, args3, kwargs3)]
loom.add_work(work)
```

2.3 3. Execute all tasks

After adding tasks, calling execute will return a dictionary of results corresponding to the keys or the order in which the tasks were added.

```
output = loom.execute()
```

key is the order in which the function was added and value is the return data of the function.

```
# Example:

def fun1():
   return "Hello World"

def fun2(a):
   return a

def fun3(a, b=0)
   return a+b

loom.add_function(fun1, [], {})
loom.add_function(fun2, [1], {})
loom.add_function(fun3, [1], {'b': 3})

output = loom.execute()
>>> output
{1: 'Hello World', 2: 1, 3: 4}
```

We can also provide a **key** to store the function return data.

```
# Example:
loom.add_function(fun1, [], {}, 'key1')
loom.add_function(fun2, [1], {}, 'fun2')
loom.add_function(fun3, [1], {'b': 3}, 'c')

output = loom.execute()
>>> output
{'key1': 'Hello World', 'fun2': 1, 'c': 4}
```