package com.twitter.cr\_mixer.module

import com.google.inject.Provides

import com.google.inject.Singleton

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

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

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.EarlybirdClientId

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.FacetsToFetch

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.GetCollectorTerminationParams

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.GetEarlybirdQuery

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.MetadataOptions

import com.twitter.finagle.memcached.{Client => MemcachedClient}

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.util.SeqLongInjection

import com.twitter.hashing.KeyHasher

import com.twitter.hermit.store.common.ObservedMemcachedReadableStore

import com.twitter.inject.TwitterModule

import com.twitter.search.common.query.thriftjava.thriftscala.CollectorParams

import com.twitter.search.earlybird.thriftscala.EarlybirdRequest

import com.twitter.search.earlybird.thriftscala.EarlybirdResponseCode

import com.twitter.search.earlybird.thriftscala.EarlybirdService

import com.twitter.search.earlybird.thriftscala.ThriftSearchQuery

import com.twitter.search.earlybird.thriftscala.ThriftSearchRankingMode

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Duration

import com.twitter.util.Future

import javax.inject.Named

object EarlybirdRecencyBasedCandidateStoreModule extends TwitterModule {

@Provides

@Singleton

@Named(ModuleNames.EarlybirdRecencyBasedWithoutRetweetsRepliesTweetsCache)

def providesEarlybirdRecencyBasedWithoutRetweetsRepliesCandidateStore(

statsReceiver: StatsReceiver,

earlybirdSearchClient: EarlybirdService.MethodPerEndpoint,

@Named(ModuleNames.EarlybirdTweetsCache) earlybirdRecencyBasedTweetsCache: MemcachedClient,

timeoutConfig: TimeoutConfig

): ReadableStore[UserId, Seq[TweetId]] = {

val stats = statsReceiver.scope("EarlybirdRecencyBasedWithoutRetweetsRepliesCandidateStore")

val underlyingStore = new ReadableStore[UserId, Seq[TweetId]] {

override def get(userId: UserId): Future[Option[Seq[TweetId]]] = {

// Home based EB filters out retweets and replies

val earlybirdRequest =

buildEarlybirdRequest(

userId,

FilterOutRetweetsAndReplies,

DefaultMaxNumTweetPerUser,

timeoutConfig.earlybirdServerTimeout)

getEarlybirdSearchResult(earlybirdSearchClient, earlybirdRequest, stats)

}

}

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = earlybirdRecencyBasedTweetsCache,

ttl = MemcacheKeyTimeToLiveDuration,

asyncUpdate = true

)(

valueInjection = SeqLongInjection,

statsReceiver = statsReceiver.scope("earlybird\_recency\_based\_tweets\_home\_memcache"),

keyToString = { k =>

f"uEBRBHM:${keyHasher.hashKey(k.toString.getBytes)}%X" // prefix = EarlyBirdRecencyBasedHoMe

}

)

}

@Provides

@Singleton

@Named(ModuleNames.EarlybirdRecencyBasedWithRetweetsRepliesTweetsCache)

def providesEarlybirdRecencyBasedWithRetweetsRepliesCandidateStore(

statsReceiver: StatsReceiver,

earlybirdSearchClient: EarlybirdService.MethodPerEndpoint,

@Named(ModuleNames.EarlybirdTweetsCache) earlybirdRecencyBasedTweetsCache: MemcachedClient,

timeoutConfig: TimeoutConfig

): ReadableStore[UserId, Seq[TweetId]] = {

val stats = statsReceiver.scope("EarlybirdRecencyBasedWithRetweetsRepliesCandidateStore")

val underlyingStore = new ReadableStore[UserId, Seq[TweetId]] {

override def get(userId: UserId): Future[Option[Seq[TweetId]]] = {

val earlybirdRequest = buildEarlybirdRequest(

userId,

// Notifications based EB keeps retweets and replies

NotFilterOutRetweetsAndReplies,

DefaultMaxNumTweetPerUser,

processingTimeout = timeoutConfig.earlybirdServerTimeout

)

getEarlybirdSearchResult(earlybirdSearchClient, earlybirdRequest, stats)

