package com.twitter.search.earlybird.archive;

import java.io.IOException;

import java.util.Date;

import java.util.List;

import java.util.concurrent.TimeUnit;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.base.Predicate;

import com.google.common.collect.Lists;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

import com.twitter.search.common.concurrent.ScheduledExecutorServiceFactory;

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

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

import com.twitter.search.common.util.GCUtil;

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.EarlybirdStatus;

import com.twitter.search.earlybird.ServerSetMember;

import com.twitter.search.earlybird.archive.ArchiveTimeSlicer.ArchiveTimeSlice;

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

import com.twitter.search.earlybird.util.ScrubGenUtil;

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

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

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

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

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

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

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

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.SegmentManager;

import com.twitter.search.earlybird.partition.SegmentManager.Filter;

import com.twitter.search.earlybird.partition.SegmentManager.Order;

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

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

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

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

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

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

import com.twitter.search.earlybird.querycache.QueryCacheManager;

import com.twitter.search.earlybird.segment.SegmentDataProvider;

import com.twitter.search.earlybird.thrift.EarlybirdStatusCode;

import com.twitter.search.earlybird.util.CoordinatedEarlybirdAction;

import com.twitter.search.earlybird.util.CoordinatedEarlybirdActionInterface;

import com.twitter.search.earlybird.util.CoordinatedEarlybirdActionLockFailed;

public class ArchiveSearchPartitionManager extends PartitionManager {

private static final Logger LOG =

LoggerFactory.getLogger(ArchiveSearchPartitionManager.class);

public static final String CONFIG\_NAME = "archive";

private static final long ONE\_DAY\_MILLIS = TimeUnit.DAYS.toMillis(1);

private final ArchiveTimeSlicer timeSlicer;

private final ArchiveSegmentDataProvider segmentDataProvider;

private final UserUpdatesStreamIndexer userUpdatesStreamIndexer;

private final UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer;

private final SegmentWarmer segmentWarmer;

private final EarlybirdIndexConfig earlybirdIndexConfig;

private final ZooKeeperTryLockFactory zkTryLockFactory;

private final Clock clock;

private final SegmentSyncConfig segmentSyncConfig;

protected final SearchCounter gcAfterIndexing;

// Used for coordinating daily updated across different replicas on the same hash partition,

// to run them one at a time, and minimize the impact on query latencies.

private final CoordinatedEarlybirdActionInterface coordinatedDailyUpdate;

private final SearchIndexingMetricSet indexingMetricSet;

// This is only used in tests where no coordination is needed.

@VisibleForTesting

public ArchiveSearchPartitionManager(

ZooKeeperTryLockFactory zooKeeperTryLockFactory,

QueryCacheManager queryCacheManager,

SegmentManager segmentManager,

DynamicPartitionConfig dynamicPartitionConfig,

UserUpdatesStreamIndexer userUpdatesStreamIndexer,

UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer,

SearchStatsReceiver searchStatsReceiver,

ArchiveEarlybirdIndexConfig earlybirdIndexConfig,

ScheduledExecutorServiceFactory executorServiceFactory,

ScheduledExecutorServiceFactory userUpdateIndexerScheduledExecutorFactory,

SearchIndexingMetricSet searchIndexingMetricSet,

SegmentSyncConfig syncConfig,

Clock clock,

CriticalExceptionHandler criticalExceptionHandler)

throws IOException {

this(

zooKeeperTryLockFactory,

queryCacheManager,

segmentManager,

dynamicPartitionConfig,

userUpdatesStreamIndexer,

userScrubGeoEventStreamIndexer,

searchStatsReceiver,

earlybirdIndexConfig,

null,

executorServiceFactory,

userUpdateIndexerScheduledExecutorFactory,

searchIndexingMetricSet,

syncConfig,

clock,

criticalExceptionHandler);

