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

import java.util.ArrayList;

import java.util.Collection;

import java.util.Iterator;

import java.util.List;

import java.util.Optional;

import com.google.common.base.Preconditions;

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

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

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

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

import com.twitter.search.common.relevance.entities.GeoObject;

import com.twitter.search.common.util.geocoding.ManhattanGeocodeRecordStore;

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

import com.twitter.stitch.Stitch;

import com.twitter.storage.client.manhattan.kv.JavaManhattanKVEndpoint;

import com.twitter.storage.client.manhattan.kv.ManhattanValue;

import com.twitter.util.Function;

import com.twitter.util.Future;

public final class ManhattanCodedLocationProvider {

private final ManhattanGeocodeRecordStore store;

private final SearchCounter locationsCounter;

private static final String LOCATIONS\_POPULATED\_STAT\_NAME = "\_locations\_populated\_count";

public static ManhattanCodedLocationProvider createWithEndpoint(

JavaManhattanKVEndpoint endpoint, String metricsPrefix, String datasetName) {

return new ManhattanCodedLocationProvider(

ManhattanGeocodeRecordStore.create(endpoint, datasetName), metricsPrefix);

}

private ManhattanCodedLocationProvider(ManhattanGeocodeRecordStore store, String metricPrefix) {

this.locationsCounter = SearchCounter.export(metricPrefix + LOCATIONS\_POPULATED\_STAT\_NAME);

this.store = store;

}

/\*\*

\* Iterates through all given messages, and for each message that has a location set, retrieves

\* the coordinates of that location from Manhattan and sets them back on that message.

\*/

public Future<Collection<IngesterTwitterMessage>> populateCodedLatLon(

Collection<IngesterTwitterMessage> messages) {

if (messages.isEmpty()) {

return Future.value(messages);

}

// Batch read requests

List<Stitch<Optional<ManhattanValue<ThriftGeocodeRecord>>>> readRequests =

new ArrayList<>(messages.size());

for (IngesterTwitterMessage message : messages) {

readRequests.add(store.asyncReadFromManhattan(message.getLocation()));

}

Future<List<Optional<ManhattanValue<ThriftGeocodeRecord>>>> batchedRequest =

Stitch.run(Stitch.collect(readRequests));

return batchedRequest.map(Function.func(optGeoLocations -> {

// Iterate over messages and responses simultaneously

Preconditions.checkState(messages.size() == optGeoLocations.size());

Iterator<IngesterTwitterMessage> messageIterator = messages.iterator();

Iterator<Optional<ManhattanValue<ThriftGeocodeRecord>>> optGeoLocationIterator =

optGeoLocations.iterator();

while (messageIterator.hasNext() && optGeoLocationIterator.hasNext()) {

IngesterTwitterMessage message = messageIterator.next();

Optional<ManhattanValue<ThriftGeocodeRecord>> optGeoLocation =

optGeoLocationIterator.next();

if (setGeoLocationForMessage(message, optGeoLocation)) {

locationsCounter.increment();

}

}

return messages;

}));

}

/\*\*

\* Returns whether a valid geolocation was successfully found and saved in the message.

\*/

private boolean setGeoLocationForMessage(

IngesterTwitterMessage message,

Optional<ManhattanValue<ThriftGeocodeRecord>> optGeoLocation) {

if (optGeoLocation.isPresent()) {

ThriftGeocodeRecord geoLocation = optGeoLocation.get().contents();

ThriftGeoPoint geoTags = geoLocation.getGeoPoint();

if ((geoTags.getLatitude() == GeoObject.DOUBLE\_FIELD\_NOT\_PRESENT)

&& (geoTags.getLongitude() == GeoObject.DOUBLE\_FIELD\_NOT\_PRESENT)) {

// This case indicates that we have "negative cache" in coded\_locations table, so

// don't try to geocode again.

message.setUncodeableLocation();

return false;

} else {

GeoObject code = new GeoObject(

geoTags.getLatitude(),

geoTags.getLongitude(),

geoTags.getAccuracy(),

ThriftGeoLocationSource.USER\_PROFILE);

message.setGeoLocation(code);

return true;

}

} else {

message.setGeocodeRequired();

return false;

}

}

}