Project 39: Tutorial to use K8s Cluster

Qiskit Advocate Mentorship Program: Fall 2021

Mentor : Hiroshi Horii, IBM Research, Japan

Mentee : Anuj Mehrotra, India

Qiskit

Project Overview

Motivation

Qiskit Aer contains a new option, **"executor",** to use a custom executor which supports **Threadpool** Executor and DASK clients. This option can be scaled up easily and speed up simulation with parallelization.

Using a DASK cluster with multiple container nodes on a Kubernetes environment , **Aer** can execute the simulation in parallel across the multiple nodes like HPC environment. Especially, the simulation time of multiple circuits and a noise simulation can much decrease based on the number of worker nodes because multiple nodes can independently run different simulations.

Goal

“Publish a Tutorial for setting up a **DASK** clustered backend for **Qiskit Aer** on **Kubernetes Cluster** (K8s) environment”

Problem Understanding

Qiskit

Scenario

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.

Use Case

Variational Quantum Eigensolver (VQE) and the other compute intensive variational algorithms already support generating circuits together for parallel gradient computation.

Available Solution

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).

Limitation(s)

In order to simulate parallel execution on premise as well as on cloud environment, dedicated compute resources might be required, but available with constraints, both on scalability and manageability.

Solution Overview

Qiskit

Suitability

For large scale job parallelization Qiskit Aer **executors** also support the distributed Clients from the **DASK** (parallel computing library for Python) as **clustered backend**. DASK natively scales Python and suitable for applications which require a distributed, auto scaling compute environment that is completely independent of application.

Scalability

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.

