package com.twitter.search.earlybird.querycache;

import java.io.IOException;

import java.util.Objects;

import java.util.Set;

import org.apache.lucene.index.IndexReader;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.index.Term;

import org.apache.lucene.search.BooleanClause;

import org.apache.lucene.search.BooleanQuery;

import org.apache.lucene.search.ConstantScoreScorer;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.search.Explanation;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.Scorer;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.search.Weight;

import com.twitter.search.common.metrics.SearchCounter;

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

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

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

/\*\*

\* Query to iterate QueryCache result (the cache)

\*/

public final class CachedFilterQuery extends Query {

private static final String STAT\_PREFIX = "querycache\_serving\_";

private static final SearchCounter REWRITE\_CALLS = SearchCounter.export(

STAT\_PREFIX + "rewrite\_calls");

private static final SearchCounter NO\_CACHE\_FOUND = SearchCounter.export(

STAT\_PREFIX + "no\_cache\_found");

private static final SearchCounter USED\_CACHE\_AND\_FRESH\_DOCS = SearchCounter.export(

STAT\_PREFIX + "used\_cache\_and\_fresh\_docs");

private static final SearchCounter USED\_CACHE\_ONLY = SearchCounter.export(

STAT\_PREFIX + "used\_cache\_only");

public static class NoSuchFilterException extends Exception {

NoSuchFilterException(String filterName) {

super("Filter [" + filterName + "] does not exists");

}

}

private static class CachedResultQuery extends Query {

private final QueryCacheResultForSegment cachedResult;

public CachedResultQuery(QueryCacheResultForSegment cachedResult) {

this.cachedResult = cachedResult;

}

@Override

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

return new DefaultFilterWeight(this) {

@Override

protected DocIdSetIterator getDocIdSetIterator(LeafReaderContext context)

throws IOException {

return cachedResult.getDocIdSet().iterator();

}

};

}

@Override

public int hashCode() {

return cachedResult == null ? 0 : cachedResult.hashCode();

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof CachedResultQuery)) {

return false;

}

CachedResultQuery query = (CachedResultQuery) obj;

return Objects.equals(cachedResult, query.cachedResult);

}

@Override

public String toString(String field) {

return "CACHED\_RESULT";

}

}

private static class CachedResultAndFreshDocsQuery extends Query {

private final Query cacheLuceneQuery;

private final QueryCacheResultForSegment cachedResult;

public CachedResultAndFreshDocsQuery(

Query cacheLuceneQuery, QueryCacheResultForSegment cachedResult) {

this.cacheLuceneQuery = cacheLuceneQuery;

this.cachedResult = cachedResult;

}

@Override

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

return new Weight(this) {

@Override

public void extractTerms(Set<Term> terms) {

}

@Override

public Explanation explain(LeafReaderContext context, int doc) throws IOException {

Scorer scorer = scorer(context);

if ((scorer != null) && (scorer.iterator().advance(doc) == doc)) {

return Explanation.match(0f, "Match on id " + doc);

}

return Explanation.match(0f, "No match on id " + doc);

}

@Override

public Scorer scorer(LeafReaderContext context) throws IOException {

Weight luceneWeight;

try {

luceneWeight = cacheLuceneQuery.createWeight(searcher, scoreMode, boost);

} catch (UnsupportedOperationException e) {

// Some queries do not support weights. This is fine, it simply means the query has

// no docs, and means the same thing as a null scorer.

return null;

}

Scorer luceneScorer = luceneWeight.scorer(context);

if (luceneScorer == null) {

return null;

}

DocIdSetIterator iterator = new CachedResultDocIdSetIterator(

cachedResult.getSmallestDocID(),

luceneScorer.iterator(),

cachedResult.getDocIdSet().iterator());

return new ConstantScoreScorer(luceneWeight, 0.0f, scoreMode, iterator);

}

@Override

public boolean isCacheable(LeafReaderContext ctx) {

return true;

}

};

}

@Override

public int hashCode() {

return (cacheLuceneQuery == null ? 0 : cacheLuceneQuery.hashCode()) \* 13

+ (cachedResult == null ? 0 : cachedResult.hashCode());

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof CachedResultAndFreshDocsQuery)) {

return false;

}

CachedResultAndFreshDocsQuery query = (CachedResultAndFreshDocsQuery) obj;

return Objects.equals(cacheLuceneQuery, query.cacheLuceneQuery)

&& Objects.equals(cachedResult, query.cachedResult);

