Project 39: Tutorial to use K8S Cluster

Qiskit Advocate Mentorship Program: Spring 2021

Mentor : Hiroshi Horii, IBM Research, Japan

Mentee : Anuj Mehrotra, India

Qiskit

Project Overview

Problem Understanding

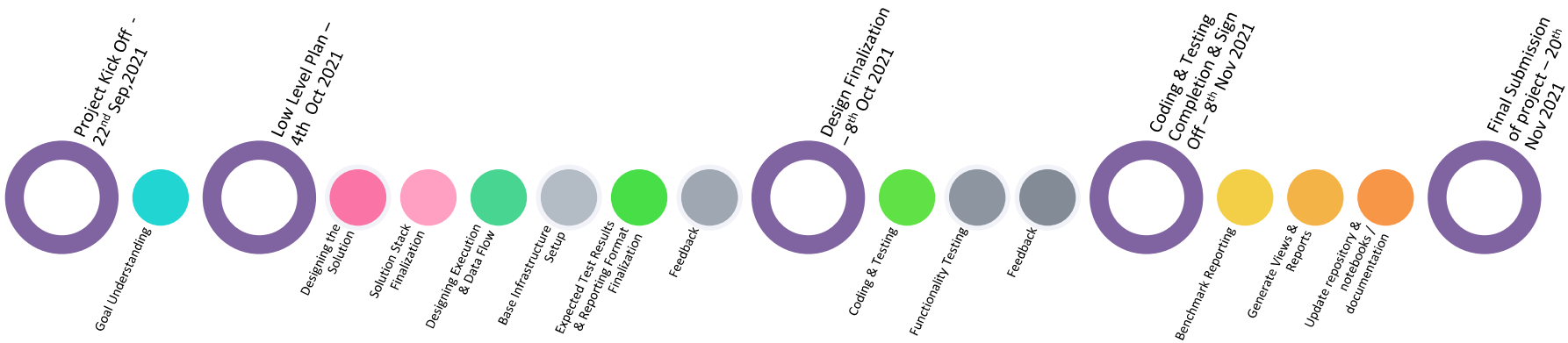
- ✓ Simulation of quantum applications like quantum chemistry, materials science, quantum biology, generates quantum systems which are much larger than computational NISQ devices, this gap can be handled by parallelizing the quantum simulation.
- ✓ Qiskit Aer is a Noisy quantum circuit simulator backend and runs simulation jobs on a single-worker Python multiprocessing ThreadPool executor so that all parallelization is handled by low-level OpenMP and CUDA code (For Multi CPU / Core & GPU environment).
- ✓ Variational Quantum Eigensolver (VQE and the other compute intensive variational algorithms) already support generating circuits together for parallel gradient computation. In order to customize job-level parallel execution of multiple circuits, a custom multiprocessing executor can be specified, which controls the splitting of circuits using the **executor** and **max_job_size** backend options of **Qiskit AerSimulator**.
- ✓ In order to simulate parallel execution on premise as well as cloud environment, dedicated compute resources might be required, but available with constraints, both on scalability and manageability.

Qiskit

Solution Overview

- ✓ For large scale job parallelization on HPC clusters, Qiskit Aer executors also support the distributed Clients from the **DASK** (parallel computing library for Python). DASK natively scales Python and suitable for applications which require a distributed, auto scaling compute environment that is completely independent of application.
- ✓ Using Kubernetes clusters, DASK worker environment can be either scaled up manually, or can be scaled as the need arises by creating auto scaling rules in Kubernetes configuration, which means DASK only need to manage scheduling across workers in Kubernetes Cluster , as work go up or down.
- ✓ The AerSimulator supports multiple simulation methods and configurable options for each simulation method. These may be set using the appropriate kwargs during initialization. They can also be set of updated using the set_options() method. Adds a new option of the backend to provide the user's executor.
- ✓ When user gives Dask client as executor, Aer can execute a simulation on the distributed machines like HPC clusters. When the executor is set, AerJobSet object is returned instead of a normal AerJob object. AerJobSet divides multiple experiments in one **qobj** into each experiment and submits each qobj to the executor as AerJob. After simulations, AerJobSet collects each result and combines them into one result object

Implementation Plan



Deliverables:

