package com.twitter.cr\_mixer.similarity\_engine

import com.twitter.cr\_mixer.config.TimeoutConfig

import com.twitter.cr\_mixer.model.ModuleNames

import com.twitter.cr\_mixer.model.TweetWithAuthor

import com.twitter.cr\_mixer.similarity\_engine.EarlybirdRecencyBasedSimilarityEngine.EarlybirdRecencyBasedSearchQuery

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.util.StatsUtil

import com.twitter.simclusters\_v2.common.TweetId

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

@Singleton

case class EarlybirdRecencyBasedSimilarityEngine @Inject() (

@Named(ModuleNames.EarlybirdRecencyBasedWithoutRetweetsRepliesTweetsCache)

earlybirdRecencyBasedWithoutRetweetsRepliesTweetsCacheStore: ReadableStore[

UserId,

Seq[TweetId]

],

@Named(ModuleNames.EarlybirdRecencyBasedWithRetweetsRepliesTweetsCache)

earlybirdRecencyBasedWithRetweetsRepliesTweetsCacheStore: ReadableStore[

UserId,

Seq[TweetId]

],

timeoutConfig: TimeoutConfig,

stats: StatsReceiver)

extends ReadableStore[EarlybirdRecencyBasedSearchQuery, Seq[TweetWithAuthor]] {

import EarlybirdRecencyBasedSimilarityEngine.\_

val statsReceiver: StatsReceiver = stats.scope(this.getClass.getSimpleName)

override def get(

query: EarlybirdRecencyBasedSearchQuery

): Future[Option[Seq[TweetWithAuthor]]] = {

Future

.collect {

if (query.filterOutRetweetsAndReplies) {

query.seedUserIds.map { seedUserId =>

StatsUtil.trackOptionItemsStats(statsReceiver.scope("WithoutRetweetsAndReplies")) {

earlybirdRecencyBasedWithoutRetweetsRepliesTweetsCacheStore

.get(seedUserId).map(\_.map(\_.map(tweetId =>

TweetWithAuthor(tweetId = tweetId, authorId = seedUserId))))

}

}

} else {

query.seedUserIds.map { seedUserId =>

StatsUtil.trackOptionItemsStats(statsReceiver.scope("WithRetweetsAndReplies")) {

earlybirdRecencyBasedWithRetweetsRepliesTweetsCacheStore

.get(seedUserId)

.map(\_.map(\_.map(tweetId =>

TweetWithAuthor(tweetId = tweetId, authorId = seedUserId))))

}

}

}

}

.map { tweetWithAuthorList =>

val earliestTweetId = SnowflakeId.firstIdFor(Time.now - query.maxTweetAge)

tweetWithAuthorList

.flatMap(\_.getOrElse(Seq.empty))

.filter(tweetWithAuthor =>

tweetWithAuthor.tweetId >= earliestTweetId // tweet age filter

&& !query.excludedTweetIds

.contains(tweetWithAuthor.tweetId)) // excluded tweet filter

.sortBy(tweetWithAuthor =>

-SnowflakeId.unixTimeMillisFromId(tweetWithAuthor.tweetId)) // sort by recency

.take(query.maxNumTweets) // take most recent N tweets

}

.map(result => Some(result))

}

}

object EarlybirdRecencyBasedSimilarityEngine {

case class EarlybirdRecencyBasedSearchQuery(

seedUserIds: Seq[UserId],

maxNumTweets: Int,

excludedTweetIds: Set[TweetId],

maxTweetAge: Duration,

filterOutRetweetsAndReplies: Boolean)

}