import argparse

import logging

import os

import pkgutil

import sys

from urllib.parse import urlsplit

import apache\_beam as beam

from apache\_beam.options.pipeline\_options import PipelineOptions

import faiss

def parse\_d6w\_config(argv=None):

"""Parse d6w config.

:param argv: d6w config

:return: dictionary containing d6w config

"""

parser = argparse.ArgumentParser(

description="See https://docbird.twitter.biz/d6w/model.html for any parameters inherited from d6w job config"

)

parser.add\_argument("--job\_name", dest="job\_name", required=True, help="d6w attribute")

parser.add\_argument("--project", dest="project", required=True, help="d6w attribute")

parser.add\_argument(

"--staging\_location", dest="staging\_location", required=True, help="d6w attribute"

)

parser.add\_argument("--temp\_location", dest="temp\_location", required=True, help="d6w attribute")

parser.add\_argument(

"--output\_location",

dest="output\_location",

required=True,

help="GCS bucket and path where resulting artifacts are uploaded",

)

parser.add\_argument(

"--service\_account\_email", dest="service\_account\_email", required=True, help="d6w attribute"

)

parser.add\_argument(

"--factory\_string",

dest="factory\_string",

required=False,

help="FAISS factory string describing index to build. See https://github.com/facebookresearch/faiss/wiki/The-index-factory",

)

parser.add\_argument(

"--metric",

dest="metric",

required=True,

help="Metric used to compute distance between embeddings. Valid values are 'l2', 'ip', 'l1', 'linf'",

)

parser.add\_argument(

"--use\_gpu",

dest="gpu",

required=True,

help="--use\_gpu=yes if you want to use GPU during index building",

)

known\_args, unknown\_args = parser.parse\_known\_args(argv)

d6w\_config = vars(known\_args)

d6w\_config["gpu"] = d6w\_config["gpu"].lower() == "yes"

d6w\_config["metric"] = parse\_metric(d6w\_config)

"""

WARNING: Currently, d6w (a Twitter tool used to deploy Dataflow jobs to GCP) and

PipelineOptions.for\_dataflow\_runner (a helper method in twitter.ml.common.apache\_beam) do not

play nicely together. The helper method will overwrite some of the config specified in the d6w

file using the defaults in https://sourcegraph.twitter.biz/git.twitter.biz/source/-/blob/src/python/twitter/ml/common/apache\_beam/\_\_init\_\_.py?L24.'

However, the d6w output message will still report that the config specified in the d6w file was used.

"""

logging.warning(

f"The following d6w config parameters will be overwritten by the defaults in "

f"https://sourcegraph.twitter.biz/git.twitter.biz/source/-/blob/src/python/twitter/ml/common/apache\_beam/\_\_init\_\_.py?L24\n"

f"{str(unknown\_args)}"

)

return d6w\_config

def get\_bq\_query():

"""

Query is expected to return rows with unique entityId

"""

return pkgutil.get\_data(\_\_name\_\_, "bq.sql").decode("utf-8")

def parse\_metric(config):

metric\_str = config["metric"].lower()

if metric\_str == "l2":

return faiss.METRIC\_L2

elif metric\_str == "ip":

return faiss.METRIC\_INNER\_PRODUCT

elif metric\_str == "l1":

return faiss.METRIC\_L1

elif metric\_str == "linf":

return faiss.METRIC\_Linf

else:

raise Exception(f"Unknown metric: {metric\_str}")

def run\_pipeline(argv=[]):

config = parse\_d6w\_config(argv)

argv\_with\_extras = argv

if config["gpu"]:

argv\_with\_extras.extend(["--experiments", "use\_runner\_v2"])

argv\_with\_extras.extend(

["--experiments", "worker\_accelerator=type:nvidia-tesla-t4;count:1;install-nvidia-driver"]

)

argv\_with\_extras.extend(

[

"--worker\_harness\_container\_image",

"gcr.io/twttr-recos-ml-prod/dataflow-gpu/beam2\_39\_0\_py3\_7",

]