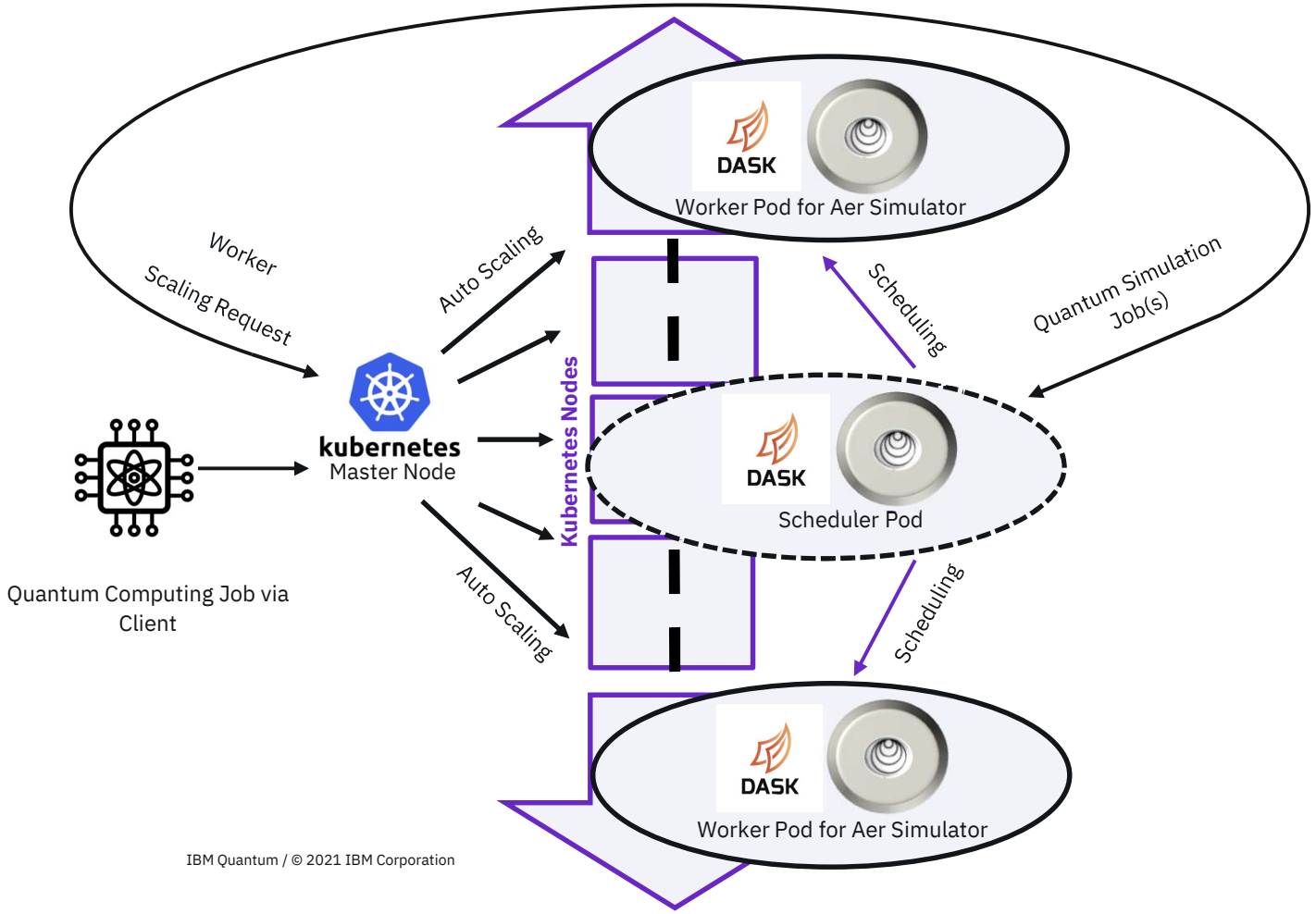
- ✓ Github Repository (with sample codes ,Tutorial, report)
- ✓ Project Report (with test results)
- ✓ Contribute to Qiskit Aer repository (raising issues /bug reporting, share resolutions)

Current Status:

- ✓ Project Goal Understanding Complete.
- ✓ Infrastructure Design & Setup Complete using DASK, Kubernetes, Python 3.8 on Ubuntu 20.04 LTS OS platform.
- ✓ Proof of Concept Testing Completed
- ✓ Contributed to Qiskit Aer repository : Issue #1364 : Serialization Error while submitting Quantum Objects to compute engine (DASK workers) of Qiskit Aer.
- ✓ Issue #1364 was fixed by #1365 PR from Hitomi Takahashi (Core Developer for Clustered Aer Backend).

