package com.twitter.search.earlybird.index;

import java.io.IOException;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Map.Entry;

import com.google.common.base.Preconditions;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.index.Term;

import org.apache.lucene.search.CollectionStatistics;

import org.apache.lucene.search.Collector;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.search.Explanation;

import org.apache.lucene.search.LeafCollector;

import org.apache.lucene.search.Scorer;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.search.TermStatistics;

import org.apache.lucene.search.Weight;

import org.apache.lucene.util.BytesRef;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

import com.twitter.search.common.constants.thriftjava.ThriftLanguage;

import com.twitter.search.common.relevance.features.EarlybirdDocumentFeatures;

import com.twitter.search.common.results.thriftjava.FieldHitAttribution;

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

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

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

import com.twitter.search.common.search.TwitterCollector;

import com.twitter.search.common.search.TwitterIndexSearcher;

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

import com.twitter.search.common.util.lang.ThriftLanguageUtil;

import com.twitter.search.core.earlybird.facets.FacetLabelProvider;

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

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

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

import com.twitter.search.earlybird.EarlybirdSearcher;

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

import com.twitter.search.earlybird.common.userupdates.UserTable;

import com.twitter.search.earlybird.search.EarlybirdLuceneSearcher;

import com.twitter.search.earlybird.search.Hit;

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

import com.twitter.search.earlybird.search.SimpleSearchResults;

import com.twitter.search.earlybird.search.facets.AbstractFacetTermCollector;

import com.twitter.search.earlybird.search.facets.TermStatisticsCollector;

import com.twitter.search.earlybird.search.facets.TermStatisticsRequestInfo;

import com.twitter.search.earlybird.search.relevance.scoring.RelevanceQuery;

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

import com.twitter.search.earlybird.thrift.ThriftFacetCount;

import com.twitter.search.earlybird.thrift.ThriftFacetCountMetadata;

import com.twitter.search.earlybird.thrift.ThriftSearchResult;

import com.twitter.search.earlybird.thrift.ThriftSearchResultMetadata;

import com.twitter.search.earlybird.thrift.ThriftSearchResults;

import com.twitter.search.earlybird.thrift.ThriftTermRequest;

import com.twitter.search.earlybird.thrift.ThriftTermResults;

import com.twitter.search.earlybird.thrift.ThriftTermStatisticsResults;

public class EarlybirdSingleSegmentSearcher extends EarlybirdLuceneSearcher {

private static final Logger LOG = LoggerFactory.getLogger(EarlybirdSingleSegmentSearcher.class);

private final EarlybirdIndexSegmentAtomicReader twitterReader;

private final ImmutableSchemaInterface schema;

private final UserTable userTable;

private final long timeSliceID;

private final EarlybirdSearcherStats searcherStats;

private Clock clock;

public EarlybirdSingleSegmentSearcher(

ImmutableSchemaInterface schema,

EarlybirdIndexSegmentAtomicReader reader,

UserTable userTable,

EarlybirdSearcherStats searcherStats,

Clock clock) {

super(reader);

this.schema = schema;

this.twitterReader = reader;

this.userTable = userTable;

this.timeSliceID = reader.getSegmentData().getTimeSliceID();

this.searcherStats = searcherStats;

this.clock = clock;

}

public final long getTimeSliceID() {

return timeSliceID;

}

public EarlybirdIndexSegmentAtomicReader getTwitterIndexReader() {

return twitterReader;

}

/\*\*

\* search() main loop.

\* This behaves exactly like IndexSearcher.search() if a stock Lucene collector passed in.

\* However, if a TwitterCollector is passed in, this class performs Twitter style early

\* termination without relying on

\* {@link org.apache.lucene.search.CollectionTerminatedException}.

\* This method is nearly identical to TwitterIndexSearcher.search() with two differences:

\* 1) advances to smallest docID before searching. Important to skip incomplete docs in

\* realtime segments.

