package com.twitter.search.earlybird.segment;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import java.util.Optional;

import java.util.concurrent.TimeUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Function;

import com.google.common.base.Preconditions;

import org.apache.thrift.TException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

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

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

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

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

import com.twitter.search.common.schema.earlybird.EarlybirdThriftDocumentUtil;

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

import com.twitter.search.common.util.io.ReaderWithStatsFactory;

import com.twitter.search.common.util.io.TransformingRecordReader;

import com.twitter.search.common.util.io.dl.DLMultiStreamReader;

import com.twitter.search.common.util.io.dl.DLReaderWriterFactory;

import com.twitter.search.common.util.io.dl.DLTimestampedReaderFactory;

import com.twitter.search.common.util.io.dl.SegmentDLUtil;

import com.twitter.search.common.util.io.recordreader.RecordReader;

import com.twitter.search.common.util.io.recordreader.RecordReaderFactory;

import com.twitter.search.common.util.thrift.ThriftUtils;

import com.twitter.search.earlybird.EarlybirdIndexConfig;

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

import com.twitter.search.earlybird.document.DocumentFactory;

import com.twitter.search.earlybird.document.TweetDocument;

import com.twitter.search.earlybird.partition.SegmentInfo;

public class DLSegmentDataReaderSet implements SegmentDataReaderSet {

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

public static final SearchRequestStats STATUS\_DL\_READ\_STATS =

SearchRequestStats.export("status\_dlreader", TimeUnit.MICROSECONDS, false);

private static final SearchRequestStats UPDATE\_EVENT\_DL\_READ\_STATS =

SearchRequestStats.export("update\_events\_dlreader", TimeUnit.MICROSECONDS, false);

// The number of tweets not indexed because they failed deserialization.

private static final SearchCounter STATUS\_SKIPPED\_DUE\_TO\_FAILED\_DESERIALIZATION\_COUNTER =

SearchCounter.export("statuses\_skipped\_due\_to\_failed\_deserialization");

@VisibleForTesting

public static final int FRESH\_READ\_THRESHOLD = (int) TimeUnit.MINUTES.toMillis(1);

private final int documentReadFreshnessThreshold =

EarlybirdConfig.getInt("documents\_reader\_freshness\_threshold\_millis", 10000);

private final int updateReadFreshnessThreshold =

EarlybirdConfig.getInt("updates\_freshness\_threshold\_millis", FRESH\_READ\_THRESHOLD);

private final int dlReaderVersion = EarlybirdConfig.getInt("dl\_reader\_version");

private final DLReaderWriterFactory dlFactory;

private final RecordReaderFactory<byte[]> dlUpdateEventsFactory;

private final EarlybirdIndexConfig indexConfig;

private final Clock clock;

private RecordReader<TweetDocument> documentReader;

// RecordReaders for update events that span all live segments.

private final RecordReader<ThriftVersionedEvents> updateEventsReader;

private final DLMultiStreamReader updateEventsMultiReader;

private final Map<Long, RecordReader<ThriftVersionedEvents>> updateEventReaders = new HashMap<>();

DLSegmentDataReaderSet(

DLReaderWriterFactory dlFactory,

final EarlybirdIndexConfig indexConfig,

Clock clock) throws IOException {

this.dlFactory = dlFactory;

this.indexConfig = indexConfig;

this.clock = clock;

this.dlUpdateEventsFactory = new ReaderWithStatsFactory(

new DLTimestampedReaderFactory(dlFactory, clock, updateReadFreshnessThreshold),

UPDATE\_EVENT\_DL\_READ\_STATS);

this.updateEventsMultiReader =

new DLMultiStreamReader("update\_events", dlUpdateEventsFactory, true, clock);

this.updateEventsReader =

new TransformingRecordReader<>(updateEventsMultiReader, record ->

(record != null) ? deserializeTVE(record.getBytes()) : null);

SearchCustomGauge.export("open\_dl\_update\_events\_streams", updateEventReaders::size);

}

private ThriftVersionedEvents deserializeTVE(byte[] bytes) {

ThriftVersionedEvents event = new ThriftVersionedEvents();

try {

ThriftUtils.fromCompactBinaryFormat(bytes, event);

return event;

} catch (TException e) {

LOG.error("error deserializing TVE", e);

return null;

}

}

@Override

public void attachDocumentReaders(SegmentInfo segmentInfo) throws IOException {

// Close any document reader left open before.

if (documentReader != null) {

LOG.warn("Previous documentReader not closed: {}", documentReader);

completeSegmentDocs(segmentInfo);

}

documentReader = newDocumentReader(segmentInfo);

