package com.twitter.simclusters\_v2.scalding

import com.twitter.scalding.\_

import com.twitter.scalding\_internal.dalv2.DALWrite.{D, WriteExtension}

import com.twitter.scalding\_internal.job.analytics\_batch.{

AnalyticsBatchExecution,

AnalyticsBatchExecutionArgs,

BatchDescription,

BatchFirstTime,

BatchIncrement,

TwitterScheduledExecutionApp

}

import com.twitter.simclusters\_v2.scalding.common.Util

import com.twitter.simclusters\_v2.hdfs\_sources.{

UserAndNeighborsFixedPathSource,

UserUserGraphScalaDataset

}

import com.twitter.simclusters\_v2.thriftscala.{NeighborWithWeights, UserAndNeighbors}

import com.twitter.wtf.scalding.jobs.common.AdhocExecutionApp

import java.util.TimeZone

/\*\*

\* This is a scheduled version of the user\_user\_normalized\_graph dataset generation job.

\*

\* The key difference in this implementation is that we donot read the ProducerNormsAndCounts dataset.

\* So we no longer store the following producer normalized scores for the edges in the NeigborWithWeights thrift:

\* followScoreNormalizedByNeighborFollowersL2, favScoreHalfLife100DaysNormalizedByNeighborFaversL2 and logFavScoreL2Normalized

\*

\*/

object UserUserGraph {

def getNeighborWithWeights(

inputEdge: Edge

): NeighborWithWeights = {

val logFavScore = UserUserNormalizedGraph.logTransformation(inputEdge.favWeight)

NeighborWithWeights(

neighborId = inputEdge.destId,

isFollowed = Some(inputEdge.isFollowEdge),

favScoreHalfLife100Days = Some(inputEdge.favWeight),

logFavScore = Some(logFavScore),

)

}

def addWeightsAndAdjListify(

input: TypedPipe[Edge],

maxNeighborsPerUser: Int

)(

implicit uniqueId: UniqueID

): TypedPipe[UserAndNeighbors] = {

val numUsersNeedingNeighborTruncation = Stat("num\_users\_needing\_neighbor\_truncation")

val numEdgesAfterTruncation = Stat("num\_edges\_after\_truncation")

val numEdgesBeforeTruncation = Stat("num\_edges\_before\_truncation")

val numFollowEdgesBeforeTruncation = Stat("num\_follow\_edges\_before\_truncation")

val numFavEdgesBeforeTruncation = Stat("num\_fav\_edges\_before\_truncation")

val numFollowEdgesAfterTruncation = Stat("num\_follow\_edges\_after\_truncation")

val numFavEdgesAfterTruncation = Stat("num\_fav\_edges\_after\_truncation")

val numRecordsInOutputGraph = Stat("num\_records\_in\_output\_graph")

input

.map { edge =>

numEdgesBeforeTruncation.inc()

if (edge.isFollowEdge) numFollowEdgesBeforeTruncation.inc()

if (edge.favWeight > 0) numFavEdgesBeforeTruncation.inc()

(edge.srcId, getNeighborWithWeights(edge))

}

.group

// .withReducers(10000)

.sortedReverseTake(maxNeighborsPerUser)(Ordering.by { x: NeighborWithWeights =>

x.favScoreHalfLife100Days.getOrElse(0.0)

})

.map {

case (srcId, neighborList) =>

if (neighborList.size >= maxNeighborsPerUser) numUsersNeedingNeighborTruncation.inc()

neighborList.foreach { neighbor =>

numEdgesAfterTruncation.inc()

if (neighbor.favScoreHalfLife100Days.exists(\_ > 0)) numFavEdgesAfterTruncation.inc()

if (neighbor.isFollowed.contains(true)) numFollowEdgesAfterTruncation.inc()

}

numRecordsInOutputGraph.inc()

UserAndNeighbors(srcId, neighborList)

}

}

def run(

followEdges: TypedPipe[(Long, Long)],

favEdges: TypedPipe[(Long, Long, Double)],

maxNeighborsPerUser: Int

)(

implicit uniqueID: UniqueID

): TypedPipe[UserAndNeighbors] = {

val combined = UserUserNormalizedGraph.combineFollowAndFav(followEdges, favEdges)

addWeightsAndAdjListify(

combined,

maxNeighborsPerUser

)

}

}

/\*\*

\*

\* capesospy-v2 update --build\_locally --start\_cron user\_user\_follow\_fav\_graph \

\* src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc.yaml

\*/

object UserUserGraphBatch extends TwitterScheduledExecutionApp {

private val firstTime: String = "2021-04-24"

implicit val tz = DateOps.UTC

implicit val parser = DateParser.default

private val batchIncrement: Duration = Days(2)

private val halfLifeInDaysForFavScore = 100

private val outputPath: String = "/user/cassowary/processed/user\_user\_graph"

private val execArgs = AnalyticsBatchExecutionArgs(

batchDesc = BatchDescription(this.getClass.getName.replace("$", "")),

firstTime = BatchFirstTime(RichDate(firstTime)),

lastTime = None,

batchIncrement = BatchIncrement(batchIncrement)

)

override def scheduledJob: Execution[Unit] = AnalyticsBatchExecution(execArgs) {

implicit dateRange =>

Execution.withId { implicit uniqueId =>

Execution.withArgs { args =>

val maxNeighborsPerUser = args.int("maxNeighborsPerUser", 2000)

Util.printCounters(

UserUserGraph

.run(

UserUserNormalizedGraph.getFollowEdges,

UserUserNormalizedGraph.getFavEdges(halfLifeInDaysForFavScore),

maxNeighborsPerUser

)

.writeDALSnapshotExecution(

UserUserGraphScalaDataset,

D.Daily,

D.Suffix(outputPath),

D.EBLzo(),

dateRange.end)

)

}

}

}

}

/\*\*

./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding:user\_user\_graph-adhoc

scalding remote run \

--user cassowary \

--keytab /var/lib/tss/keys/fluffy/keytabs/client/cassowary.keytab \

--principal service\_acoount@TWITTER.BIZ \

--cluster bluebird-qus1 \

--main-class com.twitter.simclusters\_v2.scalding.UserUserGraphAdhoc \

--target src/scala/com/twitter/simclusters\_v2/scalding:user\_user\_graph-adhoc \

-- --date 2021-04-24 --outputDir "/user/cassowary/adhoc/user\_user\_graph\_adhoc"

\*/

object UserUserGraphAdhoc extends AdhocExecutionApp {

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val maxNeighborsPerUser = args.int("maxNeighborsPerUser", 2000)

val halfLifeInDaysForFavScore = 100

val outputDir = args("outputDir")

val userAndNeighbors =

UserUserGraph

.run(

UserUserNormalizedGraph.getFollowEdges,

UserUserNormalizedGraph.getFavEdges(halfLifeInDaysForFavScore),

maxNeighborsPerUser)

Execution

.zip(

userAndNeighbors.writeExecution(UserAndNeighborsFixedPathSource(outputDir)),

userAndNeighbors.writeExecution(TypedTsv(outputDir + "\_tsv"))).unit

}

}