package com.twitter.search.earlybird;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.stream.Collectors;

import javax.annotation.Nonnull;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Joiner;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.ImmutableSet;

import com.google.common.collect.Lists;

import org.apache.commons.lang.StringUtils;

import org.apache.lucene.index.Term;

import org.apache.lucene.queryparser.classic.ParseException;

import org.apache.lucene.queryparser.classic.QueryParser;

import org.apache.lucene.search.BooleanClause.Occur;

import org.apache.lucene.search.BooleanQuery;

import org.apache.lucene.search.Query;

import org.apache.thrift.TException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

import com.twitter.decider.Decider;

import com.twitter.search.common.database.DatabaseConfig;

import com.twitter.search.common.decider.DeciderUtil;

import com.twitter.search.common.features.thrift.ThriftSearchFeatureSchema;

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

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

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

import com.twitter.search.common.partitioning.base.Segment;

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

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

import com.twitter.search.common.query.thriftjava.CollectorParams;

import com.twitter.search.common.query.thriftjava.CollectorTerminationParams;

import com.twitter.search.common.query.thriftjava.EarlyTerminationInfo;

import com.twitter.search.common.ranking.thriftjava.ThriftRankingParams;

import com.twitter.search.common.ranking.thriftjava.ThriftScoringFunctionType;

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

import com.twitter.search.common.schema.SchemaUtil;

import com.twitter.search.common.schema.SearchWhitespaceAnalyzer;

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

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

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

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

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

import com.twitter.search.common.search.TerminationTracker;

import com.twitter.search.common.search.TwitterEarlyTerminationCollector;

import com.twitter.search.common.search.termination.QueryTimeoutFactory;

import com.twitter.search.common.util.earlybird.EarlybirdResponseUtil;

import com.twitter.search.common.util.ml.tensorflow\_engine.TensorflowModelsManager;

import com.twitter.search.common.util.thrift.ThriftUtils;

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

import com.twitter.search.earlybird.common.ClientIdUtil;

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

import com.twitter.search.earlybird.exception.ClientException;

import com.twitter.search.earlybird.exception.TransientException;

import com.twitter.search.earlybird.index.facets.FacetSkipList;

import com.twitter.search.earlybird.ml.ScoringModelsManager;

import com.twitter.search.earlybird.partition.AudioSpaceTable;

import com.twitter.search.earlybird.partition.MultiSegmentTermDictionaryManager;

import com.twitter.search.earlybird.partition.PartitionConfig;

import com.twitter.search.earlybird.partition.SegmentInfo;

import com.twitter.search.earlybird.partition.SegmentManager;

import com.twitter.search.earlybird.querycache.QueryCacheConversionRules;

import com.twitter.search.earlybird.querycache.QueryCacheManager;

import com.twitter.search.earlybird.queryparser.DetectFieldAnnotationVisitor;

import com.twitter.search.earlybird.queryparser.EarlybirdLuceneQueryVisitor;

import com.twitter.search.earlybird.queryparser.HighFrequencyTermPairRewriteVisitor;

import com.twitter.search.earlybird.queryparser.LuceneRelevanceQueryVisitor;

import com.twitter.search.earlybird.queryparser.ProtectedOperatorQueryRewriter;

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

import com.twitter.search.earlybird.search.AntiGamingFilter;

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

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

import com.twitter.search.earlybird.search.EarlybirdMultiSegmentSearcher;

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

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

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

import com.twitter.search.earlybird.search.SearchResultsCollector;

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

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

import com.twitter.search.earlybird.search.SocialFilter;

import com.twitter.search.earlybird.search.SocialSearchResultsCollector;

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

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

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

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

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

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

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

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

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

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

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

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

import com.twitter.search.earlybird.search.relevance.RelevanceSearchRequestInfo;

import com.twitter.search.earlybird.search.relevance.RelevanceSearchResults;

import com.twitter.search.earlybird.search.relevance.collectors.AbstractRelevanceCollector;

import com.twitter.search.earlybird.search.relevance.collectors.BatchRelevanceTopCollector;

import com.twitter.search.earlybird.search.relevance.collectors.RelevanceAllCollector;

import com.twitter.search.earlybird.search.relevance.collectors.RelevanceTopCollector;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

import com.twitter.search.earlybird.util.EarlybirdSearchResultUtil;

import com.twitter.search.queryparser.parser.SerializedQueryParser;

import com.twitter.search.queryparser.query.Conjunction;

import com.twitter.search.queryparser.query.Disjunction;

import com.twitter.search.queryparser.query.QueryNodeUtils;

import com.twitter.search.queryparser.query.QueryParserException;

import com.twitter.search.queryparser.query.annotation.Annotation;

import com.twitter.search.queryparser.query.search.SearchOperator;

import com.twitter.search.queryparser.query.search.SearchOperatorConstants;

import com.twitter.search.queryparser.util.IdTimeRanges;

import com.twitter.search.queryparser.visitors.ConversionVisitor;

import com.twitter.search.queryparser.visitors.DetectPositiveOperatorVisitor;

import com.twitter.search.queryparser.visitors.NamedDisjunctionVisitor;

import com.twitter.search.queryparser.visitors.ProximityGroupRewriteVisitor;

import com.twitter.search.queryparser.visitors.StripAnnotationsVisitor;

import static com.twitter.search.queryparser.query.search.SearchOperator.Type.UNTIL\_TIME;

/\*\*

\* This class provides the basic search() method:

\* - converts the thrift request object into what lucene expects.

\* - gets the segment.

\* - handles all errors, and prepares the response in case of error.

\*

\* We have one instance of this class per search received.

\*/

public class EarlybirdSearcher {

public enum QueryMode {

// Please think before adding more query modes: can this be implemented in a general way?

RECENCY(new EarlybirdRPCStats("search\_recency")),

FACETS(new EarlybirdRPCStats("search\_facets")),

TERM\_STATS(new EarlybirdRPCStats("search\_termstats")),

RELEVANCE(new EarlybirdRPCStats("search\_relevance")),

TOP\_TWEETS(new EarlybirdRPCStats("search\_toptweets"));

private final EarlybirdRPCStats requestStats;

QueryMode(EarlybirdRPCStats requestStats) {

this.requestStats = requestStats;

}

public EarlybirdRPCStats getRequestStats() {

return requestStats;

}

}

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

private static final String MATCH\_ALL\_SERIALIZED\_QUERY = "(\* )";

/\*\*

\* generic field annotations can be mapped to a concrete field in the index using this mapping

\* via {@link com.twitter.search.queryparser.query.annotation.Annotation.Type#MAPPABLE\_FIELD}

\*/

private static final Map<MappableField, String> MAPPABLE\_FIELD\_MAP =

ImmutableMap.of(

MappableField.URL,

EarlybirdFieldConstant.RESOLVED\_LINKS\_TEXT\_FIELD.getFieldName());

private static final String ALLOW\_QUERY\_SPECIFIC\_SIGNAL\_DECIDER\_KEY

= "allow\_query\_specific\_score\_adjustments";

@VisibleForTesting

public static final String ALLOW\_AUTHOR\_SPECIFIC\_SIGNAL\_DECIDER\_KEY

= "allow\_author\_specific\_score\_adjustments";

private static final String USE\_MULTI\_TERM\_DISJUNCTION\_FOR\_LIKED\_BY\_USER\_IDS\_DECIDER\_KEY

= "use\_multi\_term\_disjunction\_for\_liked\_by\_user\_ids";

private static final String ALLOW\_CAMELCASE\_USERNAME\_FIELD\_WEIGHT\_OVERRIDE\_DECIDER\_KEY\_PREFIX

