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

import java.net.URI;

import java.net.URISyntaxException;

import java.util.Collection;

import java.util.Collections;

import java.util.HashSet;

import java.util.Locale;

import java.util.Map;

import java.util.Set;

import javax.naming.NamingException;

import com.google.common.base.Preconditions;

import com.google.common.collect.Maps;

import com.google.common.collect.Sets;

import org.apache.commons.lang.StringUtils;

import org.apache.commons.pipeline.StageException;

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

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

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

import com.twitter.search.common.decider.SearchDecider;

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

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

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

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

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

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

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

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

import com.twitter.search.ingester.pipeline.wire.WireModule;

import com.twitter.service.spiderduck.gen.MediaTypes;

import com.twitter.util.Duration;

import com.twitter.util.Function;

import com.twitter.util.Future;

@ConsumedTypes(IngesterTwitterMessage.class)

@ProducesConsumed

public class ResolveCompressedUrlsBatchedStage extends TwitterBatchedBaseStage

<IngesterTwitterMessage, IngesterTwitterMessage> {

private static final int PINK\_REQUEST\_TIMEOUT\_MILLIS = 500;

private static final int PINK\_REQUEST\_RETRIES = 2;

private static final String PINK\_REQUESTS\_BATCH\_SIZE\_DECIDER\_KEY = "pink\_requests\_batch\_size";

private AsyncPinkUrlsResolver urlResolver;

private int resolveUrlPercentage = 100;

private String pinkClientId;

private SearchDecider searchDecider;

// The number of URLs that we attempted to resolve.

private SearchRateCounter linksAttempted;

// The number of URLs that were successfully resolved.

private SearchRateCounter linksSucceeded;

// The number of URLs ignored because they are too long.

private SearchRateCounter linksTooLong;

// The number of URLs truncated because they are too long.

private SearchRateCounter linksTruncated;

// The number of resolved URLs without a media type.

private SearchRateCounter urlsWithoutMediaType;

// The number of resolved URLs with a specific media type.

private final Map<MediaTypes, SearchRateCounter> urlsWithMediaTypeMap =

Maps.newEnumMap(MediaTypes.class);

// The number of tweets for which all URLs were resolved.

private SearchRateCounter tweetsWithResolvedURLs;

// The number of tweets for which some URLs were not resolved.

private SearchRateCounter tweetsWithUnresolvedURLs;

// How long it takes to fully resolve all URLs in a tweet.

private Percentile<Long> millisToResolveAllTweetURLs;

// max age that a tweet can be before passed down the pipeline

private long tweetMaxAgeToResolve;

// number of times an element is within quota.

private SearchRateCounter numberOfElementsWithinQuota;

// number of times element is not within quota. If element not within quota, we dont batch.

private SearchRateCounter numberOfElementsNotWithinQuota;

// number of times element has urls.

private SearchRateCounter numberOfElementsWithUrls;

// number of times element does not have urls. If element does not have URL, we dont batch.

private SearchRateCounter numberOfElementsWithoutUrls;

// number of calls to needsToBeBatched method.

private SearchRateCounter numberOfCallsToNeedsToBeBatched;

public void setTweetMaxAgeToResolve(long tweetMaxAgeToResolve) {

this.tweetMaxAgeToResolve = tweetMaxAgeToResolve;

}

@Override

protected Class<IngesterTwitterMessage> getQueueObjectType() {

return IngesterTwitterMessage.class;

}

@Override

protected boolean needsToBeBatched(IngesterTwitterMessage element) {

numberOfCallsToNeedsToBeBatched.increment();

boolean isWithinQuota = (element.getId() % 100) < resolveUrlPercentage;

if (isWithinQuota) {

this.numberOfElementsWithinQuota.increment();

} else {

this.numberOfElementsNotWithinQuota.increment();

}

boolean hasUrls = !element.getExpandedUrlMap().isEmpty();

if (hasUrls) {

this.numberOfElementsWithUrls.increment();

} else {

this.numberOfElementsWithoutUrls.increment();

}

return hasUrls && isWithinQuota;

}

// Identity transformation. T and U types are the same

@Override

protected IngesterTwitterMessage transform(IngesterTwitterMessage element) {

return element;

}

@Override

public void initStats() {

super.initStats();

commonInnerSetupStats();

}

@Override

protected void innerSetupStats() {

super.innerSetupStats();

commonInnerSetupStats();

