package com.twitter.search.earlybird.partition;

import java.io.Closeable;

import java.util.Optional;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Stopwatch;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.config.Config;

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

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

import com.twitter.search.earlybird.EarlybirdStatus;

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

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

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

import com.twitter.search.earlybird.partition.freshstartup.FreshStartupHandler;

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

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

import com.twitter.search.queryparser.query.QueryParserException;

/\*\*

\* Handles starting an Earlybird from Kafka topics.

\*

\* Currently very unoptimized -- future versions will implement parallel indexing and loading

\* serialized data from HDFS. See http://go/removing-dl-tdd.

\*/

public class KafkaStartup implements EarlybirdStartup {

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

private final EarlybirdKafkaConsumer earlybirdKafkaConsumer;

private final StartupUserEventIndexer startupUserEventIndexer;

private final QueryCacheManager queryCacheManager;

private final SegmentManager segmentManager;

private final EarlybirdIndexLoader earlybirdIndexLoader;

private final FreshStartupHandler freshStartupHandler;

private final UserUpdatesStreamIndexer userUpdatesStreamIndexer;

private final UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer;

private final SearchIndexingMetricSet searchIndexingMetricSet;

private final SearchLongGauge loadedIndex;

private final SearchLongGauge freshStartup;

private final MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager;

private final AudioSpaceEventsStreamIndexer audioSpaceEventsStreamIndexer;

private final CriticalExceptionHandler earlybirdExceptionHandler;

private final SearchDecider decider;

private static final String FRESH\_STARTUP = "fresh startup";

private static final String INGEST\_UNTIL\_CURRENT = "ingest until current";

private static final String LOAD\_FLUSHED\_INDEX = "load flushed index";

private static final String SETUP\_QUERY\_CACHE = "setting up query cache";

private static final String USER\_UPDATES\_STARTUP = "user updates startup";

private static final String AUDIO\_SPACES\_STARTUP = "audio spaces startup";

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

"build multi segment term dictionary";

public KafkaStartup(

SegmentManager segmentManager,

EarlybirdKafkaConsumer earlybirdKafkaConsumer,

StartupUserEventIndexer startupUserEventIndexer,

UserUpdatesStreamIndexer userUpdatesStreamIndexer,

UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer,

AudioSpaceEventsStreamIndexer audioSpaceEventsStreamIndexer,

QueryCacheManager queryCacheManager,

EarlybirdIndexLoader earlybirdIndexLoader,

FreshStartupHandler freshStartupHandler,

SearchIndexingMetricSet searchIndexingMetricSet,

MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager,

CriticalExceptionHandler earlybirdExceptionHandler,

SearchDecider decider

) {

this.segmentManager = segmentManager;

this.earlybirdKafkaConsumer = earlybirdKafkaConsumer;

this.startupUserEventIndexer = startupUserEventIndexer;

this.queryCacheManager = queryCacheManager;

this.earlybirdIndexLoader = earlybirdIndexLoader;

this.freshStartupHandler = freshStartupHandler;

this.userUpdatesStreamIndexer = userUpdatesStreamIndexer;

this.userScrubGeoEventStreamIndexer = userScrubGeoEventStreamIndexer;

this.audioSpaceEventsStreamIndexer = audioSpaceEventsStreamIndexer;

this.searchIndexingMetricSet = searchIndexingMetricSet;

this.loadedIndex = SearchLongGauge.export("kafka\_startup\_loaded\_index");

this.freshStartup = SearchLongGauge.export("fresh\_startup");

this.multiSegmentTermDictionaryManager = multiSegmentTermDictionaryManager;

this.earlybirdExceptionHandler = earlybirdExceptionHandler;

this.decider = decider;

freshStartup.set(0);

}

private void userEventsStartup() {

LOG.info("Start indexing user events.");

startupUserEventIndexer.indexAllEvents();

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

// User updates are now current, keep them current by continuing to index from the stream.

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

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

if (EarlybirdConfig.consumeUserScrubGeoEvents()) {

// User scrub geo events are now current,

// keep them current by continuing to index from the stream.

