Kubeflow (Machine Learning workflows on Kubernetes)

Akshit Rameshkumar Desai
Student-Id:- 40233041
akshit.r.desai@gmail.com

Omijkumar Pravinbhai Mangukiya
Student-Id:- 40233479
om.mkp1812@yahoo.com

Ruchit Dobariya
Student-Id:- 40232238
ruchitdobariya307@gmail.com

Dharmil Mukeshbhai Vaghasiya
Student-Id:- 40230633
dharmilvaghasia3675@gmail.com

ABSTRACT

Kubeflow is an open-source project that aims to make it easy to deploy and manage machine learning (ML) workflows on Kubernetes, a popular system for managing containerized applications. With Kubeflow, users can build and deploy end to end ML systems(pipelines, jobs, etc.). It provides implementation for training operators such as TensorFlow, PyTorch, MXNet and MPI and can scale their training and inference workloads across multiple GPUs and machines. The distributed nature of Kubeflow allows users to take advantage of large amounts of compute resources to train complex and large-scale ML models. This can speed up the development and deployment of ML applications, and make it easier for organizations to use ML in their workflow.

1 INTRODUCTION

Using distributed system we will train a neural network to predict the missing citation given collection of existing citation links(ogbl-citations2 - 2.1 GB). For which we will deploy our training model on kubernetes(Google kubernetes engine API) using kubeflow. And we will analyze the result on tensorboard. Moreover, we will explore the feature of distributed system used in this project.

2 SYSTEM

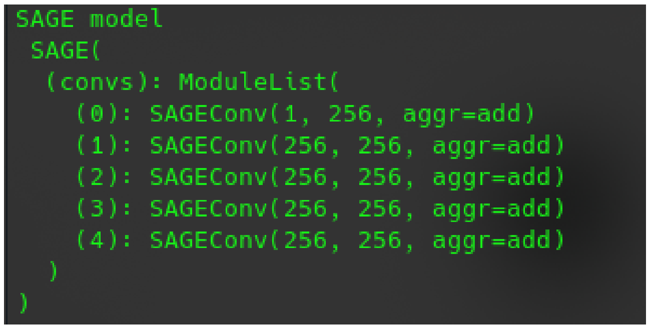
2.1 Dataset

- The **ogbl-citation2 dataset (2.1 GB)** is a directed graph, representing the citation network between a subset of papers extracted from MAG [6].
- Each node is a paper with **128-dimensional word2vec features** [5] that summarizes its title and abstract, and each directed edge indicates that one paper cites another.
- The dataset consists of **2,927,963 Nodes** and **30,561,187 edges**.

2.2 GNN model

GNN model consists of,

(1) The GraphSage GNN for generating node embeddings.
(2) A basic LinkPredictor that simply does a dot product between the two node embeddings followed by a sigmoid.
(3) This project is using the built-in SAGEConv [4] layer from PyG (torch_geometric), which allows us to quickly get our model up and running. Figure 1



```
SAGE model
 SAGE(
  (convs): ModuleList(
    (0): SAGEConv(1, 256, aggr=add)
    (1): SAGEConv(256, 256, aggr=add)
    (2): SAGEConv(256, 256, aggr=add)
    (3): SAGEConv(256, 256, aggr=add)
    (4): SAGEConv(256, 256, aggr=add)
  )
)
```

Figure 1: GNN model architracture

2.3 Pytorch distributed training

- DistributedDataParallel (DDP) implements data parallelism at the module level which can run across multiple machines.
- Applications using DDP should spawn multiple processes and create a single DDP instance per process.
- The DDP uses collective communications in the **"torch.distributed"** package to synchronize gradients and buffers.
- More specifically, DDP registers an autograd hook for each parameter given by model.parameters() and the hook will fire when the corresponding gradient is computed in the backward pass. Then DDP uses that signal to trigger gradient synchronization across processes.

2.4 Kubeflow

Kubeflow [1] uses certain deployments and jobs resources if kubernetes to achieve distributed ML/AI workflows such as,

- minio :- High Performance Object Storage to store all of its pipelines, artifacts and logs
- kubeflow-pipeline-deployer :- To deploy ML pipelines in the kubeflow.
- mysql - to store metadata about experiments

2.4.1 *Kubeflow Training Operators*.

