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

import java.util.Collection;

import java.util.Map;

import javax.naming.NamingException;

import scala.runtime.BoxedUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.Maps;

import org.apache.commons.pipeline.StageException;

import org.apache.kafka.clients.producer.ProducerRecord;

import org.apache.kafka.clients.producer.RecordMetadata;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.finatra.kafka.producers.BlockingFinagleKafkaProducer;

import com.twitter.search.common.debug.DebugEventUtil;

import com.twitter.search.common.debug.thriftjava.DebugEvents;

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

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

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

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

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

import com.twitter.search.common.schema.thriftjava.ThriftIndexingEvent;

import com.twitter.search.common.schema.thriftjava.ThriftIndexingEventType;

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

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

import com.twitter.search.ingester.pipeline.twitter.TwitterBaseStage;

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

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

import com.twitter.util.Await;

import com.twitter.util.Future;

public class KafkaProducerStage<T> extends TwitterBaseStage<T, Void> {

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

private static final Logger LATE\_EVENTS\_LOG = LoggerFactory.getLogger(

KafkaProducerStage.class.getName() + ".LateEvents");

private final Map<ThriftIndexingEventType, Percentile<Long>> processingLatenciesStats =

Maps.newEnumMap(ThriftIndexingEventType.class);

private String kafkaClientId;

private String kafkaTopicName;

private String kafkaClusterPath;

private SearchCounter sendCount;

private String perPartitionSendCountFormat;

private String deciderKey;

protected BlockingFinagleKafkaProducer<Long, ThriftVersionedEvents> kafkaProducer;

private int processingLatencyThresholdMillis = 10000;

public KafkaProducerStage() { }

public KafkaProducerStage(String topicName, String clientId, String clusterPath) {

this.kafkaTopicName = topicName;

this.kafkaClientId = clientId;

this.kafkaClusterPath = clusterPath;

}

@Override

protected void initStats() {

super.initStats();

setupCommonStats();

}

private void setupCommonStats() {

sendCount = SearchCounter.export(getStageNamePrefix() + "\_send\_count");

perPartitionSendCountFormat = getStageNamePrefix() + "\_partition\_%d\_send\_count";

for (ThriftIndexingEventType eventType : ThriftIndexingEventType.values()) {

processingLatenciesStats.put(

eventType,

PercentileUtil.createPercentile(

getStageNamePrefix() + "\_" + eventType.name().toLowerCase()

+ "\_processing\_latency\_ms"));

}

}

@Override

protected void innerSetupStats() {

setupCommonStats();

}

private boolean isEnabled() {

if (this.deciderKey != null) {

return DeciderUtil.isAvailableForRandomRecipient(decider, deciderKey);

} else {

// No decider means it's enabled.

return true;

}

}

@Override

protected void doInnerPreprocess() throws StageException, NamingException {

try {

innerSetup();

} catch (PipelineStageException e) {

throw new StageException(this, e);

}

}

@Override

protected void innerSetup() throws PipelineStageException, NamingException {

Preconditions.checkNotNull(kafkaClientId);

Preconditions.checkNotNull(kafkaClusterPath);

Preconditions.checkNotNull(kafkaTopicName);

kafkaProducer = wireModule.newFinagleKafkaProducer(

kafkaClusterPath,

new CompactThriftSerializer<ThriftVersionedEvents>(),

kafkaClientId,

IngesterPartitioner.class);

int numPartitions = wireModule.getPartitionMappingManager().getNumPartitions();

int numKafkaPartitions = kafkaProducer.partitionsFor(kafkaTopicName).size();

if (numPartitions != numKafkaPartitions) {

throw new PipelineStageException(String.format(

"Number of partitions for Kafka topic %s (%d) != number of expected partitions (%d)",

kafkaTopicName, numKafkaPartitions, numPartitions));

}

}

@Override

public void innerProcess(Object obj) throws StageException {

if (!(obj instanceof IngesterThriftVersionedEvents)) {

throw new StageException(this, "Object is not IngesterThriftVersionedEvents: " + obj);