}

private void commonInnerSetupStats() {

linksAttempted = RelevanceStats.exportRate(getStageNamePrefix() + "\_num\_links\_attempted");

linksSucceeded = RelevanceStats.exportRate(getStageNamePrefix() + "\_num\_links\_succeeded");

linksTooLong = RelevanceStats.exportRate(getStageNamePrefix() + "\_num\_links\_toolong");

linksTruncated = RelevanceStats.exportRate(getStageNamePrefix() + "\_num\_links\_truncated");

urlsWithoutMediaType = RelevanceStats.exportRate(

getStageNamePrefix() + "\_urls\_without\_media\_type");

for (MediaTypes mediaType : MediaTypes.values()) {

urlsWithMediaTypeMap.put(

mediaType,

RelevanceStats.exportRate(

getStageNamePrefix() + "\_urls\_with\_media\_type\_" + mediaType.name().toLowerCase()));

}

tweetsWithResolvedURLs = RelevanceStats.exportRate(

getStageNamePrefix() + "\_num\_tweets\_with\_resolved\_urls");

tweetsWithUnresolvedURLs = RelevanceStats.exportRate(

getStageNamePrefix() + "\_num\_tweets\_with\_unresolved\_urls");

millisToResolveAllTweetURLs = PercentileUtil.createPercentile(

getStageNamePrefix() + "\_millis\_to\_resolve\_all\_tweet\_urls");

numberOfCallsToNeedsToBeBatched = SearchRateCounter.export(getStageNamePrefix()

+ "\_calls\_to\_needsToBeBatched");

numberOfElementsWithinQuota = SearchRateCounter.export(getStageNamePrefix()

+ "\_is\_within\_quota");

numberOfElementsNotWithinQuota = SearchRateCounter.export(getStageNamePrefix()

+ "\_is\_not\_within\_quota");

numberOfElementsWithUrls = SearchRateCounter.export(getStageNamePrefix()

+ "\_has\_urls");

numberOfElementsWithoutUrls = SearchRateCounter.export(getStageNamePrefix()

+ "\_does\_not\_have\_urls");

}

@Override

protected void doInnerPreprocess() throws StageException, NamingException {

searchDecider = new SearchDecider(decider);

// We need to call this after assigning searchDecider because our updateBatchSize function

// depends on the searchDecider.

super.doInnerPreprocess();

commonInnerSetup();

}

@Override

protected void innerSetup() throws PipelineStageException, NamingException {

searchDecider = new SearchDecider(decider);

// We need to call this after assigning searchDecider because our updateBatchSize function

// depends on the searchDecider.

super.innerSetup();

commonInnerSetup();

}

private void commonInnerSetup() throws NamingException {

Preconditions.checkNotNull(pinkClientId);

urlResolver = new AsyncPinkUrlsResolver(

WireModule

.getWireModule()

.getStorer(Duration.fromMilliseconds(PINK\_REQUEST\_TIMEOUT\_MILLIS),

PINK\_REQUEST\_RETRIES),

pinkClientId);

}

@Override

protected Future<Collection<IngesterTwitterMessage>> innerProcessBatch(Collection<BatchedElement

<IngesterTwitterMessage, IngesterTwitterMessage>> batch) {

// Batch urls

Map<String, Set<IngesterTwitterMessage>> urlToTweetsMap = createUrlToTweetMap(batch);

Set<String> urlsToResolve = urlToTweetsMap.keySet();

updateBatchSize();

linksAttempted.increment(batch.size());

// Do the lookup

return urlResolver.resolveUrls(urlsToResolve).map(processResolvedUrlsFunction(batch));

}

@Override

protected void updateBatchSize() {

// update batch based on decider

int decidedBatchSize = searchDecider.featureExists(PINK\_REQUESTS\_BATCH\_SIZE\_DECIDER\_KEY)

? searchDecider.getAvailability(PINK\_REQUESTS\_BATCH\_SIZE\_DECIDER\_KEY)

: batchSize;

setBatchedStageBatchSize(decidedBatchSize);

}

//if not all urls for a message where resolved re-enqueue until maxAge is reached

private Function<Map<String, ResolveCompressedUrlsUtils.UrlInfo>,

Collection<IngesterTwitterMessage>>

processResolvedUrlsFunction(Collection<BatchedElement<IngesterTwitterMessage,

IngesterTwitterMessage>> batch) {

return Function.func(resolvedUrls -> {

linksSucceeded.increment(resolvedUrls.size());

for (ResolveCompressedUrlsUtils.UrlInfo urlInfo : resolvedUrls.values()) {

if (urlInfo.mediaType != null) {

urlsWithMediaTypeMap.get(urlInfo.mediaType).increment();

} else {

urlsWithoutMediaType.increment();

}

}

Set<IngesterTwitterMessage> successfulTweets = Sets.newHashSet();

for (BatchedElement<IngesterTwitterMessage, IngesterTwitterMessage> batchedElement : batch) {

