Running Dask on AI-Ready Enterprise Platform on VMware vSphere 7 with VMware Tanzu Kubernetes Grid Service

An End-to-end Approach to the Setup of the Machine Learning Infrastructure

vmware

Table of Contents

Executive Summary

This solution provides general design and deployment guidelines for Running Dask on AI-Ready Enterprise Platform on VMware vSphere 7 with VMware Tanzu Kubernetes Grid Service. It is showcased in this paper running on Dell VxRail. The reference architecture applies to any compatible hardware platforms running Tanzu Kubernetes Grid.

Technology Overview

VMware vSphere with Tanzu

VMware vSAN

Dell VxRail

The only fully integrated, pre-configured, and pre-tested VMware hyperconverged integrated system optimized for VMware vSAN™ and VMware Cloud Foundation™, VxRail provides a simple, cost effective hyperconverged solution that solves a wide range of operational and environmental challenges and supports almost any use case, including tier-one applications, cloud native and mixed workloads. Powered by next generation Dell PowerEdge server platforms and VxRail HCI System Software, VxRail features next-generation technology to future proof your infrastructure and enables deep integration across the VMware ecosystem. The advanced VMware hybrid cloud integration and automation simplifies the deployment of secure VxRail cloud infrastructure.

Dask

Dask is a parallel computing library that scales the existing Python ecosystem and is open source. It is developed in coordination with other community projects like NumPy, pandas, and scikit-learn. Dask provides multi-core and distributed parallel execution on larger-than-memory datasets.

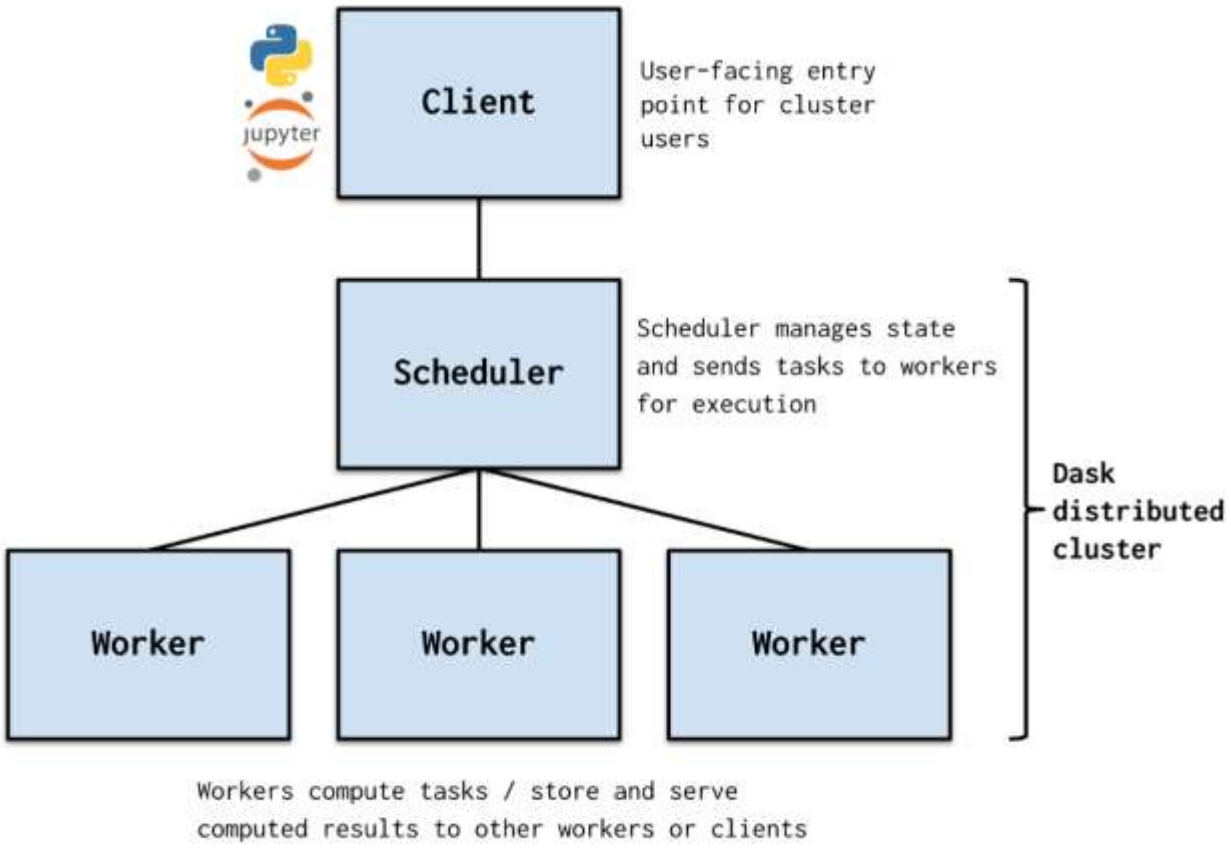

Figure 1 Dask Cluster Architecture

Solution Configuration

Hardware Resources

In this solution, we used four VxRail V670F servers. The following table shows the hardware components of each server.

Table 1 Hardware Resources

PROPERTY	SPECIFICATION
Server Model	Dell VxRail P670F
CPU	2 x Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz, 28 core each
RAM	512GB
Network Resources	1 x Intel(R) Ethernet Controller E810-XXV, 25Gbit/s, dual ports 1 x NVIDIA ConnectX-5 Ex, 100Gbit/s dual ports
Storage Resources	1 x Dell HBA355i disk controller 2 x P5600 1.6TB as vSAN Cache Devices 8 x 3.84TB Read Intensive SAS SSDs as vSAN Capacity Devices

GPU Resources	1 X NVIDIA Ampere A100 40GB PCIe

Software Resources

Table 2 Software Resources

Software	Version
vSphere	7.0 update 3c
Tanzu Kubernetes Release	v1.20.8+vmware.1
NVAIE	1.1
RAPIDS	21.10
Dask	2021.09.1
Pytorch	torch: 1.10.1+cu113, torchvision: 0.11.2+cu113, torchaudio: 0.10.1+cu113
Helm	3.7.2

Architecture Design

The Tanzu Kubernetes cluster provisioned on top of vSphere consists of ten worker nodes. Eight of them were equipped with vGPU for Dask Worker deployment. The other two worker nodes without vGPU were used for Dask Scheduler and Jupyter Notebook. NVIDIA GPU operator was installed in the Tanzu Kubernetes cluster to allow users to manage GPU nodes in the Tanzu Kubernetes cluster. Dask clusters were also deployed in the Tanzu Kubernetes cluster and configured with vSAN file service as persistent volume across all the Dask worker pods.