Architecture of Clustered Backend for Aer Simulator

Qiskit



Qiskit

Environment Overview

Application Stack

Qiskit

Ubuntu 20.04 LTS with 4vCPU

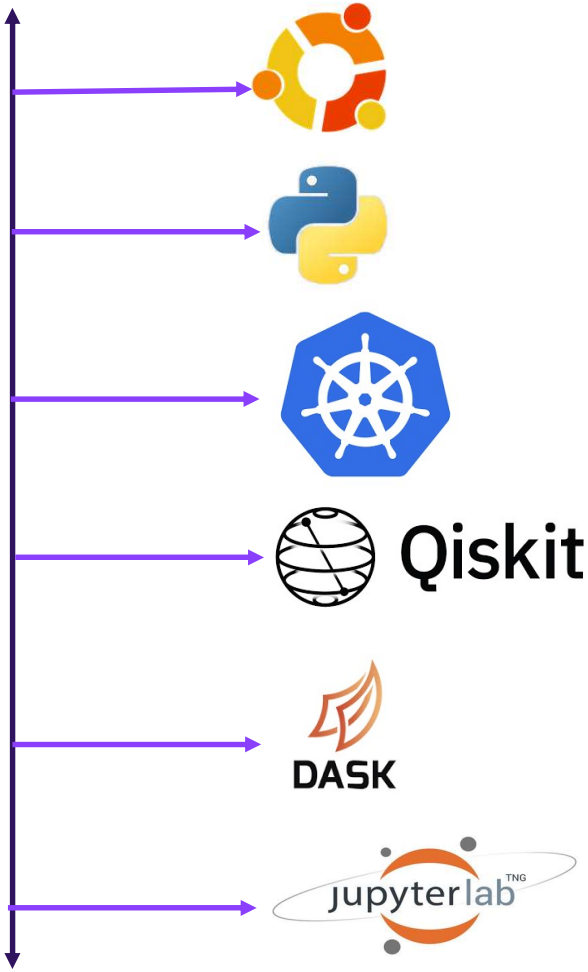
Python 3.8

Kubernetes v1.22.2

Qiskit v 0.30

Dask Kubernetes v2021.10.0

Jupyter Lab



Kubernetes Cluster Environment for Project

Qiskit

```
$ kubectl get nodes -A # One Master & 2 Worker Nodes
NAME                              STATUS   ROLES                  AGE     VERSION
ip-172-20-33-106.ec2.internal     Ready    node                   2d19h   v1.22.2
ip-172-20-51-193.ec2.internal     Ready    node                   2d19h   v1.22.2
ip-172-20-61-190.ec2.internal     Ready    control-plane,master   2d19h   v1.22.2

$ kubectl get pods -A # Pods for Kubernetes Environment
NAMESPACE     NAME                                                    READY   STATUS    RESTARTS         AGE
kube-system   coredns-5dc785954d-82nf5                                1/1     Running   0                2d19h
kube-system   coredns-5dc785954d-r5ksr                                1/1     Running   0                2d19h
kube-system   coredns-autoscaler-84d4cfd89c-hzn5k                     1/1     Running   0                2d19h
kube-system   dns-controller-c459588c4-rd7rd                          1/1     Running   0                2d19h
kube-system   ebs-csi-controller-bf875d89b-r4rcz                      6/6     Running   0                2d19h
kube-system   ebs-csi-node-dzvl9                                      3/3     Running   0                2d19h
kube-system   ebs-csi-node-smffl                                      3/3     Running   0                2d19h
kube-system   ebs-csi-node-spdzr                                      3/3     Running   0                2d19h
kube-system   etcd-manager-events-ip-172-20-61-190.ec2.internal       1/1     Running   0                2d19h
kube-system   etcd-manager-main-ip-172-20-61-190.ec2.internal         1/1     Running   0                2d19h
kube-system   kops-controller-sr5lp                                   1/1     Running   0                2d19h
kube-system   kube-apiserver-ip-172-20-61-190.ec2.internal            2/2     Running   1 (2d19h ago)    2d19h
kube-system   kube-controller-manager-ip-172-20-61-190.ec2.internal   1/1     Running   3 (2d19h ago)    2d19h
kube-system   kube-proxy-ip-172-20-33-106.ec2.internal                1/1     Running   0                2d19h
kube-system   kube-proxy-ip-172-20-51-193.ec2.internal                1/1     Running   0                2d19h
kube-system   kube-proxy-ip-172-20-61-190.ec2.internal                1/1     Running   0                2d19h
kube-system   kube-scheduler-ip-172-20-61-190.ec2.internal            1/1     Running   0                2d19h
```

