package com.twitter.timelineranker.in\_network\_tweets

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.util.FutureArrow

import com.twitter.storehaus.Store

import com.twitter.timelineranker.common.\_

import com.twitter.timelineranker.core.HydratedCandidatesAndFeaturesEnvelope

import com.twitter.timelineranker.model.RecapQuery.DependencyProvider

import com.twitter.timelineranker.model.\_

import com.twitter.timelineranker.monitoring.UsersSearchResultMonitoringTransform

import com.twitter.timelineranker.parameters.in\_network\_tweets.InNetworkTweetParams

import com.twitter.timelineranker.parameters.monitoring.MonitoringParams

import com.twitter.timelineranker.parameters.recap.RecapParams

import com.twitter.timelineranker.recap.model.ContentFeatures

import com.twitter.timelineranker.util.CopyContentFeaturesIntoHydratedTweetsTransform

import com.twitter.timelineranker.util.CopyContentFeaturesIntoThriftTweetFeaturesTransform

import com.twitter.timelineranker.util.TweetFilters

import com.twitter.timelineranker.visibility.FollowGraphDataProvider

import com.twitter.timelines.clients.gizmoduck.GizmoduckClient

import com.twitter.timelines.clients.manhattan.UserMetadataClient

import com.twitter.timelines.clients.relevance\_search.SearchClient

import com.twitter.timelines.clients.tweetypie.TweetyPieClient

import com.twitter.timelines.model.TweetId

import com.twitter.timelines.util.FailOpenHandler

import com.twitter.timelines.util.stats.RequestStatsReceiver

import com.twitter.timelines.visibility.VisibilityEnforcer

import com.twitter.util.Future

class InNetworkTweetSource(

gizmoduckClient: GizmoduckClient,

searchClient: SearchClient,

searchClientForSourceTweets: SearchClient,

tweetyPieClient: TweetyPieClient,

userMetadataClient: UserMetadataClient,

followGraphDataProvider: FollowGraphDataProvider,

contentFeaturesCache: Store[TweetId, ContentFeatures],

visibilityEnforcer: VisibilityEnforcer,

statsReceiver: StatsReceiver) {

private[this] val baseScope = statsReceiver.scope("recycledTweetSource")

private[this] val requestStats = RequestStatsReceiver(baseScope)

private[this] val failOpenScope = baseScope.scope("failOpen")

private[this] val userProfileHandler = new FailOpenHandler(failOpenScope, "userProfileInfo")

private[this] val userLanguagesHandler = new FailOpenHandler(failOpenScope, "userLanguages")

private[this] val sourceTweetSearchHandler =

new FailOpenHandler(failOpenScope, "sourceTweetSearch")

private[this] val filters = TweetFilters.ValueSet(

TweetFilters.DuplicateTweets,

TweetFilters.DuplicateRetweets,

TweetFilters.TweetsFromNotFollowedUsers,

TweetFilters.NonReplyDirectedAtNotFollowedUsers

)

private[this] val hydrateReplyRootTweetProvider =

DependencyProvider.from(InNetworkTweetParams.EnableReplyRootTweetHydrationParam)

private[this] val sourceTweetsSearchResultsTransform = new SourceTweetsSearchResultsTransform(

searchClientForSourceTweets,

sourceTweetSearchHandler,

hydrateReplyRootTweetProvider = hydrateReplyRootTweetProvider,

perRequestSourceSearchClientIdProvider = DependencyProvider.None,

baseScope

)

private[this] val visibilityEnforcingTransform = new VisibilityEnforcingTransform(

visibilityEnforcer

)

private[this] val hydratedTweetsFilter = new HydratedTweetsFilterTransform(

outerFilters = filters,

innerFilters = TweetFilters.None,

useFollowGraphData = true,

useSourceTweets = true,

statsReceiver = baseScope,

numRetweetsAllowed = HydratedTweetsFilterTransform.NumDuplicateRetweetsAllowed

)

private[this] val dynamicHydratedTweetsFilter = new TweetKindOptionHydratedTweetsFilterTransform(

useFollowGraphData = true,

useSourceTweets = true,

statsReceiver = baseScope

)

private[this] val userProfileInfoTransform =

new UserProfileInfoTransform(userProfileHandler, gizmoduckClient)

private[this] val languagesTransform =

new UserLanguagesTransform(userLanguagesHandler, userMetadataClient)

private[this] def hydratesContentFeatures(

hydratedEnvelope: HydratedCandidatesAndFeaturesEnvelope

): Boolean =

hydratedEnvelope.candidateEnvelope.query.hydratesContentFeatures.getOrElse(true)

private[this] val contentFeaturesTransformer = FutureArrow.choose(

predicate = hydratesContentFeatures,

ifTrue = contentFeaturesHydrationTransform

.andThen(CopyContentFeaturesIntoThriftTweetFeaturesTransform)

