package com.twitter.search.earlybird.partition;

import java.io.BufferedInputStream;

import java.io.IOException;

import java.time.Duration;

import java.util.List;

import java.util.Optional;

import java.util.SortedMap;

import com.google.common.base.Stopwatch;

import org.apache.commons.compress.utils.Lists;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

import com.twitter.search.common.partitioning.base.TimeSlice;

import com.twitter.search.common.util.io.flushable.DataDeserializer;

import com.twitter.search.common.util.io.flushable.FlushInfo;

import com.twitter.search.earlybird.common.NonPagingAssert;

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

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

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

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

/\*\*

\* Loads an index from HDFS, if possible, or indexes all tweets from scratch using a

\* FreshStartupHandler.

\*/

public class EarlybirdIndexLoader {

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

public static final String ENV\_FOR\_TESTS = "test\_env";

// To determine whether we should or should not load the most recent index from HDFS if available.

public static final long INDEX\_FRESHNESS\_THRESHOLD\_MILLIS = Duration.ofDays(1).toMillis();

private static final NonPagingAssert LOADING\_TOO\_MANY\_NON\_OPTIMIZED\_SEGMENTS =

new NonPagingAssert("loading\_too\_many\_non\_optimized\_segments");

private final FileSystem fileSystem;

private final Path indexPath;

private final PartitionConfig partitionConfig;

private final EarlybirdSegmentFactory earlybirdSegmentFactory;

private final SegmentSyncConfig segmentSyncConfig;

private final Clock clock;

// Aurora environment we're running in: "prod", "loadtest", "staging2" etc. etc

private final String environment;

public EarlybirdIndexLoader(

FileSystem fileSystem,

String indexHDFSPath,

String environment,

PartitionConfig partitionConfig,

EarlybirdSegmentFactory earlybirdSegmentFactory,

SegmentSyncConfig segmentSyncConfig,

Clock clock

) {

this.fileSystem = fileSystem;

this.partitionConfig = partitionConfig;

this.earlybirdSegmentFactory = earlybirdSegmentFactory;

this.segmentSyncConfig = segmentSyncConfig;

this.indexPath = EarlybirdIndexFlusher.buildPathToIndexes(indexHDFSPath, partitionConfig);

this.clock = clock;

this.environment = environment;

}

/\*\*

\* Tries to load an index from HDFS for this FlushVersion/Partition/Cluster. Returns an empty

\* option if there is no index found.

\*/

public Optional<EarlybirdIndex> loadIndex() {

try {

Optional<EarlybirdIndex> loadedIndex =

ActionLogger.call("Load index from HDFS.", this::loadFromHDFS);

if (loadedIndex.isPresent()) {

EarlybirdIndex index = loadedIndex.get();

int numOfNonOptimized = index.numOfNonOptimizedSegments();

if (numOfNonOptimized > EarlybirdIndex.MAX\_NUM\_OF\_NON\_OPTIMIZED\_SEGMENTS) {

// We should never have too many unoptimized segments. If this happens we likely have a

// bug somewhere that caused another Earlybird to flush too many unoptimized segments.

// Use NonPagingAssert to alert the oncall if this happens so they can look into it.

LOG.error("Found {} non-optimized segments when loading from disk!", numOfNonOptimized);

LOADING\_TOO\_MANY\_NON\_OPTIMIZED\_SEGMENTS.assertFailed();

// If there are too many unoptimized segments, optimize the older ones until there are

// only MAX\_NUM\_OF\_NON\_OPTIMIZED\_SEGMENTS left in the unoptimized state. The segment info

// list is always in order, so we will never try to optimize the most recent segments

// here.

int numSegmentsToOptimize =

numOfNonOptimized - EarlybirdIndex.MAX\_NUM\_OF\_NON\_OPTIMIZED\_SEGMENTS;

LOG.info("Will try to optimize {} segments", numSegmentsToOptimize);

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

if (numSegmentsToOptimize > 0 && !segmentInfo.isOptimized()) {

Stopwatch optimizationStopwatch = Stopwatch.createStarted();

LOG.info("Starting to optimize segment: {}", segmentInfo.getSegmentName());

segmentInfo.getIndexSegment().optimizeIndexes();

numSegmentsToOptimize--;

LOG.info("Optimization of segment {} finished in {}.",

segmentInfo.getSegmentName(), optimizationStopwatch);

}

}

}

int newNumOfNonOptimized = index.numOfNonOptimizedSegments();

LOG.info("Loaded {} segments. {} are unoptimized.",

index.getSegmentInfoList().size(),

newNumOfNonOptimized);

return loadedIndex;