}

IngesterThriftVersionedEvents events = (IngesterThriftVersionedEvents) obj;

tryToSendEventsToKafka(events);

}

protected void tryToSendEventsToKafka(IngesterThriftVersionedEvents events) {

if (!isEnabled()) {

return;

}

DebugEvents debugEvents = events.getDebugEvents();

// We don't propagate debug events to Kafka, because they take about 50%

// of the storage space.

events.unsetDebugEvents();

ProducerRecord<Long, ThriftVersionedEvents> record = new ProducerRecord<>(

kafkaTopicName,

null,

clock.nowMillis(),

null,

events);

sendRecordToKafka(record).ensure(() -> {

updateEventProcessingLatencyStats(events, debugEvents);

return null;

});

}

private Future<RecordMetadata> sendRecordToKafka(

ProducerRecord<Long, ThriftVersionedEvents> record) {

Future<RecordMetadata> result;

try {

result = kafkaProducer.send(record);

} catch (Exception e) {

// Even though KafkaProducer.send() returns a Future, it can throw a synchronous exception,

// so we translate synchronous exceptions into a Future.exception so we handle all exceptions

// consistently.

result = Future.exception(e);

}

return result.onSuccess(recordMetadata -> {

sendCount.increment();

SearchCounter.export(

String.format(perPartitionSendCountFormat, recordMetadata.partition())).increment();

return BoxedUnit.UNIT;

}).onFailure(e -> {

stats.incrementExceptions();

LOG.error("Sending a record failed.", e);

return BoxedUnit.UNIT;

});

}

private void updateEventProcessingLatencyStats(IngesterThriftVersionedEvents events,

DebugEvents debugEvents) {

if ((debugEvents != null) && debugEvents.isSetProcessingStartedAt()) {

// Get the one indexing event out of all events we're sending.

Collection<ThriftIndexingEvent> indexingEvents = events.getVersionedEvents().values();

Preconditions.checkState(!indexingEvents.isEmpty());

ThriftIndexingEventType eventType = indexingEvents.iterator().next().getEventType();

// Check if the event took too much time to get to this current point.

long processingLatencyMillis =

clock.nowMillis() - debugEvents.getProcessingStartedAt().getEventTimestampMillis();

processingLatenciesStats.get(eventType).record(processingLatencyMillis);

if (processingLatencyMillis >= processingLatencyThresholdMillis) {

LATE\_EVENTS\_LOG.warn("Event of type {} for tweet {} was processed in {}ms: {}",

eventType.name(),

events.getTweetId(),

processingLatencyMillis,

DebugEventUtil.debugEventsToString(debugEvents));

}

}

}

public void setProcessingLatencyThresholdMillis(int processingLatencyThresholdMillis) {

this.processingLatencyThresholdMillis = processingLatencyThresholdMillis;

}

@Override

public void innerPostprocess() throws StageException {

try {

commonCleanup();

} catch (Exception e) {

throw new StageException(this, e);

}

}

@Override

public void cleanupStageV2() {

try {

commonCleanup();

} catch (Exception e) {

LOG.error("Error trying to clean up KafkaProducerStage.", e);

}

}

private void commonCleanup() throws Exception {

Await.result(kafkaProducer.close());

}

@SuppressWarnings("unused") // set from pipeline config

public void setKafkaClientId(String kafkaClientId) {

this.kafkaClientId = kafkaClientId;

}

@SuppressWarnings("unused") // set from pipeline config

public void setKafkaTopicName(String kafkaTopicName) {

this.kafkaTopicName = kafkaTopicName;

}

@VisibleForTesting

public BlockingFinagleKafkaProducer<Long, ThriftVersionedEvents> getKafkaProducer() {

return kafkaProducer;

}

@SuppressWarnings("unused") // set from pipeline config

public void setDeciderKey(String deciderKey) {

this.deciderKey = deciderKey;

}

@SuppressWarnings("unused") // set from pipeline config

public void setKafkaClusterPath(String kafkaClusterPath) {

this.kafkaClusterPath = kafkaClusterPath;

}

}