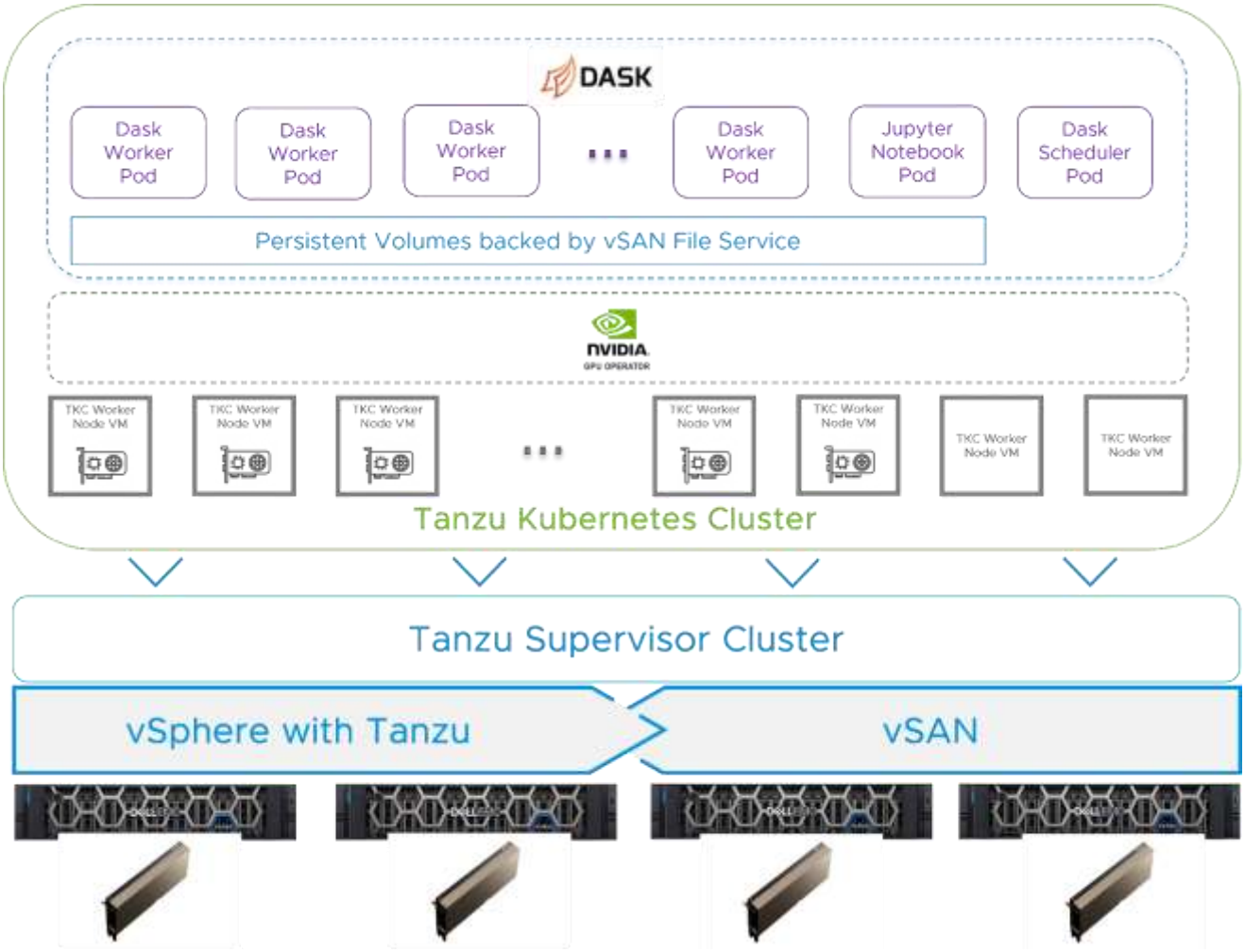

Figure 2 Solution Architecture

Network Design

The Intel 25GbE NICs were used for vSphere management, vMotion, vSAN, and Tanzu Kubernetes Grid management network, and the NVIDIA ConnectX-5 100GbE NICs were used for vSAN file service and Tanzu Kubernetes Grid workload network. In this case, the Dask cluster was physically separated from the management and vSAN network and would have higher network bandwidth for both nodes interactions and read or write data on the vSAN file share.

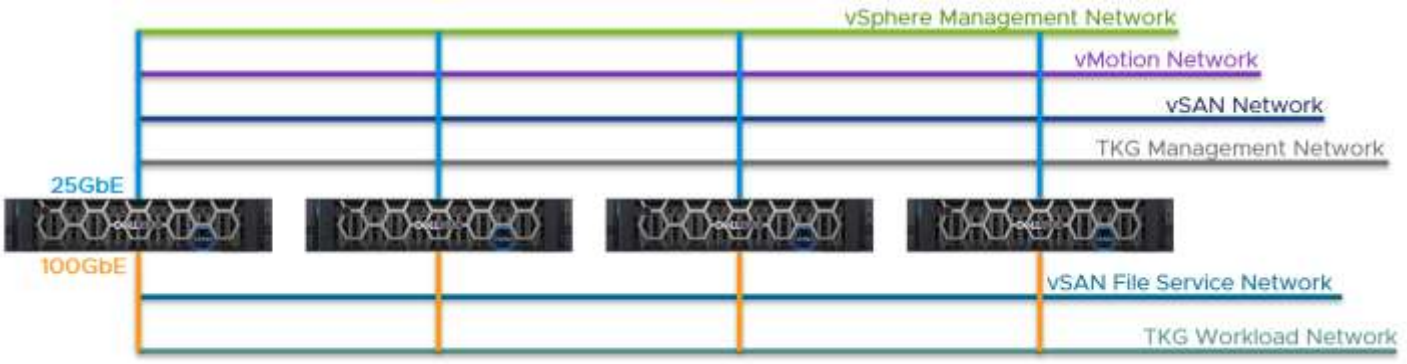

Figure 3 Networking Design

Solution Validation

Validation Tools

Monitoring Tools

Dask Dashboard

Dask provides a diagnostics dashboard where you can see your tasks as they are processed. To learn more about those graphs, look at *Diagnostics (distributed)*.