}

public ArchiveSearchPartitionManager(

ZooKeeperTryLockFactory zooKeeperTryLockFactory,

QueryCacheManager queryCacheManager,

SegmentManager segmentManager,

DynamicPartitionConfig dynamicPartitionConfig,

UserUpdatesStreamIndexer userUpdatesStreamIndexer,

UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer,

SearchStatsReceiver searchStatsReceiver,

ArchiveEarlybirdIndexConfig earlybirdIndexConfig,

ServerSetMember serverSetMember,

ScheduledExecutorServiceFactory executorServiceFactory,

ScheduledExecutorServiceFactory userUpdateIndexerExecutorFactory,

SearchIndexingMetricSet searchIndexingMetricSet,

SegmentSyncConfig syncConfig,

Clock clock,

CriticalExceptionHandler criticalExceptionHandler) throws IOException {

super(queryCacheManager, segmentManager, dynamicPartitionConfig, executorServiceFactory,

searchIndexingMetricSet, searchStatsReceiver, criticalExceptionHandler);

Preconditions.checkState(syncConfig.getScrubGen().isPresent());

Date scrubGen = ScrubGenUtil.parseScrubGenToDate(syncConfig.getScrubGen().get());

this.zkTryLockFactory = zooKeeperTryLockFactory;

final DailyStatusBatches dailyStatusBatches = new DailyStatusBatches(

zkTryLockFactory,

scrubGen);

this.earlybirdIndexConfig = earlybirdIndexConfig;

PartitionConfig curPartitionConfig = dynamicPartitionConfig.getCurrentPartitionConfig();

this.indexingMetricSet = searchIndexingMetricSet;

this.timeSlicer = new ArchiveTimeSlicer(

EarlybirdConfig.getMaxSegmentSize(), dailyStatusBatches,

curPartitionConfig.getTierStartDate(), curPartitionConfig.getTierEndDate(),

earlybirdIndexConfig);

this.segmentDataProvider =

new ArchiveSegmentDataProvider(

dynamicPartitionConfig,

timeSlicer,

this.earlybirdIndexConfig);

this.userUpdatesStreamIndexer = userUpdatesStreamIndexer;

this.userScrubGeoEventStreamIndexer = userScrubGeoEventStreamIndexer;

this.coordinatedDailyUpdate = new CoordinatedEarlybirdAction(

zkTryLockFactory,

"archive\_daily\_update",

dynamicPartitionConfig,

serverSetMember,

criticalExceptionHandler,

syncConfig);

this.segmentWarmer = new SegmentWarmer(criticalExceptionHandler);

this.clock = clock;

this.segmentSyncConfig = syncConfig;

this.gcAfterIndexing = SearchCounter.export("gc\_after\_indexing");

}

@Override

public SegmentDataProvider getSegmentDataProvider() {

return segmentDataProvider;

}

@Override

protected void startUp() throws Exception {

LOG.info("Using CompleteSegmentManager to index complete segments.");

// deferring handling of multi-segment term dictionary for the archive.

// SEARCH-11952

CompleteSegmentManager completeSegmentManager = new CompleteSegmentManager(

zkTryLockFactory,

segmentDataProvider,

userUpdatesStreamIndexer,

userScrubGeoEventStreamIndexer,

segmentManager,

null,

indexingMetricSet,

clock,

MultiSegmentTermDictionaryManager.NOOP\_INSTANCE,

segmentSyncConfig,

criticalExceptionHandler);

completeSegmentManager.indexUserEvents();

completeSegmentManager.indexCompleteSegments(

() -> segmentManager.getSegmentInfos(Filter.NeedsIndexing, Order.OLD\_TO\_NEW));

// In the archive cluster, the current segment needs to be loaded too.