= "allow\_camelcase\_username\_field\_weight\_override\_in\_";

private static final String ALLOW\_TOKENIZED\_DISPLAY\_NAME\_FIELD\_WEIGHT\_OVERRIDE\_DECIDER\_KEY\_PREFIX

= "allow\_tokenized\_display\_name\_field\_weight\_override\_in\_";

private static final boolean ALLOW\_QUERY\_SPECIFIC\_SIGNAL\_CONFIG

= EarlybirdConfig.getBool("allow\_query\_specific\_score\_adjustments", false);

private static final boolean ALLOW\_AUTHOR\_SPECIFIC\_SIGNAL\_CONFIG

= EarlybirdConfig.getBool("allow\_author\_specific\_score\_adjustments", false);

public static final int DEFAULT\_NUM\_FACET\_RESULTS = 100;

private final ImmutableSchemaInterface schemaSnapshot;

private final EarlybirdCluster cluster;

private final Clock clock;

private final Decider decider;

// The actual request thrift.

private final EarlybirdRequest request;

// searchQuery from inside the request.

private final ThriftSearchQuery searchQuery;

// CollectorParams from inside the searchQuery;

private final CollectorParams collectorParams;

// Parsed query (parsed from serialized query string in request).

private com.twitter.search.queryparser.query.Query parsedQuery;

private boolean parsedQueryAllowNullcast;

private IdTimeRanges idTimeRanges;

// Lucene version of the above. This is what we will actually be executing.

private org.apache.lucene.search.Query luceneQuery;

// Used for queries where we want to collect per-field hit attribution

@Nullable

private QueryHitAttributeHelper hitAttributeHelper;

// Debugging info can be appended to this buffer.

private final StringBuilder messageBuffer = new StringBuilder(1024);

private final EarlybirdDebugInfo debugInfo = new EarlybirdDebugInfo();

// The segment we are searching, or null for the multi-searcher.

private Segment segment = null;

// True iff we are searching all segments (multi-searcher).

private final boolean searchAllSegments;

// Tracking termination criteria for this query

private final TerminationTracker terminationTracker;

private EarlybirdLuceneSearcher searcher = null;

private final SegmentManager segmentManager;

private final QueryCacheManager queryCacheManager;

private final ScoringModelsManager scoringModelsManager;

private final TensorflowModelsManager tensorflowModelsManager;

private AntiGamingFilter antiGamingFilter = null;

private final boolean searchHighFrequencyTermPairs =

EarlybirdConfig.getBool("search\_high\_frequency\_term\_pairs", false);

// How long to allow post-termination when enforcing query timeout

private final int enforceQueryTimeoutBufferMillis =

EarlybirdConfig.getInt("enforce\_query\_timeout\_buffer\_millis", 50);

private EarlybirdRPCStats requestStats;

private QueryTimeoutFactory queryTimeoutFactory;

// Exported stats

private final EarlybirdSearcherStats searcherStats;

@VisibleForTesting

public static final SearchCounter FIELD\_WEIGHT\_OVERRIDE\_MAP\_NON\_NULL\_COUNT =

SearchCounter.export("field\_weight\_override\_map\_non\_null\_count");

@VisibleForTesting

public static final SearchCounter DROPPED\_CAMELCASE\_USERNAME\_FIELD\_WEIGHT\_OVERRIDE =

SearchCounter.export("dropped\_camelcase\_username\_field\_weight\_override");

@VisibleForTesting

public static final SearchCounter DROPPED\_TOKENIZED\_DISPLAY\_NAME\_FIELD\_WEIGHT\_OVERRIDE =

SearchCounter.export("dropped\_tokenized\_display\_name\_field\_weight\_override");

private static final SearchCounter RESPONSE\_HAS\_NO\_THRIFT\_SEARCH\_RESULTS =

SearchCounter.export("tweets\_earlybird\_searcher\_response\_has\_no\_thrift\_search\_results");

private static final SearchCounter CLIENT\_HAS\_FEATURE\_SCHEMA\_COUNTER =

SearchCounter.export("tweets\_earlybird\_searcher\_client\_has\_feature\_schema");

private static final SearchCounter CLIENT\_DOESNT\_HAVE\_FEATURE\_SCHEMA\_COUNTER =

SearchCounter.export("tweet\_earlybird\_searcher\_client\_doesnt\_have\_feature\_schema");

private static final SearchCounter COLLECTOR\_PARAMS\_MAX\_HITS\_TO\_PROCESS\_NOT\_SET\_COUNTER =

SearchCounter.export("collector\_params\_max\_hits\_to\_process\_not\_set");

private static final SearchCounter POSITIVE\_PROTECTED\_OPERATOR\_DETECTED\_COUNTER =

SearchCounter.export("positive\_protected\_operator\_detected\_counter");

// Query mode we are executing.

private final QueryMode queryMode;

// facetRequest from inside the request (or null).

private final ThriftFacetRequest facetRequest;

// termStatisticsRequest from inside the request (or null).

private final ThriftTermStatisticsRequest termStatisticsRequest;

// Results fields filled in during searchInternal().

private ThriftSearchResults searchResults = null;

private ThriftFacetResults facetResults = null;

private ThriftTermStatisticsResults termStatisticsResults = null;

private EarlyTerminationInfo earlyTerminationInfo = null;

// Partition config used to fill in debugging info.

// If null, no debug info is written into results.

@Nullable

private final PartitionConfig partitionConfig;

private final MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager;

private final QualityFactor qualityFactor;

private Set<String> queriedFields;

private final AudioSpaceTable audioSpaceTable;

public EarlybirdSearcher(

EarlybirdRequest request,

SegmentManager segmentManager,

AudioSpaceTable audioSpaceTable,

QueryCacheManager queryCacheManager,

ImmutableSchemaInterface schema,

EarlybirdCluster cluster,

@Nullable PartitionConfig partitionConfig,

Decider decider,

EarlybirdSearcherStats searcherStats,

ScoringModelsManager scoringModelsManager,

TensorflowModelsManager tensorflowModelsManager,

Clock clock,

MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager,

QueryTimeoutFactory queryTimeoutFactory,

QualityFactor qualityFactor) {

this.queryMode = getQueryMode(request);

this.schemaSnapshot = schema.getSchemaSnapshot();

// set the request stats as early as possible, so that we can track errors that happen

// early on in query processing.

this.requestStats = queryMode.getRequestStats();

this.facetRequest = request.isSetFacetRequest() ? request.getFacetRequest() : null;

this.termStatisticsRequest = request.isSetTermStatisticsRequest()

? request.getTermStatisticsRequest() : null;

this.partitionConfig = partitionConfig;

this.searcherStats = searcherStats;

this.multiSegmentTermDictionaryManager = multiSegmentTermDictionaryManager;

this.clock = clock;

this.decider = decider;

this.request = request;

this.segmentManager = segmentManager;

this.queryCacheManager = queryCacheManager;

this.cluster = cluster;

this.scoringModelsManager = scoringModelsManager;

this.tensorflowModelsManager = tensorflowModelsManager;

this.audioSpaceTable = audioSpaceTable;

// Note: we're deferring the validation/nullchecks until validateRequest()

// for more contained exception handling

this.searchQuery = request.getSearchQuery();

this.collectorParams = this.searchQuery == null ? null : this.searchQuery.getCollectorParams();

// Search all segments if searchSegmentId is unset.