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

new Thread(userScrubGeoEventStreamIndexer::run,

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

}

}

private void loadAudioSpaceEvents() {

LOG.info("Index audio space events...");

EarlybirdStatus.beginEvent(AUDIO\_SPACES\_STARTUP,

searchIndexingMetricSet.startupInAudioSpaceEventIndexer);

if (audioSpaceEventsStreamIndexer == null) {

LOG.error("Null audioSpaceEventsStreamIndexer");

return;

}

if (decider.isAvailable("enable\_reading\_audio\_space\_events")) {

Stopwatch stopwatch = Stopwatch.createStarted();

audioSpaceEventsStreamIndexer.seekToBeginning();

audioSpaceEventsStreamIndexer.readRecordsUntilCurrent();

LOG.info("Finished reading audio spaces in {}", stopwatch);

audioSpaceEventsStreamIndexer.printSummary();

new Thread(audioSpaceEventsStreamIndexer::run,

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

} else {

LOG.info("Reading audio space events not enabled");

}

EarlybirdStatus.endEvent(AUDIO\_SPACES\_STARTUP,

searchIndexingMetricSet.startupInAudioSpaceEventIndexer);

}

private void tweetsAndUpdatesStartup() throws EarlybirdStartupException {

LOG.info("Index tweets and updates...");

EarlybirdStatus.beginEvent(LOAD\_FLUSHED\_INDEX,

searchIndexingMetricSet.startupInLoadFlushedIndex);

EarlybirdIndex index;

// Set when you want to get a server from starting to ready quickly for development

// purposes.

boolean fastDevStartup = EarlybirdConfig.getBool("fast\_dev\_startup");

Optional<EarlybirdIndex> optIndex = Optional.empty();

if (!fastDevStartup) {

optIndex = earlybirdIndexLoader.loadIndex();

}

if (optIndex.isPresent()) {

loadedIndex.set(1);

LOG.info("Loaded an index.");

index = optIndex.get();

EarlybirdStatus.endEvent(LOAD\_FLUSHED\_INDEX,

searchIndexingMetricSet.startupInLoadFlushedIndex);

} else {

LOG.info("Didn't load an index, indexing from scratch.");

freshStartup.set(1);

boolean parallelIndexFromScratch = EarlybirdConfig.getBool(

"parallel\_index\_from\_scratch");

LOG.info("parallel\_index\_from\_scratch: {}", parallelIndexFromScratch);

EarlybirdStatus.beginEvent(FRESH\_STARTUP,

searchIndexingMetricSet.startupInFreshStartup);

try {

if (fastDevStartup) {

index = freshStartupHandler.fastIndexFromScratchForDevelopment();

} else if (parallelIndexFromScratch) {

index = freshStartupHandler.parallelIndexFromScratch();

} else {

index = freshStartupHandler.indexFromScratch();

}

} catch (Exception ex) {

throw new EarlybirdStartupException(ex);

} finally {

EarlybirdStatus.endEvent(FRESH\_STARTUP,

searchIndexingMetricSet.startupInFreshStartup);

}

}

LOG.info("Index has {} segments.", index.getSegmentInfoList().size());

if (index.getSegmentInfoList().size() > 0) {

LOG.info("Inserting segments into SegmentManager");

for (SegmentInfo segmentInfo : index.getSegmentInfoList()) {

segmentManager.putSegmentInfo(segmentInfo);

}

earlybirdKafkaConsumer.prepareAfterStartingWithIndex(

index.getMaxIndexedTweetId()

);

}

// Build the Multi segment term dictionary before catching up on indexing to ensure that the

// segments won't roll and delete the oldest segment while a multi segment term dictionary that

// includes that segment is being built.

buildMultiSegmentTermDictionary();

segmentManager.logState("Starting ingestUntilCurrent");

LOG.info("partial updates indexed: {}", segmentManager.getNumPartialUpdates());

EarlybirdStatus.beginEvent(INGEST\_UNTIL\_CURRENT,

searchIndexingMetricSet.startupInIngestUntilCurrent);

earlybirdKafkaConsumer.ingestUntilCurrent(index.getTweetOffset(), index.getUpdateOffset());

validateSegments();

segmentManager.logState("ingestUntilCurrent is done");

LOG.info("partial updates indexed: {}", segmentManager.getNumPartialUpdates());

EarlybirdStatus.endEvent(INGEST\_UNTIL\_CURRENT,

searchIndexingMetricSet.startupInIngestUntilCurrent);

new Thread(earlybirdKafkaConsumer::run, "earlybird-kafka-consumer").start();

