package com.twitter.search.earlybird.partition;

import com.google.common.annotations.VisibleForTesting;

import org.apache.kafka.clients.consumer.ConsumerRecord;

import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

import com.twitter.search.common.util.io.kafka.FinagleKafkaClientUtils;

import com.twitter.search.common.util.io.kafka.ThriftDeserializer;

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

import com.twitter.search.earlybird.exception.MissingKafkaTopicException;

import com.twitter.tweetypie.thriftjava.TweetEvent;

import com.twitter.tweetypie.thriftjava.UserScrubGeoEvent;

public class UserScrubGeoEventStreamIndexer extends SimpleStreamIndexer<Long, TweetEvent> {

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

protected static String kafkaClientId = "earlybird\_user\_scrub\_geo\_kafka\_consumer";

private static final SearchCounter NUM\_MISSING\_DATA\_ERRORS =

SearchCounter.export("num\_user\_scrub\_geo\_event\_kafka\_consumer\_num\_missing\_data\_errors");

private final SegmentManager segmentManager;

private final SearchIndexingMetricSet searchIndexingMetricSet;

public UserScrubGeoEventStreamIndexer(KafkaConsumer<Long, TweetEvent> kafkaConsumer,

String topic,

SearchIndexingMetricSet searchIndexingMetricSet,

SegmentManager segmentManager)

throws MissingKafkaTopicException {

super(kafkaConsumer, topic);

this.segmentManager = segmentManager;

this.searchIndexingMetricSet = searchIndexingMetricSet;

indexingSuccesses = SearchRateCounter.export("user\_scrub\_geo\_indexing\_successes");

indexingFailures = SearchRateCounter.export("user\_scrub\_geo\_indexing\_failures");

}

/\*\*

\* Provides UserScrubGeoEvent Kafka Consumer to EarlybirdWireModule.

\* @return

\*/

public static KafkaConsumer<Long, TweetEvent> provideKafkaConsumer() {

return FinagleKafkaClientUtils.newKafkaConsumerForAssigning(

EarlybirdProperty.TWEET\_EVENTS\_KAFKA\_PATH.get(),

new ThriftDeserializer<>(TweetEvent.class),

kafkaClientId,

MAX\_POLL\_RECORDS);

}

@VisibleForTesting

protected void validateAndIndexRecord(ConsumerRecord<Long, TweetEvent> record) {

TweetEvent event = record.value();

UserScrubGeoEvent geoEvent;

try {

geoEvent = event.getData().getUser\_scrub\_geo\_event();

} catch (Exception e) {

LOG.warn("TweetEventData is null for TweetEvent: " + event.toString());

indexingFailures.increment();

return;

}

if (geoEvent == null) {

LOG.warn("UserScrubGeoEvent is null");

indexingFailures.increment();

} else if (!geoEvent.isSetMax\_tweet\_id() || !geoEvent.isSetUser\_id()) {

// We should not consume an event that does not contain both a maxTweetId & userId since we

// we won't have enough data to properly store them in the map. We should, however, keep

// track of these cases since we don't want to miss out on users who have scrubbed their

// geo data from their tweets when applying the UserScrubGeoFilter.

LOG.warn("UserScrubGeoEvent is missing fields: " + geoEvent.toString());

indexingFailures.increment();

NUM\_MISSING\_DATA\_ERRORS.increment();

} else {

SearchTimer timer = searchIndexingMetricSet.userScrubGeoIndexingStats.startNewTimer();

segmentManager.indexUserScrubGeoEvent(geoEvent);

indexingSuccesses.increment();

searchIndexingMetricSet.userScrubGeoIndexingStats.stopTimerAndIncrement(timer);

}

}

}