package com.twitter.recosinjector.clients

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.util.DefaultTimer

import com.twitter.frigate.common.util.{SnowflakeUtils, UrlInfo}

import com.twitter.storehaus.{FutureOps, ReadableStore}

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

class UrlResolver(

urlInfoStore: ReadableStore[String, UrlInfo]

)(

implicit statsReceiver: StatsReceiver) {

private val EmptyFutureMap = Future.value(Map.empty[String, String])

private val stats = statsReceiver.scope(this.getClass.getSimpleName)

private val twitterResolvedUrlCounter = stats.counter("twitterResolvedUrl")

private val resolvedUrlCounter = stats.counter("resolvedUrl")

private val noResolvedUrlCounter = stats.counter("noResolvedUrl")

private val numNoDelayCounter = stats.counter("urlResolver\_no\_delay")

private val numDelayCounter = stats.counter("urlResolver\_delay")

implicit val timer: Timer = DefaultTimer

/\*\*

\* Get the resolved URL map of the input raw URLs

\*

\* @param rawUrls list of raw URLs to query

\* @return map of raw URL to resolved URL

\*/

def getResolvedUrls(rawUrls: Set[String]): Future[Map[String, String]] = {

FutureOps

.mapCollect(urlInfoStore.multiGet[String](rawUrls))

.map { resolvedUrlsMap =>

resolvedUrlsMap.flatMap {

case (

url,

Some(

UrlInfo(

Some(resolvedUrl),

Some(\_),

Some(domain),

\_,

\_,

\_,

\_,

Some(\_),

\_,

\_,

\_,

\_))) =>

if (domain == "Twitter") { // Filter out Twitter based URLs

twitterResolvedUrlCounter.incr()

None

} else {

resolvedUrlCounter.incr()

Some(url -> resolvedUrl)

}

case \_ =>

noResolvedUrlCounter.incr()

None

}

}

}

/\*\*

\* Get resolved url maps given a list of urls, grouping urls that point to the same webpage

\*/

def getResolvedUrls(urls: Seq[String], tweetId: Long): Future[Map[String, String]] = {

if (urls.isEmpty) {

EmptyFutureMap

} else {

Future

.sleep(getUrlResolverDelayDuration(tweetId))

.before(getResolvedUrls(urls.toSet))

}

}

/\*\*

\* Given a tweet, return the amount of delay needed before attempting to resolve the Urls

\*/

private def getUrlResolverDelayDuration(

tweetId: Long

): Duration = {

val urlResolverDelaySinceCreation = 12.seconds

val urlResolverDelayDuration = 4.seconds

val noDelay = 0.seconds

// Check whether the tweet was created more than the specified delay duration before now.

// If the tweet ID is not based on Snowflake, this is false, and the delay is applied.

val isCreatedBeforeDelayThreshold = SnowflakeUtils

.tweetCreationTime(tweetId)

.map(\_.untilNow)

.exists(\_ > urlResolverDelaySinceCreation)

if (isCreatedBeforeDelayThreshold) {

numNoDelayCounter.incr()

noDelay

} else {

numDelayCounter.incr()

urlResolverDelayDuration

}

}

}