vSAN Performance Service

vSAN performance service is for monitoring the performance of the vSAN environment and helping users to investigate potential problems. The performance service collects and analyzes performance statistics and displays the data in a graphical format. You can use the performance charts to manage your workload and determine the root cause of problems.

Testing Tools

XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the *Gradient Boosting* framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems quickly and accurately.

RAPIDS

The RAPIDS suite of open-source software libraries and APIs allows you to execute end-to-end data science and analytics pipelines entirely on GPUs.

RAPIDS also focuses on common data preparation tasks for analytics and data science. This includes a familiar dataframe API that integrates with a variety of machine learning algorithms for end-to-end pipeline accelerations without paying typical serialization costs. RAPIDS also includes support for multi-node, multi-GPU deployments, enabling vastly accelerated processing and training on much larger dataset sizes.

Pytorch

Pytorch is a popular open-source machine learning framework that accelerates the path from research prototyping to production deployment.

vSphere Administrator Personnel

in this solution, a vSphere cluster should be pre-configured with vSAN enabled, and the ESXi hosts inside should have NVIDIA GPUs equipped.

vSphere administrator is responsible for preparing the environment for the following configuration aspects:

Enable vGPU on ESXi hosts

vSphere administrator can follow *this article* to install NVIDIA Virtual GPU manager from NVAIE 1.1 package and enable vGPU on ESXi hosts.

Configure Tanzu Kubernetes Grid Service with vGPU access

To configure Tanzu Kubernetes Grid service with vGPU access, the vSphere administrator should **create the Supervisor Cluster** and **Content Library** that subscribed to *https://wp-content.vmware.com/v2/latest/lib.json*, **create the VM Classes with vGPU access,** and **create the Namespace with the VM Classes configured with vGPU**.

Please visit *here* for detailed procedures and steps.

In this solution, we configured the Tanzu Supervisor Cluster with *haproxy v0.2.0*, added the pre-defined **best-effort-medium**, **best-effort-large** and **best-effort-2xlarge** VM Classes to the Namespace, and created and added a VM Class(16cpu-64gram-ts-20c-vmxnet3) with the following specifications to the Namespace.

- 16 vCPU
- 64GB RAM
- 1 NVIDIA A100 vGPU with 20c in time sharing mode. (Rapidsai images doesn't support Multi-Instances GPU when we were conducting the test *https://github.com/rapidsai/dask-cuda/issues/583*)

From the storage perspective, we added the two vSAN storage policies to the Namespace; one is **vsan-r5** that is with RAID-5 configured to reduce the storage consumption while maintaining good performance, another is **vsan-r1-sw8** which is configured with RAID-1 and StripeWidth=8 to maximize the performance for the Tanzu Kubernetes cluster worker nodes.

Configure vSAN File Service for NFS

Most machine learning platforms need a data lake, a centralized repository to store all the structured and unstructured data. With vSAN file service, the Tanzu Kubernetes cluster can be configured with NFS backed ReadWriteMany persistent volume across the pods to store the data. In this solution, the size of NFS file share provided by vSAN file service is 2TB, and the storage policy is configured with RAID1 with StripeWidth=8 to guarantee the performance by distributing the data across all the vSAN disk groups while not compromising the resiliency. For more information regarding vSAN file service, please visit *here*.

Developer Personnel

Once the Namespace is prepared and configured with the developer account for access, the developer should use a client to access the Namespace. The client should have **kubectl** and **helm** installed. We also mounted the NFS file share created earlier to this client to simplify the data preparation.

Data Preparation

In this solution, we mounted the file share to the client machine with path /ml-share to prepare data.

Here we demonstrated two Machine learning scenarios:

Run Pytorch with Dask for image classification inference

For this scenario, we downloaded the images from *Stanford Dogs Dataset*, and unzipped the folder to the file share from the client machine; the path is /ml-share/dogs/Images. Also, we downloaded the *imagenet 1000 classes human readable file* to /ml-share/dogs/imagenet1000.txt.

Train XGBoost with RAPIDS with Dask

For this scenario, we downloaded three years (2017, 2018, and 2019) yellow taxi trip data by using the following command to /ml-share/taxi-csv.

```
for yr in `seq 2017 2019`; do for mth in 01 02 03 04 05 06 07 08 09 10 11 12; do nohup wget https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_$yr-$mth.csv > ${yr}_${mth} 2>&1 & done; done
```

Provision Tanzu Kubernetes Cluster

While provision the Tanzu Kubernetes Cluster for Dask, we defined the control planes and worker nodes as following.

Table 3 Tanzu Kubernetes Cluster Definition

Role	Replica	Storage Class	VM Class	Tanzu Kubernetes Release (TKR)
Control Plane	3	vsan-r5	best-effort-medium	v1.20.8---vmware.1-tkg.2
Dask GPU Worker Nodes	8	vsan-r1-sw8	16cpu-64gram-ts-20c-vmxnet3	v1.20.8---vmware.1-tkg.2
Dask Non-GPU Worker Nodes	2	vsan-r5	best-effort-xlarge	v1.20.8---vmware.1-tkg.2

Table 4

Additionally, for each of the worker nodes, we configured 100GB volume for containers and 50GB volume for the kubelet.