}

@Override

public void attachUpdateReaders(SegmentInfo segmentInfo) throws IOException {

if (updateEventsMultiReader == null) {

return;

}

String segmentName = segmentInfo.getSegmentName();

if (getUpdateEventsReaderForSegment(segmentInfo) != null) {

LOG.info("Update events reader for segment {} is already attached.", segmentName);

return;

}

long updateEventStreamOffsetTimestamp = segmentInfo.getUpdatesStreamOffsetTimestamp();

LOG.info("Attaching update events reader for segment {} with timestamp: {}.",

segmentName, updateEventStreamOffsetTimestamp);

String topic = SegmentDLUtil.getDLTopicForUpdateEvents(segmentName, dlReaderVersion);

RecordReader<byte[]> recordReader =

dlUpdateEventsFactory.newRecordReaderForTimestamp(topic, updateEventStreamOffsetTimestamp);

updateEventsMultiReader.addRecordReader(recordReader, topic);

updateEventReaders.put(segmentInfo.getTimeSliceID(),

new TransformingRecordReader<>(recordReader, this::deserializeTVE));

}

@Override

public void stopAll() {

if (documentReader != null) {

documentReader.close();

}

if (updateEventsReader != null) {

updateEventsReader.close();

}

try {

dlFactory.close();

} catch (IOException e) {

LOG.error("Exception while closing DL factory", e);

}

}

@Override

public void completeSegmentDocs(SegmentInfo segmentInfo) {

if (documentReader != null) {

documentReader.close();

documentReader = null;

}

}

@Override

public void stopSegmentUpdates(SegmentInfo segmentInfo) {

if (updateEventsMultiReader != null) {

updateEventsMultiReader.removeStream(

SegmentDLUtil.getDLTopicForUpdateEvents(segmentInfo.getSegmentName(), dlReaderVersion));

updateEventReaders.remove(segmentInfo.getTimeSliceID());

}

}

@Override

public RecordReader<TweetDocument> newDocumentReader(SegmentInfo segmentInfo) throws IOException {

String topic = SegmentDLUtil.getDLTopicForTweets(segmentInfo.getSegmentName(),

EarlybirdConfig.getPenguinVersion(), dlReaderVersion);

final long timeSliceId = segmentInfo.getTimeSliceID();

final DocumentFactory<ThriftIndexingEvent> docFactory = indexConfig.createDocumentFactory();

// Create the underlying DLRecordReader wrapped with the tweet reader stats.

RecordReader<byte[]> dlReader = new ReaderWithStatsFactory(

new DLTimestampedReaderFactory(

dlFactory,

clock,

documentReadFreshnessThreshold),

STATUS\_DL\_READ\_STATS)

.newRecordReader(topic);

// Create the wrapped reader which transforms serialized byte[] to TweetDocument.

return new TransformingRecordReader<>(

dlReader,

new Function<byte[], TweetDocument>() {

@Override

public TweetDocument apply(byte[] input) {

ThriftIndexingEvent event = new ThriftIndexingEvent();

try {

ThriftUtils.fromCompactBinaryFormat(input, event);

} catch (TException e) {

LOG.error("Could not deserialize status document", e);

STATUS\_SKIPPED\_DUE\_TO\_FAILED\_DESERIALIZATION\_COUNTER.increment();

return null;

}

Preconditions.checkNotNull(event.getDocument());

return new TweetDocument(

docFactory.getStatusId(event),

timeSliceId,

EarlybirdThriftDocumentUtil.getCreatedAtMs(event.getDocument()),

docFactory.newDocument(event));

}

});

}

@Override

public RecordReader<TweetDocument> getDocumentReader() {

return documentReader;

}

@Override

public RecordReader<ThriftVersionedEvents> getUpdateEventsReader() {

return updateEventsReader;

}

@Override

public RecordReader<ThriftVersionedEvents> getUpdateEventsReaderForSegment(

SegmentInfo segmentInfo) {

return updateEventReaders.get(segmentInfo.getTimeSliceID());

}

@Override

public Optional<Long> getUpdateEventsStreamOffsetForSegment(SegmentInfo segmentInfo) {

String topic =

SegmentDLUtil.getDLTopicForUpdateEvents(segmentInfo.getSegmentName(), dlReaderVersion);

return updateEventsMultiReader.getUnderlyingOffsetForSegmentWithTopic(topic);

}

@Override

public boolean allCaughtUp() {

return ((getDocumentReader() == null) || getDocumentReader().isCaughtUp())

&& ((getUpdateEventsReader() == null) || getUpdateEventsReader().isCaughtUp());

}

}