package com.twitter.search.earlybird.querycache;

import java.io.IOException;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.util.BitDocIdSet;

import org.apache.lucene.util.BitSet;

import org.apache.lucene.util.FixedBitSet;

import org.apache.lucene.util.SparseFixedBitSet;

import com.twitter.common.util.Clock;

import com.twitter.decider.Decider;

import com.twitter.search.common.schema.base.ImmutableSchemaInterface;

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

import com.twitter.search.earlybird.RecentTweetRestriction;

import com.twitter.search.earlybird.search.AbstractResultsCollector;

import com.twitter.search.earlybird.search.SearchRequestInfo;

import com.twitter.search.earlybird.search.SearchResultsInfo;

import com.twitter.search.earlybird.search.queries.SinceUntilFilter;

import com.twitter.search.earlybird.stats.EarlybirdSearcherStats;

import static org.apache.lucene.search.DocIdSetIterator.NO\_MORE\_DOCS;

import static com.twitter.search.core.earlybird.index.TimeMapper.ILLEGAL\_TIME;

/\*\*

\* Collector to update the query cache (one segment for a filter)

\*/

public class QueryCacheResultCollector

extends AbstractResultsCollector<SearchRequestInfo, SearchResultsInfo> {

private static final int UNSET = -1;

private final QueryCacheFilter queryCacheFilter;

private final Decider decider;

private BitSet bitSet;

private long cardinality = 0L;

private int startingDocID = UNSET;

public QueryCacheResultCollector(

ImmutableSchemaInterface schema,

QueryCacheFilter queryCacheFilter,

EarlybirdSearcherStats searcherStats,

Decider decider,

Clock clock,

int requestDebugMode) {

super(schema,

queryCacheFilter.createSearchRequestInfo(),

clock,

searcherStats,

requestDebugMode);

this.queryCacheFilter = queryCacheFilter;

this.decider = decider;

}

@Override

public void startSegment() throws IOException {

// The doc IDs in the optimized segments are always in the 0 .. (segmentSize - 1) range, so we

// can use a dense bitset to collect the hits. However, unoptimized segments can use any int

// doc IDs, so we have to use a sparse bitset to collect the hits in those segments.

if (currTwitterReader.getSegmentData().isOptimized()) {

switch (queryCacheFilter.getResultSetType()) {

case FixedBitSet:

bitSet = new FixedBitSet(currTwitterReader.maxDoc());

break;

case SparseFixedBitSet:

bitSet = new SparseFixedBitSet(currTwitterReader.maxDoc());

break;

default:

throw new IllegalStateException(

"Unknown ResultSetType: " + queryCacheFilter.getResultSetType().name());

}

} else {

bitSet = new SparseFixedBitSet(currTwitterReader.maxDoc());

}

startingDocID = findStartingDocID();

cardinality = 0;

}

@Override

protected void doCollect(long tweetID) {

bitSet.set(curDocId);

cardinality++;

}

@Override

protected SearchResultsInfo doGetResults() {

return new SearchResultsInfo();

}

public QueryCacheResultForSegment getCachedResult() {

// Note that BitSet.cardinality takes linear time in the size of the maxDoc, so we track

// cardinality separately.

return new QueryCacheResultForSegment(new BitDocIdSet(bitSet, cardinality),

cardinality, startingDocID);

}

/\*\*

\* We don't want to return results less than 15 seconds older than the most recently indexed tweet,

\* as they might not be completely indexed.

\* We can't simply use the first hit, as some cached filters might not have any hits,

\* e.g. has\_engagement in the protected cluster.

\* We can't use a clock because streams can lag.

\*/

private int findStartingDocID() throws IOException {

int lastTime = currTwitterReader.getSegmentData().getTimeMapper().getLastTime();

if (lastTime == ILLEGAL\_TIME) {

return NO\_MORE\_DOCS;

}

int untilTime = RecentTweetRestriction.queryCacheUntilTime(decider, lastTime);

if (untilTime == 0) {

return currTwitterReader.getSmallestDocID();

}

return SinceUntilFilter.getUntilQuery(untilTime)

.createWeight(new IndexSearcher(currTwitterReader), ScoreMode.COMPLETE\_NO\_SCORES, 1.0f)

.scorer(currTwitterReader.getContext())

.iterator()

.nextDoc();

}

}