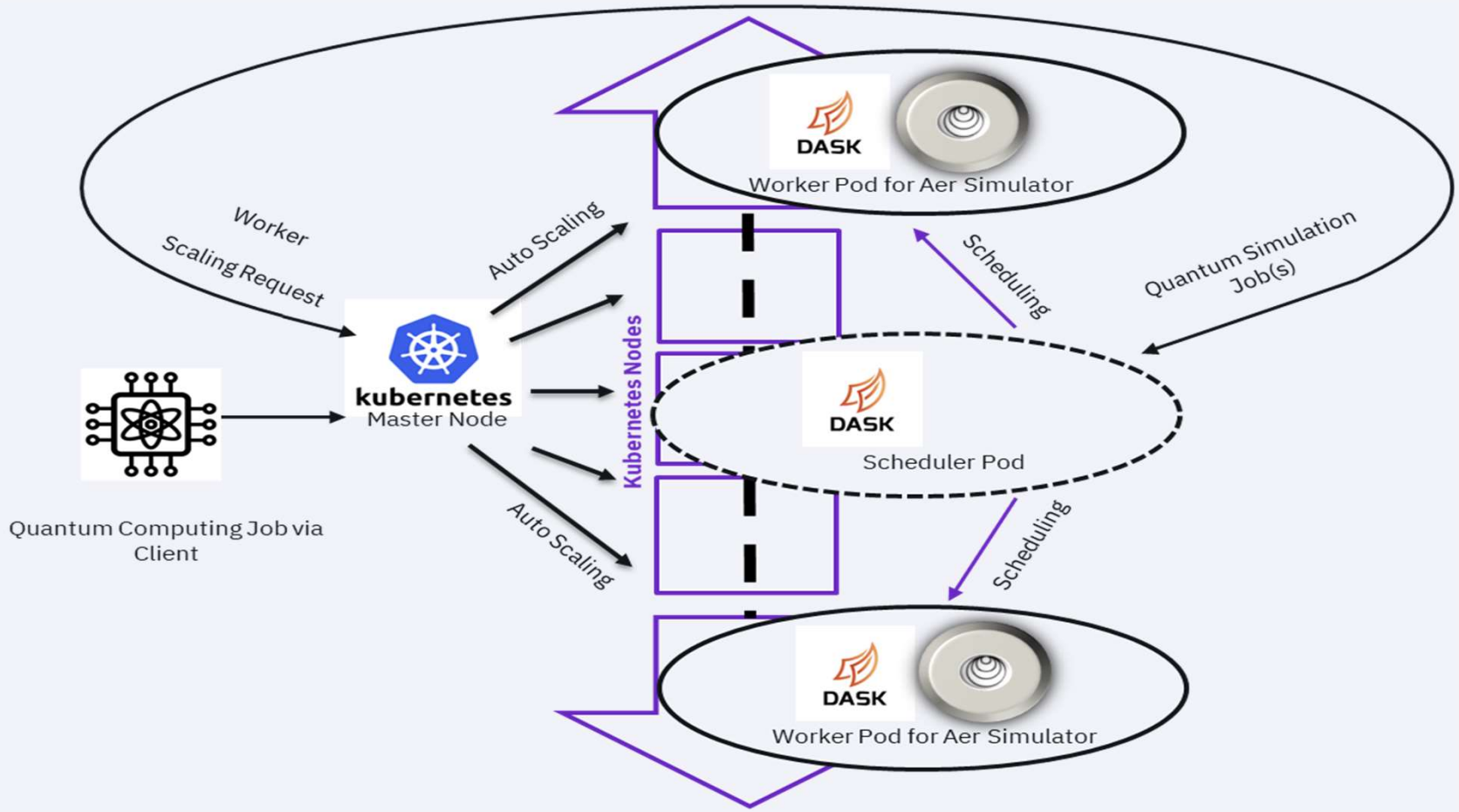
Supportability

The **Aer** supports multiple simulation methods and configurable options for each simulation method. These may be set using the appropriate arguments 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.

Serviceability

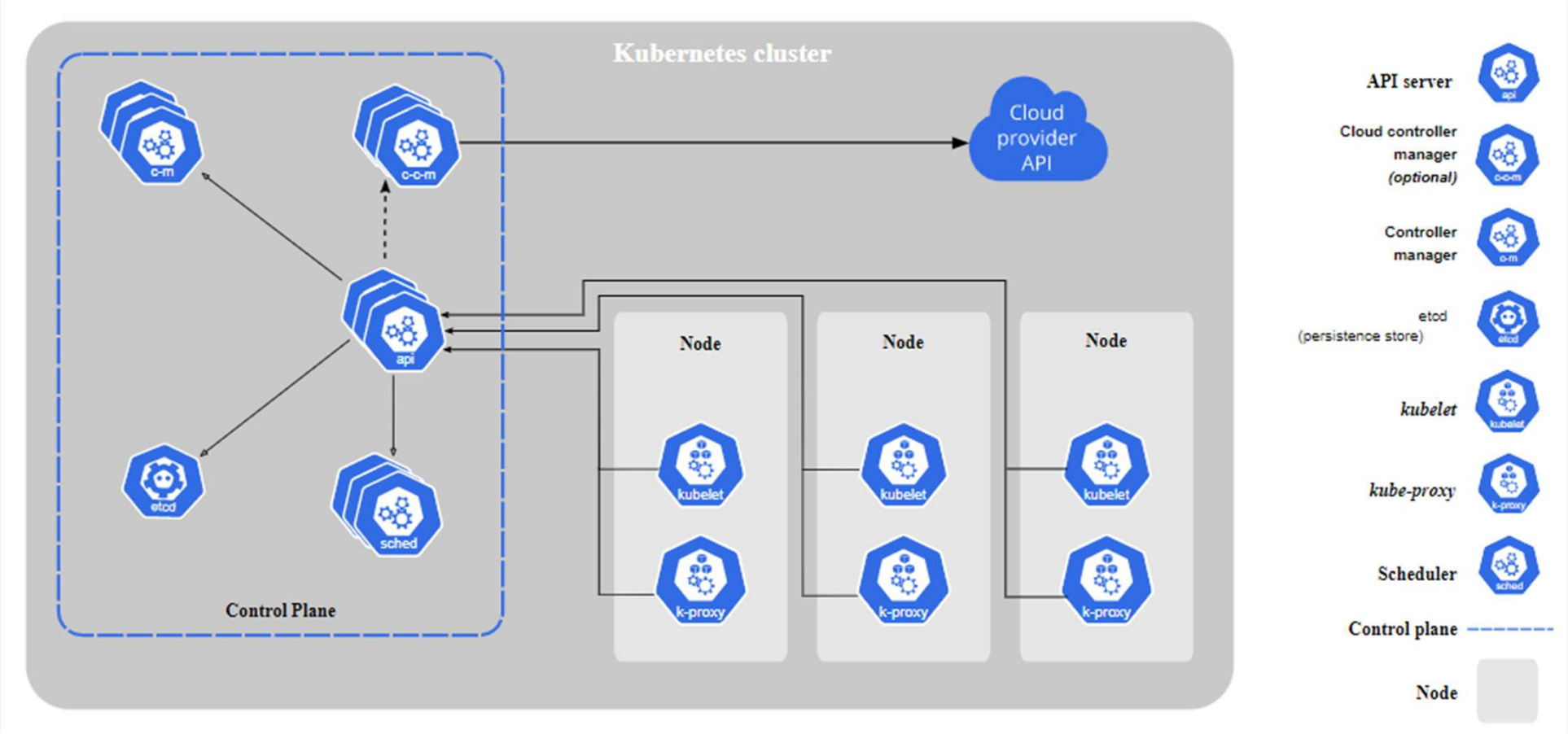
When user gives Dask client as **executor**, **AerJobSet** object is returned instead of a normal **AerJob** object. **AerJobSet** divides multiple experiments within one **qobj** into each experiment and submits each qobj to the executor as AerJob. After simulations, **AerJobSet** collects each result and combines them into one result object

