package com.twitter.search.core.earlybird.index;

import java.io.File;

import java.io.IOException;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.slf4j.Marker;

import org.slf4j.MarkerFactory;

import org.apache.lucene.document.Document;

import org.apache.lucene.index.IndexWriter;

import org.apache.lucene.index.IndexWriterConfig;

import org.apache.lucene.search.Query;

import org.apache.lucene.store.Directory;

import org.apache.lucene.store.FSDirectory;

import org.apache.lucene.store.LockObtainFailedException;

/\*\*

\* EarlybirdIndexWriter implementation that's a wrapper around Lucene's {@link IndexWriter}

\* and writes Lucene segments into a {@link Directory}.

\*/

public class EarlybirdLuceneIndexSegmentWriter extends EarlybirdIndexSegmentWriter {

private static final Logger LOG =

LoggerFactory.getLogger(EarlybirdLuceneIndexSegmentWriter.class);

private static final Marker FATAL = MarkerFactory.getMarker("FATAL");

private final EarlybirdLuceneIndexSegmentData segmentData;

private final IndexWriter indexWriter;

@Override

public EarlybirdIndexSegmentData getSegmentData() {

return segmentData;

}

/\*\*

\* Construct a lucene IndexWriter-based Earlybird segment writer.

\* This will open a Lucene IndexWriter on segmentData.getLuceneDirectory().

\* This constructor will throw LockObtainFailedException if it cannot obtain the "write.lock"

\* inside the directory segmentData.getLuceneDirectory().

\*

\* Don't add public constructors to this class. EarlybirdLuceneIndexSegmentWriter instances should

\* be created only by calling EarlybirdLuceneIndexSegmentData.createEarlybirdIndexSegmentWriter(),

\* to make sure everything is set up properly (such as CSF readers).

\*/

EarlybirdLuceneIndexSegmentWriter(

EarlybirdLuceneIndexSegmentData segmentData,

IndexWriterConfig indexWriterConfig) throws IOException {

Preconditions.checkNotNull(segmentData);

this.segmentData = segmentData;

try {

this.indexWriter = new IndexWriter(segmentData.getLuceneDirectory(), indexWriterConfig);

} catch (LockObtainFailedException e) {

logDebuggingInfoUponFailureToObtainLuceneWriteLock(segmentData, e);

// Rethrow the exception, and this Earlybird will trigger critical alerts

throw e;

}

}

private void logDebuggingInfoUponFailureToObtainLuceneWriteLock(

EarlybirdLuceneIndexSegmentData luceneIndexSegmentData,

LockObtainFailedException e) throws IOException {

// Every day, we create a new Lucene dir---we do not append into existing Lucene dirs.

// Supposedly, we should never fail to obtain the write lock from a fresh and empty

// Lucene directory.

// Adding debugging information for SEARCH-4454, where a timeslice roll failed because

// Earlybird failed to get the write lock for a new timeslice.

Directory dir = luceneIndexSegmentData.getLuceneDirectory();

LOG.error(

FATAL,

"Unable to obtain write.lock for Lucene directory. The Lucene directory is: " + dir,

e);

if (dir instanceof FSDirectory) { // this check should always be true in our current setup.

FSDirectory fsDir = (FSDirectory) dir;

// Log if the underlying directory on disk does not exist.

File underlyingDir = fsDir.getDirectory().toFile();

if (underlyingDir.exists()) {

LOG.info("Lucene directory contains the following files: "

+ Lists.newArrayList(fsDir.listAll()));

} else {

LOG.error(

FATAL,

"Directory " + underlyingDir + " does not exist on disk.",

e);

}

if (!underlyingDir.canWrite()) {

LOG.error(

FATAL,

"Cannot write into directory " + underlyingDir,

e);

}

File writeLockFile = new File(underlyingDir, "write.lock");

if (writeLockFile.exists()) {

LOG.error(

FATAL,

"Write lock file " + writeLockFile + " already exists.",

e);

}

if (!writeLockFile.canWrite()) {

LOG.error(

FATAL,

"No write access to lock file: " + writeLockFile

+ " Usable space: " + underlyingDir.getUsableSpace(),

e);

}

// List all files in the segment directory

File segmentDir = underlyingDir.getParentFile();

LOG.warn("Segment directory contains the following files: "

+ Lists.newArrayList(segmentDir.list()));

} else {

LOG.warn("Unable to log debugging info upon failing to acquire Lucene write lock."

+ "The class of the directory is: " + dir.getClass().getName());

}

}

@Override

public void addDocument(Document doc) throws IOException {

indexWriter.addDocument(doc);

}

@Override

public void addTweet(Document doc, long tweetId, boolean docIdOffensive) throws IOException {

indexWriter.addDocument(doc);

}

@Override

protected void appendOutOfOrder(Document doc, int docId) throws IOException {

throw new UnsupportedOperationException("This Lucene-based IndexWriter does not support "

+ "updates and out-of-order appends.");

}

@Override

public int numDocs() {

return indexWriter.getDocStats().maxDoc;

}

@Override

public int numDocsNoDelete() throws IOException {

return numDocs();

}

@Override

public void deleteDocuments(Query query) throws IOException {

super.deleteDocuments(query);

indexWriter.deleteDocuments(query);

}

@Override

public void addIndexes(Directory... dirs) throws IOException {

indexWriter.addIndexes(dirs);

}

@Override

public void forceMerge() throws IOException {

indexWriter.forceMerge(1);

}

@Override

public void close() throws IOException {

indexWriter.close();

}

}