}

}

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = earlybirdRecencyBasedTweetsCache,

ttl = MemcacheKeyTimeToLiveDuration,

asyncUpdate = true

)(

valueInjection = SeqLongInjection,

statsReceiver = statsReceiver.scope("earlybird\_recency\_based\_tweets\_notifications\_memcache"),

keyToString = { k =>

f"uEBRBN:${keyHasher.hashKey(k.toString.getBytes)}%X" // prefix = EarlyBirdRecencyBasedNotifications

}

)

}

private val keyHasher: KeyHasher = KeyHasher.FNV1A\_64

/\*\*

\* Note the DefaultMaxNumTweetPerUser is used to adjust the result size per cache entry.

\* If the value changes, it will increase the size of the memcache.

\*/

private val DefaultMaxNumTweetPerUser: Int = 100

private val FilterOutRetweetsAndReplies = true

private val NotFilterOutRetweetsAndReplies = false

private val MemcacheKeyTimeToLiveDuration: Duration = Duration.fromMinutes(15)

private def buildEarlybirdRequest(

seedUserId: UserId,

filterOutRetweetsAndReplies: Boolean,

maxNumTweetsPerSeedUser: Int,

processingTimeout: Duration

): EarlybirdRequest =

EarlybirdRequest(

searchQuery = getThriftSearchQuery(

seedUserId = seedUserId,

filterOutRetweetsAndReplies = filterOutRetweetsAndReplies,

maxNumTweetsPerSeedUser = maxNumTweetsPerSeedUser,

processingTimeout = processingTimeout

),

clientId = Some(EarlybirdClientId),

timeoutMs = processingTimeout.inMilliseconds.intValue(),

getOlderResults = Some(false),

adjustedProtectedRequestParams = None,

adjustedFullArchiveRequestParams = None,

getProtectedTweetsOnly = Some(false),

skipVeryRecentTweets = true,

)

private def getThriftSearchQuery(

seedUserId: UserId,

filterOutRetweetsAndReplies: Boolean,

maxNumTweetsPerSeedUser: Int,

processingTimeout: Duration

): ThriftSearchQuery = ThriftSearchQuery(

serializedQuery = GetEarlybirdQuery(

None,

None,

Set.empty,

filterOutRetweetsAndReplies

).map(\_.serialize),

fromUserIDFilter64 = Some(Seq(seedUserId)),

numResults = maxNumTweetsPerSeedUser,

rankingMode = ThriftSearchRankingMode.Recency,

collectorParams = Some(

CollectorParams(

// numResultsToReturn defines how many results each EB shard will return to search root

numResultsToReturn = maxNumTweetsPerSeedUser,

// terminationParams.maxHitsToProcess is used for early terminating per shard results fetching.

terminationParams =

GetCollectorTerminationParams(maxNumTweetsPerSeedUser, processingTimeout)

)),

facetFieldNames = Some(FacetsToFetch),

resultMetadataOptions = Some(MetadataOptions),

searchStatusIds = None

)

private def getEarlybirdSearchResult(

earlybirdSearchClient: EarlybirdService.MethodPerEndpoint,

request: EarlybirdRequest,

statsReceiver: StatsReceiver

): Future[Option[Seq[TweetId]]] = earlybirdSearchClient

.search(request)

.map { response =>

response.responseCode match {

case EarlybirdResponseCode.Success =>

val earlybirdSearchResult =

response.searchResults

.map {

\_.results

.map(searchResult => searchResult.id)

}

statsReceiver.scope("result").stat("size").add(earlybirdSearchResult.size)

earlybirdSearchResult

case e =>

statsReceiver.scope("failures").counter(e.getClass.getSimpleName).incr()

Some(Seq.empty)

}

}

}