Sample Dask Worker Pod Specification for Aer Simulator

Qiskit

 # worker-spec.yml

```
kind: Pod
metadata:
  labels:
    foo: bar
spec:
  restartPolicy: Never
  containers:
  - image: daskdev/dask:latest
    imagePullPolicy: IfNotPresent
    args: [dask-worker, --nthreads, '3', --no-dashboard, --memory-limit, 1GB, --death-timeout, '60']
    name: dask
    env:
      - name: EXTRA_PIP_PACKAGES
        value: git+https://github.com/dask/distributed
        name: EXTRA_PIP_PACKAGES
        value: git+https://github.com/Qiskit/qiskit
    resources:
      limits:
        cpu: "1"
        memory: 1G
      requests:
        cpu: "1"
        memory: 1G
```

Qiskit

Test Results

Running Simple Script for Testing Dask Cluster

Qiskit

#Python Script for Dask Array Mean Calculation

```
from dask_kubernetes import KubeCluster
import time
cluster = KubeCluster('worker-spec.yml')
cluster.scale(3)  # specify number of workers
explicitly

cluster.adapt(minimum=1, maximum=10)  # or
dynamically scale based on current workload
time.sleep(60)
# Example usage
from dask.distributed import Client
import dask.array as da

# Connect Dask to the cluster
client = Client(cluster)

# Create a large array and calculate the mean
array = da.ones((1000, 1000, 1000))
print("Ärray Mean", array.mean().compute())# Should
print 1.0
```

#During Script execution Dask Worker Pod(s) are automatically spawned on Kubernetes Nodes as per load

```
$ kubectl get pods -A|grep -i dask
default       dask-ubuntu-c3d47733-3xx6k8  1/1     Running   0  18s
```

#Script Output

```
Creating scheduler pod on cluster. This may take some time.
distributed.deploy.adaptive - INFO - Adaptive scaling started:
minimum=1 maximum=10
distributed.deploy.adaptive - INFO - Retiring workers [0, 1]
/usr/local/lib/python3.8/dist-packages/distributed/client.py:1128:
VersionMismatchWarning: Mismatched versions found

+-------------+-----------------------+-----------+---------+
| Package     | client                | scheduler | workers |
+-------------+-----------------------+-----------+---------+
| blosc       | None                  | 1.10.2    | None    |
| dask        | 2021.09.1             | 2021.10.0 | None    |
| distributed | 2021.09.1+43.g842cc758 | 2021.10.0 | None    |
| lz4         | None                  | 3.1.3     | None    |
+-------------+-----------------------+-----------+---------+

warnings.warn(version_module.VersionMismatchWarning(msg[0]["warning
"]))
Ärray Mean 1.0
distributed.deploy.adaptive_core - INFO - Adaptive stop
distributed.deploy.adaptive_core - INFO - Adaptive stop
```

Sample VQE Script for Testing Dask Cluster Performance


```
import numpy as np
import multiprocessing
from time import time
import timeit
from qiskit_nature.circuit.library import HartreeFock, UCCSD
from qiskit_nature.converters.second_quantization import QubitConverter
from qiskit_nature.drivers import UnitsType, Molecule
from qiskit_nature.drivers.second_quantization.pyscfd import PySCFDriver
from qiskit_nature.mappers.second_quantization import JordanWignerMapper, ParityMapper
from qiskit_nature.problems.second_quantization import ElectronicStructureProblem
from qiskit_nature.algorithms import GroundStateEigensolver
mol_string='H .0 .0 .0; Li .0 .0 2.5'
threads = 12
max_evals_grouped = 128
driver = PySCFDriver(atom=mol_string, unit=UnitsType.ANGSTROM, basis='sto3g')
es_problem = ElectronicStructureProblem(driver)
qubit_converter = QubitConverter(JordanWignerMapper())

from qiskit.algorithms.optimizers import SLSQP
optimizer = SLSQP(maxiter=5000)
from qiskit.providers.aer import AerSimulator
from qiskit.utils import QuantumInstance
import logging
logging.basicConfig(format='%(asctime)s %(levelname)-8s %(message)s',
                    level=logging.DEBUG,
                    datefmt='%Y-%m-%d %H:%M:%S')
from qiskit_nature.algorithms import VQEUCCFactory
simulator = AerSimulator()
quantum_instance = QuantumInstance(backend=simulator)
start = timeit.default_timer()
```

