package com.twitter.interaction\_graph.scio.ml.labels

import com.google.api.services.bigquery.model.TimePartitioning

import com.spotify.scio.ScioContext

import com.spotify.scio.values.SCollection

import com.twitter.beam.io.dal.DAL

import com.twitter.beam.io.fs.multiformat.DiskFormat

import com.twitter.beam.io.fs.multiformat.PathLayout

import com.twitter.beam.io.fs.multiformat.WriteOptions

import com.twitter.beam.job.ServiceIdentifierOptions

import com.twitter.cde.scio.dal\_read.SourceUtil

import com.twitter.conversions.DurationOps.\_

import com.twitter.dal.client.dataset.TimePartitionedDALDataset

import com.twitter.interaction\_graph.scio.agg\_client\_event\_logs.InteractionGraphAggClientEventLogsEdgeDailyScalaDataset

import com.twitter.interaction\_graph.scio.agg\_direct\_interactions.InteractionGraphAggDirectInteractionsEdgeDailyScalaDataset

import com.twitter.interaction\_graph.scio.agg\_notifications.InteractionGraphAggNotificationsEdgeDailyScalaDataset

import com.twitter.interaction\_graph.thriftscala.Edge

import com.twitter.interaction\_graph.thriftscala.EdgeLabel

import com.twitter.scio\_internal.job.ScioBeamJob

import com.twitter.socialgraph.event.thriftscala.FollowEvent

import com.twitter.socialgraph.hadoop.SocialgraphFollowEventsScalaDataset

import com.twitter.statebird.v2.thriftscala.Environment

import com.twitter.tcdc.bqblaster.beam.syntax.\_

import com.twitter.tcdc.bqblaster.core.avro.TypedProjection

import com.twitter.tcdc.bqblaster.core.transform.RootTransform

import org.apache.beam.sdk.io.gcp.bigquery.BigQueryIO

import org.joda.time.Interval

object InteractionGraphLabelsJob extends ScioBeamJob[InteractionGraphLabelsOption] {

override protected def configurePipeline(

scioContext: ScioContext,

pipelineOptions: InteractionGraphLabelsOption

): Unit = {

@transient

implicit lazy val sc: ScioContext = scioContext

implicit lazy val dateInterval: Interval = pipelineOptions.interval

val bqTableName: String = pipelineOptions.getBqTableName

val dalEnvironment: String = pipelineOptions

.as(classOf[ServiceIdentifierOptions])

.getEnvironment()

val dalWriteEnvironment = if (pipelineOptions.getDALWriteEnvironment != null) {

pipelineOptions.getDALWriteEnvironment

} else {

dalEnvironment

}

def readPartition[T: Manifest](dataset: TimePartitionedDALDataset[T]): SCollection[T] = {

SourceUtil.readDALDataset[T](

dataset = dataset,

interval = dateInterval,

dalEnvironment = dalEnvironment

)

}

val follows = readPartition[FollowEvent](SocialgraphFollowEventsScalaDataset)

.flatMap(LabelUtil.fromFollowEvent)

val directInteractions =

readPartition[Edge](InteractionGraphAggDirectInteractionsEdgeDailyScalaDataset)

.flatMap(LabelUtil.fromInteractionGraphEdge)

val clientEvents =

readPartition[Edge](InteractionGraphAggClientEventLogsEdgeDailyScalaDataset)

.flatMap(LabelUtil.fromInteractionGraphEdge)

val pushEvents =

readPartition[Edge](InteractionGraphAggNotificationsEdgeDailyScalaDataset)

.flatMap(LabelUtil.fromInteractionGraphEdge)

val labels = groupLabels(

follows ++

directInteractions ++

clientEvents ++

pushEvents)

labels.saveAsCustomOutput(

"Write Edge Labels",

DAL.write[EdgeLabel](

InteractionGraphLabelsDailyScalaDataset,

PathLayout.DailyPath(pipelineOptions.getOutputPath),

dateInterval,

DiskFormat.Parquet,

Environment.valueOf(dalWriteEnvironment),

writeOption = WriteOptions(numOfShards = Some(pipelineOptions.getNumberOfShards))

)

)

// save to BQ

if (pipelineOptions.getBqTableName != null) {

val ingestionTime = pipelineOptions.getDate().value.getStart.toDate

val bqFieldsTransform = RootTransform

.Builder()

.withPrependedFields("dateHour" -> TypedProjection.fromConstant(ingestionTime))

val timePartitioning = new TimePartitioning()

.setType("DAY").setField("dateHour").setExpirationMs(90.days.inMilliseconds)

val bqWriter = BigQueryIO

.write[EdgeLabel]

.to(bqTableName)

.withExtendedErrorInfo()

.withTimePartitioning(timePartitioning)

.withLoadJobProjectId("twttr-recos-ml-prod")

.withThriftSupport(bqFieldsTransform.build(), AvroConverter.Legacy)

.withCreateDisposition(BigQueryIO.Write.CreateDisposition.CREATE\_IF\_NEEDED)

.withWriteDisposition(BigQueryIO.Write.WriteDisposition.WRITE\_APPEND)

labels

.saveAsCustomOutput(

s"Save Recommendations to BQ $bqTableName",

bqWriter

)

}

}

def groupLabels(labels: SCollection[EdgeLabel]): SCollection[EdgeLabel] = {

labels

.map { e: EdgeLabel => ((e.sourceId, e.destinationId), e.labels.toSet) }

.sumByKey

.map { case ((srcId, destId), labels) => EdgeLabel(srcId, destId, labels) }

}

}