package com.twitter.search.earlybird.search.queries;

import java.io.IOException;

import com.google.common.annotations.VisibleForTesting;

import org.apache.lucene.index.LeafReader;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.search.BooleanClause;

import org.apache.lucene.search.BooleanQuery;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.search.Weight;

import com.twitter.search.common.query.DefaultFilterWeight;

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

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

import com.twitter.search.core.earlybird.index.util.AllDocsIterator;

import com.twitter.search.core.earlybird.index.util.RangeFilterDISI;

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

/\*\*

\* Filters tweet ids according to since\_id and max\_id parameter.

\*

\* Note that since\_id is exclusive and max\_id is inclusive.

\*/

public final class SinceMaxIDFilter extends Query {

public static final long NO\_FILTER = -1;

private final long sinceIdExclusive;

private final long maxIdInclusive;

public static Query getSinceMaxIDQuery(long sinceIdExclusive, long maxIdInclusive) {

return new BooleanQuery.Builder()

.add(new SinceMaxIDFilter(sinceIdExclusive, maxIdInclusive), BooleanClause.Occur.FILTER)

.build();

}

public static Query getSinceIDQuery(long sinceIdExclusive) {

return new BooleanQuery.Builder()

.add(new SinceMaxIDFilter(sinceIdExclusive, NO\_FILTER), BooleanClause.Occur.FILTER)

.build();

}

public static Query getMaxIDQuery(long maxIdInclusive) {

return new BooleanQuery.Builder()

.add(new SinceMaxIDFilter(NO\_FILTER, maxIdInclusive), BooleanClause.Occur.FILTER)

.build();

}

private SinceMaxIDFilter(long sinceIdExclusive, long maxIdInclusive) {

this.sinceIdExclusive = sinceIdExclusive;

this.maxIdInclusive = maxIdInclusive;

}

@Override

public int hashCode() {

return (int) (sinceIdExclusive \* 13 + maxIdInclusive);

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof SinceMaxIDFilter)) {

return false;

}

SinceMaxIDFilter filter = SinceMaxIDFilter.class.cast(obj);

return (sinceIdExclusive == filter.sinceIdExclusive)

&& (maxIdInclusive == filter.maxIdInclusive);

}

@Override

public String toString(String field) {

if (sinceIdExclusive != NO\_FILTER && maxIdInclusive != NO\_FILTER) {

return "SinceIdFilter:" + sinceIdExclusive + ",MaxIdFilter:" + maxIdInclusive;

} else if (maxIdInclusive != NO\_FILTER) {

return "MaxIdFilter:" + maxIdInclusive;

} else {

return "SinceIdFilter:" + sinceIdExclusive;

}

}

/\*\*

\* Determines if this segment is at least partially covered by the given tweet ID range.

\*/

public static boolean sinceMaxIDsInRange(

TweetIDMapper tweetIdMapper, long sinceIdExclusive, long maxIdInclusive) {

// Check for since id out of range. Note that since this ID is exclusive,

// equality is out of range too.

if (sinceIdExclusive != NO\_FILTER && sinceIdExclusive >= tweetIdMapper.getMaxTweetID()) {

return false;

}

// Check for max id in range.

return maxIdInclusive == NO\_FILTER || maxIdInclusive >= tweetIdMapper.getMinTweetID();

}

// Returns true if this segment is completely covered by these id filters.

private static boolean sinceMaxIdsCoverRange(

TweetIDMapper tweetIdMapper, long sinceIdExclusive, long maxIdInclusive) {

// Check for since\_id specified AND since\_id newer than than first tweet.

if (sinceIdExclusive != NO\_FILTER && sinceIdExclusive >= tweetIdMapper.getMinTweetID()) {

return false;

}

// Check for max id in range.

return maxIdInclusive == NO\_FILTER || maxIdInclusive > tweetIdMapper.getMaxTweetID();

}

@Override

public Weight createWeight(IndexSearcher searcher, ScoreMode scoreMode, float boost)

throws IOException {

return new DefaultFilterWeight(this) {

@Override

protected DocIdSetIterator getDocIdSetIterator(LeafReaderContext context) throws IOException {