List<SegmentInfo> allSegments =

Lists.newArrayList(segmentManager.getSegmentInfos(Filter.All, Order.OLD\_TO\_NEW));

completeSegmentManager.loadCompleteSegments(allSegments);

completeSegmentManager.buildMultiSegmentTermDictionary();

completeSegmentManager.warmSegments(allSegments);

LOG.info("Starting to run UserUpdatesKafkaConsumer");

new Thread(userUpdatesStreamIndexer::run, "userupdates-stream-indexer").start();

if (EarlybirdConfig.consumeUserScrubGeoEvents()) {

LOG.info("Starting to run UserScrubGeoEventKafkaConsumer");

new Thread(userScrubGeoEventStreamIndexer::run,

"userScrubGeoEvent-stream-indexer").start();

}

}

private static List<ArchiveTimeSlice> truncateSegmentList(List<ArchiveTimeSlice> segmentList,

int maxNumSegments) {

// Maybe cut-off the beginning of the sorted list of IDs.

if (maxNumSegments > 0 && maxNumSegments < segmentList.size()) {

return segmentList.subList(segmentList.size() - maxNumSegments, segmentList.size());

} else {

return segmentList;

}

}

@Override

protected void indexingLoop(boolean firstLoop) throws Exception {

if (firstLoop) {

EarlybirdStatus.beginEvent(

INDEX\_CURRENT\_SEGMENT, getSearchIndexingMetricSet().startupInCurrentSegment);

}

List<ArchiveTimeSlice> timeSlices = timeSlicer.getTimeSlicesInTierRange();

PartitionConfig curPartitionConfig = dynamicPartitionConfig.getCurrentPartitionConfig();

timeSlices = truncateSegmentList(timeSlices, curPartitionConfig.getMaxEnabledLocalSegments());

for (final ArchiveTimeSlice timeSlice : timeSlices) {

// If any timeslice build failed, do not try to build timeslice after that to prevent

// possible holes between timeslices.

try {

if (!processArchiveTimeSlice(timeSlice)) {

LOG.warn("Building timeslice {} has failed, stopping future builds.",

timeSlice.getDescription());

indexingMetricSet.archiveTimeSliceBuildFailedCounter.increment();

return;

}

} catch (CoordinatedEarlybirdActionLockFailed e) {

// If the timeslice build failed because of lock coordination, we can wait for the next

// iteration to build again.

return;

}

}

if (firstLoop) {

EarlybirdStatus.endEvent(

INDEX\_CURRENT\_SEGMENT, getSearchIndexingMetricSet().startupInCurrentSegment);

LOG.info("First indexing loop complete. Setting up query cache...");

EarlybirdStatus.beginEvent(

SETUP\_QUERY\_CACHE, getSearchIndexingMetricSet().startupInQueryCacheUpdates);

}

setupQueryCacheIfNeeded();

if (EarlybirdStatus.isStarting() && queryCacheManager.allTasksRan()) {

LOG.info("Query cache setup complete. Becoming current now...");

EarlybirdStatus.endEvent(

SETUP\_QUERY\_CACHE, getSearchIndexingMetricSet().startupInQueryCacheUpdates);

becomeCurrent();

EarlybirdStatus.recordEarlybirdEvent("Archive Earlybird is current");

}

updateIndexFreshnessStats(timeSlices);

}

@VisibleForTesting

protected boolean processArchiveTimeSlice(final ArchiveTimeSlice timeSlice)

throws CoordinatedEarlybirdActionLockFailed, IOException {

PartitionConfig curPartitionConfig = dynamicPartitionConfig.getCurrentPartitionConfig();

long minStatusID = timeSlice.getMinStatusID(curPartitionConfig.getIndexingHashPartitionID());

SegmentInfo segmentInfo = segmentManager.getSegmentInfo(minStatusID);

if (segmentInfo == null) {

return indexSegmentFromScratch(timeSlice);

} else if (existingSegmentNeedsUpdating(timeSlice, segmentInfo)) {

return indexNewDayAndAppendExistingSegment(timeSlice, segmentInfo);

}

return true;