Figure 4 Tanzu Kubernetes Cluster is Running

Install GPU Operator

After the Tanzu Kubernetes cluster is up and running, developers should login into the Tanzu Kubernetes cluster that was just created and follow this *link* to install NVIDIA GPU operator on the Tanzu Kubernetes cluster. The installation process needs the developer to provide the NVIDIA CLS or DLS license token and the NGC account information.

In this solution, we installed GPU operator v1.9.1.

Configure Persistent Volume Claims

We configured two Persistent Volume Claims on the NFS file share, one is for sharing the dataset across pods(mounted to /ml-share) and another is for the PIP installation cache(mounted to /root/.cache/pip) to save the time of re-downloading the additional PIP packages while any of the pods being rebuilt.

Install Dask Cluster

We used *Rapidsai Helm chart* to install Dask. The table below demonstrates the resource definition of each role.

Table 5 Dask Cluster Definition

Role	Replica	CPU	Memory	GPU per Pod	Service Type
Dask Scheduler	1	3	8GB	N/A	Load Balancer
Dask Worker	8	6	48GB	1	N/A
Jupyter Notebook	1	3	8GB	N/A	Load Balancer

We used the rapidsai container image(21.10-cuda11.2-runtime-ubuntu20.04) from *NVIDIA NGC*, and mounted those two Persistent Volume Claims created above to **Dask Worker** pods and **Jupyter Notebook** pod. The number of threads per Dask worker is adjustable. By default, number is the number of cores assigned to the pod.

We set the nodeSelector of **Dask Scheduler** and **Jupyter Notebook** to assign them those two non-GPU worker nodes.

The installation script can be found *here*.

```
root@photon-HCIBench [ ~/rapid/rapidsai ]# kubectl get all -n rapidsai
NAME                                                READY   STATUS    RESTARTS   AGE
pod/nfs-subdir-external-provisioner-779889496c-7x521   1/1     Running   0          33m
pod/rapidsai-jupyter-6584cfcc9b-rrhmr               1/1     Running   0          4ml4s
pod/rapidsai-scheduler-c7f4cc845-jbr71              1/1     Running   0          4ml4s
pod/rapidsai-worker-55d5b9d5bd-58hs7                1/1     Running   0          4ml4s
pod/rapidsai-worker-55d5b9d5bd-9vg45                1/1     Running   0          4ml4s
pod/rapidsai-worker-55d5b9d5bd-h2gnx                1/1     Running   0          4ml4s
pod/rapidsai-worker-55d5b9d5bd-jk9pl                1/1     Running   0          4ml4s
pod/rapidsai-worker-55d5b9d5bd-kw7gd                1/1     Running   0          4ml4s
pod/rapidsai-worker-55d5b9d5bd-mlstp                1/1     Running   0          4ml4s
pod/rapidsai-worker-55d5b9d5bd-tn48v                1/1     Running   0          4ml4s
pod/rapidsai-worker-55d5b9d5bd-vm9qj                1/1     Running   0          4ml4s

NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)                         AGE
service/rapidsai-jupyter    LoadBalancer   10.107.112.7     192.168.105.38   80:30246/TCP                    4ml4s
service/rapidsai-scheduler  LoadBalancer   10.105.118.200   192.168.105.39   8786:30247/TCP,8787:30014/TCP   4ml4s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nfs-subdir-external-provisioner   1/1     1            1           33m
deployment.apps/rapidsai-jupyter                  1/1     1            1           4ml4s
deployment.apps/rapidsai-scheduler                1/1     1            1           4ml4s
deployment.apps/rapidsai-worker                   8/8     8            8           4ml4s

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/nfs-subdir-external-provisioner-779889496c   1         1         1       33m
replicaset.apps/rapidsai-jupyter-6584cfcc9b                  1         1         1       4ml4s
replicaset.apps/rapidsai-scheduler-c7f4cc845                 1         1         1       4ml4s
replicaset.apps/rapidsai-worker-55d5b9d5bd                   8         8         8       4ml4s
```

Figure 5 Dask is Running

After the Dask cluster is up and running, from the client, inside the /ml-share directory, there are two directories created for the Persistent Volume claims, one is for the PIP installation cache, and another is for the data-set placement. We moved both *Stanford Dogs Dataset* and yellow taxi trip data into a directory for placing the data-set. In this case, all the Dask Worker nodes and Jupyter Notebook can access these two directories(**dogs** and **taxi-csv**) from /ml-share.

```
root@photon-HCIBench [ ~ ]# ls /ml-share/*
/ml-share/rapidsai-external-nfs-pip-cache-pvc-pvc-958764d2-7ce3-4445-90b4-ba15d4fa3126:
http  selfcheck

/ml-share/rapidsai-external-nfs-pvc-pvc-d345a4fa-a55d-4692-bb4a-ae4c273e31ab:
dogs  taxi-csv
```

Figure 6 Persistent Volume Claims

Then you can open a browser and go to *http://rapidsai-Jupyter-EXTERNAL-IP* (192.168.105.38 in our case) to access Jupyter Notebook, by default, the password is **rapidsai**. The installation contains the dashboard to show the status and metrics of Dask cluster, open browser and go to *http://Dask-Scheduler_External-IP:8787*, in our case, it should be ***http://192.168.105.39:8787***, the dashboard allows user to monitor the resource

utilization, task progress and performance, and nodes' information and logs for both Dask Worker nodes and scheduler.

