package com.twitter.search.earlybird.partition;

import java.io.IOException;

import java.util.Iterator;

import java.util.List;

import java.util.function.Supplier;

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.indexing.thriftjava.ThriftVersionedEvents;

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

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

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

import com.twitter.search.earlybird.EarlybirdStatus;

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

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

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

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

/\*\*

\* CompleteSegmentManager is used to parallelize indexing of complete (not partial) segments

\* on startup. It also populates the fields used by the PartitionManager.

\*/

public class CompleteSegmentManager {

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

private static final String INDEX\_COMPLETED\_SEGMENTS =

"indexing, optimizing and flushing complete segments";

private static final String LOAD\_COMPLETED\_SEGMENTS = "loading complete segments";

private static final String INDEX\_UPDATES\_FOR\_COMPLETED\_SEGMENTS =

"indexing updates for complete segments";

private static final String BUILD\_MULTI\_SEGMENT\_TERM\_DICT =

"build multi segment term dictionaries";

// Max number of segments being loaded / indexed concurrently.

private final int maxConcurrentSegmentIndexers =

EarlybirdProperty.MAX\_CONCURRENT\_SEGMENT\_INDEXERS.get(3);

// The state we are building.

protected final SegmentDataProvider segmentDataProvider;

private final InstrumentedQueue<ThriftVersionedEvents> retryQueue;

private final UserUpdatesStreamIndexer userUpdatesStreamIndexer;

private final UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer;

private final SegmentManager segmentManager;

private final ZooKeeperTryLockFactory zkTryLockFactory;

private final SearchIndexingMetricSet searchIndexingMetricSet;

private final Clock clock;

private MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager;

private final SegmentSyncConfig segmentSyncConfig;

private final CriticalExceptionHandler criticalExceptionHandler;

private boolean interrupted = false;

public CompleteSegmentManager(

ZooKeeperTryLockFactory zooKeeperTryLockFactory,

SegmentDataProvider segmentDataProvider,

UserUpdatesStreamIndexer userUpdatesStreamIndexer,

UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer,

SegmentManager segmentManager,

InstrumentedQueue<ThriftVersionedEvents> retryQueue,

SearchIndexingMetricSet searchIndexingMetricSet,

Clock clock,

MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager,

SegmentSyncConfig segmentSyncConfig,

CriticalExceptionHandler criticalExceptionHandler) {

this.zkTryLockFactory = zooKeeperTryLockFactory;

this.segmentDataProvider = segmentDataProvider;

this.userUpdatesStreamIndexer = userUpdatesStreamIndexer;

this.userScrubGeoEventStreamIndexer = userScrubGeoEventStreamIndexer;

this.segmentManager = segmentManager;

this.searchIndexingMetricSet = searchIndexingMetricSet;

this.clock = clock;

this.multiSegmentTermDictionaryManager = multiSegmentTermDictionaryManager;

this.segmentSyncConfig = segmentSyncConfig;

this.retryQueue = retryQueue;

this.criticalExceptionHandler = criticalExceptionHandler;

}

/\*\*

\* Indexes all user events.

\*/

public void indexUserEvents() {

LOG.info("Loading/indexing user events.");

StartupUserEventIndexer startupUserEventIndexer = new StartupUserEventIndexer(

searchIndexingMetricSet,

userUpdatesStreamIndexer,

userScrubGeoEventStreamIndexer,

segmentManager,

clock

);

startupUserEventIndexer.indexAllEvents();

LOG.info("Finished loading/indexing user events.");

}

/\*\*

\* Loads or indexes from scratch all complete segments.

\*

\* @param segmentsToIndexProvider A supplier that provides the list of all complete segments.

\*/

public void indexCompleteSegments(

Supplier<Iterable<SegmentInfo>> segmentsToIndexProvider) throws Exception {

List<Thread> segmentIndexers = Lists.newArrayList();

EarlybirdStatus.beginEvent(

INDEX\_COMPLETED\_SEGMENTS, searchIndexingMetricSet.startupInIndexCompletedSegments);

while (!interrupted && !Thread.currentThread().isInterrupted()) {

try {

// Get the refreshed list of local segment databases.

segmentManager.updateSegments(segmentDataProvider.newSegmentList());

Iterator<SegmentInfo> segmentsToIndex = segmentsToIndexProvider.get().iterator();

// Start up to max concurrent segment indexers.

