package com.twitter.search.ingester.pipeline.twitter;

import java.net.MalformedURLException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Set;

import javax.naming.NamingException;

import com.google.common.collect.Maps;

import org.apache.commons.pipeline.StageException;

import org.apache.commons.pipeline.stage.StageTimer;

import org.apache.commons.pipeline.validation.ConsumedTypes;

import org.apache.commons.pipeline.validation.ProducesConsumed;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.text.language.LocaleUtil;

import com.twitter.expandodo.thriftjava.Card2;

import com.twitter.mediaservices.commons.tweetmedia.thrift\_java.MediaInfo;

import com.twitter.search.common.indexing.thriftjava.ThriftExpandedUrl;

import com.twitter.search.common.metrics.SearchRateCounter;

import com.twitter.search.ingester.model.IngesterTwitterMessage;

import com.twitter.search.ingester.pipeline.util.BatchingClient;

import com.twitter.search.ingester.pipeline.util.CardFieldUtil;

import com.twitter.search.ingester.pipeline.util.IngesterStageTimer;

import com.twitter.search.ingester.pipeline.util.ResponseNotReturnedException;

import com.twitter.spiderduck.common.URLUtils;

import com.twitter.tweetypie.thriftjava.GetTweetOptions;

import com.twitter.tweetypie.thriftjava.GetTweetResult;

import com.twitter.tweetypie.thriftjava.GetTweetsRequest;

import com.twitter.tweetypie.thriftjava.MediaEntity;

import com.twitter.tweetypie.thriftjava.StatusState;

import com.twitter.tweetypie.thriftjava.Tweet;

import com.twitter.tweetypie.thriftjava.TweetService;

import com.twitter.util.Function;

import com.twitter.util.Future;

@ConsumedTypes(IngesterTwitterMessage.class)

@ProducesConsumed

public class RetrieveCardBatchedStage extends TwitterBaseStage

<IngesterTwitterMessage, IngesterTwitterMessage> {

private static final Logger LOG = LoggerFactory.getLogger(RetrieveCardBatchedStage.class);

private static final String CARDS\_PLATFORM\_KEY = "iPhone-13";

private int batchSize = 10;

private SearchRateCounter totalTweets;

private SearchRateCounter tweetsWithCards;

private SearchRateCounter tweetsWithoutCards;

private SearchRateCounter tweetsWithAnimatedGifMediaInfo;

private SearchRateCounter cardsWithName;

private SearchRateCounter cardsWithDomain;

private SearchRateCounter cardsWithTitles;

private SearchRateCounter cardsWithDescriptions;

private SearchRateCounter cardsWithUnknownLanguage;

private SearchRateCounter tweetsNotFound;

private SearchRateCounter malformedUrls;

private SearchRateCounter urlMismatches;

private SearchRateCounter cardExceptions;

private SearchRateCounter cardExceptionTweets;

private StageTimer retrieveCardsTimer;

private String cardNamePrefix;

// Since there is only one thread executing this stage (although that could potentially be

// changed in the pipeline config), no need to be thread safe.

private static final Map<String, SearchRateCounter> CARD\_NAME\_STATS = new HashMap<>();

private static TweetService.ServiceToClient tweetyPieService;

private BatchingClient<Long, Card2> cardsClient;

private String tweetypieClientId = null;

// Can be overridden in the corresponding pipeline-ingester.\*.xml config.

// By default protected tweets are filtered out.

// Only in the protected ingester pipeline is this set to false.

private boolean filterProtected = true;

@Override

public void initStats() {

super.initStats();

cardNamePrefix = getStageNamePrefix() + "\_card\_name\_";

totalTweets = SearchRateCounter.export(getStageNamePrefix() + "\_total\_tweets");

tweetsWithCards = SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_with\_cards");

tweetsWithoutCards = SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_without\_cards");

tweetsWithAnimatedGifMediaInfo =

SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_with\_animated\_gif\_media\_info");

cardsWithName = SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_with\_card\_name");

cardsWithDomain = SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_with\_card\_domain");

cardsWithTitles = SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_with\_card\_titles");

cardsWithDescriptions =

SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_with\_card\_descriptions");

cardsWithUnknownLanguage =

SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_with\_unknown\_card\_lanuage");

tweetsNotFound = SearchRateCounter.export(getStageNamePrefix() + "\_tweets\_not\_found");

malformedUrls = SearchRateCounter.export(getStageNamePrefix() + "\_malformed\_urls");

urlMismatches = SearchRateCounter.export(getStageNamePrefix() + "\_url\_mismatches");

cardExceptions = SearchRateCounter.export(getStageNamePrefix() + "\_card\_exceptions");

cardExceptionTweets =

SearchRateCounter.export(getStageNamePrefix() + "\_card\_exception\_tweets");

retrieveCardsTimer = new IngesterStageTimer(getStageNamePrefix() + "\_request\_timer");

}

@Override

protected void doInnerPreprocess() throws StageException, NamingException {

super.doInnerPreprocess();

tweetyPieService = wireModule.getTweetyPieClient(tweetypieClientId);

cardsClient = new BatchingClient<>(this::batchRetrieveURLs, batchSize);

}

@Override

public void innerProcess(Object obj) throws StageException {

if (!(obj instanceof IngesterTwitterMessage)) {

throw new StageException(this,

"Received object of incorrect type: " + obj.getClass().getName());

}

IngesterTwitterMessage message = (IngesterTwitterMessage) obj;

cardsClient.call(message.getTweetId())

.onSuccess(Function.cons(card -> {

updateMessage(message, card);

emitAndCount(message);

}))

