package com.twitter.tweetypie

package handler

import com.twitter.flockdb.client.Cursor

import com.twitter.flockdb.client.PageResult

import com.twitter.flockdb.client.Select

import com.twitter.flockdb.client.StatusGraph

import com.twitter.flockdb.client.UserTimelineGraph

import com.twitter.flockdb.client.thriftscala.EdgeState

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.storage.TweetStorageClient

import com.twitter.tweetypie.storage.TweetStorageClient.GetStoredTweet

import com.twitter.tweetypie.thriftscala.GetStoredTweetsByUserOptions

import com.twitter.tweetypie.thriftscala.GetStoredTweetsByUserRequest

import com.twitter.tweetypie.thriftscala.GetStoredTweetsByUserResult

import com.twitter.tweetypie.thriftscala.GetStoredTweetsOptions

import com.twitter.tweetypie.thriftscala.GetStoredTweetsRequest

object GetStoredTweetsByUserHandler {

type Type = FutureArrow[GetStoredTweetsByUserRequest, GetStoredTweetsByUserResult]

def apply(

getStoredTweetsHandler: GetStoredTweetsHandler.Type,

getStoredTweet: TweetStorageClient.GetStoredTweet,

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

maxPages: Int

): Type = {

FutureArrow { request =>

val options = request.options.getOrElse(GetStoredTweetsByUserOptions())

val startTimeMsec: Long = options.startTimeMsec.getOrElse(0L)

val endTimeMsec: Long = options.endTimeMsec.getOrElse(Time.now.inMillis)

val cursor = options.cursor.map(Cursor(\_)).getOrElse {

if (options.startFromOldest) Cursor.lowest else Cursor.highest

}

getNextTweetIdsInTimeRange(

request.userId,

startTimeMsec,

endTimeMsec,

cursor,

selectPage,

getStoredTweet,

maxPages,

numTries = 0

).flatMap {

case (tweetIds, cursor) =>

val getStoredTweetsRequest = toGetStoredTweetsRequest(tweetIds, request.userId, options)

getStoredTweetsHandler(getStoredTweetsRequest)

.map { getStoredTweetsResults =>

GetStoredTweetsByUserResult(

storedTweets = getStoredTweetsResults.map(\_.storedTweet),

cursor = if (cursor.isEnd) None else Some(cursor.value)

)

}

}

}

}

private def toGetStoredTweetsRequest(

tweetIds: Seq[TweetId],

userId: UserId,

getStoredTweetsByUserOptions: GetStoredTweetsByUserOptions

): GetStoredTweetsRequest = {

val options: GetStoredTweetsOptions = GetStoredTweetsOptions(

bypassVisibilityFiltering = getStoredTweetsByUserOptions.bypassVisibilityFiltering,

forUserId = if (getStoredTweetsByUserOptions.setForUserId) Some(userId) else None,

additionalFieldIds = getStoredTweetsByUserOptions.additionalFieldIds

)

GetStoredTweetsRequest(

tweetIds = tweetIds,

options = Some(options)

)

}

private def getNextTweetIdsInTimeRange(

userId: UserId,

startTimeMsec: Long,

endTimeMsec: Long,

cursor: Cursor,

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

getStoredTweet: TweetStorageClient.GetStoredTweet,

maxPages: Int,

numTries: Int

): Future[(Seq[TweetId], Cursor)] = {

val select = Select(

sourceId = userId,

graph = UserTimelineGraph,

stateIds =

Some(Seq(EdgeState.Archived.value, EdgeState.Positive.value, EdgeState.Removed.value))

).withCursor(cursor)

def inTimeRange(timestamp: Long): Boolean =

timestamp >= startTimeMsec && timestamp <= endTimeMsec

def pastTimeRange(timestamps: Seq[Long]) = {

if (cursor.isAscending) {

timestamps.max > endTimeMsec

} else {

timestamps.min < startTimeMsec

}

}

val pageResultFuture: Future[PageResult[Long]] = selectPage(select)

pageResultFuture.flatMap { pageResult =>

val groupedIds = pageResult.entries.groupBy(SnowflakeId.isSnowflakeId)

val nextCursor = if (cursor.isAscending) pageResult.previousCursor else pageResult.nextCursor

// Timestamps for the creation of Tweets with snowflake IDs can be calculated from the IDs

// themselves.

val snowflakeIdsTimestamps: Seq[(Long, Long)] = groupedIds.getOrElse(true, Seq()).map { id =>

val snowflakeTimeMillis = SnowflakeId.unixTimeMillisFromId(id)

(id, snowflakeTimeMillis)

}

// For non-snowflake Tweets, we need to fetch the Tweet data from Manhattan to see when the

// Tweet was created.

val nonSnowflakeIdsTimestamps: Future[Seq[(Long, Long)]] = Stitch.run(

Stitch

.traverse(groupedIds.getOrElse(false, Seq()))(getStoredTweet)

.map {

\_.flatMap {

case GetStoredTweet.Response.FoundAny(tweet, \_, \_, \_, \_) => {

if (tweet.coreData.exists(\_.createdAtSecs > 0)) {

Some((tweet.id, tweet.coreData.get.createdAtSecs))

} else None

}

case \_ => None

}

})

nonSnowflakeIdsTimestamps.flatMap { nonSnowflakeList =>

val allTweetIdsAndTimestamps = snowflakeIdsTimestamps ++ nonSnowflakeList

val filteredTweetIds = allTweetIdsAndTimestamps

.filter {

case (\_, ts) => inTimeRange(ts)

}

.map(\_.\_1)

if (nextCursor.isEnd) {

// We've considered the last Tweet for this User. There are no more Tweets to return.

Future.value((filteredTweetIds, Cursor.end))

} else if (allTweetIdsAndTimestamps.nonEmpty &&

pastTimeRange(allTweetIdsAndTimestamps.map(\_.\_2))) {

// At least one Tweet returned from Tflock has a timestamp past our time range, i.e.

// greater than the end time (if we're fetching in an ascending order) or lower than the

// start time (if we're fetching in a descending order). There is no point in looking at

// any more Tweets from this User as they'll all be outside the time range.

Future.value((filteredTweetIds, Cursor.end))

} else if (filteredTweetIds.isEmpty) {

// We're here because one of two things happened:

// 1. allTweetIdsAndTimestamps is empty: Either Tflock has returned an empty page of Tweets

// or we weren't able to fetch timestamps for any of the Tweets Tflock returned. In this

// case, we fetch the next page of Tweets.

// 2. allTweetIdsAndTimestamps is non-empty but filteredTweetIds is empty: The current page

// has no Tweets inside the requested time range. We fetch the next page of Tweets and

// try again.

// If we hit the limit for the maximum number of pages from tflock to be requested, we

// return an empty list of Tweets with the cursor for the caller to try again.

if (numTries == maxPages) {

Future.value((filteredTweetIds, nextCursor))

} else {

getNextTweetIdsInTimeRange(

userId = userId,

startTimeMsec = startTimeMsec,

endTimeMsec = endTimeMsec,

cursor = nextCursor,

selectPage = selectPage,

getStoredTweet = getStoredTweet,

maxPages = maxPages,

numTries = numTries + 1

)

}

} else {

// filteredTweetIds is non-empty: There are some Tweets in this page that are within the

// requested time range, and we aren't out of the time range yet. We return the Tweets we

// have and set the cursor forward for the next request.

Future.value((filteredTweetIds, nextCursor))

}

}

}

}

}