package com.twitter.search.earlybird.search;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import com.twitter.common.util.Clock;

import com.twitter.search.common.constants.thriftjava.ThriftLanguage;

import com.twitter.search.common.features.thrift.ThriftSearchResultFeatures;

import com.twitter.search.common.schema.base.ImmutableSchemaInterface;

import com.twitter.search.common.schema.earlybird.EarlybirdCluster;

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

import com.twitter.search.common.search.EarlyTerminationState;

import com.twitter.search.common.util.LongIntConverter;

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

import com.twitter.search.earlybird.common.userupdates.UserTable;

import com.twitter.search.earlybird.stats.EarlybirdSearcherStats;

import com.twitter.search.earlybird.thrift.ThriftSearchResultMetadata;

import com.twitter.search.earlybird.thrift.ThriftSearchResultMetadataOptions;

import com.twitter.search.earlybird.thrift.ThriftSearchResultType;

/\*\*

\* This class collects results for Recency queries for delegation to collectors based on query mode

\*/

public class SearchResultsCollector

extends AbstractResultsCollector<SearchRequestInfo, SimpleSearchResults> {

private static final EarlyTerminationState TERMINATED\_COLLECTED\_ENOUGH\_RESULTS =

new EarlyTerminationState("terminated\_collected\_enough\_results", true);

protected final List<Hit> results;

private final Set<Integer> requestedFeatureIds;

private final EarlybirdCluster cluster;

private final UserTable userTable;

public SearchResultsCollector(

ImmutableSchemaInterface schema,

SearchRequestInfo searchRequestInfo,

Clock clock,

EarlybirdSearcherStats searcherStats,

EarlybirdCluster cluster,

UserTable userTable,

int requestDebugMode) {

super(schema, searchRequestInfo, clock, searcherStats, requestDebugMode);

results = new ArrayList<>();

this.cluster = cluster;

this.userTable = userTable;

ThriftSearchResultMetadataOptions options =

searchRequestInfo.getSearchQuery().getResultMetadataOptions();

if (options != null && options.isReturnSearchResultFeatures()) {

requestedFeatureIds = schema.getSearchFeatureSchema().getEntries().keySet();

} else if (options != null && options.isSetRequestedFeatureIDs()) {

requestedFeatureIds = new HashSet<>(options.getRequestedFeatureIDs());

} else {

requestedFeatureIds = null;

}

}

@Override

public void startSegment() throws IOException {

featuresRequested = requestedFeatureIds != null;

}

@Override

public void doCollect(long tweetID) throws IOException {

Hit hit = new Hit(currTimeSliceID, tweetID);

ThriftSearchResultMetadata metadata =

new ThriftSearchResultMetadata(ThriftSearchResultType.RECENCY)

.setPenguinVersion(EarlybirdConfig.getPenguinVersionByte());

// Set tweet language in metadata

ThriftLanguage thriftLanguage = ThriftLanguage.findByValue(

(int) documentFeatures.getFeatureValue(EarlybirdFieldConstant.LANGUAGE));

metadata.setLanguage(thriftLanguage);

// Check and collect hit attribution data, if it's available.

fillHitAttributionMetadata(metadata);

// Set the nullcast flag in metadata

metadata.setIsNullcast(documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_NULLCAST\_FLAG));

if (searchRequestInfo.isCollectConversationId()) {

long conversationId =

documentFeatures.getFeatureValue(EarlybirdFieldConstant.CONVERSATION\_ID\_CSF);

if (conversationId != 0) {

ensureExtraMetadataIsSet(metadata);

metadata.getExtraMetadata().setConversationId(conversationId);

}

}

fillResultGeoLocation(metadata);

collectRetweetAndReplyMetadata(metadata);

long fromUserId = documentFeatures.getFeatureValue(EarlybirdFieldConstant.FROM\_USER\_ID\_CSF);

if (requestedFeatureIds != null) {

ThriftSearchResultFeatures features = documentFeatures.getSearchResultFeatures(

getSchema(), requestedFeatureIds::contains);

ensureExtraMetadataIsSet(metadata);

metadata.getExtraMetadata().setFeatures(features);

metadata.setFromUserId(fromUserId);

if (documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_CARD\_FLAG)) {

metadata.setCardType(

(byte) documentFeatures.getFeatureValue(EarlybirdFieldConstant.CARD\_TYPE\_CSF\_FIELD));

}

}

if (searchRequestInfo.isGetFromUserId()) {

metadata.setFromUserId(fromUserId);

}

collectExclusiveConversationAuthorId(metadata);

collectFacets(metadata);

collectFeatures(metadata);

collectIsProtected(metadata, cluster, userTable);

hit.setMetadata(metadata);

results.add(hit);

updateHitCounts(tweetID);

}

private final void collectRetweetAndReplyMetadata(ThriftSearchResultMetadata metadata)

throws IOException {

if (searchRequestInfo.isGetInReplyToStatusId() || searchRequestInfo.isGetReferenceAuthorId()) {

boolean isRetweet = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_RETWEET\_FLAG);

boolean isReply = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_REPLY\_FLAG);

// Set the isRetweet and isReply metadata so that clients who request retweet and reply

// metadata know whether a result is a retweet or reply or neither.

metadata.setIsRetweet(isRetweet);

metadata.setIsReply(isReply);

// Only store the shared status id if the hit is a reply or a retweet and

// the getInReplyToStatusId flag is set.

if (searchRequestInfo.isGetInReplyToStatusId() && (isReply || isRetweet)) {

long sharedStatusID =

documentFeatures.getFeatureValue(EarlybirdFieldConstant.SHARED\_STATUS\_ID\_CSF);

if (sharedStatusID != 0) {

metadata.setSharedStatusId(sharedStatusID);

}

}

// Only store the reference tweet author ID if the hit is a reply or a retweet and the

// getReferenceAuthorId flag is set.

if (searchRequestInfo.isGetReferenceAuthorId() && (isReply || isRetweet)) {

// the REFERENCE\_AUTHOR\_ID\_CSF stores the source tweet author id for all retweets

long referenceAuthorId =

documentFeatures.getFeatureValue(EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_CSF);

if (referenceAuthorId != 0) {

metadata.setReferencedTweetAuthorId(referenceAuthorId);

} else if (cluster != EarlybirdCluster.FULL\_ARCHIVE) {

// we also store the reference author id for retweets, directed at tweets, and self

// threaded tweets separately on Realtime/Protected Earlybirds. This data will be moved to

// the REFERENCE\_AUTHOR\_ID\_CSF and these fields will be deprecated in SEARCH-34958.

referenceAuthorId = LongIntConverter.convertTwoIntToOneLong(

(int) documentFeatures.getFeatureValue(

EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_MOST\_SIGNIFICANT\_INT),

(int) documentFeatures.getFeatureValue(

EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_LEAST\_SIGNIFICANT\_INT));

if (referenceAuthorId > 0) {

metadata.setReferencedTweetAuthorId(referenceAuthorId);

}

}

}

}

}

/\*\*

\* This differs from base class because we check against num results collected instead of

\* num hits collected.

\*/

@Override

public EarlyTerminationState innerShouldCollectMore() throws IOException {

if (results.size() >= searchRequestInfo.getNumResultsRequested()) {

collectedEnoughResults();

if (shouldTerminate()) {

return setEarlyTerminationState(TERMINATED\_COLLECTED\_ENOUGH\_RESULTS);

}

}

return EarlyTerminationState.COLLECTING;

}

@Override

public SimpleSearchResults doGetResults() {

// Sort hits by tweet id.

Collections.sort(results);

return new SimpleSearchResults(results);

}

}