package com.twitter.search.earlybird.archive;

import java.io.IOException;

import java.util.Date;

import com.google.common.base.Preconditions;

import com.google.common.base.Predicate;

import org.apache.commons.lang.time.FastDateFormat;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

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

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

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

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

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

import com.twitter.search.common.util.zktrylock.ZooKeeperTryLockFactory;

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.exception.CriticalExceptionHandler;

import com.twitter.search.earlybird.index.EarlybirdSegmentFactory;

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

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

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

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

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

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

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

import com.twitter.search.earlybird.stats.EarlybirdSearcherStats;

/\*\*

\* Given a segment, this class checks if the segment has an index built on HDFS:

\* if not, use SimpleSegmentIndexer to build an index

\* if yes, load the HDFS index, build a new index for the new status data which has dates newer

\* than the HDFS index, then append the loaded HDFS index.

\*/

public class ArchiveSegmentUpdater {

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

private final SegmentSyncConfig sync;

private final EarlybirdIndexConfig earlybirdIndexConfig;

private final ZooKeeperTryLockFactory zkTryLockFactory;

private final SearchStatsReceiver statsReceiver = new SearchStatsReceiverImpl();

private final SearchIndexingMetricSet searchIndexingMetricSet =

new SearchIndexingMetricSet(statsReceiver);

private final EarlybirdSearcherStats searcherStats =

new EarlybirdSearcherStats(statsReceiver);

private final SearchRateCounter indexNewSegment =

new SearchRateCounter("index\_new\_segment");

private final SearchRateCounter updateExistingSegment =

new SearchRateCounter("update\_existing\_segment");

private final SearchRateCounter skipExistingSegment =

new SearchRateCounter("skip\_existing\_segment");

private Clock clock;

public ArchiveSegmentUpdater(ZooKeeperTryLockFactory zooKeeperTryLockFactory,

SegmentSyncConfig sync,

EarlybirdIndexConfig earlybirdIndexConfig,

Clock clock) {

this.sync = sync;

this.earlybirdIndexConfig = earlybirdIndexConfig;

this.zkTryLockFactory = zooKeeperTryLockFactory;

this.clock = clock;

}

private boolean canUpdateSegment(SegmentInfo segmentInfo) {

if (!(segmentInfo.getSegment() instanceof ArchiveSegment)) {

LOG.info("only ArchiveSegment is available for updating now: "

+ segmentInfo);

return false;

}

if (!segmentInfo.isEnabled()) {

LOG.debug("Segment is disabled: " + segmentInfo);

return false;

}

if (segmentInfo.isComplete() || segmentInfo.isIndexing()

|| segmentInfo.getSyncInfo().isLoaded()) {

LOG.debug("Cannot update already indexed segment: " + segmentInfo);

return false;

}

return true;

}

/\*\*

\* Given a segment, checks if the segment has an index built on HDFS:

\* if not, use SimpleSegmentIndexer to build an index

\* if yes, load the HDFS index, build a new index for the new status data which has dates newer

\* than the HDFS index, then append the loaded HDFS index.

\*

\* Returns whether the segment was successfully updated.

\*/

public boolean updateSegment(SegmentInfo segmentInfo) {

Preconditions.checkArgument(segmentInfo.getSegment() instanceof ArchiveSegment);

if (!canUpdateSegment(segmentInfo)) {

return false;

}

if (segmentInfo.isIndexing()) {

LOG.error("Segment is already being indexed: " + segmentInfo);

return false;

}

final Date hdfsEndDate = ArchiveHDFSUtils.getSegmentEndDateOnHdfs(sync, segmentInfo);

if (hdfsEndDate == null) {

indexNewSegment.increment();

if (!indexSegment(segmentInfo, ArchiveSegment.MATCH\_ALL\_DATE\_PREDICATE)) {

return false;

}

} else {

final Date curEndDate = ((ArchiveSegment) segmentInfo.getSegment()).getDataEndDate();

if (!hdfsEndDate.before(curEndDate)) {

skipExistingSegment.increment();

LOG.info("Segment is up-to-date: " + segmentInfo.getSegment().getTimeSliceID()

+ " Found flushed segment on HDFS with end date: "

+ FastDateFormat.getInstance("yyyyMMdd").format(hdfsEndDate));

segmentInfo.setComplete(true);

segmentInfo.getSyncInfo().setFlushed(true);

return true;

}

updateExistingSegment.increment();

LOG.info("Updating segment: " + segmentInfo.getSegment().getTimeSliceID()

+ "; new endDate will be " + FastDateFormat.getInstance("yyyyMMdd").format(curEndDate));

if (!updateSegment(segmentInfo, hdfsEndDate)) {

return false;

}

}

boolean success = SegmentOptimizer.optimize(segmentInfo);

if (!success) {

// Clean up the segment dir on local disk

segmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

LOG.info("Error optimizing segment: " + segmentInfo);

return false;

}

// Verify segment before uploading.