.andThen(CopyContentFeaturesIntoHydratedTweetsTransform),

ifFalse = FutureArrow[

HydratedCandidatesAndFeaturesEnvelope,

HydratedCandidatesAndFeaturesEnvelope

](Future.value) // empty transformer

)

private[this] val contentFeaturesHydrationTransform =

new ContentFeaturesHydrationTransformBuilder(

tweetyPieClient = tweetyPieClient,

contentFeaturesCache = contentFeaturesCache,

enableContentFeaturesGate =

RecapQuery.paramGate(InNetworkTweetParams.EnableContentFeaturesHydrationParam),

enableTokensInContentFeaturesGate =

RecapQuery.paramGate(InNetworkTweetParams.EnableTokensInContentFeaturesHydrationParam),

enableTweetTextInContentFeaturesGate =

RecapQuery.paramGate(InNetworkTweetParams.EnableTweetTextInContentFeaturesHydrationParam),

enableConversationControlContentFeaturesGate = RecapQuery.paramGate(

InNetworkTweetParams.EnableConversationControlInContentFeaturesHydrationParam),

enableTweetMediaHydrationGate = RecapQuery.paramGate(

InNetworkTweetParams.EnableTweetMediaHydrationParam

),

hydrateInReplyToTweets = true,

statsReceiver = baseScope

).build()

private[this] val candidateGenerationTransform = new CandidateGenerationTransform(baseScope)

private[this] val maxFollowedUsersProvider =

DependencyProvider.from(InNetworkTweetParams.MaxFollowedUsersParam)

private[this] val earlybirdReturnAllResultsProvider =

DependencyProvider.from(InNetworkTweetParams.EnableEarlybirdReturnAllResultsParam)

private[this] val relevanceOptionsMaxHitsToProcessProvider =

DependencyProvider.from(InNetworkTweetParams.RelevanceOptionsMaxHitsToProcessParam)

private[this] val followGraphDataTransform =

new FollowGraphDataTransform(followGraphDataProvider, maxFollowedUsersProvider)

private[this] val enableRealGraphUsersProvider =

DependencyProvider.from(RecapParams.EnableRealGraphUsersParam)

private[this] val maxRealGraphAndFollowedUsersProvider =

DependencyProvider.from(RecapParams.MaxRealGraphAndFollowedUsersParam)

private[this] val maxRealGraphAndFollowedUsersFSOverrideProvider =

DependencyProvider.from(RecapParams.MaxRealGraphAndFollowedUsersFSOverrideParam)

private[this] val imputeRealGraphAuthorWeightsProvider =

DependencyProvider.from(RecapParams.ImputeRealGraphAuthorWeightsParam)

private[this] val imputeRealGraphAuthorWeightsPercentileProvider =

DependencyProvider.from(RecapParams.ImputeRealGraphAuthorWeightsPercentileParam)

private[this] val maxRealGraphAndFollowedUsersFromDeciderAndFS = DependencyProvider { envelope =>

maxRealGraphAndFollowedUsersFSOverrideProvider(envelope).getOrElse(

maxRealGraphAndFollowedUsersProvider(envelope))

}

private[this] val followAndRealGraphCombiningTransform = new FollowAndRealGraphCombiningTransform(

followGraphDataProvider = followGraphDataProvider,

maxFollowedUsersProvider = maxFollowedUsersProvider,

enableRealGraphUsersProvider = enableRealGraphUsersProvider,

maxRealGraphAndFollowedUsersProvider = maxRealGraphAndFollowedUsersFromDeciderAndFS,

imputeRealGraphAuthorWeightsProvider = imputeRealGraphAuthorWeightsProvider,

imputeRealGraphAuthorWeightsPercentileProvider = imputeRealGraphAuthorWeightsPercentileProvider,

statsReceiver = baseScope

)

private[this] val maxCountProvider = DependencyProvider { query =>

query.maxCount.getOrElse(query.params(InNetworkTweetParams.DefaultMaxTweetCount))

}

private[this] val maxCountWithMarginProvider = DependencyProvider { query =>

val maxCount = query.maxCount.getOrElse(query.params(InNetworkTweetParams.DefaultMaxTweetCount))

val multiplier = query.params(InNetworkTweetParams.MaxCountMultiplierParam)

