package com.twitter.tweetypie

package handler

import com.twitter.finagle.stats.Stat

import com.twitter.flockdb.client.\_

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.thriftscala.\_

trait EraseUserTweetsHandler {

val eraseUserTweetsRequest: FutureArrow[EraseUserTweetsRequest, Unit]

val asyncEraseUserTweetsRequest: FutureArrow[AsyncEraseUserTweetsRequest, Unit]

}

/\*\*

\* This library allows you to erase all of a users's tweets. It's used to clean up

\* tweets after a user deletes their account.

\*/

object EraseUserTweetsHandler {

/\*\*

\* Build a FutureEffect which, when called, deletes one page worth of tweets at the

\* specified flock cursor. When the page of tweets has been deleted another asyncEraseUserTweets

\* request is made with the updated cursor location so that the next page of tweets can be processed.

\*/

def apply(

selectPage: FutureArrow[Select[StatusGraph], PageResult[Long]],

deleteTweet: FutureEffect[(TweetId, UserId)],

asyncEraseUserTweets: FutureArrow[AsyncEraseUserTweetsRequest, Unit],

stats: StatsReceiver,

sleep: () => Future[Unit] = () => Future.Unit

): EraseUserTweetsHandler =

new EraseUserTweetsHandler {

val latencyStat: Stat = stats.stat("latency\_ms")

val deletedTweetsStat: Stat = stats.stat("tweets\_deleted\_for\_erased\_user")

val selectUserTweets: AsyncEraseUserTweetsRequest => Select[StatusGraph] =

(request: AsyncEraseUserTweetsRequest) =>

UserTimelineGraph

.from(request.userId)

.withCursor(Cursor(request.flockCursor))

// For a provided list of tweetIds, delete each one sequentially, sleeping between each call

// This is a rate limiting mechanism to slow down deletions.

def deletePage(page: PageResult[Long], expectedUserId: UserId): Future[Unit] =

page.entries.foldLeft(Future.Unit) { (previousFuture, nextId) =>

for {

\_ <- previousFuture

\_ <- sleep()

\_ <- deleteTweet((nextId, expectedUserId))

} yield ()

}

/\*\*

\* If we aren't on the last page, make another EraseUserTweets request to delete

\* the next page of tweets

\*/

val nextRequestOrEnd: (AsyncEraseUserTweetsRequest, PageResult[Long]) => Future[Unit] =

(request: AsyncEraseUserTweetsRequest, page: PageResult[Long]) =>

if (page.nextCursor.isEnd) {

latencyStat.add(Time.fromMilliseconds(request.startTimestamp).untilNow.inMillis)

deletedTweetsStat.add(request.tweetCount + page.entries.size)

Future.Unit

} else {

asyncEraseUserTweets(

request.copy(

flockCursor = page.nextCursor.value,

tweetCount = request.tweetCount + page.entries.size

)

)

}

override val eraseUserTweetsRequest: FutureArrow[EraseUserTweetsRequest, Unit] =

FutureArrow { request =>

asyncEraseUserTweets(

AsyncEraseUserTweetsRequest(

userId = request.userId,

flockCursor = Cursor.start.value,

startTimestamp = Time.now.inMillis,

tweetCount = 0L

)

)

}

override val asyncEraseUserTweetsRequest: FutureArrow[AsyncEraseUserTweetsRequest, Unit] =

FutureArrow { request =>

for {

\_ <- sleep()

// get one page of tweets

page <- selectPage(selectUserTweets(request))

// delete tweets

\_ <- deletePage(page, request.userId)

// make call to delete the next page of tweets

\_ <- nextRequestOrEnd(request, page)

} yield ()

}

}

}