Architecture of Clustered Backend for Aer Simulator



Solution Component : Kubernetes Cluster

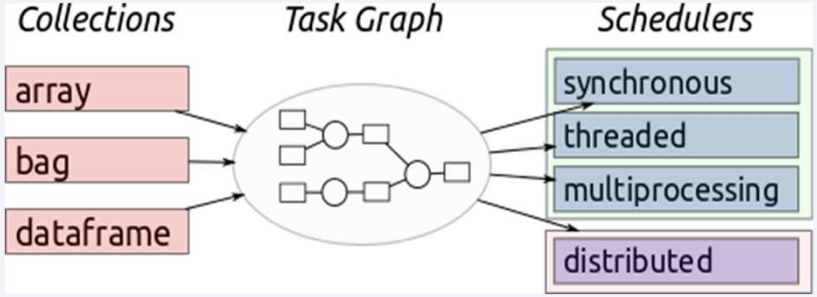
Qiskit

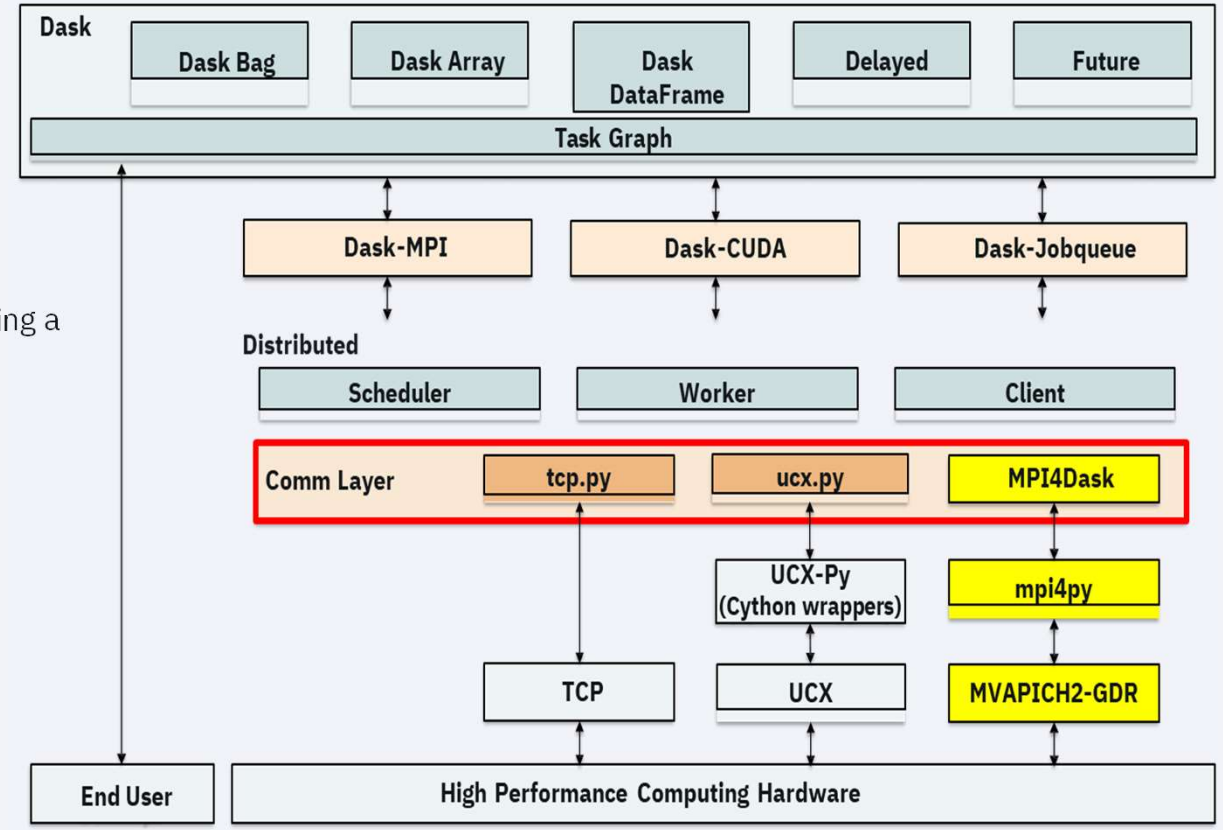


Solution Component : DASK

Qiskit

- Dask has three primary data structures:
  - Array (modeled after NumPy)
  - DataFrame (modeled after pandas)
  - Bag (modeled after lists)
- Uses delayed and futures to perform lazy evaluation while building a dependency graph of tasks
- The scheduler then executes the task graph—in sequence, multithreaded, multiprocessed, or distributed.





Solution Component : DASK Kubernetes

Qiskit

Dask Kubernetes Module provides cluster managers for Kubernetes.

HelmCluster is for managing an existing Dask cluster which has been deployed using Helm.

```
from dask_kubernetes import HelmCluster
cluster = HelmCluster(release_name="myrelease")
cluster.scale(10)
```

Typical Dask Kubernetes **(HelmCluster)** configuration for Qiskit Aer used in Project

```
#Configuration
import qiskit
from qiskit.providers.aer import AerSimulator
from dask.distributed import Client
from dask_kubernetes import HelmCluster
cluster = HelmCluster(release_name="my-dask")
exc = Client(cluster)
qbackend = AerSimulator()
qbackend.set_options(executor=exc)
qbackend.set_options(max_job_size=1)
```

KubeCluster deploys Dask clusters on Kubernetes clusters using native Kubernetes APIs. It is designed to dynamically launch ad-hoc deployments.

```
from dask_kubernetes import KubeCluster
cluster = KubeCluster.from_yaml('worker-template.yaml')
cluster.scale(20)  # add 20 workers
cluster.adapt()    # or create and destroy workers dynamically based on workload

from dask.distributed import Client
client = Client(cluster)
```

