package com.twitter.search.earlybird.partition;

import java.io.IOException;

import java.util.EnumMap;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.atomic.AtomicLong;

import com.google.common.collect.HashBasedTable;

import com.google.common.collect.Table;

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.SearchRateCounter;

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

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

import com.twitter.search.common.partitioning.snowflakeparser.SnowflakeIdParser;

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

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

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.index.EarlybirdSegment;

import com.twitter.util.Time;

public class SegmentWriter implements ISegmentWriter {

// helper, used for collecting stats

enum FailureReason {

FAILED\_INSERT,

FAILED\_FOR\_TWEET\_IN\_INDEX,

FAILED\_FOR\_COMPLETE\_SEGMENT

}

private static final String STAT\_PREFIX = "segment\_writer\_";

private static final String EVENT\_COUNTER = STAT\_PREFIX + "%s\_%s\_segment\_%s";

private static final String EVENT\_COUNTER\_ALL\_SEGMENTS = STAT\_PREFIX + "%s\_%s\_all\_segments";

private static final String EVENT\_TIMERS = STAT\_PREFIX + "%s\_timing";

private static final String DROPPED\_UPDATES\_FOR\_DISABLED\_SEGMENTS =

STAT\_PREFIX + "%s\_dropped\_updates\_for\_disabled\_segments";

private static final String INDEXING\_LATENCY =

STAT\_PREFIX + "%s\_indexing\_latency\_ms";

private final byte penguinVersion;

private final DocumentFactory<ThriftIndexingEvent> updateFactory;

private final DocumentFactory<ThriftIndexingEvent> documentFactory;

private final SearchRateCounter missingPenguinVersion;

private final EarlybirdSegment earlybirdSegment;

private final SegmentInfo segmentInfo;

// Stores per segment counters for each (indexing event type, result) pair

// Example stat name

// "segment\_writer\_partial\_update\_success\_segment\_twttr\_search\_test\_start\_%d\_p\_0\_of\_1"

private final Table<ThriftIndexingEventType, Result, SearchRateCounter> statsForUpdateType =

HashBasedTable.create();

// Stores aggregated counters for each (indexing event type, result) pair across all segments

// Example stat name

// "segment\_writer\_partial\_update\_success\_all\_segments"

private final Table<ThriftIndexingEventType, Result, SearchRateCounter>

aggregateStatsForUpdateType = HashBasedTable.create();

// Stores per segment counters for each (indexing event type, non-retryable failure reason) pair

// Example stat name

// "segment\_writer\_partial\_update\_failed\_for\_tweet\_in\_index\_segment\_twttr\_search\_t\_%d\_p\_0\_of\_1"

private final Table<ThriftIndexingEventType, FailureReason, SearchRateCounter>

failureStatsForUpdateType = HashBasedTable.create();

// Stores aggregated counters for each (indexing event type, non-retryable failure reason) pair

// Example stat name

// "segment\_writer\_partial\_update\_failed\_for\_tweet\_in\_index\_all\_segments"

private final Table<ThriftIndexingEventType, FailureReason, SearchRateCounter>

aggregateFailureStatsForUpdateType = HashBasedTable.create();

private final EnumMap<ThriftIndexingEventType, SearchTimerStats> eventTimers =

new EnumMap<>(ThriftIndexingEventType.class);

private final EnumMap<ThriftIndexingEventType, SearchRateCounter>

droppedUpdatesForDisabledSegments = new EnumMap<>(ThriftIndexingEventType.class);

// We pass this stat from the SearchIndexingMetricSet so that we can share the atomic longs

// between all SegmentWriters and export the largest freshness value across all segments.

private final EnumMap<ThriftIndexingEventType, AtomicLong> updateFreshness;

private final EnumMap<ThriftIndexingEventType, Percentile<Long>> indexingLatency =

new EnumMap<>(ThriftIndexingEventType.class);

public SegmentWriter(

SegmentInfo segmentInfo,

EnumMap<ThriftIndexingEventType, AtomicLong> updateFreshness

) {

this.segmentInfo = segmentInfo;

this.updateFreshness = updateFreshness;

this.earlybirdSegment = segmentInfo.getIndexSegment();

this.penguinVersion = EarlybirdConfig.getPenguinVersionByte();

this.updateFactory = segmentInfo.getEarlybirdIndexConfig().createUpdateFactory();

this.documentFactory = segmentInfo.getEarlybirdIndexConfig().createDocumentFactory();

String segmentName = segmentInfo.getSegmentName();

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

for (Result result : Result.values()) {

String stat = String.format(EVENT\_COUNTER, type, result, segmentName).toLowerCase();

statsForUpdateType.put(type, result, SearchRateCounter.export(stat));

String aggregateStat =

String.format(EVENT\_COUNTER\_ALL\_SEGMENTS, type, result).toLowerCase();

aggregateStatsForUpdateType.put(type, result, SearchRateCounter.export(aggregateStat));

}

for (FailureReason reason : FailureReason.values()) {

String stat = String.format(EVENT\_COUNTER, type, reason, segmentName).toLowerCase();

failureStatsForUpdateType.put(type, reason, SearchRateCounter.export(stat));

