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

import java.util.List;

import java.util.Map;

import java.util.Set;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.decider.Decider;

import com.twitter.pink\_floyd.thrift.ClientIdentifier;

import com.twitter.pink\_floyd.thrift.Mask;

import com.twitter.pink\_floyd.thrift.Storer;

import com.twitter.pink\_floyd.thrift.UrlData;

import com.twitter.pink\_floyd.thrift.UrlReadRequest;

import com.twitter.pink\_floyd.thrift.UrlReadResponse;

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

import com.twitter.util.Await;

import com.twitter.util.Future;

import com.twitter.util.Throw;

import com.twitter.util.Throwables;

import com.twitter.util.Try;

import static com.twitter.search.ingester.pipeline.twitter.ResolveCompressedUrlsUtils.getUrlInfo;

/\*\*

\* Resolve compressed URL via Pink

\*/

public class ResolveCompressedUrlsPink {

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

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

private final Storer.ServiceIface storerClient;

private final ClientIdentifier pinkClientId;

private final Mask requestMask;

private final SearchDecider decider;

// Use ServerSet to construct a metadata store client

public ResolveCompressedUrlsPink(Storer.ServiceIface storerClient,

String pinkClientId,

Decider decider) {

this.storerClient = storerClient;

this.pinkClientId = ClientIdentifier.valueOf(pinkClientId);

this.decider = new SearchDecider(Preconditions.checkNotNull(decider));

requestMask = new Mask();

requestMask.setResolution(true);

requestMask.setHtmlBasics(true);

requestMask.setUrlDirectInfo(true);

}

/\*\*

\* Resolve a set of URLs using PinkFloyd.

\*/

public Map<String, ResolveCompressedUrlsUtils.UrlInfo> resolveUrls(Set<String> urls) {

if (urls == null || urls.size() == 0) {

return null;

}

List<String> urlsList = ImmutableList.copyOf(urls);

int batchSize = decider.featureExists(PINK\_REQUESTS\_BATCH\_SIZE\_DECIDER\_KEY)

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

: 10000;

int numRequests = (int) Math.ceil(1.0 \* urlsList.size() / batchSize);

List<Future<UrlReadResponse>> responseFutures = Lists.newArrayList();

for (int i = 0; i < numRequests; ++i) {

UrlReadRequest request = new UrlReadRequest();

request.setUrls(

urlsList.subList(i \* batchSize, Math.min(urlsList.size(), (i + 1) \* batchSize)));

request.setMask(requestMask);

request.setClientId(pinkClientId);

// Send all requests in parallel.

responseFutures.add(storerClient.read(request));

}

Map<String, ResolveCompressedUrlsUtils.UrlInfo> resultMap = Maps.newHashMap();

for (Future<UrlReadResponse> responseFuture : responseFutures) {

Try<UrlReadResponse> tryResponse = getResponseTry(responseFuture);

if (tryResponse.isThrow()) {

continue;

}

UrlReadResponse response = tryResponse.get();

for (UrlData urlData : response.getData()) {

if (ResolveCompressedUrlsUtils.isResolved(urlData)) {

resultMap.put(urlData.url, getUrlInfo(urlData));

}

}

}

return resultMap;

}

private Try<UrlReadResponse> getResponseTry(Future<UrlReadResponse> responseFuture) {

try {

Try<UrlReadResponse> tryResponse = Await.result(responseFuture.liftToTry());

if (tryResponse.isThrow()) {

Throwable throwable = ((Throw) tryResponse).e();

LOG.warn("Failed to resolve URLs with Pink Storer.", throwable);

}

return tryResponse;

} catch (Exception e) {

return Throwables.unchecked(e);

}

}

}