distex

Release 0.7.2

Ewald de Wit

Sep 28, 2022

CONTENTS

Release 0.7.2.

CHAPTER

ONE

POOL

class distex.pool.**Pool**(*num_workers=0*, *hosts=None*, *qsize=2*, *initializer=None*, *initargs=()*, *localhost=''*, *localport=0*, *lazy_create=False*, *worker_loop=LoopType.default*, *func_pickle=PickleType.dill*, *data_pickle=PickleType.pickle*)

Pool of local and remote workers that can run tasks.

To create a process pool of 4 local workers:

```
pool = Pool(4)
```

To create 8 remote workers on host `maxi`, using SSH (unix only):

```
pool = Pool(0, 'ssh://maxi/8')
```

`distex` must be installed on all remote hosts and the `distex_proc` script must be in the path. Test this with `ssh <host> distex_proc`. When using SSH it is not necessary to have a distex server running on the hosts.

When not using SSH a spawning server has to be started first on all hosts involved:

```
python3 -m distex.server
```

Warning: Only use this in a trusted network environment.

With the server running on host `mini`, to create a pool of 2 workers running there:

```
pool = Pool(0, 'mini/2')
```

Local, remote SSH and remote non-SSH workers can all be combined in one pool:

```
pool = Pool(4, ['ssh://maxi/8', 'mini/2'])
```

To give a SSH username or a non-default port such as 10022, specify the host as `'ssh://username@maxi:10022/8'`. It is not possible to give a password, use SSH keys instead: ssh-keygen can be used to create a key and ssh-copy-id to copy it to all hosts.

Parameters

- **num_workers** (`int`) – Number of local process workers. The default of 0 will use the number of CPUs.
- **hosts** – List of remote host specification strings in the format `[ssh://][username@]hostname[:portnumber]/num_workers`.

- **qsize** (int) – Number of pending tasks per worker. To improve the throughput of small tasks this can be increased from the default of 2. If no queueing is desired then it can be set to 1.
- **initializer** – Callable to initialize worker processes.
- **initargs** (tuple) – Arguments tuple that is unpacked into the initializer.
- **localhost** (str) – Local TCP server (if any) will listen on this address.
- **localport** (int) – Local TCP server (if any) will listen on this port (default: random open port).
- **lazy_create** (bool) – If True then no workers will be created until the first task is submitted.
- **worker_loop** (int) – LoopType to use for workers:
  0. default (=uvloop when available, proactor on Windows)
  1. asyncio (standard selector event loop)
  2. uvloop (Unix only)
  3. proactor (Windows only)
  4. quamash (PyQt)
- **func_pickle** (int) – PickleType to to use for serializing functions:
  0. pickle
  1. cloudpickle
  2. dill
- **data_pickle** (int) – PickleType to to use for data:
  0. pickle
  1. cloudpickle
  2. dill

distex.Pool implements the concurrent.futures.Executor interface and can be used in the place of ProcessPoolExecutor.

CHAPTER

TWO

POOLMAP

class distex.poolmap.**PoolMap**(*pool*, *func*, *timeout=None*, *chunksize=1*, *ordered=True*, *source=None*)

Map a function using a distributed pool with eventkit.

Parameters

- **func** – Function to map. If it returns an awaitable then the result will be awaited and returned.
- **timeout** – Timeout in seconds since map is started.
- **chunksize** – Source emits are chunked up to this size. A larger chunksize can greatly improve efficiency for small tasks.
- **ordered** –
  - `True`: The order of results preserves the source order.
  - `False`: Results are in order of completion.

CHAPTER

THREE

INTRODUCTION

Distex offers a distributed process pool to utilize multiple CPUs or machines. It uses asyncio to efficiently manage the worker processes.

Features:

- Scales from 1 to 1000's of processors;
- Can handle in the order of 50.000 small tasks per second;
- Easy to use with SSH (secure shell) hosts;
- Full async support;
- Maps over unbounded iterables;
- Compatible with concurrent.futures.ProcessPool (or PEP3148).

3.1 Installation

```
pip3 install -U distex
```

When using remote hosts then distex must be installed on those too. Make sure that the `distex_proc` script can be found in the path.

For SSH hosts: Authentication should be done with SSH keys since there is no support for passwords. The remote installation can be tested with:

```
ssh <host> distex_proc
```

Dependencies:

- Python version 3.6 or higher;
- On Unix the `uvloop` package is recommended: `pip3 install uvloop`
- SSH client and server (optional).

3.2 Examples

A process pool can have local and remote workers. Here is a pool that uses 4 local workers:

```
from distex import Pool

def f(x):
    return x*x

pool = Pool(4)
for y in pool.map(f, range(100)):
    print(y)
```

To create a pool that also uses 8 workers on host `maxi`, using ssh:

```
pool = Pool(4, 'ssh://maxi/8')
```

To use a pool in combination with eventkit:

```
from distex import Pool
import eventkit as ev
import bz2

pool = Pool()
# await pool  # un-comment in Jupyter
data = [b'A' * 1000000] * 1000

pipe = ev.Sequence(data).poolmap(pool, bz2.compress).map(len).mean().last()

print(pipe.run())  # in Jupyter: print(await pipe)
pool.shutdown()
```

There is full support for every asynchronous construct imaginable:

```
import asyncio
from distex import Pool

def init():
    # pool initializer: set the start time for every worker
    import time
    import builtins
    builtins.t0 = time.time()

async def timer(i=0):
    # async code running in the pool
    import time
    import asyncio
    await asyncio.sleep(1)
    return time.time() - t0

async def ait():
    # async iterator running on the user side
    for i in range(20):
```

(continues on next page)

(continued from previous page)

```
        await asyncio.sleep(0.1)
        yield i

async def main():
    async with Pool(4, initializer=init, qsize=1) as pool:
        async for t in pool.map_async(timer, ait()):
            print(t)
        print(await pool.run_on_all_async(timer))


asyncio.run(main())
```

3.3 High level architecture

Distex does not use remote 'task servers'. Instead it is done the other way around: A local server is started first; Then the local and remote workers are started and each of them will connect on its own back to the server. When all workers have connected then the pool is ready for duty.

Each worker consists of a single-threaded process that is running an asyncio event loop. This loop is used both for communication and for running asynchronous tasks. Synchronous tasks are run in a blocking fashion.

When using ssh, a remote (or 'reverse') tunnel is created from a remote Unix socket to the local Unix socket that the local server is listening on. Multiple workers on a remote machine will use the same Unix socket and share the same ssh tunnel.

The plain `ssh` executable is used instead of much nicer solutions such as AsyncSSH. This is to keep the CPU usage of encrypting/decrypting outside of the event loop and offload it to the `ssh` process(es).

3.4 Documentation

Distex documentation

author
Ewald de Wit <ewald.de.wit@gmail.com>

INDEX

P