)

options = PipelineOptions(argv\_with\_extras)

output\_bucket\_name = urlsplit(config["output\_location"]).netloc

with beam.Pipeline(options=options) as p:

input\_data = p | "Read from BigQuery" >> beam.io.ReadFromBigQuery(

method=beam.io.ReadFromBigQuery.Method.DIRECT\_READ,

query=get\_bq\_query(),

use\_standard\_sql=True,

)

index\_built = input\_data | "Build and upload index" >> beam.CombineGlobally(

MergeAndBuildIndex(

output\_bucket\_name,

config["output\_location"],

config["factory\_string"],

config["metric"],

config["gpu"],

)

)

# Make linter happy

index\_built

class MergeAndBuildIndex(beam.CombineFn):

def \_\_init\_\_(self, bucket\_name, gcs\_output\_path, factory\_string, metric, gpu):

self.bucket\_name = bucket\_name

self.gcs\_output\_path = gcs\_output\_path

self.factory\_string = factory\_string

self.metric = metric

self.gpu = gpu

def create\_accumulator(self):

return []

def add\_input(self, accumulator, element):

accumulator.append(element)

return accumulator

def merge\_accumulators(self, accumulators):

merged = []

for accum in accumulators:

merged.extend(accum)

return merged

def extract\_output(self, rows):

# Reimports are needed on workers

import glob

import subprocess

import faiss

from google.cloud import storage

import numpy as np

client = storage.Client()

bucket = client.get\_bucket(self.bucket\_name)

logging.info("Building FAISS index")

logging.info(f"There are {len(rows)} rows")

ids = np.array([x["entityId"] for x in rows]).astype("long")

embeds = np.array([x["embedding"] for x in rows]).astype("float32")

dimensions = len(embeds[0])

N = ids.shape[0]

logging.info(f"There are {dimensions} dimensions")

if self.factory\_string is None:

M = 48

divideable\_dimensions = (dimensions // M) \* M

if divideable\_dimensions != dimensions:

opq\_prefix = f"OPQ{M}\_{divideable\_dimensions}"

else:

opq\_prefix = f"OPQ{M}"

clusters = N // 20

self.factory\_string = f"{opq\_prefix},IVF{clusters},PQ{M}"

logging.info(f"Factory string is {self.factory\_string}, metric={self.metric}")

if self.gpu:

logging.info("Using GPU")

res = faiss.StandardGpuResources()

cpu\_index = faiss.index\_factory(dimensions, self.factory\_string, self.metric)

cpu\_index = faiss.IndexIDMap(cpu\_index)

gpu\_index = faiss.index\_cpu\_to\_gpu(res, 0, cpu\_index)

gpu\_index.train(embeds)

gpu\_index.add\_with\_ids(embeds, ids)

cpu\_index = faiss.index\_gpu\_to\_cpu(gpu\_index)

else:

logging.info("Using CPU")

cpu\_index = faiss.index\_factory(dimensions, self.factory\_string, self.metric)

cpu\_index = faiss.IndexIDMap(cpu\_index)

cpu\_index.train(embeds)

cpu\_index.add\_with\_ids(embeds, ids)

logging.info("Built faiss index")

local\_path = "/indices"

logging.info(f"Writing indices to local {local\_path}")

subprocess.run(f"mkdir -p {local\_path}".strip().split())

local\_index\_path = os.path.join(local\_path, "result.index")

faiss.write\_index(cpu\_index, local\_index\_path)

logging.info(f"Done writing indices to local {local\_path}")

logging.info(f"Uploading to GCS with path {self.gcs\_output\_path}")

assert os.path.isdir(local\_path)

for local\_file in glob.glob(local\_path + "/\*"):

remote\_path = os.path.join(

self.gcs\_output\_path.split("/")[-1], local\_file[1 + len(local\_path) :]

)

blob = bucket.blob(remote\_path)

blob.upload\_from\_filename(local\_file)

if \_\_name\_\_ == "\_\_main\_\_":

logging.getLogger().setLevel(logging.INFO)

run\_pipeline(sys.argv)