package com.twitter.tsp.stores

import com.twitter.tsp.stores.TopicTweetsCosineSimilarityAggregateStore.ScoreKey

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.simclusters\_v2.thriftscala.\_

import com.twitter.storehaus.ReadableStore

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

import com.twitter.tsp.stores.SemanticCoreAnnotationStore.\_

import com.twitter.tsp.stores.TopicSocialProofStore.TopicSocialProof

import com.twitter.util.Future

/\*\*

\* Provides a session-less Topic Social Proof information which doesn't rely on any User Info.

\* This store is used by MemCache and In-Memory cache to achieve a higher performance.

\* One Consumer embedding and Producer embedding are used to calculate raw score.

\*/

case class TopicSocialProofStore(

representationScorerStore: ReadableStore[ScoreId, Score],

semanticCoreAnnotationStore: ReadableStore[TweetId, Seq[TopicAnnotation]]

)(

statsReceiver: StatsReceiver)

extends ReadableStore[TopicSocialProofStore.Query, Seq[TopicSocialProof]] {

import TopicSocialProofStore.\_

// Fetches the tweet's topic annotations from SemanticCore's Annotation API

override def get(query: TopicSocialProofStore.Query): Future[Option[Seq[TopicSocialProof]]] = {

StatsUtil.trackOptionStats(statsReceiver) {

for {

annotations <-

StatsUtil.trackItemsStats(statsReceiver.scope("semanticCoreAnnotationStore")) {

semanticCoreAnnotationStore.get(query.cacheableQuery.tweetId).map(\_.getOrElse(Nil))

}

filteredAnnotations = filterAnnotationsByAllowList(annotations, query)

scoredTopics <-

StatsUtil.trackItemMapStats(statsReceiver.scope("scoreTopicTweetsTweetLanguage")) {

// de-dup identical topicIds

val uniqueTopicIds = filteredAnnotations.map { annotation =>

TopicId(annotation.topicId, Some(query.cacheableQuery.tweetLanguage), country = None)

}.toSet

if (query.cacheableQuery.enableCosineSimilarityScoreCalculation) {

scoreTopicTweets(query.cacheableQuery.tweetId, uniqueTopicIds)

} else {

Future.value(uniqueTopicIds.map(id => id -> Map.empty[ScoreKey, Double]).toMap)

}

}

} yield {

if (scoredTopics.nonEmpty) {

val versionedTopicProofs = filteredAnnotations.map { annotation =>

val topicId =

TopicId(annotation.topicId, Some(query.cacheableQuery.tweetLanguage), country = None)

TopicSocialProof(

topicId,

scores = scoredTopics.getOrElse(topicId, Map.empty),

annotation.ignoreSimClustersFilter,

annotation.modelVersionId

)

}

Some(versionedTopicProofs)

} else {

None

}

}

}

}

/\*\*\*

\* When the allowList is not empty (e.g., TSP handler call, CrTopic handler call),

\* the filter will be enabled and we will only keep annotations that have versionIds existing

\* in the input allowedSemanticCoreVersionIds set.

\* But when the allowList is empty (e.g., some debugger calls),

\* we will not filter anything and pass.

\* We limit the number of versionIds to be K = MaxNumberVersionIds

\*/

private def filterAnnotationsByAllowList(

annotations: Seq[TopicAnnotation],

query: TopicSocialProofStore.Query

): Seq[TopicAnnotation] = {

val trimmedVersionIds = query.allowedSemanticCoreVersionIds.take(MaxNumberVersionIds)

annotations.filter { annotation =>

trimmedVersionIds.isEmpty || trimmedVersionIds.contains(annotation.modelVersionId)

}

}

private def scoreTopicTweets(

tweetId: TweetId,

topicIds: Set[TopicId]

): Future[Map[TopicId, Map[ScoreKey, Double]]] = {

Future.collect {

topicIds.map { topicId =>

val scoresFut = TopicTweetsCosineSimilarityAggregateStore.getRawScoresMap(

topicId,

tweetId,

TopicTweetsCosineSimilarityAggregateStore.DefaultScoreKeys,

representationScorerStore

)

topicId -> scoresFut

}.toMap

}

}

}

object TopicSocialProofStore {

private val MaxNumberVersionIds = 9

case class Query(

cacheableQuery: CacheableQuery,

allowedSemanticCoreVersionIds: Set[Long] = Set.empty) // overridden by FS

case class CacheableQuery(

tweetId: TweetId,

tweetLanguage: String,

enableCosineSimilarityScoreCalculation: Boolean = true)

case class TopicSocialProof(

topicId: TopicId,

scores: Map[ScoreKey, Double],

ignoreSimClusterFiltering: Boolean,

semanticCoreVersionId: Long)

}