Qiskit

Environment Overview

Application Stack

Qiskit

Ubuntu 20.04 LTS

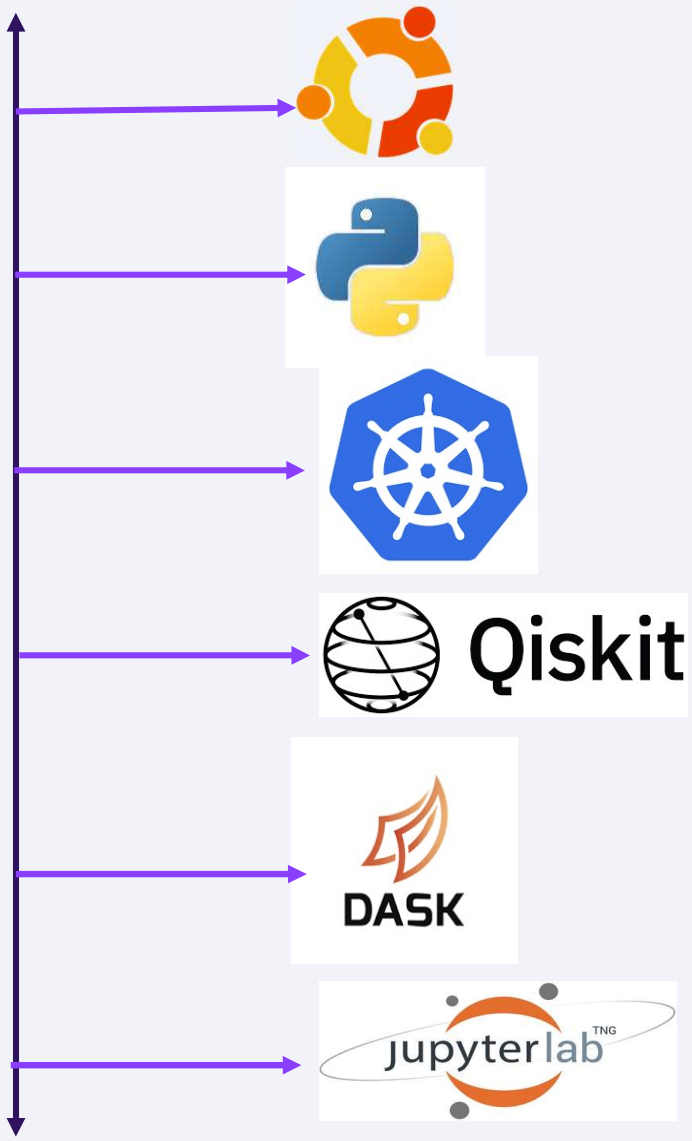
Python 3.8

Kubernetes v1.22.2

Qiskit v 0.30

Dask Kubernetes v2021.10.0

Jupyter Lab



Kubernetes Cluster Environment

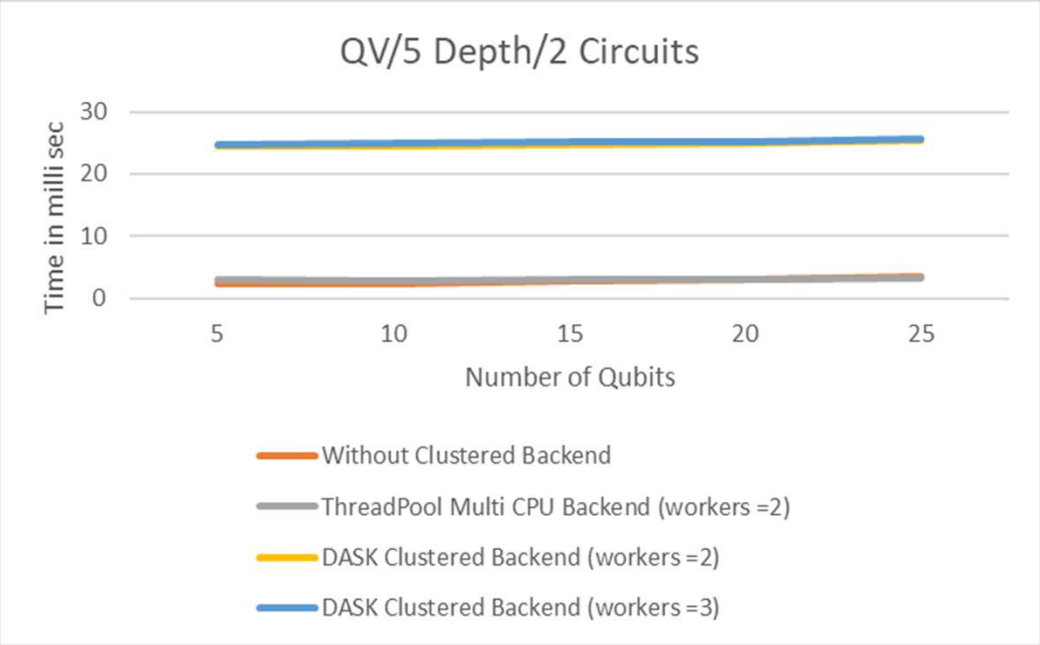
Qiskit

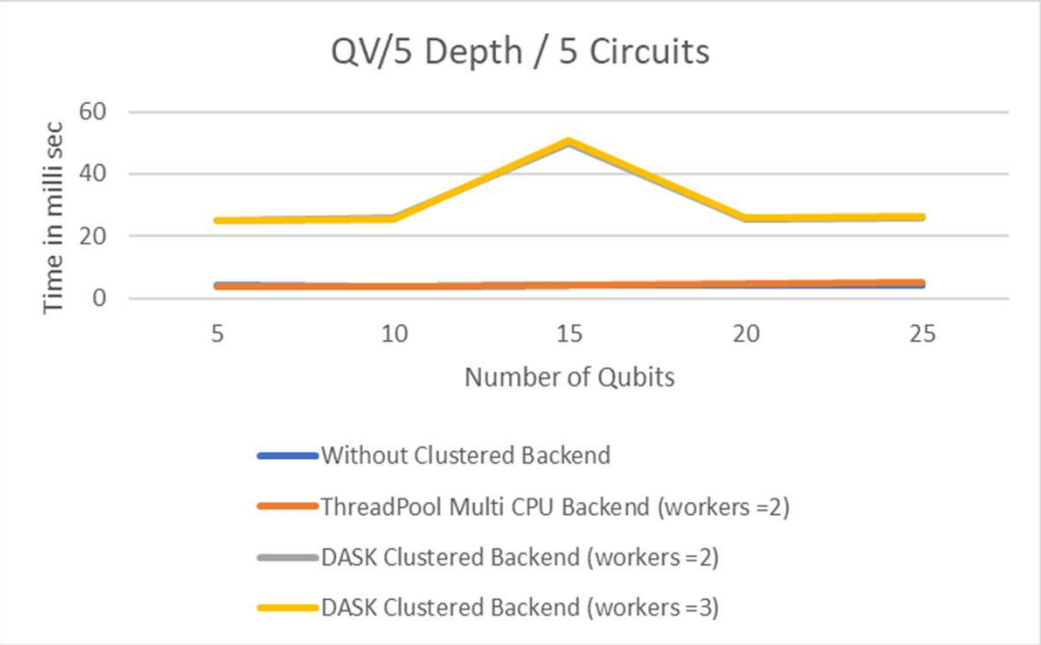
```
$ kubectl get nodes -A # One Master & 2 Worker Nodes
NAME                              STATUS   ROLES                  AGE     VERSION
ip-172-20-33-106.ec2.internal     Ready    node                   2d19h   v1.22.2
ip-172-20-51-193.ec2.internal     Ready    node                   2d19h   v1.22.2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

Qiskit

Test Results

Non-Clustered vs Clustered Backend





- ✓ Smaller Quantum Objects (with low depth & number of circuits) doesn't get performance benefit from clustered backend
- ✓ Network latency across DASK nodes on K8s, does contribute to the execution time, but at same time give access to scalable environment which is not possible with Single CPU or ThreadPool.
- ✓ Seamless scaling of DASK nodes on K8s environment, if autoscaling is enabled on basis of resource utilization.

Qiskit

Wrapping Up

Summarizing the Journey

Qiskit