}

@VisibleForTesting

SegmentInfo newSegmentInfo(ArchiveTimeSlice timeSlice) throws IOException {

return new SegmentInfo(segmentDataProvider.newArchiveSegment(timeSlice),

segmentManager.getEarlybirdSegmentFactory(), segmentSyncConfig);

}

private boolean indexNewDayAndAppendExistingSegment(final ArchiveTimeSlice timeSlice,

SegmentInfo segmentInfo)

throws CoordinatedEarlybirdActionLockFailed, IOException {

LOG.info("Updating segment: {}; new endDate will be {} segmentInfo: {}",

segmentInfo.getSegment().getTimeSliceID(), timeSlice.getEndDate(), segmentInfo);

// Create another new SegmentInfo for indexing

final SegmentInfo newSegmentInfoForIndexing = newSegmentInfo(timeSlice);

// make a final reference of the old segment info to be passed into closure.

final SegmentInfo oldSegmentInfo = segmentInfo;

// Sanity check: the old and new segment should not share the same lucene directory.

Preconditions.checkState(

!newSegmentInfoForIndexing.getSyncInfo().getLocalLuceneSyncDir().equals(

oldSegmentInfo.getSyncInfo().getLocalLuceneSyncDir()));

Preconditions.checkState(

!newSegmentInfoForIndexing.getSyncInfo().getLocalSyncDir().equals(

oldSegmentInfo.getSyncInfo().getLocalSyncDir()));

final ArchiveSegment oldSegment = (ArchiveSegment) segmentInfo.getSegment();

return indexSegment(newSegmentInfoForIndexing, oldSegmentInfo, input -> {

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

// we're in the on-disk archive, and we're sure that the previous days have already

// been indexed.

return !earlybirdIndexConfig.isIndexStoredOnDisk()

// First time around, and the segment has not been indexed and optimized yet,

// we will want to add all the days

|| !oldSegmentInfo.isOptimized()

|| oldSegmentInfo.getIndexSegment().getIndexStats().getStatusCount() == 0

|| !oldSegment.getDataEndDate().before(timeSlice.getEndDate())

// Index any new days

|| input.after(oldSegment.getDataEndDate());

});

}

private boolean existingSegmentNeedsUpdating(ArchiveTimeSlice timeSlice,

SegmentInfo segmentInfo) {

return ((ArchiveSegment) segmentInfo.getSegment())

.getDataEndDate().before(timeSlice.getEndDate())

// First time around, the end date is the same as the timeSlice end date, but

// the segment has not been indexed and optimized yet

|| (!segmentInfo.isOptimized() && !segmentInfo.wasIndexed())

// If indexing failed, this index will not be marked as complete, and we will want

// to reindex

|| !segmentInfo.isComplete();

}

private boolean indexSegmentFromScratch(ArchiveTimeSlice timeSlice) throws

CoordinatedEarlybirdActionLockFailed, IOException {

SegmentInfo segmentInfo = newSegmentInfo(timeSlice);

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

+ "; new endDate will be " + timeSlice.getEndDate() + " segmentInfo: " + segmentInfo);

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

}

private void updateIndexFreshnessStats(List<ArchiveTimeSlice> timeSlices) {

if (!timeSlices.isEmpty()) {

ArchiveTimeSlice lastTimeslice = timeSlices.get(timeSlices.size() - 1);

// Add ~24 hours to start of end date to estimate freshest tweet time.

indexingMetricSet.freshestTweetTimeMillis.set(

lastTimeslice.getEndDate().getTime() + ONE\_DAY\_MILLIS);

PartitionConfig curPartitionConfig = dynamicPartitionConfig.getCurrentPartitionConfig();

long maxStatusId = lastTimeslice.getMaxStatusID(

curPartitionConfig.getIndexingHashPartitionID());

if (maxStatusId > indexingMetricSet.highestStatusId.get()) {

indexingMetricSet.highestStatusId.set(maxStatusId);

}

}

}

@Override

public void shutDownIndexing() {

LOG.info("Shutting down.");

userUpdatesStreamIndexer.close();

userScrubGeoEventStreamIndexer.close();

LOG.info("Closed User Event Kafka Consumers. Now Shutting down reader set.");

getSegmentDataProvider().getSegmentDataReaderSet().stopAll();

}

/\*\*

\* Attempts to index new days of data into the provided segment, indexing only the days that

\* match the "dateFilter" predicate.