segmentIndexers.clear();

while (segmentsToIndex.hasNext() && segmentIndexers.size() < maxConcurrentSegmentIndexers) {

SegmentInfo nextSegment = segmentsToIndex.next();

if (!nextSegment.isComplete()) {

Thread thread = new Thread(new SingleSegmentIndexer(nextSegment),

"startup-segment-indexer-" + nextSegment.getSegmentName());

thread.start();

segmentIndexers.add(thread);

}

}

// No remaining indexer threads, we're done.

if (segmentIndexers.size() == 0) {

LOG.info("Finished indexing complete segments");

EarlybirdStatus.endEvent(

INDEX\_COMPLETED\_SEGMENTS, searchIndexingMetricSet.startupInIndexCompletedSegments);

break;

}

// Wait for threads to complete fully.

LOG.info("Started {} indexing threads", segmentIndexers.size());

for (Thread thread : segmentIndexers) {

thread.join();

}

LOG.info("Joined all {} indexing threads", segmentIndexers.size());

} catch (IOException e) {

LOG.error("IOException in SegmentStartupManager loop", e);

} catch (InterruptedException e) {

interrupted = true;

LOG.error("Interrupted joining segment indexer thread", e);

}

}

}

/\*\*

\* Loads all given complete segments.

\*

\* @param completeSegments The list of all complete segments to be loaded.

\*/

public void loadCompleteSegments(List<SegmentInfo> completeSegments) throws Exception {

if (!interrupted && !Thread.currentThread().isInterrupted()) {

LOG.info("Starting to load {} complete segments.", completeSegments.size());

EarlybirdStatus.beginEvent(

LOAD\_COMPLETED\_SEGMENTS, searchIndexingMetricSet.startupInLoadCompletedSegments);

List<Thread> segmentThreads = Lists.newArrayList();

List<SegmentInfo> segmentsToBeLoaded = Lists.newArrayList();

for (SegmentInfo segmentInfo : completeSegments) {

if (segmentInfo.isEnabled()) {

segmentsToBeLoaded.add(segmentInfo);

Thread segmentLoaderThread = new Thread(

() -> new SegmentLoader(segmentSyncConfig, criticalExceptionHandler)

.load(segmentInfo),

"startup-segment-loader-" + segmentInfo.getSegmentName());

segmentThreads.add(segmentLoaderThread);

segmentLoaderThread.start();

} else {

LOG.info("Will not load segment {} because it's disabled.", segmentInfo.getSegmentName());

}

}

for (Thread segmentLoaderThread : segmentThreads) {

segmentLoaderThread.join();

}

for (SegmentInfo segmentInfo : segmentsToBeLoaded) {

if (!segmentInfo.getSyncInfo().isLoaded()) {

// Throw an exception if a segment could not be loaded: We do not want earlybirds to

// startup with missing segments.

throw new RuntimeException("Could not load segment " + segmentInfo.getSegmentName());

}

}

LOG.info("Loaded all complete segments, starting indexing all updates.");

EarlybirdStatus.beginEvent(

INDEX\_UPDATES\_FOR\_COMPLETED\_SEGMENTS,

searchIndexingMetricSet.startupInIndexUpdatesForCompletedSegments);

// Index all updates for all complete segments until we're fully caught up.

if (!EarlybirdCluster.isArchive(segmentManager.getEarlybirdIndexConfig().getCluster())) {

segmentThreads.clear();

for (SegmentInfo segmentInfo : completeSegments) {

if (segmentInfo.isEnabled()) {

Thread segmentUpdatesThread = new Thread(

() -> new SimpleUpdateIndexer(

segmentDataProvider.getSegmentDataReaderSet(),

searchIndexingMetricSet,

retryQueue,

criticalExceptionHandler).indexAllUpdates(segmentInfo),

"startup-complete-segment-update-indexer-" + segmentInfo.getSegmentName());

segmentThreads.add(segmentUpdatesThread);

segmentUpdatesThread.start();

} else {

LOG.info("Will not index updates for segment {} because it's disabled.",

segmentInfo.getSegmentName());

}

}

for (Thread segmentUpdatesThread : segmentThreads) {

segmentUpdatesThread.join();

}

}

LOG.info("Indexed updates for all complete segments.");

EarlybirdStatus.endEvent(

INDEX\_UPDATES\_FOR\_COMPLETED\_SEGMENTS,

searchIndexingMetricSet.startupInIndexUpdatesForCompletedSegments);

EarlybirdStatus.endEvent(

LOAD\_COMPLETED\_SEGMENTS, searchIndexingMetricSet.startupInLoadCompletedSegments);

}

}

/\*\*

\* Builds the term dictionary that spans all earlybird segments. Some fields share the term

\* dictionary across segments as an optimization.