LeafReader reader = context.reader();

if (!(reader instanceof EarlybirdIndexSegmentAtomicReader)) {

return new AllDocsIterator(reader);

}

EarlybirdIndexSegmentAtomicReader twitterInMemoryIndexReader =

(EarlybirdIndexSegmentAtomicReader) reader;

TweetIDMapper tweetIdMapper =

(TweetIDMapper) twitterInMemoryIndexReader.getSegmentData().getDocIDToTweetIDMapper();

// Important to return a null DocIdSetIterator here, so the Scorer will skip searching

// this segment completely.

if (!sinceMaxIDsInRange(tweetIdMapper, sinceIdExclusive, maxIdInclusive)) {

return null;

}

// Optimization: just return a match-all iterator when the whole segment is in range.

// This avoids having to do so many status id lookups.

if (sinceMaxIdsCoverRange(tweetIdMapper, sinceIdExclusive, maxIdInclusive)) {

return new AllDocsIterator(reader);

}

return new SinceMaxIDDocIdSetIterator(

twitterInMemoryIndexReader, sinceIdExclusive, maxIdInclusive);

}

};

}

@VisibleForTesting

static class SinceMaxIDDocIdSetIterator extends RangeFilterDISI {

private final DocIDToTweetIDMapper docIdToTweetIdMapper;

private final long sinceIdExclusive;

private final long maxIdInclusive;

public SinceMaxIDDocIdSetIterator(EarlybirdIndexSegmentAtomicReader reader,

long sinceIdExclusive,

long maxIdInclusive) throws IOException {

super(reader,

findMaxIdDocID(reader, maxIdInclusive),

findSinceIdDocID(reader, sinceIdExclusive));

this.docIdToTweetIdMapper = reader.getSegmentData().getDocIDToTweetIDMapper();

this.sinceIdExclusive = sinceIdExclusive; // sinceStatusId == NO\_FILTER is OK, it's exclusive

this.maxIdInclusive = maxIdInclusive != NO\_FILTER ? maxIdInclusive : Long.MAX\_VALUE;

}

/\*\*

\* This is a necessary check when we have out of order tweets in the archive.

\* When tweets are out of order, this guarantees that no false positive results are returned.

\* I.e. we can still miss some tweets in the specified range, but we never incorrectly return

\* anything that's not in the range.

\*/

@Override

protected boolean shouldReturnDoc() {

final long statusID = docIdToTweetIdMapper.getTweetID(docID());

return statusID > sinceIdExclusive && statusID <= maxIdInclusive;

}

private static int findSinceIdDocID(

EarlybirdIndexSegmentAtomicReader reader, long sinceIdExclusive) throws IOException {

TweetIDMapper tweetIdMapper =

(TweetIDMapper) reader.getSegmentData().getDocIDToTweetIDMapper();

if (sinceIdExclusive != SinceMaxIDFilter.NO\_FILTER) {

// We use this as an upper bound on the search, so we want to find the highest possible

// doc ID for this tweet ID.

boolean findMaxDocID = true;

return tweetIdMapper.findDocIdBound(

sinceIdExclusive,

findMaxDocID,

reader.getSmallestDocID(),

reader.maxDoc() - 1);

} else {

return DocIDToTweetIDMapper.ID\_NOT\_FOUND;

}

}

private static int findMaxIdDocID(

EarlybirdIndexSegmentAtomicReader reader, long maxIdInclusive) throws IOException {

TweetIDMapper tweetIdMapper =

(TweetIDMapper) reader.getSegmentData().getDocIDToTweetIDMapper();

if (maxIdInclusive != SinceMaxIDFilter.NO\_FILTER) {

// We use this as a lower bound on the search, so we want to find the lowest possible

// doc ID for this tweet ID.

boolean findMaxDocID = false;

return tweetIdMapper.findDocIdBound(

maxIdInclusive,

findMaxDocID,

reader.getSmallestDocID(),

reader.maxDoc() - 1);

} else {

return DocIDToTweetIDMapper.ID\_NOT\_FOUND;

}

}

}

}