(maxCount \* multiplier).toInt

}

private[this] val debugAuthorsMonitoringProvider =

DependencyProvider.from(MonitoringParams.DebugAuthorsAllowListParam)

private[this] val retrieveSearchResultsTransform = new RecapSearchResultsTransform(

searchClient = searchClient,

maxCountProvider = maxCountWithMarginProvider,

returnAllResultsProvider = earlybirdReturnAllResultsProvider,

relevanceOptionsMaxHitsToProcessProvider = relevanceOptionsMaxHitsToProcessProvider,

enableExcludeSourceTweetIdsProvider = DependencyProvider.True,

enableSettingTweetTypesWithTweetKindOptionProvider =

DependencyProvider.from(RecapParams.EnableSettingTweetTypesWithTweetKindOption),

perRequestSearchClientIdProvider = DependencyProvider.None,

statsReceiver = baseScope,

logSearchDebugInfo = false

)

private[this] val preTruncateSearchResultsTransform =

new UsersSearchResultMonitoringTransform(

name = "RecapSearchResultsTruncationTransform",

new RecapSearchResultsTruncationTransform(

extraSortBeforeTruncationGate = DependencyProvider.True,

maxCountProvider = maxCountWithMarginProvider,

statsReceiver = baseScope.scope("afterSearchResultsTransform")

),

baseScope.scope("afterSearchResultsTransform"),

debugAuthorsMonitoringProvider

)

private[this] val finalTruncationTransform = new UsersSearchResultMonitoringTransform(

name = "RecapSearchResultsTruncationTransform",

new RecapSearchResultsTruncationTransform(

extraSortBeforeTruncationGate = DependencyProvider.True,

maxCountProvider = maxCountProvider,

statsReceiver = baseScope.scope("finalTruncation")

),

baseScope.scope("finalTruncation"),

debugAuthorsMonitoringProvider

)

// Fetch source tweets based on search results present in the envelope

// and hydrate them.

private[this] val fetchAndHydrateSourceTweets =

sourceTweetsSearchResultsTransform

.andThen(SourceTweetHydrationTransform)

// Hydrate candidate tweets and fetch source tweets in parallel

private[this] val hydrateTweetsAndSourceTweetsInParallel =

new HydrateTweetsAndSourceTweetsInParallelTransform(

candidateTweetHydration = CandidateTweetHydrationTransform,

sourceTweetHydration = fetchAndHydrateSourceTweets

)

private[this] val trimToMatchSearchResultsTransform = new TrimToMatchSearchResultsTransform(

hydrateReplyRootTweetProvider = hydrateReplyRootTweetProvider,

statsReceiver = baseScope

)

private[this] val hydrationAndFilteringPipeline =

CreateCandidateEnvelopeTransform // Create empty CandidateEnvelope

.andThen(followGraphDataTransform) // Fetch follow graph data

.andThen(followAndRealGraphCombiningTransform) // Experiment: expand seed author set

.andThen(retrieveSearchResultsTransform) // Fetch search results

.andThen(

preTruncateSearchResultsTransform

) // truncate the search result up to maxCount + some margin, preserving the random tweet

.andThen(SearchResultDedupAndSortingTransform) // dedups, and sorts reverse-chron

.andThen(hydrateTweetsAndSourceTweetsInParallel) // candidates + source tweets in parallel

.andThen(visibilityEnforcingTransform) // filter hydrated tweets to visible ones

.andThen(hydratedTweetsFilter) // filter hydrated tweets based on predefined filter

.andThen(dynamicHydratedTweetsFilter) // filter hydrated tweets based on query TweetKindOption

.andThen(TrimToMatchHydratedTweetsTransform) // trim searchResult to match with hydratedTweets

.andThen(

finalTruncationTransform

) // truncate the searchResult to exactly up to maxCount, preserving the random tweet

.andThen(

trimToMatchSearchResultsTransform

) // trim other fields to match with the final searchResult

// runs the main pipeline in parallel with fetching user profile info and user languages

private[this] val featureHydrationDataTransform = new FeatureHydrationDataTransform(

hydrationAndFilteringPipeline,

languagesTransform,

userProfileInfoTransform

)

private[this] val featureHydrationPipeline =

featureHydrationDataTransform

.andThen(InNetworkTweetsSearchFeaturesHydrationTransform)

.andThen(contentFeaturesTransformer)

.andThen(candidateGenerationTransform)

def get(query: RecapQuery): Future[CandidateTweetsResult] = {

requestStats.addEventStats {

featureHydrationPipeline(query)

}

}

def get(queries: Seq[RecapQuery]): Future[Seq[CandidateTweetsResult]] = {

Future.collect(queries.map(get))

}

}