package com.twitter.graph\_feature\_service.worker.util

import com.twitter.bijection.Injection

import com.twitter.concurrent.AsyncSemaphore

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.constdb\_util.{

AutoUpdatingReadOnlyGraph,

ConstDBImporter,

Injections

}

import com.twitter.graph\_feature\_service.common.Configs

import com.twitter.util.{Duration, Future, Timer}

import java.nio.ByteBuffer

/\*\*

\* @param dataPath the path to the data on HDFS

\* @param hdfsCluster cluster where we check for updates and download graph files from

\* @param hdfsClusterUrl url to HDFS cluster

\* @param shard The shard of the graph to download

\* @param minimumSizeForCompleteGraph minimumSize for complete graph - otherwise we don't load it

\* @param updateIntervalMin The interval after which the first update is tried and the interval between such updates

\* @param updateIntervalMax the maximum time before an update is triggered

\* @param deleteInterval The interval after which older data is deleted from disk

\* @param sharedSemaphore The semaphore controls the number of graph loads at same time on the instance.

\*/

case class AutoUpdatingGraph(

dataPath: String,

hdfsCluster: String,

hdfsClusterUrl: String,

shard: Int,

minimumSizeForCompleteGraph: Long,

updateIntervalMin: Duration = 1.hour,

updateIntervalMax: Duration = 12.hours,

deleteInterval: Duration = 2.seconds,

sharedSemaphore: Option[AsyncSemaphore] = None

)(

implicit statsReceiver: StatsReceiver,

timer: Timer)

extends AutoUpdatingReadOnlyGraph[Long, ByteBuffer](

hdfsCluster,

hdfsClusterUrl,

shard,

minimumSizeForCompleteGraph,

updateIntervalMin,

updateIntervalMax,

deleteInterval,

sharedSemaphore

)

with ConstDBImporter[Long, ByteBuffer] {

override def numGraphShards: Int = Configs.NumGraphShards

override def basePath: String = dataPath

override val keyInj: Injection[Long, ByteBuffer] = Injections.long2Varint

override val valueInj: Injection[ByteBuffer, ByteBuffer] = Injection.identity

override def get(targetId: Long): Future[Option[ByteBuffer]] =

super

.get(targetId)

.map { res =>

res.foreach(r => arraySizeStat.add(r.remaining()))

res

}

private val arraySizeStat = stats.scope("get").stat("size")

}