package com.twitter.search.earlybird.partition;

import java.time.Duration;

import java.util.Arrays;

import java.util.Collections;

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

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

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

import org.apache.kafka.common.TopicPartition;

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

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

/\*\*

\* BalancingKafkaConsumer is designed to read from the tweets and updates streams in proportion to

\* the rates that those streams are written to, i.e. both topics should have nearly the same amount

\* of lag. This is important because if one stream gets too far ahead of the other, we could end up

\* in a situation where:

\* 1. If the tweet stream is ahead of the updates stream, we couldn't apply an update because a

\* segment has been optimized, and one of those fields became frozen.

\* 2. If the updates stream is ahead of the tweet stream, we might drop updates because they are

\* more than a minute old, but the tweets might still not be indexed.

\*

\* Also see 'Consumption Flow Control' in

\* https://kafka.apache.org/23/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html

\*/

public class BalancingKafkaConsumer {

// If one of the topic-partitions lags the other by more than 10 seconds,

// it's worth it to pause the faster one and let the slower one catch up.

private static final long BALANCE\_THRESHOLD\_MS = Duration.ofSeconds(10).toMillis();

private final KafkaConsumer<Long, ThriftVersionedEvents> kafkaConsumer;

private final TopicPartition tweetTopic;

private final TopicPartition updateTopic;

private final SearchRateCounter tweetsPaused;

private final SearchRateCounter updatesPaused;

private final SearchRateCounter resumed;

private long tweetTimestamp = 0;

private long updateTimestamp = 0;

private long pausedAt = 0;

private boolean paused = false;

public BalancingKafkaConsumer(

KafkaConsumer<Long, ThriftVersionedEvents> kafkaConsumer,

TopicPartition tweetTopic,

TopicPartition updateTopic

) {

this.kafkaConsumer = kafkaConsumer;

this.tweetTopic = tweetTopic;

this.updateTopic = updateTopic;

String prefix = "balancing\_kafka\_";

String suffix = "\_topic\_paused";

tweetsPaused = SearchRateCounter.export(prefix + tweetTopic.topic() + suffix);

updatesPaused = SearchRateCounter.export(prefix + updateTopic.topic() + suffix);

resumed = SearchRateCounter.export(prefix + "topics\_resumed");

}

/\*\*

\* Calls poll on the underlying consumer and pauses topics as necessary.

\*/

public ConsumerRecords<Long, ThriftVersionedEvents> poll(Duration timeout) {

ConsumerRecords<Long, ThriftVersionedEvents> records = kafkaConsumer.poll(timeout);

topicFlowControl(records);

return records;

}

private void topicFlowControl(ConsumerRecords<Long, ThriftVersionedEvents> records) {

for (ConsumerRecord<Long, ThriftVersionedEvents> record : records) {

long timestamp = record.timestamp();

if (updateTopic.topic().equals(record.topic())) {

updateTimestamp = Math.max(updateTimestamp, timestamp);

} else if (tweetTopic.topic().equals(record.topic())) {

tweetTimestamp = Math.max(tweetTimestamp, timestamp);

} else {

throw new IllegalStateException(

"Unexpected partition " + record.topic() + " in BalancingKafkaConsumer");

}

}

if (paused) {

// If we paused and one of the streams is still below the pausedAt point, we want to continue

// reading from just the lagging stream.

if (tweetTimestamp >= pausedAt && updateTimestamp >= pausedAt) {

// We caught up, resume reading from both topics.

paused = false;

kafkaConsumer.resume(Arrays.asList(tweetTopic, updateTopic));

resumed.increment();

}

} else {

long difference = Math.abs(tweetTimestamp - updateTimestamp);

if (difference < BALANCE\_THRESHOLD\_MS) {

// The streams have approximately the same lag, so no need to pause anything.

return;

}

// The difference is too great, one of the streams is lagging behind the other so we need to

// pause one topic so the other can catch up.

paused = true;

pausedAt = Math.max(updateTimestamp, tweetTimestamp);

if (tweetTimestamp > updateTimestamp) {

kafkaConsumer.pause(Collections.singleton(tweetTopic));

tweetsPaused.increment();

} else {

kafkaConsumer.pause(Collections.singleton(updateTopic));

updatesPaused.increment();

}

}

}

public void close() {

kafkaConsumer.close();

}

}