Bringing Scaling Transparency To Proteomic Applications With Serverless Computing

Presented at: Sixth International Workshop on Serverless Computing (WoSC6) 2020

Gil Vernik, Pedro Garcia-López, Mariano Mirabelli

Tuesday, December 8, 2020
IN WHAT CONTEXT DID THIS WORK TAKE PLACE?

➤ CloudButton Project

➤ In collaboration with URV CLOUDLAB

➤ In collaboration with IBM
WHAT ARE THE AIMS OF THIS WORK?

➤ Move Proteomics Work Queue application to Serverless.

➤ Demonstrate scaling transparency.

➤ Show cost/performance optimisations.
WHAT SCALING TRANSPARENCY MEANS?

“Scaling transparency means that applications can expand in scale without changes to the system structure or the application algorithms.”[1]

WHAT IS OUR WORK BASED ON?

- Work Queue framework
- Replica Exchange algorithm
- ProtoMol framework
REPLICA EXCHANGE WORK 
QUEUE ARCHITECTURE
REPLICA EXCHANGE
SERVERLESS ARCHITECTURE

Starting Main Program

Upload Proto/Map files in Deaggregated Storage

While Monte Carlo Step < Total Steps

Try/Merge Functions

Accept Results

Deaggregated Data Storage
SERVERLESS PROTOTYPES

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

COS Prototype

Redis Prototype
# EXPERIMENT SETUP

## Replica Exchange Configuration and Replicas by Execution

<table>
<thead>
<tr>
<th>Replica Exchange Values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Monte Carlo Steps</td>
<td>100</td>
</tr>
<tr>
<td>Default MD Steps</td>
<td>10000</td>
</tr>
<tr>
<td>Default Boundary Conditions</td>
<td>Vacuum</td>
</tr>
<tr>
<td>Default Output Frequency</td>
<td>10000</td>
</tr>
<tr>
<td>Default Physical Temperature</td>
<td>300</td>
</tr>
<tr>
<td>Minimum Temperature</td>
<td>300</td>
</tr>
<tr>
<td>Maximum Temperature</td>
<td>400</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Replicas Variation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Number of Replicas</td>
<td>12</td>
</tr>
<tr>
<td>Replicas Delta</td>
<td>12</td>
</tr>
<tr>
<td>Ending Number of Replicas</td>
<td>192</td>
</tr>
</tbody>
</table>
## RESULTS OBTAINED

Why is Redis Prototype cheaper?

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Work Queue</th>
<th>COS</th>
<th>Redis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg Function Time(secons)</td>
<td>0</td>
<td>38</td>
<td>30</td>
</tr>
<tr>
<td>Experiment Total Time(hours)</td>
<td>42.19</td>
<td>26.55</td>
<td>18.21</td>
</tr>
<tr>
<td>Total Function Cost</td>
<td>0</td>
<td>213.63</td>
<td>166.46</td>
</tr>
<tr>
<td>Worker Nodes Cost(VMs)</td>
<td>188.139</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Master Node Cost(VM)</td>
<td>8.43</td>
<td>5.31</td>
<td>3.46</td>
</tr>
<tr>
<td>Redis Node Cost(VM)</td>
<td>0</td>
<td>0</td>
<td>3.27</td>
</tr>
<tr>
<td>Total Price</td>
<td>196.57</td>
<td>218.94</td>
<td>173.38</td>
</tr>
</tbody>
</table>
RESULTS OBTAINED

Why Do Serverless Prototypes scale transparently?

```python
# Start of serverless function code
def serverless_task_process(task, ibm_cos):
    # ProtoMol invocation inside serverless through task object.
    cmd = "./" + task.input_remote_execn_file
    subprocess.call(cmd, shell = True)
# End of serverless function code

# For each Monte Carlo Step
# For each replica
    task = create_task(replica_id, local_temp_dir, bucket)
    tasks_list.append(task)
# End For each replica
lithops.map(serverless_task_process, tasks_list)
result_list = lithops.get_result()
# End For Monte Carlo Step
```

How many functions do we launch?

<table>
<thead>
<tr>
<th>Index</th>
<th>Réplicas</th>
<th>Functions Triggered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>1200</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>2400</td>
</tr>
<tr>
<td>`</td>
<td>replicas[index-1]+12</td>
<td>replicas[index]*Monte_Carlo_Steps</td>
</tr>
</tbody>
</table>

| 12 | 192 | 19200 |
HOW DOES LITHOPS FACILITATE ACCESS TO CLOUD RESOURCES?

Invoking Serverless Functions

```python
from lithops_multiprocessing import Pool

pool_client = Pool()
activation_list = pool_client.map(serverless_task_process, task_list_iterdata)
```

Accessing COS

```python
from lithops_multiprocessing.cloud_proxy import open as cloud_open

def read_from_remote_storage(filename):
    with cloud_open(filename, 'rb') as f:
        lines = f.readlines()
    return lines
```

Accessing Redis

```python
from lithops_multiprocessing import Manager

shared_map = Manager().dict()
res = shared_map['key']
```
MAIN CONCLUSIONS

➤ Serverless prototypes reduce the total execution time of the Replica Exchange by around forty percent.

➤ Serverless prototypes scale transparently.

➤ Serverless solution can obtain a comparable or even cheaper cost than a serverful one using Work Queue over VM.
THANK YOU!

- **Links to our work:**
  - [https://www.serverlesscomputing.org/wosc6/#p10](https://www.serverlesscomputing.org/wosc6/#p10)
  - [https://github.com/faas-prototypes/protomol](https://github.com/faas-prototypes/protomol)

- **Contact:**
  - mmirabelli@uoc.edu
  - pedro.garcia@urv.cat
  - gilv@il.ibm.com