package com.twitter.interaction\_graph.scio.agg\_notifications

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.ReadOptions

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

import com.twitter.client\_event\_filtering.FrigateFilteredClientEventsDataflowScalaDataset

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.interaction\_graph.scio.common.FeatureGeneratorUtil

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

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

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

import com.twitter.tweetsource.public\_tweets.PublicTweetsScalaDataset

object InteractionGraphNotificationsJob extends ScioBeamJob[InteractionGraphNotificationsOption] {

override protected def configurePipeline(

sc: ScioContext,

opts: InteractionGraphNotificationsOption

): Unit = {

val pushClientEvents: SCollection[LogEvent] = sc

.customInput(

name = "Read Push Client Events",

DAL

.read(

FrigateFilteredClientEventsDataflowScalaDataset,

opts.interval,

DAL.Environment.Prod,

)

)

val pushNtabEvents =

pushClientEvents.flatMap(InteractionGraphNotificationUtil.getPushNtabEvents)

// look back tweets for 2 days because MR gets tweets from 2 days ago.

// Allow a grace period of 24 hours to reduce oncall workload

val graceHours = 24

val interval2DaysBefore =

opts.interval.withStart(opts.interval.getStart.minusDays(2).plusHours(graceHours))

val tweetAuthors: SCollection[(Long, Long)] = sc

.customInput(

name = "Read Tweets",

DAL

.read(

dataset = PublicTweetsScalaDataset,

interval = interval2DaysBefore,

environmentOverride = DAL.Environment.Prod,

readOptions = ReadOptions(projections = Some(Seq("tweetId", "userId")))

)

).map { t => (t.tweetId, t.userId) }

val pushNtabEdgeCounts = pushNtabEvents

.join(tweetAuthors)

.map {

case (\_, ((srcId, feature), destId)) => ((srcId, destId, feature), 1L)

}

.withName("summing edge feature counts")

.sumByKey

val aggPushEdges = pushNtabEdgeCounts

.map {

case ((srcId, destId, featureName), count) =>

(srcId, destId) -> Seq(

EdgeFeature(featureName, FeatureGeneratorUtil.initializeTSS(count)))

}

.sumByKey

.map {

case ((srcId, destId), edgeFeatures) =>

Edge(srcId, destId, None, edgeFeatures.sortBy(\_.name.value))

}

aggPushEdges.saveAsCustomOutput(

"Write Edge Records",

DAL.write[Edge](

InteractionGraphAggNotificationsEdgeDailyScalaDataset,

PathLayout.DailyPath(opts.getOutputPath + "/aggregated\_notifications\_edge\_daily"),

opts.interval,

DiskFormat.Parquet,

Environment.valueOf(opts.getDALWriteEnvironment),

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

)

)

}

}