IngesterTwitterMessage message = batchedElement.getItem();

Set<String> tweetUrls = message.getExpandedUrlMap().keySet();

int resolvedUrlCounter = 0;

for (String url : tweetUrls) {

ResolveCompressedUrlsUtils.UrlInfo urlInfo = resolvedUrls.get(url);

// if the url didn't resolve move on to the next one, this might trigger a re-enqueue

// if the tweet is still kind of new. But we want to process the rest for when that

// is not the case and we are going to end up passing it to the next stage

if (urlInfo == null) {

continue;

}

String resolvedUrl = urlInfo.resolvedUrl;

Locale locale = urlInfo.language == null ? null

: LocaleUtil.getLocaleOf(urlInfo.language);

if (StringUtils.isNotBlank(resolvedUrl)) {

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

resolvedUrlCounter += 1;

enrichTweetWithUrlInfo(message, expandedUrl, urlInfo, locale);

}

}

long tweetMessageAge = clock.nowMillis() - message.getDate().getTime();

if (resolvedUrlCounter == tweetUrls.size()) {

millisToResolveAllTweetURLs.record(tweetMessageAge);

tweetsWithResolvedURLs.increment();

successfulTweets.add(message);

} else if (tweetMessageAge > tweetMaxAgeToResolve) {

tweetsWithUnresolvedURLs.increment();

successfulTweets.add(message);

} else {

//re-enqueue if all urls weren't resolved and the tweet is younger than maxAge

reEnqueueAndRetry(batchedElement);

}

}

return successfulTweets;

});

}

private Map<String, Set<IngesterTwitterMessage>> createUrlToTweetMap(

Collection<BatchedElement<IngesterTwitterMessage, IngesterTwitterMessage>> batch) {

Map<String, Set<IngesterTwitterMessage>> urlToTweetsMap = Maps.newHashMap();

for (BatchedElement<IngesterTwitterMessage, IngesterTwitterMessage> batchedElement : batch) {

IngesterTwitterMessage message = batchedElement.getItem();

for (String originalUrl : message.getExpandedUrlMap().keySet()) {

Set<IngesterTwitterMessage> messages = urlToTweetsMap.get(originalUrl);

if (messages == null) {

messages = new HashSet<>();

urlToTweetsMap.put(originalUrl, messages);

}

messages.add(message);

}

}

return Collections.unmodifiableMap(urlToTweetsMap);

}

// enrich the twitterMessage with the resolvedCounter Urls.

private void enrichTweetWithUrlInfo(IngesterTwitterMessage message,

ThriftExpandedUrl expandedUrl,

ResolveCompressedUrlsUtils.UrlInfo urlInfo,

Locale locale) {

String truncatedUrl = maybeTruncate(urlInfo.resolvedUrl);

if (truncatedUrl == null) {

return;

}

expandedUrl.setCanonicalLastHopUrl(truncatedUrl);

if (urlInfo.mediaType != null) {

// Overwrite url media type with media type from resolved url only if the media type from

// resolved url is not Unknown

if (!expandedUrl.isSetMediaType() || urlInfo.mediaType != MediaTypes.UNKNOWN) {

expandedUrl.setMediaType(urlInfo.mediaType);

}

}

if (urlInfo.linkCategory != null) {

expandedUrl.setLinkCategory(urlInfo.linkCategory);

}

// Note that if there are multiple links in one tweet message, the language of the

// link that got examined later in this for loop will overwrite the values that were

// written before. This is not an optimal design but considering most tweets have

// only one link, or same-language links, this shouldn't be a big issue.

if (locale != null) {

message.setLinkLocale(locale);

}

if (urlInfo.description != null) {

expandedUrl.setDescription(urlInfo.description);

}

if (urlInfo.title != null) {

expandedUrl.setTitle(urlInfo.title);

}

}

// test methods

public void setResolveUrlPercentage(int percentage) {

this.resolveUrlPercentage = percentage;

}

public void setPinkClientId(String pinkClientId) {

this.pinkClientId = pinkClientId;

}

public static final int MAX\_URL\_LENGTH = 1000;

private String maybeTruncate(String fullUrl) {

if (fullUrl.length() <= MAX\_URL\_LENGTH) {

return fullUrl;

}

try {

URI parsed = new URI(fullUrl);

// Create a URL with an empty query and fragment.

String simplified = new URI(parsed.getScheme(),

parsed.getAuthority(),

parsed.getPath(),

null,

null).toString();

if (simplified.length() < MAX\_URL\_LENGTH) {

linksTruncated.increment();

return simplified;

}

} catch (URISyntaxException e) {

}

linksTooLong.increment();

return null;

}

}