this.searchAllSegments = !request.isSetSearchSegmentId();

if (this.collectorParams == null

|| !this.collectorParams.isSetTerminationParams()) {

this.terminationTracker = new TerminationTracker(clock);

} else if (request.isSetClientRequestTimeMs()) {

this.terminationTracker = new TerminationTracker(collectorParams.getTerminationParams(),

request.getClientRequestTimeMs(), clock,

getPostTerminationOverheadMillis(collectorParams.getTerminationParams()));

} else {

this.terminationTracker = new TerminationTracker(

collectorParams.getTerminationParams(), clock,

getPostTerminationOverheadMillis(collectorParams.getTerminationParams()));

}

this.queryTimeoutFactory = queryTimeoutFactory;

this.qualityFactor = qualityFactor;

}

private int getPostTerminationOverheadMillis(CollectorTerminationParams terminationParams) {

// If enforcing timeouts, set the post-termination buffer to the smaller of the timeout or the

// configured buffer. This ensures that timeout >= buffer, and a request with a smaller timeout

// should just time out immediately (because timeout == buffer).

return (terminationParams.isEnforceQueryTimeout() && terminationParams.getTimeoutMs() > 0)

? Math.min(enforceQueryTimeoutBufferMillis, terminationParams.getTimeoutMs()) : 0;

}

// Appends a debug string to the buffer.

private void appendMessage(String message) {

messageBuffer.append(message).append("\n");

}

/\*\*

\* Processes an Earlybird search request.

\* @return the earlybird response for this search request.

\*/

public EarlybirdResponse search() {

try {

debugInfo.setHost(DatabaseConfig.getLocalHostname());

// Throws transient exception for invalid requests.

validateRequest();

// Throws client exception for bad queries,

parseEarlybirdRequest();

// Modify the Lucene query if necessary.

luceneQuery = postLuceneQueryProcess(luceneQuery);

// Might return PARTITION\_NOT\_FOUND or PARTITION\_DISABLED.

EarlybirdResponseCode code = initSearcher();

if (code != EarlybirdResponseCode.SUCCESS) {

return respondError(code);

}

return searchInternal();

} catch (TransientException e) {

LOG.error(String.format("Transient exception in search() for EarlybirdRequest:\n%s", request),

e);

appendMessage(e.getMessage());

return respondError(EarlybirdResponseCode.TRANSIENT\_ERROR);

} catch (ClientException e) {

LOG.warn(String.format("Client exception in search() %s for EarlybirdRequest:\n %s",

e, request));

appendMessage(e.getMessage());

return respondError(EarlybirdResponseCode.CLIENT\_ERROR);

} catch (Exception e) {

LOG.warn(String.format("Uncaught exception in search() for EarlybirdRequest:\n%s", request),

e);

appendMessage(e.getMessage());

return respondError(EarlybirdResponseCode.TRANSIENT\_ERROR);

} catch (AssertionError e) {

LOG.warn(String.format("Assertion error in search() for EarlybirdRequest:\n%s", request), e);

appendMessage(e.getMessage());

return respondError(EarlybirdResponseCode.TRANSIENT\_ERROR);

} catch (Error e) {

// SEARCH-33166: If we got here, it means what was thrown was not an Exception, or anything

// we know how to handle. Log the Error for diagnostic purposes and propagate it.

LOG.error("Re-throwing uncaught error", e);

throw e;

}

}

public EarlybirdRPCStats getRequestStats() {

return requestStats;

}

/\*\*

\* Wraps the given query with the provided filter queries.

\*

\* @param query the query to wrap with filters.

\* @param filters the filters to wrap the query with.

\* @return a BooleanQuery wrapped with filters

\*/

public static Query wrapFilters(Query query, Query... filters) {

boolean filtersEmpty = filters == null || filters.length == 0;

if (!filtersEmpty) {

filtersEmpty = true;

for (Query f : filters) {

if (f != null) {

filtersEmpty = false;

break;

}

}

}

if (filtersEmpty) {

if (query == null) {

return new MatchAllDocsQuery();

} else {

return query;

}

}

BooleanQuery.Builder bqBuilder = new BooleanQuery.Builder();

if (query != null) {

bqBuilder.add(query, Occur.MUST);

}

for (Query f : filters) {

if (f != null) {

bqBuilder.add(f, Occur.FILTER);

}

}

return bqBuilder.build();

}

// Examine all fields in the request for sanity.

private void validateRequest() throws TransientException, ClientException {

// First try thrift's internal validate. Should always succeed.

try {

request.validate();

} catch (TException e) {

throw new TransientException(e.getMessage(), e);

}

if (searchQuery == null) {

throw new TransientException("No ThriftSearchQuery specified");

}

if (collectorParams == null) {

throw new TransientException("No CollectorParams specified");

}

validateTermStatsRequest();

if (!searchAllSegments) {

if (request.getSearchSegmentId() <= 0) {

String msg = "Bad time slice ID: " + request.getSearchSegmentId();

throw new TransientException(msg);

}

// Initialize the segment.

SegmentInfo segmentInfo = this.segmentManager.getSegmentInfo(request.getSearchSegmentId());

segment = segmentInfo != null ? segmentInfo.getSegment() : null;

}

if (collectorParams.getNumResultsToReturn() < 0) {

String msg = "Invalid numResults: " + collectorParams.getNumResultsToReturn();

throw new TransientException(msg);

}

if (searchQuery.getNamedDisjunctionMapSize() > 0 && searchQuery.isSetLuceneQuery()) {

throw new ClientException("namedMultiTermDisjunctionMap does not support with luceneQuery");

}

}

private void validateTermStatsRequest() throws ClientException {

// Validate the field names and values for all ThriftTermRequests.

if (request.isSetTermStatisticsRequest()

&& request.getTermStatisticsRequest().isSetTermRequests()) {

for (ThriftTermRequest termRequest : request.getTermStatisticsRequest().getTermRequests()) {

// If termRequest.fieldName is not set, it defaults to 'text', which is a string field,

// so we don't need to check the term.

if (termRequest.isSetFieldName()) {

String fieldName = termRequest.getFieldName();

Schema.FieldInfo facetFieldInfo = schemaSnapshot.getFacetFieldByFacetName(fieldName);

if (facetFieldInfo != null) {

// Facet fields are string fields, so we don't need to check the term.

continue;

}

Schema.FieldInfo fieldInfo = schemaSnapshot.getFieldInfo(fieldName);

if (fieldInfo == null) {

throw new ClientException("Field " + fieldName + " is not present in the schema.");

}

try {

SchemaUtil.toBytesRef(fieldInfo, termRequest.getTerm());

} catch (UnsupportedOperationException e) {

throw new ClientException("Term " + termRequest.getTerm() + " is not compatible with "

+ "the type of field " + fieldName);

}

}

}

}

}

private void setQueriesInDebugInfo(

com.twitter.search.queryparser.query.Query parsedQ,

org.apache.lucene.search.Query luceneQ) {

debugInfo.setParsedQuery(parsedQ == null ? null : parsedQ.serialize());

debugInfo.setLuceneQuery(luceneQ == null ? null : luceneQ.toString());

}

/\*\*

\* Takes the EarlybirdRequest that came into the service and after various parsing and processing

\* steps ultimately produces a Lucene query.