success = ArchiveSegmentVerifier.verifySegment(segmentInfo);

if (!success) {

segmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

LOG.info("Segment not uploaded to HDFS because it did not pass verification: " + segmentInfo);

return false;

}

// upload the index to HDFS

success = new SegmentHdfsFlusher(zkTryLockFactory, sync, false)

.flushSegmentToDiskAndHDFS(segmentInfo);

if (success) {

ArchiveHDFSUtils.deleteHdfsSegmentDir(sync, segmentInfo, false, true);

} else {

// Clean up the segment dir on hdfs

ArchiveHDFSUtils.deleteHdfsSegmentDir(sync, segmentInfo, true, false);

LOG.info("Error uploading segment to HDFS: " + segmentInfo);

}

segmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

return success;

}

/\*\*

\* Build index for the given segmentInfo. Only those statuses passing the dateFilter are indexed.

\*/

private boolean indexSegment(final SegmentInfo segmentInfo, Predicate<Date> dateFilter) {

Preconditions.checkArgument(segmentInfo.getSegment() instanceof ArchiveSegment);

RecordReader<TweetDocument> documentReader = null;

try {

ArchiveSegment archiveSegment = (ArchiveSegment) segmentInfo.getSegment();

DocumentFactory<ThriftIndexingEvent> documentFactory =

earlybirdIndexConfig.createDocumentFactory();

documentReader = archiveSegment.getStatusRecordReader(documentFactory, dateFilter);

// Read and index the statuses

boolean success = new SimpleSegmentIndexer(documentReader, searchIndexingMetricSet)

.indexSegment(segmentInfo);

if (!success) {

// Clean up segment dir on local disk

segmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

LOG.info("Error indexing segment: " + segmentInfo);

}

return success;

} catch (IOException e) {

segmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

LOG.info("Exception while indexing segment: " + segmentInfo, e);

return false;

} finally {

if (documentReader != null) {

documentReader.stop();

}

}

}

/\*\*

\* Load the index built on HDFS for the given segmentInfo, index the new data and append the

\* HDFS index to the new indexed segment

\*/

private boolean updateSegment(final SegmentInfo segmentInfo, final Date hdfsEndDate) {

SegmentInfo hdfsSegmentInfo = loadSegmentFromHdfs(segmentInfo, hdfsEndDate);

if (hdfsSegmentInfo == null) {

return indexSegment(segmentInfo, ArchiveSegment.MATCH\_ALL\_DATE\_PREDICATE);

}

boolean success = indexSegment(segmentInfo, input -> {

// we're updating the segment - only index days after the old end date,

// and we're sure that the previous days have already been indexed.

return input.after(hdfsEndDate);

});

if (!success) {

LOG.error("Error indexing new data: " + segmentInfo);

return indexSegment(segmentInfo, ArchiveSegment.MATCH\_ALL\_DATE\_PREDICATE);

}

// Now, append the index loaded from hdfs

try {

segmentInfo.getIndexSegment().append(hdfsSegmentInfo.getIndexSegment());

hdfsSegmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

LOG.info("Deleted local segment directories with end date " + hdfsEndDate + " : "

+ segmentInfo);

} catch (IOException e) {

LOG.warn("Caught IOException while appending segment " + hdfsSegmentInfo.getSegmentName(), e);

hdfsSegmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

segmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

return false;

}

segmentInfo.setComplete(true);

return true;

}

/\*\*

\* Load the index built on HDFS for the given segmentInfo and end date

\*/

private SegmentInfo loadSegmentFromHdfs(final SegmentInfo segmentInfo, final Date hdfsEndDate) {

Preconditions.checkArgument(segmentInfo.getSegment() instanceof ArchiveSegment);

ArchiveSegment segment = new ArchiveSegment(

segmentInfo.getTimeSliceID(),

EarlybirdConfig.getMaxSegmentSize(),

segmentInfo.getNumPartitions(),

segmentInfo.getSegment().getHashPartitionID(),

hdfsEndDate);

EarlybirdSegmentFactory factory = new EarlybirdSegmentFactory(

earlybirdIndexConfig,

searchIndexingMetricSet,

searcherStats,

clock);

SegmentInfo hdfsSegmentInfo;

try {

hdfsSegmentInfo = new SegmentInfo(segment, factory, sync);

CriticalExceptionHandler criticalExceptionHandler =

new CriticalExceptionHandler();

boolean success = new SegmentLoader(sync, criticalExceptionHandler)

.load(hdfsSegmentInfo);

if (!success) {

// If not successful, segmentLoader has already cleaned up the local dir.

LOG.info("Error loading hdfs segment " + hdfsSegmentInfo

+ ", building segment from scratch.");

hdfsSegmentInfo = null;

}

} catch (IOException e) {

LOG.error("Exception while loading segment from hdfs: " + segmentInfo, e);

hdfsSegmentInfo = null;

}

return hdfsSegmentInfo;

}

}