Sistemes Distribuïts

LITHOPS

Què és Lithops?

“Lithops es pot definir com un toolkit python multi-cloud que permet escalar transparentment aplicacións locals, multi-procés, a recursos massius al Cloud”

Computació tradicional

Programació en serie → Programació en paral·lel

```python
import time

def f(x):
    time.sleep(10)
    return x*x

if __name__ == '__main__':
    for i in [1, 2, 3]:
        print(f(i))
```

```
josep@cloudlab04:~$ time python3.8 example_serial.py
1
4
9

real    0m30,055s
user    0m0,029s
sys     0m0,000s
```

```python
import time
from multiprocessing import Pool

def f(x):
    time.sleep(10)
    return x*x

if __name__ == '__main__':
    with Pool() as p:
        print(p.map(f, [1, 2, 3]))
```

```
josep@cloudlab04:~$ time python3.8 example_parallel.py
[1, 4, 9]

real    0m10,073s
user    0m0,050s
sys     0m0,021s
```

Computació tradicional

Programació en paral·lel (multiprocessing)

```
import time
from multiprocessing import Pool

def f(x):
    time.sleep(10)
    return x*x

if __name__ == '__main__':
    with Pool() as p:
        print(p.map(f, range(100)))
```

100 Tasques

```
josep@cloudlab04:~$ time python3.8 example_parallel.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 7
84, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2
500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 504
1, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464,
 8649, 8836, 9025, 9216, 9409, 9604, 9801]

real    4m40,288s
user    0m0,084s
sys     0m0,009s
```

Computació al Cloud

Programació en paral·lel (Lithops)

```python
import time
# from multiprocessing import Pool
from lithops.multiprocessing import Pool

def f(x):
    time.sleep(10)
    return x*x

if __name__ == '__main__':
    with Pool() as p:
        print(p.map(f, range(100)))
```

100 Tasques

```
on3.8 examples/tests/test.py
.dev2
ibm_cos -- IBM COS Storage client created - Region: us-east
f.ibm_cf -- IBM CF client created - Region: us-east - Namespace: josep_dev
xecutor created with ID: 73bbe9-0
2021-02-26 11:57:49,677 [INFO] lithops.invokers -- ExecutorID 73bbe9-0 | JobID M000 - Selected Runtime: lithopscloud/ibmcf-python-v38 - 256MB
2021-02-26 11:57:49,679 [INFO] lithops.job.job -- ExecutorID 73bbe9-0 | JobID M000 - Uploading function and data - Total: 4.4KiB
2021-02-26 11:57:50,686 [INFO] lithops.invokers -- ExecutorID 73bbe9-0 | JobID M000 - Starting function invocation: f() - Total: 100 activations
2021-02-26 11:57:50,868 [INFO] lithops.invokers -- ExecutorID 73bbe9-0 | JobID M000 - View execution logs at /tmp/lithops/logs/73bbe9-0-M000.log
2021-02-26 11:57:50,870 [INFO] lithops.executors -- ExecutorID 73bbe9-0 - Waiting for functions to complete

  100%|                                                                                                                                  | 100/100

2021-02-26 11:58:06,192 [INFO] lithops.executors -- ExecutorID 73bbe9-0 - Getting results
2021-02-26 11:58:07,010 [INFO] lithops.executors -- ExecutorID 73bbe9-0 - Cleaning temporary data
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024,
 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3
364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 688
9, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801]

real    0m17,792s
user    0m4,573s
sys     0m0,423s
```

Perquè Lithops?

- Una diferència clau amb els frameworks de computació tradicionals és que Lithops no requereix tasques de provisió i gestió de clústers.
- Lithops fa ús de plataformes serverless disponibles públicament, com IBM Cloud Functions.
- Un dels avantatges principals de l'ús de plataformes serverless és que ofereixen grans quantitats de CPU i memòria en pocs segons, pagant només pel temps exacte que les feu servir.

Com funciona Lithops?

- Lithops agafa les funcions, classes i variables definides localment i les transfereix al núvol. Això passa en temps d'execució.
- En segon pla, Lithops fa ús d'un servei d'emmagatzematge d'objectes (per exemple, IBM COS) per emmagatzemar aquesta informació i altres dades intermèdies.
- Lithops lliura el codi i les dades de l'usuari al núvol sense necessitat de conèixer com es desplega i s'executa a la plataforma serverless.

On encaixa Lithops?

