package com.twitter.search.earlybird.index;

import java.io.IOException;

import com.google.common.base.Preconditions;

import org.apache.lucene.index.LeafReader;

import org.apache.lucene.index.NumericDocValues;

import org.apache.lucene.search.DocIdSetIterator;

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

import com.twitter.search.common.util.analysis.IntTermAttributeImpl;

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

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

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

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

import com.twitter.search.core.earlybird.index.DocIDToTweetIDMapper;

import com.twitter.search.core.earlybird.index.TimeMapper;

import com.twitter.search.core.earlybird.index.column.ColumnStrideFieldIndex;

/\*\*

\* A few caveats when using this class:

\* - This class only supports in-order createdAt!

\* - Before actually using this class, one must call prepareToRead() with a Lucene AtomicReader

\* - prepareToRead() will load docID to createdAt mapping into memory, if not already done.

\*/

public class DocValuesBasedTimeMapper implements TimeMapper {

private LeafReader reader;

private ColumnStrideFieldIndex docValues;

protected int minTimestamp = ILLEGAL\_TIME;

protected int maxTimestamp = ILLEGAL\_TIME;

/\*\*

\* When indexing finishes, this method should be called with a index reader that

\* can see all documents.

\* @param leafReader Lucene index reader used to access "TweetID" to "createdAt" mapping.

\*/

public void initializeWithLuceneReader(LeafReader leafReader, ColumnStrideFieldIndex csf)

throws IOException {

reader = Preconditions.checkNotNull(leafReader);

docValues = Preconditions.checkNotNull(csf);

// Find the min and max timestamps.

// See SEARCH-5534

// In the archive, tweets are always sorted in descending order by tweet ID, but

// that does not mean that the documents are necessarily sorted by time. We've observed tweet ID

// generation be decoupled from timestamp creation (i.e. a larger tweet ID having a smaller

// created\_at time).

minTimestamp = Integer.MAX\_VALUE;

maxTimestamp = Integer.MIN\_VALUE;

NumericDocValues onDiskDocValues = reader.getNumericDocValues(

EarlybirdFieldConstants.EarlybirdFieldConstant.CREATED\_AT\_CSF\_FIELD.getFieldName());

for (int i = 0; i < reader.maxDoc(); ++i) {

Preconditions.checkArgument(onDiskDocValues.advanceExact(i));

int timestamp = (int) onDiskDocValues.longValue();

docValues.setValue(i, timestamp);

if (timestamp < minTimestamp) {

minTimestamp = timestamp;

}

if (timestamp > maxTimestamp) {

maxTimestamp = timestamp;

}

}

}

@Override

public int getLastTime() {

return maxTimestamp;

}

@Override

public int getFirstTime() {

return minTimestamp;

}

@Override

public int getTime(int docID) {

if (docID < 0 || docID > reader.maxDoc()) {

return ILLEGAL\_TIME;

}

return (int) docValues.get(docID);

}

@Override

public int findFirstDocId(int timeSeconds, int smallestDocID) throws IOException {

// In the full archive, the smallest doc id corresponds to largest timestamp.

if (timeSeconds > maxTimestamp) {

return smallestDocID;

}

if (timeSeconds < minTimestamp) {

return reader.maxDoc() - 1;

}

int docId = DocValuesHelper.getLargestDocIdWithCeilOfValue(

reader,

EarlybirdFieldConstants.EarlybirdFieldConstant.CREATED\_AT\_FIELD.getFieldName(),

IntTermAttributeImpl.copyIntoNewBytesRef(timeSeconds));

if (docId == DocIdSetIterator.NO\_MORE\_DOCS) {

return ILLEGAL\_TIME;

}

return docId;

}

@Override

public TimeMapper optimize(DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) {

// DocValuesBasedTimerMapper instances are not flushed or loaded,

// so their optimization is a no-op.

return this;

}

@Override

public Flushable.Handler<DocValuesBasedTimeMapper> getFlushHandler() {

// EarlybirdIndexSegmentData will still try to flush the DocValuesBasedTimeMapper for the

// respective segment, so we need to pass in a DocValuesBasedTimeMapper instance to this

// flusher: otherwise, Flushable.Handler.flush() will throw a NullPointerException.

return new FlushHandler(new DocValuesBasedTimeMapper());

}

// Full archive earlybirds don't actually flush or load the DocValuesBasedTimeMapper. This is

// why doFlush() is a no-op, and doLoad() returns a new DocValuesBasedTimeMapper instance

// (initializeWithLuceneReader() will be called at load time to initialize this new

// DocValuesBasedTimeMapper instance).

public static class FlushHandler extends Flushable.Handler<DocValuesBasedTimeMapper> {

public FlushHandler() {

super();

}

public FlushHandler(DocValuesBasedTimeMapper objectToFlush) {

super(objectToFlush);

}

@Override

protected void doFlush(FlushInfo flushInfo, DataSerializer out) {

}

@Override

protected DocValuesBasedTimeMapper doLoad(FlushInfo flushInfo, DataDeserializer in) {

return new DocValuesBasedTimeMapper();

}

}

}