\*/

private void parseEarlybirdRequest() throws ClientException {

SerializedQueryParser parser = new SerializedQueryParser(EarlybirdConfig.getPenguinVersion());

try {

// if the deprecated iterativeQueries field is set, return an error to the client

// indicating that support for it has been removed.

if (searchQuery.isSetDeprecated\_iterativeQueries()) {

throw new ClientException("Invalid request: iterativeQueries feature has been removed");

}

// we parse the actual query from the user, if any

luceneQuery = null;

parsedQuery = null; // this will be set by parseQueryHelper()

if (searchQuery.getLikedByUserIDFilter64Size() > 0

&& searchQuery.isSetLuceneQuery()) {

throw new ClientException("likedByUserIDFilter64 does not support with luceneQuery");

}

if (!StringUtils.isBlank(request.getSearchQuery().getSerializedQuery())) {

searcherStats.thriftQueryWithSerializedQuery.increment();

luceneQuery = parseSerializedQuery(searchQuery.getSerializedQuery(), parser, true);

} else if (!StringUtils.isBlank(request.getSearchQuery().getLuceneQuery())) {

searcherStats.thriftQueryWithLuceneQuery.increment();

luceneQuery = parseLuceneQuery(searchQuery.getLuceneQuery());

LOG.info("lucene query: {}", searchQuery.getLuceneQuery());

if (luceneQuery != null) {

LOG.info("Using lucene query directly from the request: " + luceneQuery.toString());

}

} else {

searcherStats.thriftQueryWithoutTextQuery.increment();

luceneQuery = parseSerializedQuery(

MATCH\_ALL\_SERIALIZED\_QUERY,

parser,

queryMode != QueryMode.TERM\_STATS);

}

} catch (QueryParserException | BooleanQuery.TooManyClauses e) {

LOG.info("Exception parsing query during search", e);

appendMessage(e.getMessage());

throw new ClientException(e);

}

}

/\*\*

\* Parses a serialized query and creates a Lucene query out of it.

\*

\* To see how serialized queries look like, go to go/searchsyntax.

\*/

private Query parseSerializedQuery(

String serializedQuery,

SerializedQueryParser parser,

boolean shouldAdjustQueryBasedOnRequestParameters) throws QueryParserException {

// Parse the serialized query.

parsedQuery = parser.parse(serializedQuery);

if (parsedQuery == null) {

return null;

}

// rewrite query if positive 'protected' operator is detected

if (parsedQuery.accept(new DetectPositiveOperatorVisitor(SearchOperatorConstants.PROTECTED))) {

POSITIVE\_PROTECTED\_OPERATOR\_DETECTED\_COUNTER.increment();

ProtectedOperatorQueryRewriter rewriter = new ProtectedOperatorQueryRewriter();

parsedQuery = rewriter.rewrite(

parsedQuery,

request.followedUserIds,

segmentManager.getUserTable());

}

ThriftSearchRelevanceOptions options = searchQuery.getRelevanceOptions();

if (shouldAdjustQueryBasedOnRequestParameters) {

// If likedByUserIDFilter64 is set, combine it with query

// Note: we deal with likedByUserIDFilter64 here instead of in postLuceneQueryProcess as we

// want annotate query with ranks.

if (searchQuery.isSetLikedByUserIDFilter64()

&& searchQuery.getLikedByUserIDFilter64Size() > 0) {

parsedQuery = combineWithLikedByUserIdFilter64(

parsedQuery, searchQuery.getLikedByUserIDFilter64());

}

// If namedListMap field is set, replace the named lists in the serialized query.

if (searchQuery.getNamedDisjunctionMapSize() > 0) {

parsedQuery = parsedQuery.accept(

new NamedDisjunctionVisitor(searchQuery.getNamedDisjunctionMap()));

}

if (searchQuery.isSetRelevanceOptions()

&& searchQuery.getRelevanceOptions().isCollectFieldHitAttributions()) {

// NOTE: Before we do any modifications to the serialized query tree, annotate the query

// nodes with their node rank in the original query.

this.hitAttributeHelper =

QueryHitAttributeHelper.from(parsedQuery, schemaSnapshot);

parsedQuery = hitAttributeHelper.getAnnotatedQuery();

}

// Currently antisocial/nullcast tweets are dropped when we build index, but some tweets may

// become antisocial with realtime updates. For consistency, we should always filter out

// antisocial/nullcast tweets if the user is not explicitly including it.

final boolean allowAntisocial =

parsedQuery.accept(new DetectPositiveOperatorVisitor(SearchOperatorConstants.ANTISOCIAL));

if (!allowAntisocial) {

parsedQuery = QueryNodeUtils.appendAsConjunction(

parsedQuery,

QueryCacheConversionRules.CACHED\_EXCLUDE\_ANTISOCIAL);

}

parsedQueryAllowNullcast =

parsedQuery.accept(new DetectPositiveOperatorVisitor(SearchOperatorConstants.NULLCAST));

if (!parsedQueryAllowNullcast) {

parsedQuery = QueryNodeUtils.appendAsConjunction(

parsedQuery, new SearchOperator("filter", SearchOperatorConstants.NULLCAST).negate());

}

// Strip all annotations from the filters that will be converted to query cache filters.

// See SEARCH-15552.

parsedQuery = parsedQuery.accept(

new StripAnnotationsVisitor(QueryCacheConversionRules.STRIP\_ANNOTATIONS\_QUERIES));

// Convert certain filters into cached filters, also consolidate them.

parsedQuery = parsedQuery.accept(

new ConversionVisitor(QueryCacheConversionRules.DEFAULT\_RULES));

// add proximity if needed

if (options != null

&& options.isProximityScoring()

&& searchQuery.getRankingMode() != ThriftSearchRankingMode.RECENCY) {

parsedQuery = parsedQuery.accept(new ProximityGroupRewriteVisitor()).simplify();

}

}

if (request.isSkipVeryRecentTweets()) {

parsedQuery = restrictQueryToFullyIndexedTweets(parsedQuery);

}

parsedQuery = parsedQuery.simplify();

debugInfo.setParsedQuery(parsedQuery.serialize());

// Extract top-level since-id for pagination optimizations.

idTimeRanges = IdTimeRanges.fromQuery(parsedQuery);

// Does any final processing specific to EarlybirdSearch class.

parsedQuery = preLuceneQueryProcess(parsedQuery);

// Convert to a lucene query.

EarlybirdLuceneQueryVisitor luceneVisitor = getLuceneVisitor(

options == null ? null : options.getFieldWeightMapOverride());

if (options != null) {

luceneVisitor

.setProximityPhraseWeight((float) options.getProximityPhraseWeight())

.setProximityPhraseSlop(options.getProximityPhraseSlop());

}

// Propagate hit attribute helper to the lucene visitor if it has been setup.

luceneVisitor.setFieldHitAttributeHelper(this.hitAttributeHelper);

org.apache.lucene.search.Query query = parsedQuery.accept(luceneVisitor);

if (query != null) {

debugInfo.setLuceneQuery(query.toString());

}

queriedFields = luceneVisitor.getQueriedFields();

return query;

}

private Query parseLuceneQuery(String query) {

QueryParser parser = new QueryParser(

EarlybirdFieldConstant.TEXT\_FIELD.getFieldName(),

new SearchWhitespaceAnalyzer());

parser.setSplitOnWhitespace(true);

try {

return parser.parse(query);

} catch (ParseException e) {

LOG.error("Cannot parse raw lucene query: " + query, e);

} catch (NullPointerException e) {

LOG.error("NullPointerException while parsing raw lucene query: " + query

+ ", probably your grammar is wrong.\n", e);

}

return null;

}

private com.twitter.search.queryparser.query.Query combineWithLikedByUserIdFilter64(

com.twitter.search.queryparser.query.Query query,

List<Long> ids) throws QueryParserException {

return QueryNodeUtils.appendAsConjunction(query, getLikedByUserIdQuery(ids));

}

/\*\*

\* initSearcher initializes the segmentSearcher, and returns SUCCESS if OK

\* or some other response code it not OK.