.onFailure(Function.cons(exception -> {

if (!(exception instanceof ResponseNotReturnedException)) {

cardExceptionTweets.increment();

}

emitAndCount(message);

}));

}

private Future<Map<Long, Card2>> batchRetrieveURLs(Set<Long> keys) {

retrieveCardsTimer.start();

totalTweets.increment(keys.size());

GetTweetOptions options = new GetTweetOptions()

.setInclude\_cards(true)

.setCards\_platform\_key(CARDS\_PLATFORM\_KEY)

.setBypass\_visibility\_filtering(!filterProtected);

GetTweetsRequest request = new GetTweetsRequest()

.setOptions(options)

.setTweet\_ids(new ArrayList<>(keys));

return tweetyPieService.get\_tweets(request)

.onFailure(throwable -> {

cardExceptions.increment();

LOG.error("TweetyPie server threw an exception while requesting tweetIds: "

+ request.getTweet\_ids(), throwable);

return null;

})

.map(this::createIdToCardMap);

}

private void updateMessage(IngesterTwitterMessage message, Card2 card) {

tweetsWithCards.increment();

String cardName = card.getName().toLowerCase();

addCardNameToStats(cardName);

message.setCardName(cardName);

cardsWithName.increment();

message.setCardUrl(card.getUrl());

String url = getLastHop(message, card.getUrl());

if (url != null) {

try {

String domain = URLUtils.getDomainFromURL(url);

message.setCardDomain(domain.toLowerCase());

cardsWithDomain.increment();

} catch (MalformedURLException e) {

malformedUrls.increment();

if (LOG.isDebugEnabled()) {

LOG.debug("Tweet ID {} has a malformed card last hop URL: {}", message.getId(), url);

}

}

} else {

// This happens with retweet. Basically when retrieve card for a retweet, we

// get a card associated with the original tweet, so the tco won't match.

// As of Sep 2014, this seems to be the intended behavior and has been running

// like this for over a year.

urlMismatches.increment();

}

message.setCardTitle(

CardFieldUtil.extractBindingValue(CardFieldUtil.TITLE\_BINDING\_KEY, card));

if (message.getCardTitle() != null) {

cardsWithTitles.increment();

}

message.setCardDescription(

CardFieldUtil.extractBindingValue(CardFieldUtil.DESCRIPTION\_BINDING\_KEY, card));

if (message.getCardDescription() != null) {

cardsWithDescriptions.increment();

}

CardFieldUtil.deriveCardLang(message);

if (LocaleUtil.UNKNOWN.getLanguage().equals(message.getCardLang())) {

cardsWithUnknownLanguage.increment();

}

}

private Map<Long, Card2> createIdToCardMap(List<GetTweetResult> listResult) {

Map<Long, Card2> responseMap = Maps.newHashMap();

for (GetTweetResult entry : listResult) {

if (entry.isSetTweet()

&& entry.isSetTweet\_state()

&& (entry.getTweet\_state() == StatusState.FOUND)) {

long id = entry.getTweet\_id();

if (entry.getTweet().isSetCard2()) {

responseMap.put(id, entry.getTweet().getCard2());

} else {

// Short-term fix for removal of animated GIF cards --

// if the tweet contains an animated GIF, create a card based on media entity data

Card2 card = createCardForAnimatedGif(entry.getTweet());

if (card != null) {

responseMap.put(id, card);

tweetsWithAnimatedGifMediaInfo.increment();

} else {

tweetsWithoutCards.increment();

}

}

} else {

tweetsNotFound.increment();

}

}

return responseMap;

}

private Card2 createCardForAnimatedGif(Tweet tweet) {

if (tweet.getMediaSize() > 0) {

for (MediaEntity mediaEntity : tweet.getMedia()) {

MediaInfo mediaInfo = mediaEntity.getMedia\_info();

if (mediaInfo != null && mediaInfo.getSetField() == MediaInfo.\_Fields.ANIMATED\_GIF\_INFO) {

Card2 card = new Card2();

card.setName("animated\_gif");

// Use the original compressed URL for the media entity to match existing card URLs

card.setUrl(mediaEntity.getUrl());

card.setBinding\_values(Collections.emptyList());

return card;

}

}

}

return null;

}

// Unfortunately the url returned in the card data is not the last hop

private String getLastHop(IngesterTwitterMessage message, String url) {

if (message.getExpandedUrlMap() != null) {

ThriftExpandedUrl expanded = message.getExpandedUrlMap().get(url);

if ((expanded != null) && expanded.isSetCanonicalLastHopUrl()) {

return expanded.getCanonicalLastHopUrl();

}

}

return null;

}

// Used by commons-pipeline and set via the xml config

public void setFilterProtected(boolean filterProtected) {

LOG.info("Filtering protected tweets: {}", filterProtected);

this.filterProtected = filterProtected;

}

public void setTweetypieClientId(String tweetypieClientId) {

LOG.info("Using tweetypieClientId: {}", tweetypieClientId);

this.tweetypieClientId = tweetypieClientId;

}

public void setInternalBatchSize(int internalBatchSize) {

this.batchSize = internalBatchSize;

}

/\*\*

\* For each card name, we add a rate counter to observe what kinds of card we're actually

\* indexing, and with what rate.

\*/

private void addCardNameToStats(String cardName) {

SearchRateCounter cardNameCounter = CARD\_NAME\_STATS.get(cardName);

if (cardNameCounter == null) {

cardNameCounter = SearchRateCounter.export(cardNamePrefix + cardName);

CARD\_NAME\_STATS.put(cardName, cardNameCounter);

}

cardNameCounter.increment();

}

}