package com.twitter.simclusters\_v2.scio

package multi\_type\_graph.common

import com.spotify.scio.ScioContext

import com.spotify.scio.values.SCollection

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

import com.twitter.common.util.Clock

import com.twitter.scalding\_internal.job.RequiredBinaryComparators.ordSer

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

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

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

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

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

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

import com.twitter.util.Duration

object MultiTypeGraphUtil {

val RootMHPath: String = "manhattan\_sequence\_files/multi\_type\_graph/"

val RootThriftPath: String = "processed/multi\_type\_graph/"

val AdhocRootPath = "adhoc/multi\_type\_graph/"

val nounOrdering: Ordering[Noun] = new Ordering[Noun] {

// We define an ordering for each noun type as specified in simclusters\_v2/multi\_type\_graph.thrift

// Please make sure we don't remove anything here that's still a part of the union Noun thrift and

// vice versa, if we add a new noun type to thrift, an ordering for it needs to added here as well.

def nounTypeOrder(noun: Noun): Int = noun match {

case \_: Noun.UserId => 0

case \_: Noun.Country => 1

case \_: Noun.Language => 2

case \_: Noun.Query => 3

case \_: Noun.TopicId => 4

case \_: Noun.TweetId => 5

}

override def compare(x: Noun, y: Noun): Int = nounTypeOrder(x) compare nounTypeOrder(y)

}

val rightNodeTypeOrdering: Ordering[RightNodeType] = ordSer[RightNodeType]

val rightNodeOrdering: Ordering[RightNode] =

new Ordering[RightNode] {

override def compare(x: RightNode, y: RightNode): Int = {

Ordering

.Tuple2(rightNodeTypeOrdering, nounOrdering)

.compare((x.rightNodeType, x.noun), (y.rightNodeType, y.noun))

}

}

def getTruncatedMultiTypeGraph(

noOlderThan: Duration = Duration.fromDays(14)

)(

implicit sc: ScioContext

): SCollection[(Long, RightNode, Double)] = {

sc.customInput(

"ReadTruncatedMultiTypeGraph",

DAL

.readMostRecentSnapshotNoOlderThan(

TruncatedMultiTypeGraphScioScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod

)

).flatMap {

case KeyVal(LeftNode.UserId(userId), rightNodesList) =>

rightNodesList.rightNodeWithEdgeWeightList.map(rightNodeWithWeight =>

(userId, rightNodeWithWeight.rightNode, rightNodeWithWeight.weight))

}

}

}