UniFaaS: Programming across Distributed Cyberinfrastructure with Federated Function Serving

Yifei Li*, Ryan Chard†, Yadu Babuji††, Kyle Chard†††, Ian Foster†††, Zhuozhao Li*

* Dept. of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China
† Dept. of Computer Science, University of Chicago, Chicago, IL, USA
‡ Data Science and Learning Division, Argonne National Laboratory, Lemont, IL, USA
Motivation

- Executing scientific workflows across cyberinfrastructure (CI) amortizing queue times, distributed data, specialized accelerator etc.

- When executing distributed scientific workflows
  - **funcX**
    - Pros: Easy to build a distributed computing resource pool
    - Cons: Independent execution, manual data staging, limitations of input/output size
  - **Parsl**
    - Pros: Support the DAG workflow, data staging (e.g. FTP, HTTP)
    - Cons: Complicated to execute workflows on distributed CI simultaneously

- What about funcX as an executor of Parsl?
  - Things can be resolved immediately
    - easy to program (in Parsl’s way), distributed execution
  - Things to be resolved
    - data management, performance (scheduling)
@function
def compute_fingerprint(GlobusFile: mol_file):
    from rdkit import *
    import GlobusFile
    mol_path = mol_file.get_remote_file_path()
molecule = open(mol_path).readline()
fp = AllChem.GetMorganFingerprint(Chem.MolFromSmiles(molecule), 2)
out_file = GlobusFile.create("fp.txt")
out_path = out_file.get_remote_file_path()
open(out_path, 'w').write(fp)
return out_file

- a shim layer to wrap data and R/W ops.
- a decorator like @python_app in Parsl
UniFaaS Scheduling

Goal: to minimize workflow’s makespan

Challenges: varying data staging time, dynamic resource capacity.

Intuition:
- Data staging problem: start it as early as possible
- Dynamic resource capacity: real-time scheduling

Dynamic heterogeneity-aware scheduling (DHA in short)
UniFaaS Scheduling

Goal: to minimize workflow’s makespan

Challenges: varying data staging time, dynamic resource capacity.

Intuition:
- Data staging problem: start it as early as possible
- Dynamic resource capacity: real-time scheduling

Dynamic heterogeneity-aware scheduling (DHA in short)
UniFaaS Scheduling

No prior knowledge

• Locality-aware scheduling for dynamic resource capacity
  schedule based on real-time status (real-time)

• Capacity-aware scheduling for static resource capacity
  schedule when the DAG enters our system (offline)

<table>
<thead>
<tr>
<th>Scheduling type</th>
<th>Capacity</th>
<th>Locality</th>
<th>DHA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic DAG supported</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dynamic resource supported</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Knowledge required</td>
<td>✗</td>
<td>❌</td>
<td>✓</td>
</tr>
</tbody>
</table>
Experiment

Latency

< 1 ms  →  < 1 ms  →  < 1 ms  →  < 1 ms
Scheduling  →  Data management  →  Submission  →  Remote execution

< 1 ms  →  < 1 ms  →  < 1 ms
Result logging  →  Result polling  →  t_e = 62 ms

\( t_h = 2 \text{ ms} \)

\( t_d = 726 \text{ ms} \)

\( t_{\text{sub}} = 4 \text{ ms} \)

\( t_e = 62 \text{ ms} \)

\( t_{\text{h}} = 2 \text{ ms} \)

One “hello world” task with a 1 MB input totally costs 1087 ms.

**OVERHEAD OF DIFFERENT ALGORITHMS.**

<table>
<thead>
<tr>
<th>Scheduling algorithm</th>
<th>Overhead (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>( 1.72 \times 10^{-4} )</td>
</tr>
<tr>
<td>Locality</td>
<td>( 3.00 \times 10^{-3} )</td>
</tr>
<tr>
<td>DHA</td>
<td>( 3.46 \times 10^{-3} )</td>
</tr>
</tbody>
</table>

All algorithms have a modest overhead.
Experiment

Scalability

Scalability of 5-second tasks is close to the ideal for up to 12 endpoints. Longer-duration tasks, better scaling.
Experiment

Case study

1. DHA has the best performance and highest worker utilization.
2. Improved performance by 22.99%, while utilizing only an additional 19.48% of resources.

Execute the drug screening workflow under static resource capacity.
Questions?

Yifei Li
12232396@mail.sustech.edu.cn
Case study: dynamic capacity

1. DHA has the best performance.
2. Locality is better than DHA without re-scheduling.

### Experiment Results

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Makespan (s)</th>
<th>Transfer size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>3,070</td>
<td>3.54</td>
</tr>
<tr>
<td>Locality</td>
<td>2,507</td>
<td>58.93</td>
</tr>
<tr>
<td>DHA without re-scheduling</td>
<td>2,880</td>
<td>55.36</td>
</tr>
<tr>
<td>DHA</td>
<td>1,727</td>
<td>43.32</td>
</tr>
</tbody>
</table>

**Execute the drug screening workflow under dynamic resource capacity.**