\* 2) skips deletes using twitterReader

\*/

@Override

protected void search(List<LeafReaderContext> leaves, Weight weight, Collector coll)

throws IOException {

// If an TwitterCollector is passed in, we can do a few extra things in here, such

// as early termination. Otherwise we can just fall back to IndexSearcher.search().

if (!(coll instanceof TwitterCollector)) {

super.search(leaves, weight, coll);

return;

}

TwitterCollector collector = (TwitterCollector) coll;

if (collector.isTerminated()) {

return;

}

LOG.debug("Starting segment {}", timeSliceID);

// Notify the collector that we're starting this segment, and check for early

// termination criteria again. setNextReader() performs 'expensive' early

// termination checks in some implementations such as TwitterEarlyTerminationCollector.

LeafCollector leafCollector = collector.getLeafCollector(twitterReader.getContext());

if (collector.isTerminated()) {

return;

}

// Initialize the scorer:

// Note that constructing the scorer may actually do real work, such as advancing to the

// first hit.

// The scorer may be null if we can tell right away that the query has no hits: e.g. if the

// first hit does not actually exist.

Scorer scorer = weight.scorer(twitterReader.getContext());

if (scorer == null) {

LOG.debug("Scorer was null, not searching segment {}", timeSliceID);

collector.finishSegment(DocIdSetIterator.NO\_MORE\_DOCS);

return;

}

leafCollector.setScorer(scorer);

// Make sure to start searching at the smallest docID.

DocIdSetIterator docIdSetIterator = scorer.iterator();

int smallestDocId = twitterReader.getSmallestDocID();

int docID = docIdSetIterator.advance(smallestDocId);

// Collect results.

while (docID != DocIdSetIterator.NO\_MORE\_DOCS) {

// Exclude deleted docs.

if (!twitterReader.getDeletesView().isDeleted(docID)) {

leafCollector.collect(docID);

}

// Check if we're done after we consumed the document.

if (collector.isTerminated()) {

break;

}

docID = docIdSetIterator.nextDoc();

}

// Always finish the segment, providing the last docID advanced to.

collector.finishSegment(docID);

}

@Override

public void fillFacetResults(

AbstractFacetTermCollector collector, ThriftSearchResults searchResults)

throws IOException {

if (searchResults == null || searchResults.getResultsSize() == 0) {

return;

}

EarlybirdIndexSegmentData segmentData = twitterReader.getSegmentData();

collector.resetFacetLabelProviders(

segmentData.getFacetLabelProviders(), segmentData.getFacetIDMap());

DocIDToTweetIDMapper docIdMapper = segmentData.getDocIDToTweetIDMapper();

for (ThriftSearchResult result : searchResults.getResults()) {

int docId = docIdMapper.getDocID(result.getId());

if (docId < 0) {

continue;

}

segmentData.getFacetCountingArray().collectForDocId(docId, collector);

collector.fillResultAndClear(result);

}

}

@Override

public TermStatisticsCollector.TermStatisticsSearchResults collectTermStatistics(

TermStatisticsRequestInfo searchRequestInfo,

EarlybirdSearcher searcher, int requestDebugMode) throws IOException {

TermStatisticsCollector collector = new TermStatisticsCollector(

schema, searchRequestInfo, searcherStats, clock, requestDebugMode);

search(searchRequestInfo.getLuceneQuery(), collector);

searcher.maybeSetCollectorDebugInfo(collector);

return collector.getResults();

}

/\*\* This method is only used for debugging, so it's not optimized for speed \*/

@Override

public void explainSearchResults(SearchRequestInfo searchRequestInfo,

SimpleSearchResults hits,

ThriftSearchResults searchResults) throws IOException {

Weight weight =

createWeight(rewrite(searchRequestInfo.getLuceneQuery()), ScoreMode.COMPLETE, 1.0f);

DocIDToTweetIDMapper docIdMapper = twitterReader.getSegmentData().getDocIDToTweetIDMapper();

for (int i = 0; i < hits.numHits(); i++) {

final Hit hit = hits.getHit(i);

Preconditions.checkState(hit.getTimeSliceID() == timeSliceID,

"hit: " + hit.toString() + " is not in timeslice: " + timeSliceID);

final ThriftSearchResult result = searchResults.getResults().get(i);

if (!result.isSetMetadata()) {

result.setMetadata(new ThriftSearchResultMetadata()

