package com.twitter.search.earlybird.document;

import java.io.IOException;

import java.util.concurrent.TimeUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import org.apache.lucene.document.Document;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

import com.twitter.decider.Decider;

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

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

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

import com.twitter.search.common.schema.SchemaDocumentFactory;

import com.twitter.search.common.schema.base.FieldNameToIdMapping;

import com.twitter.search.common.schema.base.ImmutableSchemaInterface;

import com.twitter.search.common.schema.base.Schema;

import com.twitter.search.common.schema.base.ThriftDocumentUtil;

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

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

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

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

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

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

import com.twitter.search.common.util.text.filter.NormalizedTokenFilter;

import com.twitter.search.common.util.text.splitter.HashtagMentionPunctuationSplitter;

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

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

public class ThriftIndexingEventDocumentFactory extends DocumentFactory<ThriftIndexingEvent> {

private static final Logger LOG =

LoggerFactory.getLogger(ThriftIndexingEventDocumentFactory.class);

private static final FieldNameToIdMapping ID\_MAPPING = new EarlybirdFieldConstants();

private static final long TIMESTAMP\_ALLOWED\_FUTURE\_DELTA\_MS = TimeUnit.SECONDS.toMillis(60);

private static final String FILTER\_TWEETS\_WITH\_FUTURE\_TWEET\_ID\_AND\_CREATED\_AT\_DECIDER\_KEY =

"filter\_tweets\_with\_future\_tweet\_id\_and\_created\_at";

private static final SearchCounter NUM\_TWEETS\_WITH\_FUTURE\_TWEET\_ID\_AND\_CREATED\_AT\_MS =

SearchCounter.export("num\_tweets\_with\_future\_tweet\_id\_and\_created\_at\_ms");

private static final SearchCounter NUM\_TWEETS\_WITH\_INCONSISTENT\_TWEET\_ID\_AND\_CREATED\_AT\_MS\_FOUND =

SearchCounter.export("num\_tweets\_with\_inconsistent\_tweet\_id\_and\_created\_at\_ms\_found");

private static final SearchCounter

NUM\_TWEETS\_WITH\_INCONSISTENT\_TWEET\_ID\_AND\_CREATED\_AT\_MS\_ADJUSTED =

SearchCounter.export("num\_tweets\_with\_inconsistent\_tweet\_id\_and\_created\_at\_ms\_adjusted");

private static final SearchCounter NUM\_TWEETS\_WITH\_INCONSISTENT\_TWEET\_ID\_AND\_CREATED\_AT\_MS\_DROPPED

= SearchCounter.export("num\_tweets\_with\_inconsistent\_tweet\_id\_and\_created\_at\_ms\_dropped");

@VisibleForTesting

static final String ENABLE\_ADJUST\_CREATED\_AT\_TIME\_IF\_MISMATCH\_WITH\_SNOWFLAKE =

"enable\_adjust\_created\_at\_time\_if\_mismatch\_with\_snowflake";

@VisibleForTesting

static final String ENABLE\_DROP\_CREATED\_AT\_TIME\_IF\_MISMATCH\_WITH\_SNOWFLAKE =

"enable\_drop\_created\_at\_time\_if\_mismatch\_with\_snowflake";

private final SchemaDocumentFactory schemaDocumentFactory;

private final EarlybirdCluster cluster;

private final SearchIndexingMetricSet searchIndexingMetricSet;

private final Decider decider;

private final Schema schema;

private final Clock clock;

public ThriftIndexingEventDocumentFactory(

Schema schema,

EarlybirdCluster cluster,

Decider decider,

SearchIndexingMetricSet searchIndexingMetricSet,

CriticalExceptionHandler criticalExceptionHandler) {

this(

schema,

getSchemaDocumentFactory(schema, cluster, decider),

cluster,

searchIndexingMetricSet,

decider,

Clock.SYSTEM\_CLOCK,

criticalExceptionHandler

);

}

/\*\*

\* Returns a document factory that knows how to convert ThriftDocuments to Documents based on the

\* provided schema.

\*/

public static SchemaDocumentFactory getSchemaDocumentFactory(

Schema schema,

EarlybirdCluster cluster,

Decider decider) {

return new SchemaDocumentFactory(schema,

Lists.newArrayList(

new TruncationTokenStreamWriter(cluster, decider),

(fieldInfo, stream) -> {

// Strip # @ $ symbols, and break up underscore connected tokens.

if (fieldInfo.getFieldType().useTweetSpecificNormalization()) {

return new HashtagMentionPunctuationSplitter(new NormalizedTokenFilter(stream));

}

return stream;

}));

}

@VisibleForTesting

protected ThriftIndexingEventDocumentFactory(

Schema schema,

SchemaDocumentFactory schemaDocumentFactory,

EarlybirdCluster cluster,

SearchIndexingMetricSet searchIndexingMetricSet,

Decider decider,

Clock clock,

CriticalExceptionHandler criticalExceptionHandler) {

super(criticalExceptionHandler);

this.schema = schema;

this.schemaDocumentFactory = schemaDocumentFactory;

this.cluster = cluster;

this.searchIndexingMetricSet = searchIndexingMetricSet;

this.decider = decider;

this.clock = clock;

}

@Override

public long getStatusId(ThriftIndexingEvent event) {

Preconditions.checkNotNull(event);

if (event.isSetDocument() && event.getDocument() != null) {

ThriftDocument thriftDocument = event.getDocument();

try {

// Ideally, we should not call getSchemaSnapshot() here. But, as this is called only to

// retrieve status id and the ID field is static, this is fine for the purpose.

