package com.twitter.simclusters\_v2.scalding.offline\_job

import com.twitter.scalding.\_

import com.twitter.scalding\_internal.dalv2.DAL

import com.twitter.scalding\_internal.dalv2.DALWrite.\_

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

import com.twitter.simclusters\_v2.scalding.offline\_job.SimClustersOfflineJob.\_

import com.twitter.simclusters\_v2.scalding.offline\_job.SimClustersOfflineJobUtil.\_

import com.twitter.simclusters\_v2.thriftscala.TweetAndClusterScores

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

import java.util.TimeZone

/\*\*

\* The offline job runs every 12 hours, and save these two data sets to HDFS.

\*

\* capesospy-v2 update --build\_locally --start\_cron \

\* --start\_cron offline\_tweet\_job src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object SimClustersOfflineJobScheduledApp extends ScheduledExecutionApp {

import com.twitter.simclusters\_v2.scalding.common.TypedRichPipe.\_

private val tweetClusterScoresDatasetPath: String =

"/user/cassowary/processed/simclusters/tweet\_cluster\_scores"

private val tweetTopKClustersDatasetPath: String =

"/user/cassowary/processed/simclusters/tweet\_top\_k\_clusters"

private val clusterTopKTweetsDatasetPath: String =

"/user/cassowary/processed/simclusters/cluster\_top\_k\_tweets"

override def batchIncrement: Duration = Hours(12)

override def firstTime: RichDate = RichDate("2020-05-25")

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val previousTweetClusterScores: TypedPipe[TweetAndClusterScores] =

if (firstTime.timestamp == dateRange.start.timestamp) { // if it is the first batch

TypedPipe.from(Nil)

} else {

DAL

.readMostRecentSnapshot(

SimclustersOfflineTweetClusterScoresScalaDataset,

dateRange - batchIncrement

)

.toTypedPipe

.count("NumPreviousTweetClusterScores")

}

// we have to use some way to throw away old tweets, otherwise the data set will be growing

// all the time. We only keep the tweets that received at least 1 engagement in the last day.

// This parameter can be adjusted

val tweetsToKeep = getSubsetOfValidTweets(Days(1))

.count("NumTweetsToKeep")

val updatedTweetClusterScores = computeAggregatedTweetClusterScores(

dateRange,

readInterestedInScalaDataset(dateRange),

readTimelineFavoriteData(dateRange),

previousTweetClusterScores

).map { tweetClusterScore =>

tweetClusterScore.tweetId -> tweetClusterScore

}

.count("NumUpdatedTweetClusterScoresBeforeFiltering")

.join(tweetsToKeep.asKeys) // filter out invalid tweets

.map {

case (\_, (tweetClusterScore, \_)) => tweetClusterScore

}

.count("NumUpdatedTweetClusterScores")

.forceToDisk

val tweetTopKClusters = computeTweetTopKClusters(updatedTweetClusterScores)

.count("NumTweetTopKSaved")

val clusterTopKTweets = computeClusterTopKTweets(updatedTweetClusterScores)

.count("NumClusterTopKSaved")

val writeTweetClusterScoresExec = updatedTweetClusterScores

.writeDALSnapshotExecution(

SimclustersOfflineTweetClusterScoresScalaDataset,

D.Hourly, // note that we use hourly in order to make it flexible for hourly batch size

D.Suffix(tweetClusterScoresDatasetPath),

D.EBLzo(),

dateRange.end

)

val writeTweetTopKClustersExec = tweetTopKClusters

.writeDALSnapshotExecution(

SimclustersOfflineTweetTopKClustersScalaDataset,

D.Hourly, // note that we use hourly in order to make it flexible for hourly batch size

D.Suffix(tweetTopKClustersDatasetPath),

D.EBLzo(),

dateRange.end

)

val writeClusterTopKTweetsExec = clusterTopKTweets

.writeDALSnapshotExecution(

SimclustersOfflineClusterTopKTweetsScalaDataset,

D.Hourly, // note that we use hourly in order to make it flexible for hourly batch size

D.Suffix(clusterTopKTweetsDatasetPath),

D.EBLzo(),

dateRange.end

)

Execution

.zip(writeTweetClusterScoresExec, writeTweetTopKClustersExec, writeClusterTopKTweetsExec)

.unit

}

}