\*/

private EarlybirdResponseCode initSearcher() throws IOException {

searcher = null;

if (searchAllSegments) {

return initMultiSegmentSearcher();

} else {

return initSingleSegmentSearcher();

}

}

private EarlybirdResponseCode initSingleSegmentSearcher() throws IOException {

if (segment == null) {

String message = "Segment not found for time slice: " + request.getSearchSegmentId();

LOG.warn(message);

appendMessage(message);

return EarlybirdResponseCode.PARTITION\_NOT\_FOUND;

}

EarlybirdResponseCode code = this.segmentManager.checkSegment(segment);

if (code != EarlybirdResponseCode.SUCCESS) {

String message = "Segment " + segment + " either disabled or dropped";

LOG.warn(message);

appendMessage(message);

return code;

}

searcher = segmentManager.getSearcher(segment, schemaSnapshot);

if (searcher == null) {

String message = "Could not construct searcher for segment " + segment;

LOG.error(message);

appendMessage(message);

return EarlybirdResponseCode.PERSISTENT\_ERROR;

} else {

appendMessage("Searching segment: " + segment);

return EarlybirdResponseCode.SUCCESS;

}

}

private EarlybirdResponseCode initMultiSegmentSearcher() throws IOException {

EarlybirdMultiSegmentSearcher multiSearcher =

segmentManager.getMultiSearcher(schemaSnapshot);

searcher = multiSearcher;

Preconditions.checkNotNull(searcher);

// Set a top level since id to skip entire segments when possible.

multiSearcher.setIdTimeRanges(idTimeRanges);

return EarlybirdResponseCode.SUCCESS;

}

private com.twitter.search.queryparser.query.Query

restrictQueryToFullyIndexedTweets(com.twitter.search.queryparser.query.Query query) {

long untilTimeSeconds =

RecentTweetRestriction.recentTweetsUntilTime(decider, (int) (clock.nowMillis() / 1000));

if (untilTimeSeconds == 0) {

return query;

}

SearchOperator timeLimit = new SearchOperator(UNTIL\_TIME, untilTimeSeconds);

return new Conjunction(query, timeLimit);

}

private EarlybirdResponse newResponse(EarlybirdResponseCode code, boolean setDebugInfo) {

EarlybirdResponse response = new EarlybirdResponse();

response.setResponseCode(code);

if (setDebugInfo) {

response.setDebugInfo(debugInfo);

if (messageBuffer.length() > 0) {

response.setDebugString(DatabaseConfig.getLocalHostname()

+ ":\n" + messageBuffer.toString());

}

}

return response;

}

private EarlybirdResponse respondError(EarlybirdResponseCode code) {

appendMessage("Responding with error code " + code);

// Always respond with an error message, even when request.debug is false

return newResponse(code, true);

}

@VisibleForTesting

public TerminationTracker getTerminationTracker() {

return terminationTracker;

}

public void maybeSetCollectorDebugInfo(TwitterEarlyTerminationCollector collector) {

if (request.isSetDebugOptions() && request.getDebugOptions().isIncludeCollectorDebugInfo()) {

debugInfo.setCollectorDebugInfo(collector.getDebugInfo());

}

}

public void setTermStatisticsDebugInfo(List<String> termStatisticsDebugInfo) {

debugInfo.setTermStatisticsDebugInfo(termStatisticsDebugInfo);

}

private EarlybirdResponse searchInternal() throws TransientException, ClientException {

searchResults = new ThriftSearchResults();

SearchResultsInfo searchResultsInfo;

try {

switch (queryMode) {

case RECENCY:

searchResultsInfo = processRealtimeQuery();

break;

case RELEVANCE:

// Relevance search and Model-based search differ only on the scoring function used.

SearchTimer timer = searcherStats.createTimer();

timer.start();

searchResultsInfo = processRelevanceQuery();

timer.stop();

searcherStats.recordRelevanceStats(timer, request);

break;

case FACETS:

searchResultsInfo = processFacetsQuery();

break;

case TERM\_STATS:

searchResultsInfo = processTermStatsQuery();

break;

case TOP\_TWEETS:

searchResultsInfo = processTopTweetsQuery();

break;

default:

throw new TransientException("Unknown query mode " + queryMode);

}

return respondSuccess(searchResults, facetResults, termStatisticsResults,

earlyTerminationInfo, searchResultsInfo);

} catch (IOException e) {

throw new TransientException(e.getMessage(), e);

}

}

/\*\*

\* Helper method to process facets query.

\*/

private SearchResultsInfo processFacetsQuery() throws ClientException, IOException {

// figure out which fields we need to count

FacetCountState facetCountState = newFacetCountState();

// Additionally wrap our query into a skip list boolean query for faster counting.

if (!facetRequest.isUsingQueryCache()) {

// Only if all fields to be counted use skip lists, then we can add a required clause

// that filters out all results that do not contain those fields

boolean cannotAddRequiredClause = facetCountState.hasFieldToCountWithoutSkipList();

final Query facetSkipListFilter =

cannotAddRequiredClause ? null : FacetSkipList.getSkipListQuery(facetCountState);

final Query antisocialFilter = UserFlagsExcludeFilter.getUserFlagsExcludeFilter(

segmentManager.getUserTable(), true, true, false);

luceneQuery = wrapFilters(luceneQuery,

facetSkipListFilter,

antisocialFilter);

}

facetResults = new ThriftFacetResults(new HashMap<>());

FacetSearchRequestInfo searchRequestInfo =

new FacetSearchRequestInfo(searchQuery, facetRequest.getFacetRankingOptions(),

luceneQuery, facetCountState, terminationTracker);

searchRequestInfo.setIdTimeRanges(idTimeRanges);

if (searchQuery.getMaxHitsPerUser() > 0) {

antiGamingFilter = new AntiGamingFilter(

searchQuery.getMaxHitsPerUser(),

searchQuery.getMaxTweepcredForAntiGaming(),

luceneQuery);

}

AbstractResultsCollector<

FacetSearchRequestInfo, EarlybirdLuceneSearcher.FacetSearchResults> collector;

if (request.getDebugMode() > 2) {

collector = new ExplainFacetResultsCollector(schemaSnapshot,

searchRequestInfo, antiGamingFilter, searcherStats, clock, request.debugMode);

} else {

collector = new FacetResultsCollector(schemaSnapshot,

searchRequestInfo, antiGamingFilter, searcherStats, clock, request.debugMode);

}

setQueriesInDebugInfo(parsedQuery, searchRequestInfo.getLuceneQuery());

searcher.search(searchRequestInfo.getLuceneQuery(), collector);

EarlybirdLuceneSearcher.FacetSearchResults hits = collector.getResults();

EarlybirdSearchResultUtil.setResultStatistics(searchResults, hits);

earlyTerminationInfo = EarlybirdSearchResultUtil.prepareEarlyTerminationInfo(hits);

Set<Long> userIDWhitelist =

antiGamingFilter != null ? antiGamingFilter.getUserIDWhitelist() : null;

prepareFacetResults(facetResults, hits, facetCountState, userIDWhitelist,

request.getDebugMode());

facetResults.setUserIDWhitelist(userIDWhitelist);

maybeSetCollectorDebugInfo(collector);

if (collector instanceof ExplainFacetResultsCollector) {

((ExplainFacetResultsCollector) collector).setExplanations(facetResults);

}

return hits;

}

/\*\*

\* Helper method to process term-stats query.