thriftDocument = ThriftDocumentPreprocessor.preprocess(

thriftDocument, cluster, schema.getSchemaSnapshot());

} catch (IOException e) {

throw new IllegalStateException("Unable to obtain tweet ID from ThriftDocument", e);

}

return ThriftDocumentUtil.getLongValue(

thriftDocument, EarlybirdFieldConstant.ID\_FIELD.getFieldName(), ID\_MAPPING);

} else {

throw new IllegalArgumentException("ThriftDocument is null inside ThriftIndexingEvent.");

}

}

@Override

protected Document innerNewDocument(ThriftIndexingEvent event) throws IOException {

Preconditions.checkNotNull(event);

Preconditions.checkNotNull(event.getDocument());

ImmutableSchemaInterface schemaSnapshot = schema.getSchemaSnapshot();

// If the tweet id and create\_at are in the future, do not index it.

if (areTweetIDAndCreateAtInTheFuture(event)

&& DeciderUtil.isAvailableForRandomRecipient(decider,

FILTER\_TWEETS\_WITH\_FUTURE\_TWEET\_ID\_AND\_CREATED\_AT\_DECIDER\_KEY)) {

NUM\_TWEETS\_WITH\_FUTURE\_TWEET\_ID\_AND\_CREATED\_AT\_MS.increment();

return null;

}

if (isNullcastBitAndFilterConsistent(schemaSnapshot, event)) {

ThriftDocument thriftDocument =

adjustOrDropIfTweetIDAndCreatedAtAreInconsistent(

ThriftDocumentPreprocessor.preprocess(event.getDocument(), cluster, schemaSnapshot));

if (thriftDocument != null) {

return schemaDocumentFactory.newDocument(thriftDocument);

} else {

return null;

}

} else {

return null;

}

}

private ThriftDocument adjustOrDropIfTweetIDAndCreatedAtAreInconsistent(ThriftDocument document) {

final long tweetID = EarlybirdThriftDocumentUtil.getID(document);

// Thrift document is storing created at in seconds.

final long createdAtMs = EarlybirdThriftDocumentUtil.getCreatedAtMs(document);

if (!SnowflakeIdParser.isTweetIDAndCreatedAtConsistent(tweetID, createdAtMs)) {

// Increment found counter.

NUM\_TWEETS\_WITH\_INCONSISTENT\_TWEET\_ID\_AND\_CREATED\_AT\_MS\_FOUND.increment();

LOG.error(

"Found inconsistent tweet ID and created at timestamp: [tweetID={}], [createdAtMs={}]",

tweetID, createdAtMs);

if (DeciderUtil.isAvailableForRandomRecipient(

decider, ENABLE\_ADJUST\_CREATED\_AT\_TIME\_IF\_MISMATCH\_WITH\_SNOWFLAKE)) {

// Update created at (and csf) with the time stamp in snow flake ID.

final long createdAtMsInID = SnowflakeIdParser.getTimestampFromTweetId(tweetID);

EarlybirdThriftDocumentUtil.replaceCreatedAtAndCreatedAtCSF(

document, (int) (createdAtMsInID / 1000));

// Increment adjusted counter.

NUM\_TWEETS\_WITH\_INCONSISTENT\_TWEET\_ID\_AND\_CREATED\_AT\_MS\_ADJUSTED.increment();

LOG.error(

"Updated created at to match tweet ID: createdAtMs={}, tweetID={}, createdAtMsInID={}",

createdAtMs, tweetID, createdAtMsInID);

} else if (DeciderUtil.isAvailableForRandomRecipient(

decider, ENABLE\_DROP\_CREATED\_AT\_TIME\_IF\_MISMATCH\_WITH\_SNOWFLAKE)) {

// Drop and increment counter!

NUM\_TWEETS\_WITH\_INCONSISTENT\_TWEET\_ID\_AND\_CREATED\_AT\_MS\_DROPPED.increment();

LOG.error(

"Dropped tweet with inconsistent ID and timestamp: createdAtMs={}, tweetID={}",

createdAtMs, tweetID);

return null;

}

}

return document;

}

private boolean isNullcastBitAndFilterConsistent(

ImmutableSchemaInterface schemaSnapshot,

ThriftIndexingEvent event) {

return ThriftDocumentPreprocessor.isNullcastBitAndFilterConsistent(

event.getDocument(), schemaSnapshot);

}

/\*\*

\* Check if the tweet ID and create\_at are in the future and beyond the allowed

\* TIMESTAMP\_ALLOWED\_FUTURE\_DELTA\_MS range from current time stamp.

\*/

private boolean areTweetIDAndCreateAtInTheFuture(ThriftIndexingEvent event) {

ThriftDocument document = event.getDocument();

final long tweetID = EarlybirdThriftDocumentUtil.getID(document);

if (tweetID < SnowflakeIdParser.SNOWFLAKE\_ID\_LOWER\_BOUND) {

return false;

}

final long tweetIDTimestampMs = SnowflakeIdParser.getTimestampFromTweetId(tweetID);

final long allowedFutureTimestampMs = clock.nowMillis() + TIMESTAMP\_ALLOWED\_FUTURE\_DELTA\_MS;

final long createdAtMs = EarlybirdThriftDocumentUtil.getCreatedAtMs(document);

if (tweetIDTimestampMs > allowedFutureTimestampMs && createdAtMs > allowedFutureTimestampMs) {

LOG.error(

"Found future tweet ID and created at timestamp: "

+ "[tweetID={}], [createdAtMs={}], [compareDeltaMs={}]",

tweetID, createdAtMs, TIMESTAMP\_ALLOWED\_FUTURE\_DELTA\_MS);

return true;

}

return false;

}

}