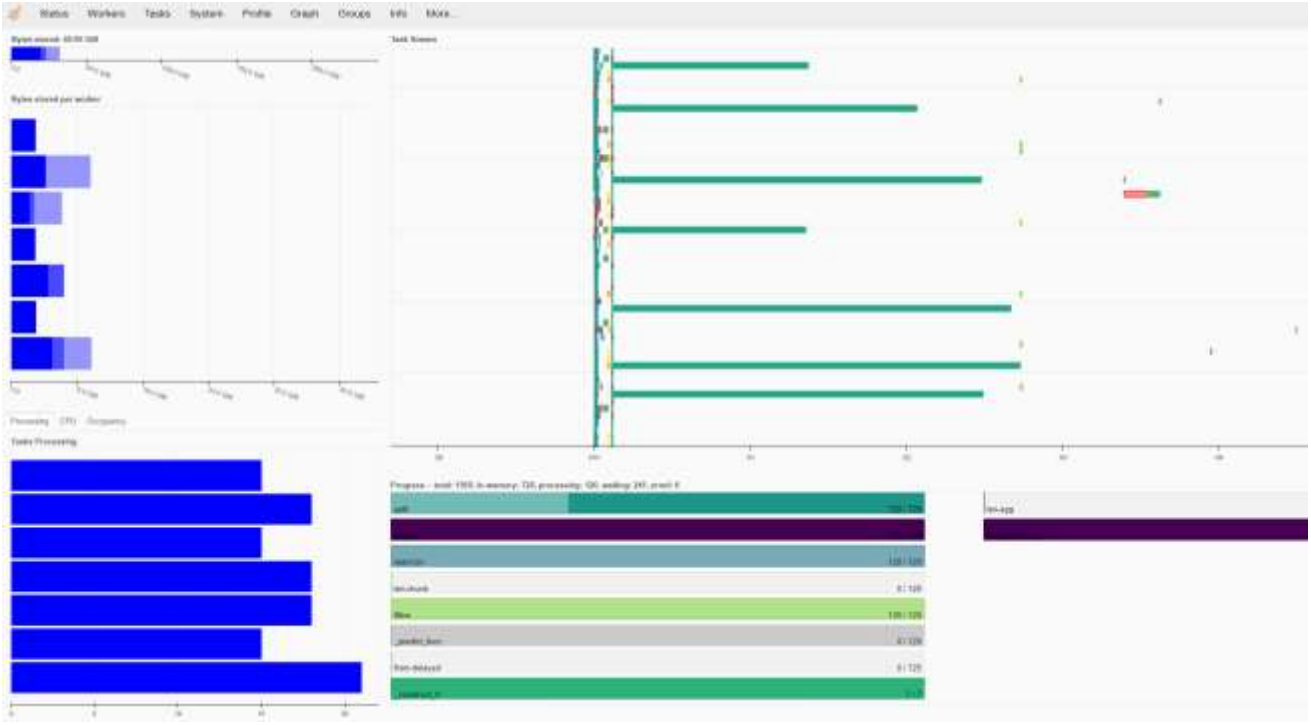

Figure 7 Dask Dashboard

The Jupyter Notebook is also configured with Dask plug-in, click on the Dask icon on the left, and put the same *http://Dask-Scheduler_External-IP:8787* to get the metrics.

Run Machine learning Workload with Dask

Run Pytorch with Dask for Image Classification Inference

By referring to this *blog*, we ran image classification inference with *resnet50* model of Pytorch with Dask on different numbers of Dask Workers by using vGPU or vCPU(by setting device to torch.device("cuda") for vGPU and torch.device("cpu") for vCPU). In total, there are 20580 photos to be classified. These photos are placed in the shared Persistent Volume Claim and can be accessed by all the Dask Worker nodes and Jupyter Notebook. In our testing, resnet50 can correctly classify 13802 photos, and the correct ratio is around 67%. Since we were using a pre-trained model resnet50 and did not do data cleansing, the accuracy is not overwhelming. However, as an example, we can see the significant performance improvement by Tanzu Kubernetes cluster with vGPU configured and the scalability from VxRail infrastructure all the way up to Dask platform deployed Tanzu Kubernetes cluster.

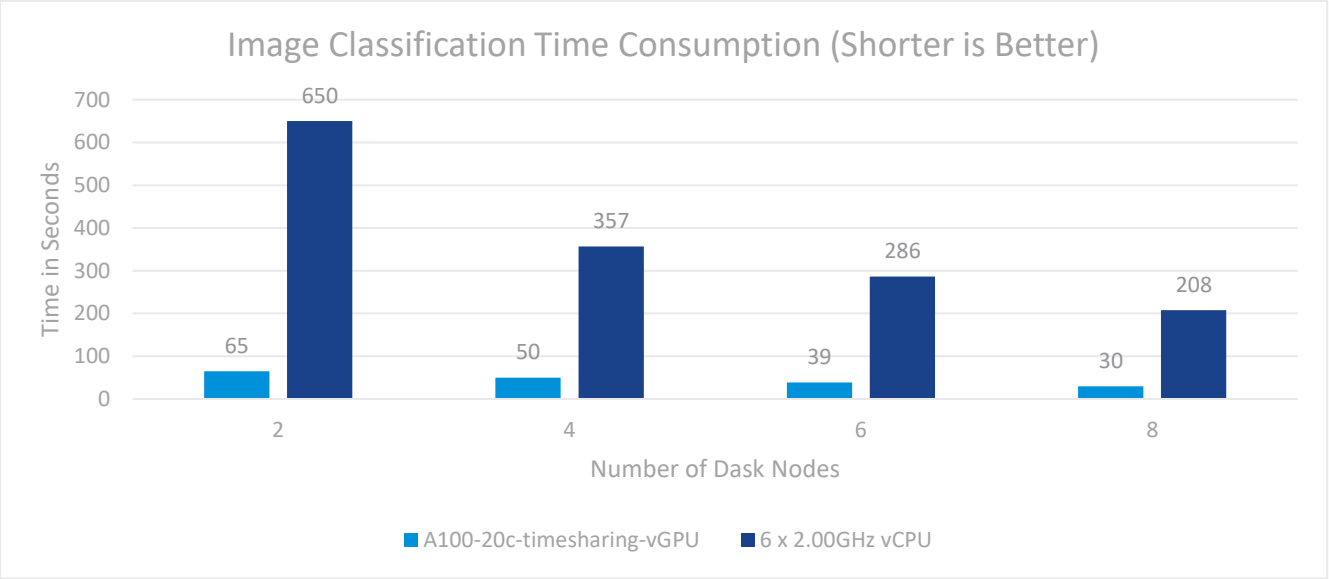

Figure 8 Image Classification Performance

We configured the Dask Worker node with six vCPU and one vGPU, the ratio of 20GB vGPU to 2.00GHz vCPU is 1:6. We used the time consumption as the performance metrics, in general, six 2.00GHz vCPU would spend 7x to 10x of time than one vGPU with 20GB memory buffer.