\*/

public void buildMultiSegmentTermDictionary() {

EarlybirdStatus.beginEvent(

BUILD\_MULTI\_SEGMENT\_TERM\_DICT,

searchIndexingMetricSet.startupInMultiSegmentTermDictionaryUpdates);

if (!interrupted && !Thread.currentThread().isInterrupted()) {

LOG.info("Building multi segment term dictionaries.");

boolean built = multiSegmentTermDictionaryManager.buildDictionary();

LOG.info("Done building multi segment term dictionaries, result: {}", built);

}

EarlybirdStatus.endEvent(

BUILD\_MULTI\_SEGMENT\_TERM\_DICT,

searchIndexingMetricSet.startupInMultiSegmentTermDictionaryUpdates);

}

/\*\*

\* Warms up the data in the given segments. The warm up will usually make sure that all necessary

\* is loaded in RAM and all relevant data structures are created before the segments starts

\* serving real requests.

\*

\* @param segments The list of segments to warm up.

\*/

public final void warmSegments(Iterable<SegmentInfo> segments) throws InterruptedException {

int threadId = 1;

Iterator<SegmentInfo> it = segments.iterator();

try {

List<Thread> segmentWarmers = Lists.newLinkedList();

while (it.hasNext()) {

segmentWarmers.clear();

while (it.hasNext() && segmentWarmers.size() < maxConcurrentSegmentIndexers) {

final SegmentInfo segment = it.next();

Thread t = new Thread(() ->

new SegmentWarmer(criticalExceptionHandler).warmSegmentIfNecessary(segment),

"startup-warmer-" + threadId++);

t.start();

segmentWarmers.add(t);

}

for (Thread t : segmentWarmers) {

t.join();

}

}

} catch (InterruptedException e) {

LOG.error("Interrupted segment warmer thread", e);

Thread.currentThread().interrupt();

throw e;

}

}

/\*\*

\* Indexes a complete segment.

\*/

private class SingleSegmentIndexer implements Runnable {

private final SegmentInfo segmentInfo;

public SingleSegmentIndexer(SegmentInfo segmentInfo) {

this.segmentInfo = segmentInfo;

}

@Override

public void run() {

// 0) Check if the segment can be loaded. This might copy the segment from HDFS.

if (new SegmentLoader(segmentSyncConfig, criticalExceptionHandler)

.downloadSegment(segmentInfo)) {

LOG.info("Will not index segment {} because it was downloaded from HDFS.",

segmentInfo.getSegmentName());

segmentInfo.setComplete(true);

return;

}

LOG.info("SingleSegmentIndexer starting for segment: " + segmentInfo);

// 1) Index all tweets in this segment.

RecordReader<TweetDocument> tweetReader;

try {

tweetReader = segmentDataProvider.getSegmentDataReaderSet().newDocumentReader(segmentInfo);

if (tweetReader != null) {

tweetReader.setExhaustStream(true);

}

} catch (Exception e) {

throw new RuntimeException("Could not create tweet reader for segment: " + segmentInfo, e);

}

new SimpleSegmentIndexer(tweetReader, searchIndexingMetricSet).indexSegment(segmentInfo);

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

throw new RuntimeException("Segment does not appear to be complete: " + segmentInfo);

}

// 2) Index all updates in this segment (archive earlybirds don't have updates).

if (!EarlybirdCluster.isArchive(segmentManager.getEarlybirdIndexConfig().getCluster())) {

new SimpleUpdateIndexer(

segmentDataProvider.getSegmentDataReaderSet(),

searchIndexingMetricSet,

retryQueue,

criticalExceptionHandler).indexAllUpdates(segmentInfo);

}

// 3) Optimize the segment.

SegmentOptimizer.optimize(segmentInfo);

// 4) Flush to HDFS if necessary.

new SegmentHdfsFlusher(zkTryLockFactory, segmentSyncConfig)

.flushSegmentToDiskAndHDFS(segmentInfo);

// 5) Unload the segment from memory.

segmentInfo.getIndexSegment().close();

}

}

}