.setPenguinVersion(EarlybirdConfig.getPenguinVersionByte()));

}

final int docIdToExplain = docIdMapper.getDocID(hit.getStatusID());

if (docIdToExplain == DocIDToTweetIDMapper.ID\_NOT\_FOUND) {

result.getMetadata().setExplanation(

"ERROR: Could not find doc ID to explain for " + hit.toString());

} else {

Explanation explanation;

FieldHitAttribution fieldHitAttribution = result.getMetadata().getFieldHitAttribution();

if (weight instanceof RelevanceQuery.RelevanceWeight && fieldHitAttribution != null) {

RelevanceQuery.RelevanceWeight relevanceWeight =

(RelevanceQuery.RelevanceWeight) weight;

explanation = relevanceWeight.explain(

twitterReader.getContext(), docIdToExplain, fieldHitAttribution);

} else {

explanation = weight.explain(twitterReader.getContext(), docIdToExplain);

}

hit.setHasExplanation(true);

result.getMetadata().setExplanation(explanation.toString());

}

}

}

@Override

public void fillFacetResultMetadata(Map<Term, ThriftFacetCount> facetResults,

ImmutableSchemaInterface documentSchema,

byte debugMode) throws IOException {

FacetLabelProvider provider = twitterReader.getFacetLabelProviders(

documentSchema.getFacetFieldByFacetName(EarlybirdFieldConstant.TWIMG\_FACET));

FacetLabelProvider.FacetLabelAccessor photoAccessor = null;

if (provider != null) {

photoAccessor = provider.getLabelAccessor();

}

for (Entry<Term, ThriftFacetCount> facetResult : facetResults.entrySet()) {

Term term = facetResult.getKey();

ThriftFacetCount facetCount = facetResult.getValue();

ThriftFacetCountMetadata metadata = facetCount.getMetadata();

if (metadata == null) {

metadata = new ThriftFacetCountMetadata();

facetCount.setMetadata(metadata);

}

fillTermMetadata(term, metadata, photoAccessor, debugMode);

}

}

@Override

public void fillTermStatsMetadata(ThriftTermStatisticsResults termStatsResults,

ImmutableSchemaInterface documentSchema,

byte debugMode) throws IOException {

FacetLabelProvider provider = twitterReader.getFacetLabelProviders(

documentSchema.getFacetFieldByFacetName(EarlybirdFieldConstant.TWIMG\_FACET));

FacetLabelProvider.FacetLabelAccessor photoAccessor = null;

if (provider != null) {

photoAccessor = provider.getLabelAccessor();

}

for (Map.Entry<ThriftTermRequest, ThriftTermResults> entry

: termStatsResults.termResults.entrySet()) {

ThriftTermRequest termRequest = entry.getKey();

if (termRequest.getFieldName().isEmpty()) {

continue;

}

Schema.FieldInfo facetField = schema.getFacetFieldByFacetName(termRequest.getFieldName());

Term term = null;

if (facetField != null) {

term = new Term(facetField.getName(), termRequest.getTerm());

}

if (term == null) {

continue;

}

ThriftFacetCountMetadata metadata = entry.getValue().getMetadata();

if (metadata == null) {

metadata = new ThriftFacetCountMetadata();

entry.getValue().setMetadata(metadata);

}

fillTermMetadata(term, metadata, photoAccessor, debugMode);

}

}

private void fillTermMetadata(Term term, ThriftFacetCountMetadata metadata,

FacetLabelProvider.FacetLabelAccessor photoAccessor,

byte debugMode) throws IOException {

boolean isTwimg = term.field().equals(EarlybirdFieldConstant.TWIMG\_LINKS\_FIELD.getFieldName());

int internalDocID = DocIDToTweetIDMapper.ID\_NOT\_FOUND;

long statusID = -1;

long userID = -1;

Term facetTerm = term;

// Deal with the from\_user\_id facet.

if (term.field().equals(EarlybirdFieldConstant.FROM\_USER\_ID\_CSF.getFieldName())) {

userID = Long.parseLong(term.text());

facetTerm = new Term(EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName(),