Execution Time without Parallelism

```
#Results
=== GROUND STATE ENERGY ===

* Electronic ground state energy (Hartree): -8.458691730495
  - computed part:      -8.458691730495
~ Nuclear repulsion energy (Hartree): 0.635012653104
> Total ground state energy (Hartree): -7.823679077391

=== MEASURED OBSERVABLES ===

  0:  # Particles: 4.000 S: 0.000 S^2: 0.000 M: -0.000

=== DIPOLE MOMENTS ===

~ Nuclear dipole moment (a.u.): [0.0  0.0  14.17294593]

  0:
  * Electronic dipole moment (a.u.): [-0.00000216  -0.0000072  12.7343175]
    - computed part:      [-0.00000216  -0.0000072  12.7343175]
  > Dipole moment (a.u.): [0.00000216  0.0000072  1.43862843]  Total: 1.43862843
                 (debye): [0.0000055  0.0000183  3.6566284]  Total: 3.6566284

Execution Time without Parallelism:  727.362411745009
```

Execution Time with ThreadPool Parallelism (=2 Workers)

Qiskit

```
#Configuration
from concurrent.futures import ThreadPoolExecutor
exc = ThreadPoolExecutor(max_workers=2)
simulator = AerSimulator(executor=exc)
```

```
#Results
=== GROUND STATE ENERGY ===

* Electronic ground state energy (Hartree): -8.458691735856
  - computed part:      -8.458691735856
~ Nuclear repulsion energy (Hartree): 0.635012653104
> Total ground state energy (Hartree): -7.823679082752

=== MEASURED OBSERVABLES ===

  0:  # Particles: 4.000 S: 0.000 S^2: 0.000 M: -0.000

=== DIPOLE MOMENTS ===

~ Nuclear dipole moment (a.u.): [0.0  0.0  14.17294593]

  0:
  * Electronic dipole moment (a.u.): [-0.00000348  0.00000499  12.73429606]
    - computed part:      [-0.00000348  0.00000499  12.73429606]
  > Dipole moment (a.u.): [0.00000348  -0.00000499  1.43864987]  Total: 1.43864987
                  (debye): [0.00000884  -0.00001268  3.65668289]  Total: 3.65668289

Execution Time with ThreadPool Parallelism across 2 workers:  734.7169829360209
```

Execution Time with Dask Cluster Parallelism (=3 Workers)

Qiskit

```
#Configuration
from dask_kubernetes import KubeCluster
    import time
    cluster = KubeCluster('worker-spec.yml')
    cluster.scale(3)  # specify number of workers explicitly

    cluster.adapt(minimum=2, maximum=10)  # or dynamically scale based on current workload
    time.sleep(10)
    from dask.distributed import Client
    exc_dask_nw3 = Client(cluster)
```

#During Script execution Dask Worker Pod(s) are automatically spawned on Kubernetes Nodes as per load

```
$ kubectl get pods -A -o wide|grep -i dask
default       dask-ubuntu-2e887251-7nbtx9                       1/1     Running   0             2m11s   100.96.2.24
ip-172-20-33-106.ec2.internal   <none>           <none>
default       dask-ubuntu-2e887251-7p8rb2                       0/1     Pending   0             2m11s   <none>
<none>                          <none>           <none>
default       dask-ubuntu-2e887251-7xtcsz                       1/1     Running   0             5m6s    100.96.1.18
ip-172-20-51-193.ec2.internal   <none>           <none>
```

Execution Time with Dask Cluster Parallelism (=3 Workers)

Qiskit

```
#Results
SymPyDeprecationWarning(feature="expr_free_symbols method",
=== GROUND STATE ENERGY ===

* Electronic ground state energy (Hartree): -8.458691722957
  - computed part:      -8.458691722957
~ Nuclear repulsion energy (Hartree): 0.635012653104
> Total ground state energy (Hartree): -7.823679069853

=== MEASURED OBSERVABLES ===

  0:  # Particles: 4.000 S: 0.000 S^2: 0.000 M: -0.000

=== DIPOLE MOMENTS ===

~ Nuclear dipole moment (a.u.): [0.0  0.0  14.17294593]

  0:
  * Electronic dipole moment (a.u.): [0.00000941  0.00000262  12.73432137]
    - computed part:      [0.00000941  0.00000262  12.73432137]
  > Dipole moment (a.u.): [-0.00000941  -0.00000262  1.43862456]  Total: 1.43862456
                 (debye): [-0.00002391  -0.00000666  3.65661855]  Total: 3.65661855

Execution Time with three  Dask Cluster worker(s):  1041.7258470460074
distributed.deploy.adaptive_core - INFO - Adaptive stop
distributed.deploy.adaptive_core - INFO - Adaptive stop
```