Performance considerations on execution of large scale workflow applications on cloud functions

M. Pawlik, K. Figiela, M. Malawski
m.pawlik@cyfronet.pl, {kfigiela,malawski}@agh.edu.pl

AGH University of Science and Technology
Kraków, Poland
Presentation plan

1. Introduction
2. Objectives
3. Methodology
4. Results
5. Conclusions and future work
Scientific workflows and cloud

- **Workflow - paradigm of implementing and preserving scientific process.**
  - graph representation
  - allow for modeling complex procedures
  - provide a level abstraction over implementation details and infrastructure
  - enable parallelization
  - Workflow Management System is required to execute the workflow

- **IaaS Cloud as an execution environment:**
  - dynamic infrastructure provisioning
  - elastic billing models

Montage workflow
Scientific workflows and FaaS

- Even greater elasticity.
- No need to manually provision the infrastructure.
- Very fine billing granularity.

- Some limitations:
  - single task runtime limit
  - limited set of function configurations (memory tied to cpu etc.)
  - reduced control (eg. cold starts)
  - introduction of overheads etc.
Objectives

Measure parameters significant for assumed scenario: computation offloading to FaaS:

1. Performance specific for scientific applications
2. Infrastructure provisioning time
3. Overhead of the API (REST)
4. Other interesting characteristics

Provide basis for constructing performance models of scientific large scale workflows, which will allow for improvements in scheduling algorithms.
Studied infrastructures

- Amazon: AWS Lambda (eu-west-1, 256MB, 512MB, 1024MB, 1536MB, 2048MB, 3008MB)
- Google Cloud Functions (us-central1, 256MB, 512MB, 1024MB, 2048MB)
- IBM Cloud Functions (UK, 256MB, 512MB)

- Most providers don’t supply exact information about infrastructure
- Even if they do it’s just a rough estimate
Benchmarking toolkit

- In-house developed workflow management system: HyperFlow
  - https://github.com/hyperflow-wms
  - written in Node.js, easy to use and extensible
  - supports FaaS

- Testing application: a “bag of tasks” workflow with 5120 tasks
  - exceeds limits imposed by most providers

- Testing load: Linpack
  - problem size of 3408x3408
  - concentrates on raw computing power (FLOPS)

- Automation and reproducibility
Experiment setup

- Local workflow orchestrator, workload was offloaded to FaaS.
- Instrumentation was added to WMS and Cloud functions to gather detailed traces of execution.
- Task execution was divided into three stages:
Performance results
Performance results, cont.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>256</td>
<td>2.95</td>
<td>1.38</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>4.62</td>
<td>1.40</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>10.10</td>
<td>4.27</td>
</tr>
<tr>
<td></td>
<td>1536</td>
<td>14.04</td>
<td>7.18</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>27.26</td>
<td>8.37</td>
</tr>
<tr>
<td></td>
<td>3008</td>
<td>27.05</td>
<td>8.40</td>
</tr>
<tr>
<td>GCF</td>
<td>256</td>
<td>6.92</td>
<td>6.23</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>9.54</td>
<td>3.03</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>16.11</td>
<td>1.60</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>20.23</td>
<td>2.03</td>
</tr>
<tr>
<td>IBM</td>
<td>256</td>
<td>7.35</td>
<td>3.47</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>7.15</td>
<td>3.50</td>
</tr>
</tbody>
</table>
Task start delay
Infrastructure provisioning, Gantt charts, AWS
Infrastructure provisioning, Gantt charts, Google
Infrastructure provisioning, Gantt charts, IBM
Response time
Response time
Patterns in measured run time

Concept of “remainder time”: \( t_r = t \mod 100\text{ms} \)

- functions are billed for each 100ms
- runtime is rounded to nearest 100ms
Remainder time
Conclusions and Future work

- Presented work depicts significant characteristics of offered FaaS services.
- In terms of performance:
  - it varies significantly
  - largest function usually gives the least gains
  - faster function doesn’t always translate to shortest timespan
- HTTP API overhead is significant.

- Release full source code and automation scripts.
- Construct performance model and incorporate it into scheduling.
- Scheduling validation.
References

Thank you for attention!

Any questions?

http://cloud-functions.icsr.agh.edu.pl/dashboard/db/providers

contact: m.pawlik@cyfronet.pl, malawski@agh.edu.pl