- Based on different framework there are different Kubeflow Training Operators of which we have used **PyTorch Training (PyTorchJob)**.
- PyTorchJob is a Kubernetes custom resource to run PyTorch training jobs on Kubernetes. The Kubeflow implementation of PyTorchJob is in training-operator.

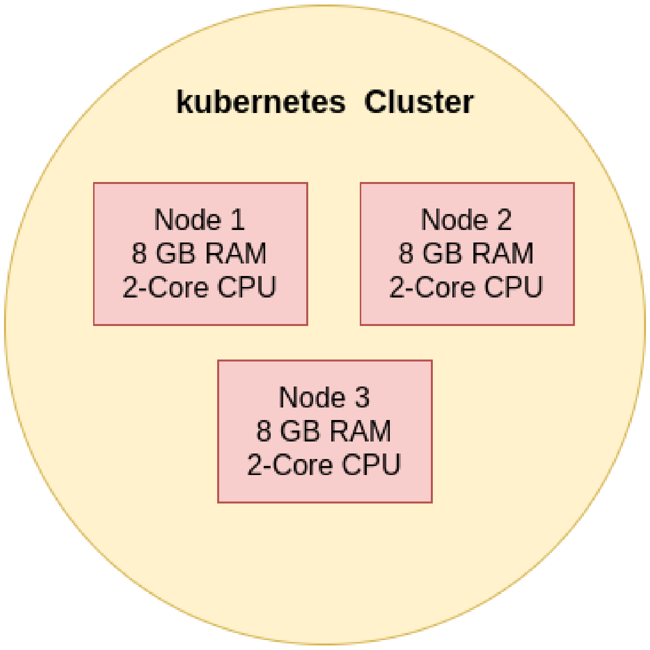


Figure 2: kubernetes cluster

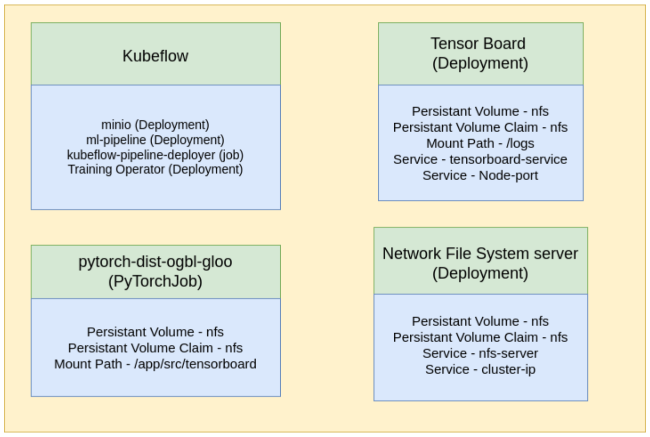


Figure 3: System architracture

2.5 Kubernetes

In this project we have used GCP(Google Cloud Provider). The kubernetes [2] cluster in GCP is provided by GKE(Google Kubernetes Engine) API. For this project we are using 3 nodes in kubernetes cluster each with 8 GB RAM and 2-core CPUs. Figure 2

- **Deployment** :- A Kubernetes deployment is a resource object in Kubernetes that provides declarative updates to applications. A Deployment provides declarative updates for Pods and ReplicaSets.
- **Job** :- To create batch transactions, Kubernetes provides the Job object. A Job object creates one or more Pods and attempts to retry the execution until a specified number of them terminate successfully.
- **Service**:- A Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). The set of Pods targeted by a Service is usually determined by a selector.
- **Persistent Volume**:- A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource.
- **Persistent Volume Claim**:- A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources.
- System is using above kubernetes resources as per Figure 3

2.5.1 ***gcloud builds***. The Google Cloud Build process involves running a series of build steps in Docker containers. It performs the series of steps on remote machine to build docker container and then push to artifactory repositories.

- Each build step can perform any task that can be accomplished from a container, regardless of the environment.
- We have created build scripts with various build steps to create custom Docker images.
- GNN model docker image: **gcr.io/dsd-demo-370521/pytorch-distributed-ogbl:1.0**
- TensorBoard docker image: **gcr.io/dsd-demo-370521/tensorboard:1.0**
- NFS server docker image: **gcr.io/google_containers/volume-nfs:latest**

2.5.2 ***Tensorboard***. It provides the visualization and tooling needed for machine learning experimentation. Here, we are Tracking and visualizing metrics such as loss and accuracy.