Challenges

- ✓ To realize the capability of DASK cluster, scalable cloud platform is required.
- ✓ Huge Qiskit circuit list simulation performance can be restricted by the architecture and network latency of underlying hosting platform
- ✓ Development work still in progress to have effective Qiskit Aer simulator which can leverage clustered backend of **"DASK on K8s"**.

Achievements

- ✓ 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).

Road Ahead

- ✓ Deploy & Test the setup on multiple Cloud & On-Premise Platforms
- ✓ Explore & optimize more use case from quantum chemistry, materials science, quantum biology & other domains.
- ✓ Regularly update the tutorial.

Tutorial Link : https://github.com/iotaisolutions/qamp-fall-2021

References

- ✓ Kubernetes Documentation: https://kubernetes.io/
- ✓ DASK Documentation : https://docs.dask.org/
- ✓ DASK Kubernetes Documentation: https://kubernetes.dask.org/en/latest/
- ✓ QISKIT Aer Documentation : https://qiskit.org/documentation/stubs/qiskit.providers.aer.AerSimulator.html

  https://qiskit.org/documentation/apidoc/parallel.html
- ✓ GitHub for QISKIT Aer : https://github.com/Qiskit/qiskit-aer
- ✓ Github for DASK : https://github.com/dask/dask

Qiskit

Demo