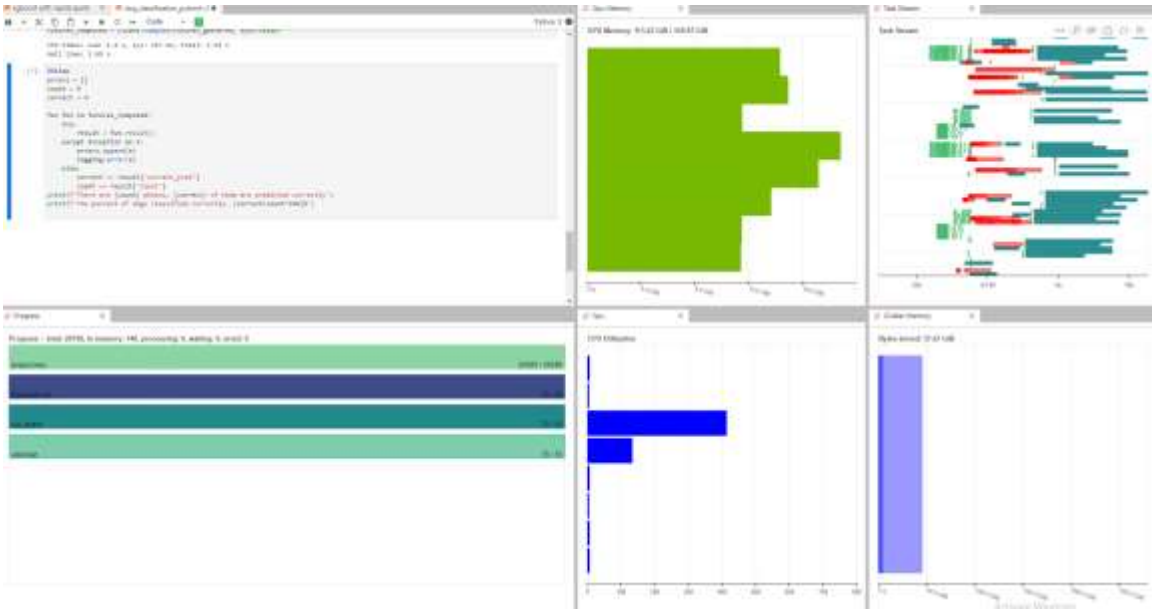

Figure 9 Run Pytorch with vGPU

While running the testing, by looking at the Dask-plugin from Jupyter Notebook, we saw both scenarios (using vGPU or vCPU) could balance the workload distribution to all the vCPUs or vGPUs across all the Dask worker nodes. For the tests handled by vGPU, we saw all the vGPU were using 80%+ memory buffer; and the tests handled by vCPU were nearly saturating all six vCPU for all the Dask Worker nodes.

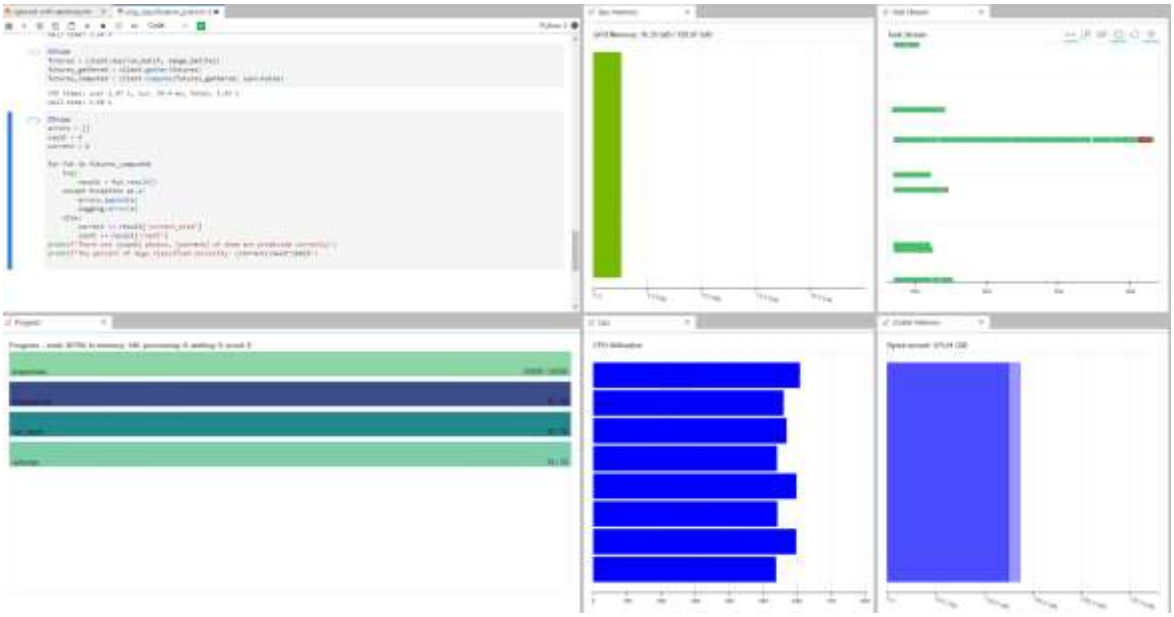

Figure 10 Run Pytorch with vCPU

Train XGBoost Regressor with RAPIDS and Dask

We also ran a typical Machine learning case: predict the New York taxi fare using XGBoost regressor. The data-set that we used includes all three years (2017-2019) New York taxi trip data. The entire process of this example includes:

1. Data cleansing: drop the rows with invalid data convert the data format if needed
2. Data loading: load the data to the memory
3. Data Splitting: split the data to train data set and test data set, the ratio is 75%:25%
4. Model Training: train the XGBoost regressor model
5. Inference: get the taxi fare predictions using the test data set
6. Error Analysis: compare the taxi fare prediction data and the taxi fare in test data set

Using the same deployment, Jupyter Notebook didn't have vGPU assigned thus we had to use *dask.dataframe* library to read the csv files, and the vCPU also handled; the data loading and split process on the Jupyter Notebook, only the XGBoost model training, and inference process were handled by vGPUs on the Dask Worker nodes. In this case, Jupyter Notebook spent 85 seconds on data loading, 13 seconds on data splitting, and 35 seconds on XGBoost Regressor model training.