- We have used custom scaler for logging.
- To log a custom scalar, we have used tf.summary.scalar() with a file writer.
- The file writer is responsible for writing data for this run to the specified directory.
- The custom tensorboard deployment image uses python:slim docker base image, along with shared NFS volume mount.
- Also to access the dashboard from outside of the cluster a NodePort service is created and attached to tensorboard deployment.

3 DEMO SCENARIO

For the demo of this project we displayed below feature of distributed system.

3.1 Scalability & Performance

we ran system with different number of pods & epoch and tried to measure the time taken by it to train the model. We can make it more scalable by increasing number of pods by keeping hardware resources in mind.

From above table it can be concluded that the model should be trained with high number of epoch. Also it can be seen that time to train the model with small number workers doesn't differentiate the time taken for the training.

No. of workers	epoch	time taken	accuracy
2	10	12.18 min	73.82%
4	10	12.04 min	73.74%
6	10	11.17 min	74.28%
2	30	35.45 min	83.2%
4	30	33.93 min	85.04%
6	30	30.76 min	85.11%

Table 1: Performance and Accuracy

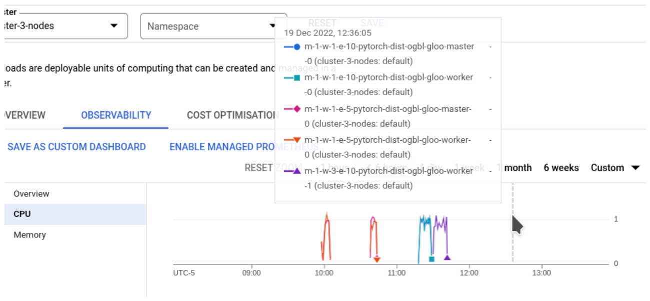

ter-3-nodes
Namespace
19 Dec 2022, 12:36:05
m-1-w-1-e-10-pytorch-dist-ogbl-gloo-master-0 (cluster-3-nodes: default)
m-1-w-1-e-10-pytorch-dist-ogbl-gloo-worker-0 (cluster-3-nodes: default)
m-1-w-1-e-5-pytorch-dist-ogbl-gloo-master-0 (cluster-3-nodes: default)
m-1-w-1-e-5-pytorch-dist-ogbl-gloo-worker-0 (cluster-3-nodes: default)
m-1-w-3-e-10-pytorch-dist-ogbl-gloo-worker-1 (cluster-3-nodes: default)
OVERVIEW
OBSERVABILITY
COST OPTIMISATION
SAVE AS CUSTOM DASHBOARD
ENABLE MANAGED PRO
RESET
month
6 weeks
Custom
Overview
CPU
Memory
1
0
UTC-5
09:00
10:00
11:00
12:00
13:00

Figure 4: CPU Usage

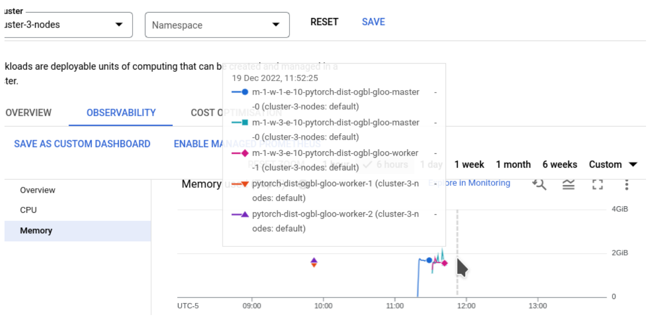

uster-3-nodes
Namespace
RESET
SAVE
19 Dec 2022, 11:52:25
m-1-w-1-e-10-pytorch-dist-ogbl-gloo-master-0 (cluster-3-nodes: default)
m-1-w-3-e-10-pytorch-dist-ogbl-gloo-master-0 (cluster-3-nodes: default)
m-1-w-3-e-10-pytorch-dist-ogbl-gloo-worker-1 (cluster-3-nodes: default)
pytorch-dist-ogbl-gloo-worker-1 (cluster-3-nodes: default)
pytorch-dist-ogbl-gloo-worker-2 (cluster-3-nodes: default)
OVERVIEW
OBSERVABILITY
SAVE AS CUSTOM DASHBOARD
1 week
1 month
6 weeks
Custom
Overview
CPU
Memory
4GiB
2GiB
0
UTC-5
09:00
10:00
11:00
12:00
13:00