- La naturalesa sense estat de les plataformes serverless actuals fa que Lithops sigui ideal per executar feines com ara:
  - Hyperparameter optimization
  - object storage data (pre-)processing
  - searching and processing logs
  - big data analytics
  - Monte Carlo simulations
  - Genomics analytics
  - web scraping
  - model scoring

Lithops APIs

- Multiprocessing API:

Lithops implementa la majoria dels mètodes i abstraccions de llibreria **multiprocessing** per executar tasques al núvol amb una API familiar. Per exemple, podem crear un pool () i utilitzar el mètode map () per generar una funció per a cada entrada d'una llista iterable:

```
from lithops.multiprocessing import Pool

def double(i):
    return i * 2

with Pool() as pool:
    result = pool.map(double, [1, 2, 3, 4, 5])
    print(result)
```

Lithops APIs

- Multiprocessing API:

multiprocessing.Process
multiprocessing.Pool
- apply()
- apply_async()
- map()
- map_async()

```
20  if __name__ == '__main__':
21      p = Process(target=function, args=(name,), kwargs={'language': 'english'})
22      p.start()
```

```
17  if __name__ == '__main__':
18      pool = Pool(processes= 4 )
19      count = pool.map(is_inside, part_count)
```

https://github.com/lithops-cloud/lithops/blob/master/docs/api_multiprocessing.md#pool

Lithops APIs

- Futures API:

Lithops implementa una API similar a la biblioteca **concurrent.futures**. Aquesta API es basa en objectes anomenats Futures, creats quan Lithops executa una funció. Amb aquest objecte Future, és possible accedir als resultats i algunes estadístiques sobre l'execució:

```
from lithops import FunctionExecutor


def hello(name):
    return 'Hello {}!'.format(name)


with FunctionExecutor() as fexec:
    future = fexec.call_async(hello, 'World')
    print(future.result())
```

Lithops APIs

- Futures API:

FunctionExecutor

call_async()

map()

map_reduce()

```python
import lithops

iterdata = [1, 2, 3, 4]


def my_map_function(x):
    return x + 7


def my_reduce_function(results):
    total = 0
    for map_result in results:
        total = total + map_result
    return total


if __name__ == "__main__":
    """
    By default the reducer will be launched within a Cloud Function
    when the local Lithops have all the results from the mappers.
    """
    fexec = lithops.FunctionExecutor()
    fexec.map_reduce(my_map_function, iterdata, my_reduce_function)
    print(fexec.get_result())
```

https://github.com/lithops-cloud/lithops/blob/master/docs/api_futures.md

Lithops APIs

- Storage API:

L'API d'emmagatzematge facilita el funcionament del backend d'emmagatzematge amb mètodes d'API senzills similars a la biblioteca python boto3.

L'API d'emmagatzematge juntament amb una API de càlcul proporciona una flexibilitat sense precedents per executar tasques al núvol com ara l'anàlisi de dades massives o qualsevol altre tipus d'aplicacions que impliquin anàlisi o gestió de dades.

Lithops APIs

- Storage API:

Storage

put_object()

get_object()

head_object()

post_object()

delete_object()

```
from lithops import FunctionExecutor, Storage

BUCKET = 'my-bucket'

def get_file(key, storage):
    return storage.get_object(bucket=BUCKET,
                              key=key))

if __name__ == "__main__":
    storage = Storage()
    storage.put_object(bucket=BUCKET,
                       key='test.txt',
                       body='Hello World')

    with FunctionExecutor() as fexec:
        fut = fexec.call_async(get_file, 'test.txt')
        print(fut.result())
```

https://github.com/lithops-cloud/lithops/blob/master/docs/api_storage.md

Lithops Multi-cloud

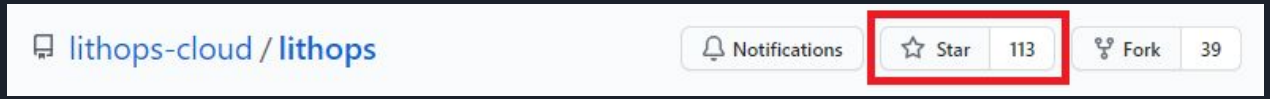
- El codi s'executa indistintament a qualsevol Cloud:

```python
from lithops import FunctionExecutor

def hello(name):
    return 'Hello {}!'.format(name)

with FunctionExecutor() as fexec:
    future = fexec.call_async(hello, 'World')
    print(future.result())
```

Lithops

open source

https://github.com/lithops-cloud/lithops



UNIVERSITAT ROVIRA I VIRGILI

IBM®