However, some improvements could be made if the data loading and splitting process can be handled by vGPU. In this case, we tested another deployment scenario to assign one vGPU to the Jupyter Notebook pod and removed one Dask Worker node.

Table 6 Dask Cluster Definition

Role	Replica	CPU	Memory	GPU per Pod	Service Type
Dask Scheduler	1	3	8GB	N/A	Load Balancer
Dask Worker	8 -> 7	6	48GB	1	N/A
Jupyter Notebook	1	3	8GB	N/A -> 1	Load Balancer

In this case, we used *dask_cudf* library of *Rapids* to replace dask.dataframe which can help to move the data preparation process from vCPU to vGPU. In this test, Jupyter Notebook spent 19 seconds on data loading, 9 seconds on data splitting, and 25 seconds on XGBoost Regressor model training. Even there were just seven Dask Worker nodes; the Model Training performance was not compromised. Giving one vGPU to Jupyter Notebook and using cudf library for the XGBoost model training shortens the entire training process by more than half!

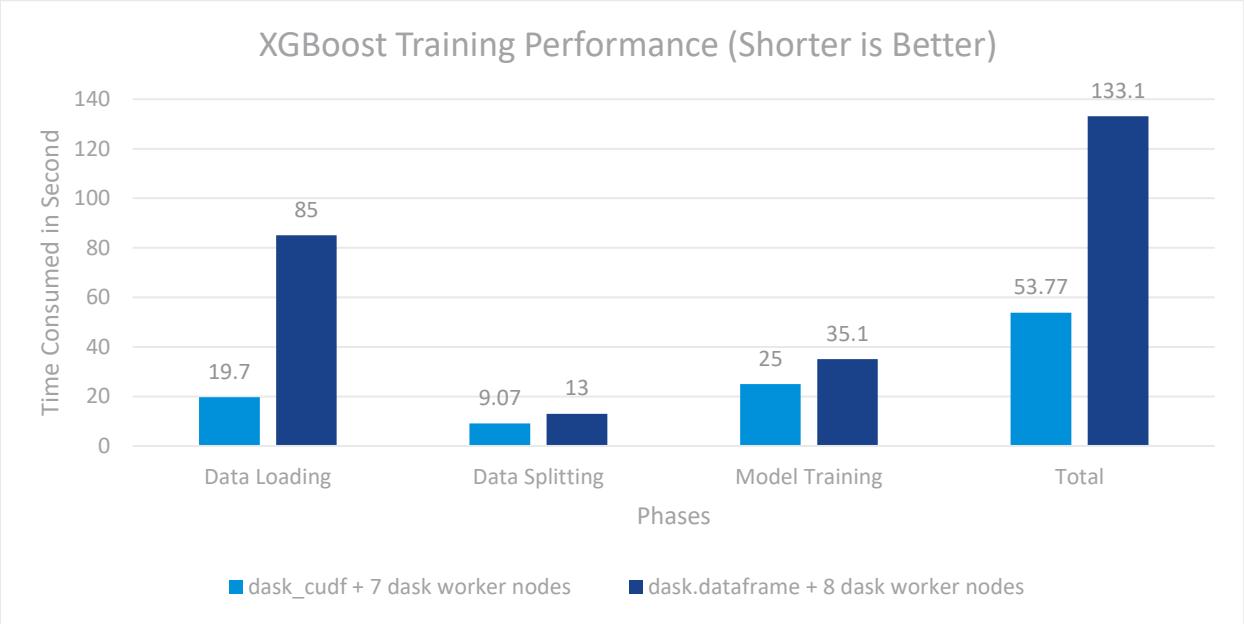

Figure 11 Train XGBoost Performance

From the resource utilization perspective, Dask distributed the workload pretty evenly across the cluster, and in total, the vGPU usage was about 73%, while the vCPU was barely used during the training process.