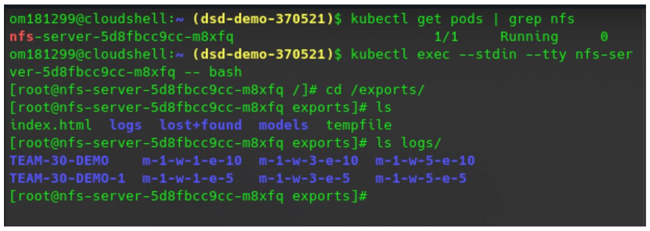
Figure 5: CPU Usage

3.2 Resource sharing

3.2.1 ***CPU***. Figure 4 displays CPU usage of each pod during experimental runs. (on average 1.16 core CPU)

3.2.2 ***Memory***. Figure 5 displays Memory usage of each pod during experimental runs. (on average 2-GB RAM))

3.2.3 ***File System***. To visualise the logs on the tensorboard it needs shared file system. In kuberenetes, two persistent volume claim can not be attached to same persistent volume. So, the data directory is shared with the help of NFS (Network File System). NFS [3] is deployed as a nfs-server deployment in our cluster. Which basically mounts a disk(provided by gke) and hosts the nfs-server on top of that. Each deployment that want to access this nfs shared volume needs to use a common persistent volume claim. And to access the tensorboard dashboard from outside of the cluster the nodeport service is used. Here, Node-port service binds the local port with container port 6006. By default GCP does not allows to expose any ports in cluster with firewall rules. Therefore to expose the nodeport service a new firewall rule is being used. Figure 6



```
om181299@cloudshell:~ (dsd-demo-370521)$ kubectl get pods | grep nfs
nfs-server-5d8fbcc9cc-m8xfq                                1/1     Running     0
om181299@cloudshell:~ (dsd-demo-370521)$ kubectl exec --stdin --tty nfs-ser
ver-5d8fbcc9cc-m8xfq -- bash
[root@nfs-server-5d8fbcc9cc-m8xfq /]# cd /exports/
[root@nfs-server-5d8fbcc9cc-m8xfq exports]# ls
index.html  logs  lost+found  models  tempfile
[root@nfs-server-5d8fbcc9cc-m8xfq exports]# ls logs/
TEAM-30-DEMO    m-1-w-1-e-10  m-1-w-3-e-10  m-1-w-5-e-10
TEAM-30-DEMO-1  m-1-w-1-e-5   m-1-w-3-e-5   m-1-w-5-e-5
[root@nfs-server-5d8fbcc9cc-m8xfq exports]#
```

Figure 6: Network file system server

3.3 Fault tolerance

By killing one of the worker pod while on-going training we can replicate the system failure. Fault tolerance behaviour of kubernetes restarts the failed pod.

4 CONCLUSION

Using kubeflow we can achieve (To solve ML/AI problems)

- **Scalability** :- training ML/AI models at scale
- **Performance** :- reduced time taken for training
- **Resource sharing** :- CPU, Memory and File System
- **Fault tolerance** :- Auto-restarts in case of failure

REFERENCES

[1] [n.d.]. Kubeflow Documentation. ([n. d.]). https://www.kubeflow.org/docs/started/
[2] [n.d.]. *Kubernetes Documentation*. https://kubernetes.io/docs/home/
[3] [n.d.]. *NFS server in GKE*. https://medium.com/platformer-blog/nfs-persistent-volumes-with-kubernetes-a-case-study-ce1ed6e2c266
[4] [n.d.]. *Torch Geometric SAGEConv*. https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/nn/conv/sage_conv.html
[5] Emre Guney. [n.d.]. The patent holder's dilemma: Buy, sell, or troll? *Reproducible drug repurposing: When similarity does not suffice. In Pacific Symposiumon Biocomputing* ([n. d.]). https://pubmed.ncbi.nlm.nih.gov/27896969/
[6] Chiyuan Huang Chieh-Han Wu Yuxiao Dong Kuansan Wang, Zhihong Shen and Anshul Kanakia. [n.d.]. Microsoft academic graph: When experts are not enough. Quantitative Science Studies. ([n. d.]). https://direct.mit.edu/qss/article/1/1/396/15572/Microsoft-Academic-Graph-When-experts-are-not