}

protected void validateSegments() throws EarlybirdStartupException {

if (!Config.environmentIsTest()) {

// Unfortunately, many tests start Earlybirds with 0 indexed documents, so we disable this

// check in tests.

validateSegmentsForNonTest();

}

}

protected void validateSegmentsForNonTest() throws EarlybirdStartupException {

// SEARCH-24123: Prevent Earlybird from starting if there are no indexed documents.

if (segmentManager.getNumIndexedDocuments() == 0) {

throw new EarlybirdStartupException("Earlybird has zero indexed documents.");

}

}

private void queryCacheStartup() throws EarlybirdStartupException {

EarlybirdStatus.beginEvent(SETUP\_QUERY\_CACHE,

searchIndexingMetricSet.startupInQueryCacheUpdates);

try {

queryCacheManager.setupTasksIfNeeded(segmentManager);

} catch (QueryParserException e) {

LOG.error("Exception when setting up query cache tasks");

throw new EarlybirdStartupException(e);

}

queryCacheManager.waitUntilAllQueryCachesAreBuilt();

// Print the sizes of the query caches so that we can see that they're built.

Iterable<SegmentInfo> segmentInfos =

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

segmentManager.logState("After building query caches");

for (SegmentInfo segmentInfo : segmentInfos) {

LOG.info("Segment: {}, Total cardinality: {}", segmentInfo.getSegmentName(),

segmentInfo.getIndexSegment().getQueryCachesCardinality());

}

// We're done building the query caches for all segments, and the earlybird is ready to become

// current. Restrict all future query cache task runs to one single core, to make sure our

// searcher threads are not impacted.

queryCacheManager.setWorkerPoolSizeAfterStartup();

EarlybirdStatus.endEvent(SETUP\_QUERY\_CACHE,

searchIndexingMetricSet.startupInQueryCacheUpdates);

}

/\*\*

\* Closes all currently running Indexers.

\*/

@VisibleForTesting

public void shutdownIndexing() {

LOG.info("Shutting down KafkaStartup.");

earlybirdKafkaConsumer.close();

userUpdatesStreamIndexer.close();

userScrubGeoEventStreamIndexer.close();

// Note that the QueryCacheManager is shut down in EarlybirdServer::shutdown.

}

private void buildMultiSegmentTermDictionary() {

EarlybirdStatus.beginEvent(BUILD\_MULTI\_SEGMENT\_TERM\_DICTIONARY,

searchIndexingMetricSet.startupInMultiSegmentTermDictionaryUpdates);

Stopwatch stopwatch = Stopwatch.createStarted();

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

multiSegmentTermDictionaryManager.buildDictionary();

LOG.info("Done with building multi segment term dictionary in {}", stopwatch);

EarlybirdStatus.endEvent(BUILD\_MULTI\_SEGMENT\_TERM\_DICTIONARY,

searchIndexingMetricSet.startupInMultiSegmentTermDictionaryUpdates);

}

private void parallelIndexingStartup() throws EarlybirdStartupException {

Thread userEventsThread = new Thread(this::userEventsStartup, "index-user-events-startup");

Thread tweetsAndUpdatesThread = new Thread(() -> {

try {

tweetsAndUpdatesStartup();

} catch (EarlybirdStartupException e) {

earlybirdExceptionHandler.handle(this, e);

}

}, "index-tweets-and-updates-startup");

Thread audioSpaceEventsThread = new Thread(this::loadAudioSpaceEvents,

"index-audio-space-events-startup");

userEventsThread.start();

tweetsAndUpdatesThread.start();

audioSpaceEventsThread.start();

try {

userEventsThread.join();

} catch (InterruptedException e) {

throw new EarlybirdStartupException("Interrupted while indexing user events");

}

try {

tweetsAndUpdatesThread.join();

} catch (InterruptedException e) {

throw new EarlybirdStartupException("Interrupted while indexing tweets and updates");

}

try {

audioSpaceEventsThread.join();

} catch (InterruptedException e) {

throw new EarlybirdStartupException("Interrupted while indexing audio space events");

}

}

/\*\*

\* Does startups and starts indexing. Returns when the earlybird

\* is current.

\*/

@Override

public Closeable start() throws EarlybirdStartupException {

parallelIndexingStartup();

queryCacheStartup();

EarlybirdStatus.setStatus(EarlybirdStatusCode.CURRENT);

return this::shutdownIndexing;

}

}