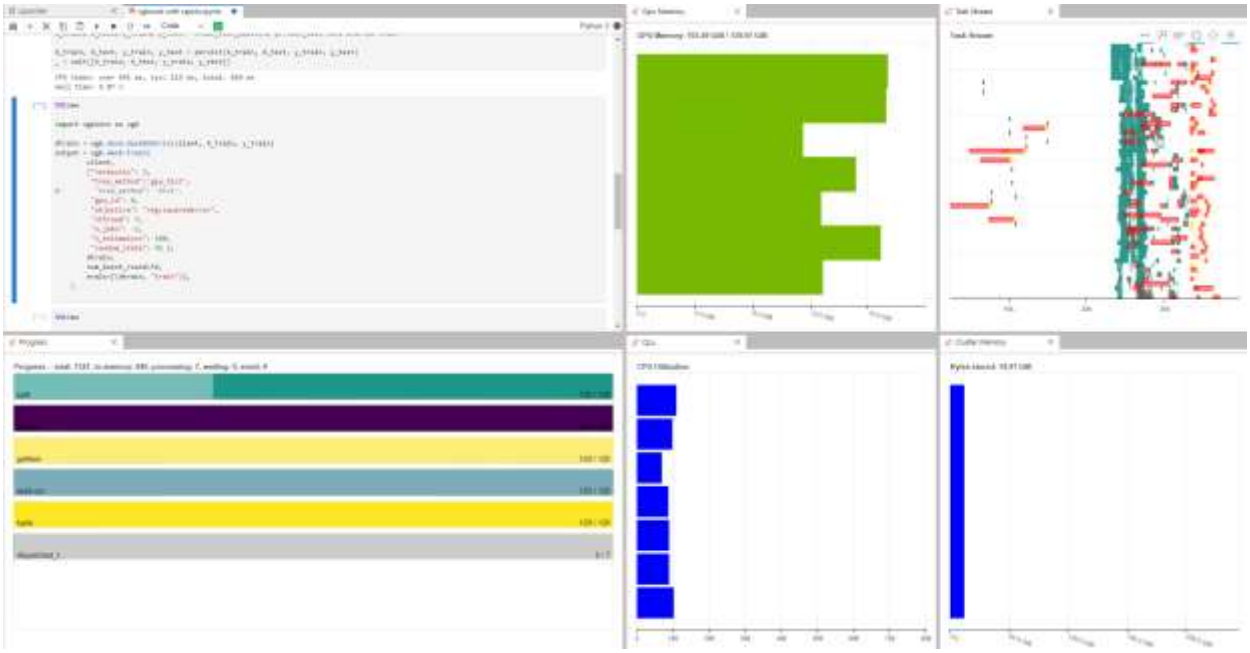

Figure 12 Resource Utilization

Best Practices

The aspects below are recommended while deploying Dask cluster on Tanzu Kubernetes cluster.

Start with small size

For the first deployment, try a smaller size of Tanzu Kubernetes cluster with a smaller Dask cluster. Not all the workload or learning needs that many GPU or other compute resources. Then use a small portion of the source data to run the Machine learning workload. If you are satisfied with the performance, increase the source data gradually until you feel the speed is too slow, then increase the Tanzu Kubernetes cluster and Dask cluster size.

Monitor the resource

A vSphere administrator can use vSphere and vSAN performance services to monitor the resources utilization from an infrastructure perspective. For example, using vSAN performance service can help to identify if there's any bottleneck on the Tanzu Kubernetes cluster volumes or file share.

A developer or data scientist can monitor the resource utilization using Dask dashboard can help to find out if there's any resource including CPU, memory, GPU, storage, and network is saturated or if any pod became the hotspot to identify the performance bottleneck.

Use vSAN for ReadWriteMany Persistent Volume

Using vSAN file service for ReadWriteMany Persistent Volume can easily scale out the file share, and the security, failure tolerance, performance, and capacity saving features can also be easily balanced by manipulating the storage policy of the file share.

Use the Image from NGC

NVIDIA has picked the latest and most stable Rapidsai image from *docker hub*, so you dont need to struggle with which build to use. When installing the Dask Helm chart, set the Dask Scheduler, Worker, and Jupyter's image repo to nvcr.io/nvidia/rapidsai/rapidsai and the image tag to 21.10-cuda11.2-runtime-ubuntu20.04.

Install Additional packages

If additional packages need to be used, like Pytorch, you need to install them on the Dask worker nodes. In this case, while installing the Dask Helm chart, set the dask.worker.env[0].name=EXTRA_PIP_PACKAGES and dask.worker.env[0].value="--upgrade torch==1.10.1+cu113 torchvision==0.11.2+cu113 torchaudio==0.10.1+cu113 -f *https://download.pytorch.org/whl/cu113/torch_stable.html --trusted-host download.pytorch.org*" to install Pytorch packages on Dask workers after Dask cluster is deployed. The process can be monitored by using "kubectl logs" to check each individual Dask worker installation process.

Also, we recommend creating a separate Persistent Volume Claim for the PIP cache directory for all the Dask worker nodes and mounting it to /root/.cache/pip within each node in case any of the Dask worker nodes needs to be recreated and the PIP wheels downloaded by other Dask worker nodes would be used to install the package directly without downloading it again.

Use Nodeselector for Dask Scheduler and Jupyter Notebook

We recommend using Nodeselector to place Dask Scheduler and Jupyter pods to the Tanzu Kubernetes cluster nodes without vGPU. In this case we can easily identify the root cause if there's any issue or bottleneck in the Dask cluster.

Plan ahead, if RAPIDS library will be used for Machine learning, then only Dask Scheduler can be allocated to a Tanzu Kubernetes cluster nodes without GPU, and all the other pods, including Jupyter Notebook should be allocated to Tanzu Kubernetes cluster nodes with GPU, in this case, we need to reduce the Dask Worker nodes by one.

NOTE: Kubernetes *doesn't allow pods to share GPU*, so we can't deploy the Jupyter Notebook and a Dask Worker within the same Tanzu Kubernetes cluster node with GPU, however, this issue is already being taken care by the Kubernetes community per *https://github.com/src-d/k8s-nvidia-gpu-overcommit/blob/overcommit/README.md*

Conclusion

Dask can parallelize and distribute the Machine learning workload by leveraging the CPU or GPU resources on multiple worker nodes, which can help to speed up the workload by adding more Dask Worker nodes.

Running on Dask cluster on Tanzu Kubernetes cluster with vGPU on VxRail simplifies the deployment, configuration, maintenance, and scaling-out for Machine learning. This end-to-end solution allows vSphere administrators to add VxRail servers into the vSphere cluster on demand to expand the vSphere cluster size with more GPU, then the developer can add more worker nodes with vGPU into the Tanzu Kubernetes cluster and scale out the Dask cluster with more Dask Workers accordingly.

The combination of Dask, Tanzu Kubernetes Grid service, and VxRail enables VMware's partners and customers to provision and manage the Machine learning platform on demand and to run Machine learning workload according to the business needs.

References

Pytorch: *https://pytorch.org/*

RAPIDS: *https://rapids.ai/index.html*

XGBoost: *https://xgboost.readthedocs.io/en/stable/*

Dask: *https://dask.org/*

About the Author

Chen Wei, Staff Solution Architect, VMware

The following reviewers also contributed to the paper contents:

Vic Dery, Sr. Principal Engineer of VxRail Technical Marketing in Dell Technologies

Jason Marques, Sr. Principal Engineer of VxRail Technical Marketing in Dell Technologies
Tony Foster, Sr. Principal Engineer of VxRail Technical Marketing in Dell Technologies


vmware

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com.