\* @return true iff indexing succeeded, false otherwise.

\*/

@VisibleForTesting

protected boolean indexSegment(final SegmentInfo segmentInfo,

@Nullable final SegmentInfo segmentToAppend,

final Predicate<Date> dateFilter)

throws CoordinatedEarlybirdActionLockFailed, IOException {

// Don't coordinate while we're starting up

if (!EarlybirdStatus.isStarting()) {

return coordinatedDailyUpdate.execute(segmentInfo.getSegmentName(),

isCoordinated -> innerIndexSegment(segmentInfo, segmentToAppend, dateFilter));

} else {

return innerIndexSegment(segmentInfo, segmentToAppend, dateFilter);

}

}

private boolean innerIndexSegment(SegmentInfo segmentInfo,

@Nullable SegmentInfo segmentToAppend,

Predicate<Date> dateFilter)

throws IOException {

// First try to load the new day from HDFS / Local disk

if (new SegmentLoader(segmentSyncConfig, criticalExceptionHandler).load(segmentInfo)) {

LOG.info("Successful loaded segment for new day: " + segmentInfo);

segmentManager.putSegmentInfo(segmentInfo);

gcAfterIndexing.increment();

GCUtil.runGC();

return true;

}

LOG.info("Failed to load segment for new day. Will index segment: " + segmentInfo);

RecordReader<TweetDocument> tweetReader = ((ArchiveSegment) segmentInfo.getSegment())

.getStatusRecordReader(earlybirdIndexConfig.createDocumentFactory(), dateFilter);

try {

// Read and index the statuses

boolean success = newSimpleSegmentIndexer(tweetReader, segmentToAppend)

.indexSegment(segmentInfo);

if (!success) {

return false;

}

} finally {

tweetReader.stop();

}

if (!SegmentOptimizer.optimize(segmentInfo)) {

// We consider the whole indexing event as failed if we fail to optimize.

LOG.error("Failed to optimize segment: " + segmentInfo);

segmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

return false;

}

if (!segmentWarmer.warmSegmentIfNecessary(segmentInfo)) {

// We consider the whole indexing event as failed if we failed to warm (because we open

// index readers in the warmer).

LOG.error("Failed to warm segment: " + segmentInfo);

segmentInfo.deleteLocalIndexedSegmentDirectoryImmediately();

return false;

}

// Flush and upload segment to HDFS. If this fails, we just log a warning and return true.

boolean success = new SegmentHdfsFlusher(zkTryLockFactory, segmentSyncConfig)

.flushSegmentToDiskAndHDFS(segmentInfo);

if (!success) {

LOG.warn("Failed to flush segment to HDFS: " + segmentInfo);

}

segmentManager.putSegmentInfo(segmentInfo);

gcAfterIndexing.increment();

GCUtil.runGC();

return true;

}

@VisibleForTesting

protected SimpleSegmentIndexer newSimpleSegmentIndexer(

RecordReader<TweetDocument> tweetReader, SegmentInfo segmentToAppend) {

return new SimpleSegmentIndexer(tweetReader, indexingMetricSet, segmentToAppend);

}

@Override

public boolean isCaughtUpForTests() {

return EarlybirdStatus.getStatusCode() == EarlybirdStatusCode.CURRENT;

}

public CoordinatedEarlybirdActionInterface getCoordinatedOptimizer() {

return this.coordinatedDailyUpdate;

}

public ArchiveTimeSlicer getTimeSlicer() {

return timeSlicer;

}

}