String aggregateStat =

String.format(EVENT\_COUNTER\_ALL\_SEGMENTS, type, reason).toLowerCase();

aggregateFailureStatsForUpdateType.put(

type, reason, SearchRateCounter.export(aggregateStat));

}

eventTimers.put(type, SearchTimerStats.export(

String.format(EVENT\_TIMERS, type).toLowerCase(),

TimeUnit.MICROSECONDS,

false));

droppedUpdatesForDisabledSegments.put(

type,

SearchRateCounter.export(

String.format(DROPPED\_UPDATES\_FOR\_DISABLED\_SEGMENTS, type).toLowerCase()));

indexingLatency.put(

type,

PercentileUtil.createPercentile(

String.format(INDEXING\_LATENCY, type).toLowerCase()));

}

this.missingPenguinVersion = SearchRateCounter.export(

"documents\_without\_current\_penguin\_version\_" + penguinVersion + "\_" + segmentName);

}

@Override

public synchronized Result indexThriftVersionedEvents(ThriftVersionedEvents tve)

throws IOException {

if (!tve.getVersionedEvents().containsKey(penguinVersion)) {

missingPenguinVersion.increment();

return Result.FAILURE\_NOT\_RETRYABLE;

}

ThriftIndexingEvent tie = tve.getVersionedEvents().get(penguinVersion);

ThriftIndexingEventType eventType = tie.getEventType();

if (!segmentInfo.isEnabled()) {

droppedUpdatesForDisabledSegments.get(eventType).increment();

return Result.SUCCESS;

}

SearchTimerStats timerStats = eventTimers.get(eventType);

SearchTimer timer = timerStats.startNewTimer();

long tweetId = tve.getId();

Result result = tryApplyIndexingEvent(tweetId, tie);

if (result == Result.SUCCESS) {

long tweetAgeInMs = SnowflakeIdParser.getTimestampFromTweetId(tweetId);

AtomicLong freshness = updateFreshness.get(tie.getEventType());

// Note that this is racy at startup because we don't do an atomic swap, but it will be

// approximately accurate, and this stat doesn't matter until we are current.

if (freshness.get() < tweetAgeInMs) {

freshness.set(tweetAgeInMs);

}

if (tie.isSetCreateTimeMillis()) {

long age = Time.now().inMillis() - tie.getCreateTimeMillis();

indexingLatency.get(tie.getEventType()).record(age);

}

}

statsForUpdateType.get(eventType, result).increment();

aggregateStatsForUpdateType.get(eventType, result).increment();

timerStats.stopTimerAndIncrement(timer);

return result;

}

public SegmentInfo getSegmentInfo() {

return segmentInfo;

}

public boolean hasTweet(long tweetId) throws IOException {

return earlybirdSegment.hasDocument(tweetId);

}

private Result tryApplyIndexingEvent(long tweetId, ThriftIndexingEvent tie) throws IOException {

if (applyIndexingEvent(tie, tweetId)) {

return Result.SUCCESS;

}

if (tie.getEventType() == ThriftIndexingEventType.INSERT) {

// We don't retry inserts

incrementFailureStats(tie, FailureReason.FAILED\_INSERT);

return Result.FAILURE\_NOT\_RETRYABLE;

}

if (earlybirdSegment.hasDocument(tweetId)) {

// An update fails to be applied for a tweet that is in the index.

incrementFailureStats(tie, FailureReason.FAILED\_FOR\_TWEET\_IN\_INDEX);

return Result.FAILURE\_NOT\_RETRYABLE;

}

if (segmentInfo.isComplete()) {

// An update is directed at a tweet that is not in the segment (hasDocument(tweetId) failed),

// and the segment is complete (i.e. there will never be new tweets for this segment).

incrementFailureStats(tie, FailureReason.FAILED\_FOR\_COMPLETE\_SEGMENT);

return Result.FAILURE\_NOT\_RETRYABLE;

}

// The tweet may arrive later for this event, so it's possible a later try will succeed

return Result.FAILURE\_RETRYABLE;

}

private void incrementFailureStats(ThriftIndexingEvent tie, FailureReason failureReason) {

failureStatsForUpdateType.get(tie.getEventType(), failureReason).increment();

aggregateFailureStatsForUpdateType.get(tie.getEventType(), failureReason).increment();

}

private boolean applyIndexingEvent(ThriftIndexingEvent tie, long tweetId) throws IOException {

switch (tie.getEventType()) {

case OUT\_OF\_ORDER\_APPEND:

return earlybirdSegment.appendOutOfOrder(updateFactory.newDocument(tie), tweetId);

case PARTIAL\_UPDATE:

return earlybirdSegment.applyPartialUpdate(tie);

case DELETE:

return earlybirdSegment.delete(tweetId);

case INSERT:

earlybirdSegment.addDocument(buildInsertDocument(tie, tweetId));

return true;

default:

throw new IllegalArgumentException("Unexpected update type: " + tie.getEventType());

}

}

private TweetDocument buildInsertDocument(ThriftIndexingEvent tie, long tweetId) {

return new TweetDocument(

tweetId,

segmentInfo.getTimeSliceID(),

tie.getCreateTimeMillis(),

documentFactory.newDocument(tie));

}

}