\*/

private SearchResultsInfo processTermStatsQuery() throws IOException {

// first extract the terms that we need to count

TermStatisticsRequestInfo searchRequestInfo =

new TermStatisticsRequestInfo(searchQuery, luceneQuery, termStatisticsRequest,

terminationTracker);

searchRequestInfo.setIdTimeRanges(idTimeRanges);

setQueriesInDebugInfo(parsedQuery, searchRequestInfo.getLuceneQuery());

TermStatisticsCollector.TermStatisticsSearchResults hits =

searcher.collectTermStatistics(searchRequestInfo, this, request.getDebugMode());

EarlybirdSearchResultUtil.setResultStatistics(searchResults, hits);

earlyTerminationInfo = EarlybirdSearchResultUtil.prepareEarlyTerminationInfo(hits);

if (hits.results != null) {

termStatisticsResults = new ThriftTermStatisticsResults();

prepareTermStatisticsResults(termStatisticsResults, hits, request.getDebugMode());

}

return hits;

}

/\*\*

\* Helper method to process realtime query.

\*/

private SearchResultsInfo processRealtimeQuery() throws IOException, ClientException {

// Disable maxHitsToProcess.

if (!collectorParams.isSetTerminationParams()) {

collectorParams.setTerminationParams(new CollectorTerminationParams());

collectorParams.getTerminationParams().setMaxHitsToProcess(-1);

COLLECTOR\_PARAMS\_MAX\_HITS\_TO\_PROCESS\_NOT\_SET\_COUNTER.increment();

}

SearchRequestInfo searchRequestInfo = new SearchRequestInfo(

searchQuery, luceneQuery, terminationTracker);

searchRequestInfo.setIdTimeRanges(idTimeRanges);

searchRequestInfo.setHitAttributeHelper(hitAttributeHelper);

searchRequestInfo.setTimestamp(getQueryTimestamp(searchQuery));

AbstractResultsCollector<SearchRequestInfo, SimpleSearchResults> collector;

if (searchQuery.isSetSocialFilterType()) {

if (!searchRequestInfo.getSearchQuery().isSetDirectFollowFilter()

|| !searchRequestInfo.getSearchQuery().isSetTrustedFilter()) {

searcherStats.unsetFiltersForSocialFilterTypeQuery.increment();

throw new ClientException(

"SocialFilterType specified without a TrustedFilter or DirectFollowFilter");

}

SocialFilter socialFilter = new SocialFilter(

searchQuery.getSocialFilterType(),

searchRequestInfo.getSearchQuery().getSearcherId(),

searchRequestInfo.getSearchQuery().getTrustedFilter(),

searchRequestInfo.getSearchQuery().getDirectFollowFilter());

collector = new SocialSearchResultsCollector(

schemaSnapshot,

searchRequestInfo,

socialFilter,

searcherStats,

cluster,

segmentManager.getUserTable(),

request.getDebugMode());

} else {

collector = new SearchResultsCollector(

schemaSnapshot,

searchRequestInfo,

clock,

searcherStats,

cluster,

segmentManager.getUserTable(),

request.getDebugMode());

}

setQueriesInDebugInfo(parsedQuery, luceneQuery);

searcher.search(luceneQuery, collector);

SimpleSearchResults hits = collector.getResults();

EarlybirdSearchResultUtil.setResultStatistics(searchResults, hits);

earlyTerminationInfo = EarlybirdSearchResultUtil.prepareEarlyTerminationInfo(hits);

EarlybirdSearchResultUtil.prepareResultsArray(

searchResults.getResults(), hits, request.debugMode > 0 ? partitionConfig : null);

searchResults.setHitCounts(collector.getHitCountMap());

maybeSetCollectorDebugInfo(collector);

addResultPayloads();

return hits;

}

/\*\*

\* Helper method to process relevance query.

\*/

private SearchResultsInfo processRelevanceQuery() throws IOException, ClientException {

if (!searchQuery.isSetRelevanceOptions()) {

LOG.warn("Relevance query with no relevance options!");

searchQuery.setRelevanceOptions(new ThriftSearchRelevanceOptions());

}

// Note: today the assumption is that if you specify hasSpecifiedTweets,

// you really do want all tweets scored and returned.

final boolean hasSpecifiedTweets = searchQuery.getSearchStatusIdsSize() > 0;

if (hasSpecifiedTweets) {

collectorParams.setNumResultsToReturn(searchQuery.getSearchStatusIdsSize());

}

// If we have explicit user ids, we will want to look at all results from those users, and will

// not need to use the AntiGamingFilter.

final boolean hasSpecifiedFromUserIds = searchQuery.getFromUserIDFilter64Size() > 0;

createRelevanceAntiGamingFilter(hasSpecifiedTweets, hasSpecifiedFromUserIds);

if (searchQuery.getRelevanceOptions().isSetRankingParams()) {

ThriftRankingParams rankingParams = searchQuery.getRelevanceOptions().getRankingParams();

// The score adjustment signals that are passed in the request are disabled for the archive

// cluster or when the features are decidered off. If the request provides those fields,

// we unset them since checking the hashmap when scoring can cause a slight bump in

// latency.

//

// Verify that the signal query specific scores for tweets signal is enabled

if (rankingParams.isSetQuerySpecificScoreAdjustments()) {

if (ALLOW\_QUERY\_SPECIFIC\_SIGNAL\_CONFIG

&& DeciderUtil.isAvailableForRandomRecipient(

decider, ALLOW\_QUERY\_SPECIFIC\_SIGNAL\_DECIDER\_KEY)) {

searcherStats.querySpecificSignalQueriesUsed.increment();

searcherStats.querySpecificSignalMapTotalSize.add(

rankingParams.getQuerySpecificScoreAdjustmentsSize());

} else {

searchQuery.getRelevanceOptions().getRankingParams().unsetQuerySpecificScoreAdjustments();

searcherStats.querySpecificSignalQueriesErased.increment();

}

}

// Verify that the signal author specific scores signal is enabled

if (rankingParams.isSetAuthorSpecificScoreAdjustments()) {

if (ALLOW\_AUTHOR\_SPECIFIC\_SIGNAL\_CONFIG

&& DeciderUtil.isAvailableForRandomRecipient(

decider, ALLOW\_AUTHOR\_SPECIFIC\_SIGNAL\_DECIDER\_KEY)) {

searcherStats.authorSpecificSignalQueriesUsed.increment();

searcherStats.authorSpecificSignalMapTotalSize.add(

rankingParams.getAuthorSpecificScoreAdjustmentsSize());

} else {

searchQuery.getRelevanceOptions().getRankingParams()

.unsetAuthorSpecificScoreAdjustments();

searcherStats.authorSpecificSignalQueriesErased.increment();

}

}

}

ScoringFunction scoringFunction =

new ScoringFunctionProvider.DefaultScoringFunctionProvider(

request, schemaSnapshot, searchQuery, antiGamingFilter,

segmentManager.getUserTable(), hitAttributeHelper,

parsedQuery, scoringModelsManager, tensorflowModelsManager)

.getScoringFunction();

scoringFunction.setDebugMode(request.getDebugMode());

RelevanceQuery relevanceQuery = new RelevanceQuery(luceneQuery, scoringFunction);