}

@Override

public String toString(String field) {

return "CACHED\_RESULT\_AND\_FRESH\_DOCS";

}

}

private static final Query DUMMY\_FILTER = wrapFilter(new Query() {

@Override

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

return new DefaultFilterWeight(this) {

@Override

protected DocIdSetIterator getDocIdSetIterator(LeafReaderContext context) {

return null;

}

};

}

@Override

public int hashCode() {

return System.identityHashCode(this);

}

@Override

public boolean equals(Object obj) {

return this == obj;

}

@Override

public String toString(String field) {

return "DUMMY\_FILTER";

}

});

private final QueryCacheFilter queryCacheFilter;

// Lucene Query used to fill the cache

private final Query cacheLuceneQuery;

public static Query getCachedFilterQuery(String filterName, QueryCacheManager queryCacheManager)

throws NoSuchFilterException {

return wrapFilter(new CachedFilterQuery(filterName, queryCacheManager));

}

private static Query wrapFilter(Query filter) {

return new BooleanQuery.Builder()

.add(filter, BooleanClause.Occur.FILTER)

.build();

}

private CachedFilterQuery(String filterName, QueryCacheManager queryCacheManager)

throws NoSuchFilterException {

queryCacheFilter = queryCacheManager.getFilter(filterName);

if (queryCacheFilter == null) {

throw new NoSuchFilterException(filterName);

}

queryCacheFilter.incrementUsageStat();

// retrieve the query that was used to populate the cache

cacheLuceneQuery = queryCacheFilter.getLuceneQuery();

}

/\*\*

\* Creates a query base on the cache situation

\*/

@Override

public Query rewrite(IndexReader reader) {

EarlybirdIndexSegmentAtomicReader twitterReader = (EarlybirdIndexSegmentAtomicReader) reader;

QueryCacheResultForSegment cachedResult =

twitterReader.getSegmentData().getQueryCacheResult(queryCacheFilter.getFilterName());

REWRITE\_CALLS.increment();

if (cachedResult == null || cachedResult.getSmallestDocID() == -1) {

// No cached result, or cache has never been updated

// This happens to the newly created segment, between the segment creation and first

// query cache update

NO\_CACHE\_FOUND.increment();

if (queryCacheFilter.getCacheModeOnly()) {

// since this query cache filter allows cache mode only, we return a query that

// matches no doc

return DUMMY\_FILTER;

}

return wrapFilter(cacheLuceneQuery);

}

if (!queryCacheFilter.getCacheModeOnly() && // is this a cache mode only filter?

// the following check is only necessary for the realtime segment, which

// grows. Since we decrement docIds in the realtime segment, a reader

// having a smallestDocID less than the one in the cachedResult indicates

// that the segment/reader has new documents.

cachedResult.getSmallestDocID() > twitterReader.getSmallestDocID()) {

// The segment has more documents than the cached result. IOW, there are new

// documents that are not cached. This happens to latest segment that we're indexing to.

USED\_CACHE\_AND\_FRESH\_DOCS.increment();

return wrapFilter(new CachedResultAndFreshDocsQuery(cacheLuceneQuery, cachedResult));

}

// The segment has not grown since the cache was last updated.

// This happens mostly to old segments that we're no longer indexing to.

USED\_CACHE\_ONLY.increment();

return wrapFilter(new CachedResultQuery(cachedResult));

}

@Override

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

throws IOException {

final Weight luceneWeight = cacheLuceneQuery.createWeight(searcher, scoreMode, boost);

return new Weight(this) {

@Override

public Scorer scorer(LeafReaderContext context) throws IOException {

return luceneWeight.scorer(context);

}

@Override

public void extractTerms(Set<Term> terms) {

luceneWeight.extractTerms(terms);

}

@Override

public Explanation explain(LeafReaderContext context, int doc) throws IOException {

return luceneWeight.explain(context, doc);

}

@Override

public boolean isCacheable(LeafReaderContext ctx) {

return luceneWeight.isCacheable(ctx);

}

};

}

@Override

public int hashCode() {

return cacheLuceneQuery == null ? 0 : cacheLuceneQuery.hashCode();

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof CachedFilterQuery)) {

return false;

}

CachedFilterQuery filter = (CachedFilterQuery) obj;

return Objects.equals(cacheLuceneQuery, filter.cacheLuceneQuery);

}

@Override

public String toString(String s) {

return "CachedFilterQuery[" + queryCacheFilter.getFilterName() + "]";

}

}