}

} catch (Throwable e) {

LOG.error("Error loading index from HDFS, will index from scratch.", e);

}

return Optional.empty();

}

private Optional<EarlybirdIndex> loadFromHDFS() throws Exception {

SortedMap<Long, Path> pathsByTime =

EarlybirdIndexFlusher.getIndexPathsByTime(indexPath, fileSystem);

if (pathsByTime.isEmpty()) {

LOG.info("Could not load index from HDFS (path: {}), will index from scratch.", indexPath);

return Optional.empty();

}

long mostRecentIndexTimeMillis = pathsByTime.lastKey();

Path mostRecentIndexPath = pathsByTime.get(mostRecentIndexTimeMillis);

if (clock.nowMillis() - mostRecentIndexTimeMillis > INDEX\_FRESHNESS\_THRESHOLD\_MILLIS) {

LOG.info("Most recent index in HDFS (path: {}) is old, will do a fresh startup.",

mostRecentIndexPath);

return Optional.empty();

}

EarlybirdIndex index = ActionLogger.call(

"loading index from " + mostRecentIndexPath,

() -> loadIndex(mostRecentIndexPath));

return Optional.of(index);

}

private EarlybirdIndex loadIndex(Path flushPath) throws Exception {

Path indexInfoPath = flushPath.suffix("/" + EarlybirdIndexFlusher.INDEX\_INFO);

FlushInfo indexInfo;

try (FSDataInputStream infoInputStream = fileSystem.open(indexInfoPath)) {

indexInfo = FlushInfo.loadFromYaml(infoInputStream);

}

FlushInfo segmentsFlushInfo = indexInfo.getSubProperties(EarlybirdIndexFlusher.SEGMENTS);

List<String> segmentNames = Lists.newArrayList(segmentsFlushInfo.getKeyIterator());

// This should only happen if you're running in stagingN and loading a prod index through

// the read\_index\_from\_prod\_location flag. In this case, we point to a directory that has

// a lot more than the number of segments we want in staging and we trim this list to the

// desired number.

if (environment.matches("staging\\d")) {

if (segmentNames.size() > partitionConfig.getMaxEnabledLocalSegments()) {

LOG.info("Trimming list of loaded segments from size {} to size {}.",

segmentNames.size(), partitionConfig.getMaxEnabledLocalSegments());

segmentNames = segmentNames.subList(

segmentNames.size() - partitionConfig.getMaxEnabledLocalSegments(),

segmentNames.size());

}

}

List<SegmentInfo> segmentInfoList = ParallelUtil.parmap("load-index", name -> {

FlushInfo subProperties = segmentsFlushInfo.getSubProperties(name);

long timesliceID = subProperties.getLongProperty(EarlybirdIndexFlusher.TIMESLICE\_ID);

return ActionLogger.call(

"loading segment " + name,

() -> loadSegment(flushPath, name, timesliceID));

}, segmentNames);

return new EarlybirdIndex(

segmentInfoList,

indexInfo.getLongProperty(EarlybirdIndexFlusher.TWEET\_KAFKA\_OFFSET),

indexInfo.getLongProperty(EarlybirdIndexFlusher.UPDATE\_KAFKA\_OFFSET));

}

private SegmentInfo loadSegment(

Path flushPath,

String segmentName,

long timesliceID

) throws IOException {

Path segmentPrefix = flushPath.suffix("/" + segmentName);

Path segmentPath = segmentPrefix.suffix(EarlybirdIndexFlusher.DATA\_SUFFIX);

TimeSlice timeSlice = new TimeSlice(

timesliceID,

EarlybirdConfig.getMaxSegmentSize(),

partitionConfig.getIndexingHashPartitionID(),

partitionConfig.getNumPartitions());

SegmentInfo segmentInfo = new SegmentInfo(

timeSlice.getSegment(),

earlybirdSegmentFactory,

segmentSyncConfig);

Path infoPath = segmentPrefix.suffix(EarlybirdIndexFlusher.INFO\_SUFFIX);

FlushInfo flushInfo;

try (FSDataInputStream infoInputStream = fileSystem.open(infoPath)) {

flushInfo = FlushInfo.loadFromYaml(infoInputStream);

}

FSDataInputStream inputStream = fileSystem.open(segmentPath);

// It's significantly slower to read from the FSDataInputStream on demand, so we

// use a buffered reader to pre-read bigger chunks.

int bufferSize = 1 << 22; // 4MB

BufferedInputStream bufferedInputStream = new BufferedInputStream(inputStream, bufferSize);

DataDeserializer in = new DataDeserializer(bufferedInputStream, segmentName);

segmentInfo.getIndexSegment().load(in, flushInfo);

return segmentInfo;

}

}