RelevanceSearchRequestInfo searchRequestInfo =

new RelevanceSearchRequestInfo(

searchQuery, relevanceQuery, terminationTracker, qualityFactor);

searchRequestInfo.setIdTimeRanges(idTimeRanges);

searchRequestInfo.setHitAttributeHelper(hitAttributeHelper);

searchRequestInfo.setTimestamp(getQueryTimestamp(searchQuery));

if (shouldUseTensorFlowCollector()

&& searchQuery.getRelevanceOptions().isUseRelevanceAllCollector()) {

throw new ClientException("Tensorflow scoring does not work with the RelevanceAllCollector");

}

final AbstractRelevanceCollector collector;

// First check if the Tensorflow results collector should be used, because the

// TensorflowBasedScoringFunction only works with the BatchRelevanceTopCollector

if (shouldUseTensorFlowCollector()) {

// Collect top numResults.

collector = new BatchRelevanceTopCollector(

schemaSnapshot,

searchRequestInfo,

scoringFunction,

searcherStats,

cluster,

segmentManager.getUserTable(),

clock,

request.getDebugMode());

} else if (hasSpecifiedTweets

|| searchQuery.getRelevanceOptions().isUseRelevanceAllCollector()) {

// Collect all.

collector = new RelevanceAllCollector(

schemaSnapshot,

searchRequestInfo,

scoringFunction,

searcherStats,

cluster,

segmentManager.getUserTable(),

clock,

request.getDebugMode());

} else {

// Collect top numResults.

collector = new RelevanceTopCollector(

schemaSnapshot,

searchRequestInfo,

scoringFunction,

searcherStats,

cluster,

segmentManager.getUserTable(),

clock,

request.getDebugMode());

}

// Make sure that the Tensorflow scoring function and the Tensorflow results collector are

// always used together. If this fails it will result in a TRANSIENT\_ERROR response.

Preconditions.checkState((collector instanceof BatchRelevanceTopCollector)

== (scoringFunction instanceof TensorflowBasedScoringFunction));

setQueriesInDebugInfo(parsedQuery, searchRequestInfo.getLuceneQuery());

searcher.search(searchRequestInfo.getLuceneQuery(), collector);

RelevanceSearchResults hits = collector.getResults();

EarlybirdSearchResultUtil.setResultStatistics(searchResults, hits);

searchResults.setScoringTimeNanos(hits.getScoringTimeNanos());

earlyTerminationInfo = EarlybirdSearchResultUtil.prepareEarlyTerminationInfo(hits);

EarlybirdSearchResultUtil.setLanguageHistogram(searchResults, collector.getLanguageHistogram());

EarlybirdSearchResultUtil.prepareRelevanceResultsArray(

searchResults.getResults(),

hits,

antiGamingFilter != null ? antiGamingFilter.getUserIDWhitelist() : null,

request.getDebugMode() > 0 ? partitionConfig : null);

searchResults.setHitCounts(collector.getHitCountMap());

searchResults.setRelevanceStats(hits.getRelevanceStats());

maybeSetCollectorDebugInfo(collector);

if (explanationsEnabled(request.getDebugMode())) {

searcher.explainSearchResults(searchRequestInfo, hits, searchResults);

}

addResultPayloads();

return hits;

}

public static boolean explanationsEnabled(int debugLevel) {

return debugLevel > 1;

}

private boolean shouldUseTensorFlowCollector() {

return tensorflowModelsManager.isEnabled()

&& searchQuery.getRelevanceOptions().isSetRankingParams()

&& searchQuery.getRelevanceOptions().getRankingParams().isSetType()

&& searchQuery.getRelevanceOptions().getRankingParams().getType()

== ThriftScoringFunctionType.TENSORFLOW\_BASED;

}

/\*\*

\* Optionally, if requested and needed, will create a new AntiGamingFilter. Otherwize, no

\* AntiGamingFilter will be used for this query.

\* @param hasSpecifiedTweets whether the request has searchStatusIds specified.

\* @param hasSpecifiedFromUserIds whether the request has fromUserIDFilter64 specified.

\*/

private void createRelevanceAntiGamingFilter(

boolean hasSpecifiedTweets, boolean hasSpecifiedFromUserIds) {

// Anti-gaming filter (turned off for specified tweets mode, or when you're explicitly asking

// for specific users' tweets).

if (searchQuery.getMaxHitsPerUser() > 0 && !hasSpecifiedTweets && !hasSpecifiedFromUserIds) {

searcherStats.relevanceAntiGamingFilterUsed.increment();

antiGamingFilter = new AntiGamingFilter(

searchQuery.getMaxHitsPerUser(),

searchQuery.getMaxTweepcredForAntiGaming(),

luceneQuery);

} else if (searchQuery.getMaxHitsPerUser() <= 0) {

searcherStats.relevanceAntiGamingFilterNotRequested.increment();

} else if (hasSpecifiedTweets && hasSpecifiedFromUserIds) {

searcherStats.relevanceAntiGamingFilterSpecifiedTweetsAndFromUserIds.increment();

} else if (hasSpecifiedTweets) {

searcherStats.relevanceAntiGamingFilterSpecifiedTweets.increment();

} else if (hasSpecifiedFromUserIds) {

searcherStats.relevanceAntiGamingFilterSpecifiedFromUserIds.increment();

}

}

/\*\*

\* Check to make sure that there are no nullcast documents in results. If there exists nullcasts

\* in results, we should log error and increment counters correspondingly.

\*/

@VisibleForTesting

public void logAndIncrementStatsIfNullcastInResults(ThriftSearchResults thriftSearchResults) {

if (!thriftSearchResults.isSetResults()) {

return;

}

Set<Long> unexpectedNullcastStatusIds =

EarlybirdResponseUtil.findUnexpectedNullcastStatusIds(thriftSearchResults, request);

if (!unexpectedNullcastStatusIds.isEmpty()) {

searcherStats.nullcastUnexpectedQueries.increment();

searcherStats.nullcastUnexpectedResults.add(unexpectedNullcastStatusIds.size());

String base64Request;

try {

base64Request = ThriftUtils.toBase64EncodedString(request);

} catch (TException e) {

base64Request = "Failed to parse base 64 request";

}

LOG.error(

"Found unexpected nullcast tweets: {} | parsedQuery: {} | request: {} | response: {} | "

+ "request base 64: {}",

Joiner.on(",").join(unexpectedNullcastStatusIds),

parsedQuery.serialize(),

request,

thriftSearchResults,

base64Request);

}

}

private void addResultPayloads() throws IOException {

if (searchQuery.getResultMetadataOptions() != null) {

if (searchQuery.getResultMetadataOptions().isGetTweetUrls()) {

searcher.fillFacetResults(new ExpandedUrlCollector(), searchResults);

}

if (searchQuery.getResultMetadataOptions().isGetNamedEntities()) {

searcher.fillFacetResults(new NamedEntityCollector(), searchResults);

}

if (searchQuery.getResultMetadataOptions().isGetEntityAnnotations()) {

searcher.fillFacetResults(new EntityAnnotationCollector(), searchResults);

}

if (searchQuery.getResultMetadataOptions().isGetSpaces()) {

searcher.fillFacetResults(new SpaceFacetCollector(audioSpaceTable), searchResults);

}

}

}

/\*\*

\* Helper method to process top tweets query.