LongTermAttributeImpl.copyIntoNewBytesRef(userID));

} else if (isTwimg) {

statusID = Long.parseLong(term.text());

internalDocID = twitterReader.getSegmentData().getDocIDToTweetIDMapper().getDocID(statusID);

}

if (internalDocID == DocIDToTweetIDMapper.ID\_NOT\_FOUND) {

// If this is not a twimg, this is how statusID should be looked up

//

// If this is a twimg but we couldn't find the internalDocID, that means this segment,

// or maybe even this earlybird, does not contain the original tweet. Then we treat this as

// a normal facet for now

internalDocID = twitterReader.getOldestDocID(facetTerm);

if (internalDocID >= 0) {

statusID =

twitterReader.getSegmentData().getDocIDToTweetIDMapper().getTweetID(internalDocID);

} else {

statusID = -1;

}

}

// make sure tweet is not deleted

if (internalDocID < 0 || twitterReader.getDeletesView().isDeleted(internalDocID)) {

return;

}

if (metadata.isSetStatusId()

&& metadata.getStatusId() > 0

&& metadata.getStatusId() <= statusID) {

// we already have the metadata for this facet from an earlier tweet

return;

}

// now check if this tweet is offensive, e.g. antisocial, nsfw, sensitive

EarlybirdDocumentFeatures documentFeatures = new EarlybirdDocumentFeatures(twitterReader);

documentFeatures.advance(internalDocID);

boolean isOffensiveFlagSet =

documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG);

boolean isSensitiveFlagSet =

documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_SENSITIVE\_CONTENT);

boolean offensive = isOffensiveFlagSet || isSensitiveFlagSet;

// also, user should not be marked as antisocial, nsfw or offensive

if (userID < 0) {

userID = documentFeatures.getFeatureValue(EarlybirdFieldConstant.FROM\_USER\_ID\_CSF);

}

offensive |= userTable.isSet(userID,

UserTable.ANTISOCIAL\_BIT

| UserTable.OFFENSIVE\_BIT

| UserTable.NSFW\_BIT);

metadata.setStatusId(statusID);

metadata.setTwitterUserId(userID);

metadata.setCreated\_at(twitterReader.getSegmentData().getTimeMapper().getTime(internalDocID));

int langId = (int) documentFeatures.getFeatureValue(EarlybirdFieldConstant.LANGUAGE);

Locale lang = ThriftLanguageUtil.getLocaleOf(ThriftLanguage.findByValue(langId));

metadata.setStatusLanguage(ThriftLanguageUtil.getThriftLanguageOf(lang));

metadata.setStatusPossiblySensitive(offensive);

if (isTwimg && photoAccessor != null && !metadata.isSetNativePhotoUrl()) {

int termID = twitterReader.getTermID(term);

if (termID != EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND) {

BytesRef termPayload = photoAccessor.getTermPayload(termID);

if (termPayload != null) {

metadata.setNativePhotoUrl(termPayload.utf8ToString());

}

}

}

if (debugMode > 3) {

StringBuilder sb = new StringBuilder(256);

if (metadata.isSetExplanation()) {

sb.append(metadata.getExplanation());

}

sb.append(String.format("TweetId=%d (%s %s), UserId=%d (%s %s), Term=%s\n",

statusID,

isOffensiveFlagSet ? "OFFENSIVE" : "",

isSensitiveFlagSet ? "SENSITIVE" : "",

userID,

userTable.isSet(userID, UserTable.ANTISOCIAL\_BIT) ? "ANTISOCIAL" : "",

userTable.isSet(userID, UserTable.NSFW\_BIT) ? "NSFW" : "",

term.toString()));

metadata.setExplanation(sb.toString());

}

}

public ImmutableSchemaInterface getSchemaSnapshot() {

return schema;

}

@Override

public CollectionStatistics collectionStatistics(String field) throws IOException {

return TwitterIndexSearcher.collectionStatistics(field, getIndexReader());

}

@Override

public TermStatistics termStatistics(Term term, int docFreq, long totalTermFreq) {

return TwitterIndexSearcher.termStats(term, docFreq, totalTermFreq);

}

}