\*/

private SearchResultsInfo processTopTweetsQuery() throws IOException, ClientException {

// set dummy relevance options if it's not available, but this shouldn't happen in prod

if (!searchQuery.isSetRelevanceOptions()) {

searchQuery.setRelevanceOptions(new ThriftSearchRelevanceOptions());

}

if (!searchQuery.getRelevanceOptions().isSetRankingParams()) {

searchQuery.getRelevanceOptions().setRankingParams(

// this is important, or it's gonna pick DefaultScoringFunction which pretty much

// does nothing.

new ThriftRankingParams().setType(ThriftScoringFunctionType.TOPTWEETS));

}

ScoringFunction scoringFunction = new ScoringFunctionProvider.DefaultScoringFunctionProvider(

request, schemaSnapshot, searchQuery, null,

segmentManager.getUserTable(), hitAttributeHelper, parsedQuery,

scoringModelsManager, tensorflowModelsManager)

.getScoringFunction();

scoringFunction.setDebugMode(request.getDebugMode());

RelevanceQuery relevanceQuery = new RelevanceQuery(luceneQuery, scoringFunction);

RelevanceSearchRequestInfo searchRequestInfo =

new RelevanceSearchRequestInfo(

searchQuery, relevanceQuery, terminationTracker, qualityFactor);

searchRequestInfo.setIdTimeRanges(idTimeRanges);

searchRequestInfo.setTimestamp(getQueryTimestamp(searchQuery));

final AbstractRelevanceCollector collector =

new RelevanceTopCollector(

schemaSnapshot,

searchRequestInfo,

scoringFunction,

searcherStats,

cluster,

segmentManager.getUserTable(),

clock,

request.getDebugMode());

setQueriesInDebugInfo(parsedQuery, searchRequestInfo.getLuceneQuery());

searcher.search(searchRequestInfo.getLuceneQuery(), collector);

RelevanceSearchResults hits = collector.getResults();

EarlybirdSearchResultUtil.setResultStatistics(searchResults, hits);

searchResults.setScoringTimeNanos(hits.getScoringTimeNanos());

earlyTerminationInfo = EarlybirdSearchResultUtil.prepareEarlyTerminationInfo(hits);

EarlybirdSearchResultUtil.setLanguageHistogram(

searchResults,

collector.getLanguageHistogram());

EarlybirdSearchResultUtil.prepareRelevanceResultsArray(

searchResults.getResults(),

hits,

null,

request.getDebugMode() > 0 ? partitionConfig : null);

searchResults.setHitCounts(collector.getHitCountMap());

searchResults.setRelevanceStats(hits.getRelevanceStats());

maybeSetCollectorDebugInfo(collector);

if (explanationsEnabled(request.getDebugMode())

&& searchQuery.isSetRelevanceOptions()

&& searchQuery.getRelevanceOptions().isSetRankingParams()) {

searcher.explainSearchResults(searchRequestInfo, hits, searchResults);

}

addResultPayloads();

return hits;

}

private FacetCountState newFacetCountState() throws ClientException {

int minNumFacetResults = DEFAULT\_NUM\_FACET\_RESULTS;

if (facetRequest.isSetFacetRankingOptions()

&& facetRequest.getFacetRankingOptions().isSetNumCandidatesFromEarlybird()) {

minNumFacetResults = facetRequest.getFacetRankingOptions().getNumCandidatesFromEarlybird();

}

// figure out which fields we need to count

FacetCountState facetCountState = new FacetCountState(schemaSnapshot, minNumFacetResults);

// all categories if none!

if (facetRequest.getFacetFields() == null || facetRequest.getFacetFields().isEmpty()) {

for (Schema.FieldInfo facetField : schemaSnapshot.getFacetFields()) {

facetCountState.addFacet(

facetField.getFieldType().getFacetName(), DEFAULT\_NUM\_FACET\_RESULTS);

}

} else {

Iterator<ThriftFacetFieldRequest> it = facetRequest.getFacetFieldsIterator();

while (it.hasNext()) {

ThriftFacetFieldRequest facetFieldRequest = it.next();

Schema.FieldInfo facet = schemaSnapshot.getFacetFieldByFacetName(

facetFieldRequest.getFieldName());

if (facet != null) {

facetCountState.addFacet(

facet.getFieldType().getFacetName(), facetFieldRequest.getNumResults());

} else {

throw new ClientException("Unknown facet field: " + facetFieldRequest.getFieldName());

}

}

}

return facetCountState;

}

private com.twitter.search.queryparser.query.Query preLuceneQueryProcess(

com.twitter.search.queryparser.query.Query twitterQuery) throws QueryParserException {

com.twitter.search.queryparser.query.Query query = twitterQuery;

if (searchHighFrequencyTermPairs && !includesCardField(searchQuery, query)) {

// Process high frequency term pairs. Works best when query is as flat as possible.

query = HighFrequencyTermPairRewriteVisitor.safeRewrite(

query,

DeciderUtil.isAvailableForRandomRecipient(

decider, "enable\_hf\_term\_pair\_negative\_disjunction\_rewrite"));

}

return query.simplify();

}

private Query postLuceneQueryProcess(final Query query) throws ClientException {

if (StringUtils.isBlank(request.getSearchQuery().getSerializedQuery())

&& StringUtils.isBlank(request.getSearchQuery().getLuceneQuery())) {

searcherStats.numRequestsWithBlankQuery.get(queryMode).increment();

if (searchQuery.getSearchStatusIdsSize() == 0

&& searchQuery.getFromUserIDFilter64Size() == 0

&& searchQuery.getLikedByUserIDFilter64Size() == 0) {

// No query or ids to search. This is only allowed in some modes.

if (queryMode == QueryMode.RECENCY

|| queryMode == QueryMode.RELEVANCE

|| queryMode == QueryMode.TOP\_TWEETS) {

throw new ClientException(

"No query or status ids for " + queryMode.toString().toLowerCase() + " query");

}

}

}

// Wrap the query as needed with additional query filters.

List<Query> filters = Lists.newArrayList();

// Min tweep cred filter.

if (searchQuery.isSetMinTweepCredFilter()) {

searcherStats.addedFilterBadUserRep.increment();

filters.add(BadUserRepFilter.getBadUserRepFilter(searchQuery.getMinTweepCredFilter()));

}

if (searchQuery.getFromUserIDFilter64Size() > 0) {

this.queriedFields.add(EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName());

this.searcherStats.addedFilterFromUserIds.increment();

try {

filters.add(UserIdMultiSegmentQuery.createIdDisjunctionQuery(

"from\_user\_id\_filter",

searchQuery.getFromUserIDFilter64(),

EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName(),

schemaSnapshot,

multiSegmentTermDictionaryManager,

decider,

cluster,

Lists.newArrayList(),

null,

queryTimeoutFactory.createQueryTimeout(request, terminationTracker, clock)));

} catch (QueryParserException e) {

throw new ClientException(e);

}

}

// Wrap the lucene query with these filters.

Query wrappedQuery = wrapFilters(query, filters.toArray(new Query[filters.size()]));

// If searchStatusIds is set, additionally modify the query to search exactly these

// ids, using the luceneQuery only for scoring.

if (searchQuery.getSearchStatusIdsSize() > 0) {

this.searcherStats.addedFilterTweetIds.increment();

final Query queryForScoring = wrappedQuery;

final Query queryForRetrieval =

RequiredStatusIDsFilter.getRequiredStatusIDsQuery(searchQuery.getSearchStatusIds());

return new BooleanQuery.Builder()

.add(queryForRetrieval, Occur.MUST)

.add(queryForScoring, Occur.SHOULD)

.build();

}

return wrappedQuery;

}

private com.twitter.search.queryparser.query.Query getLikedByUserIdQuery(

List<Long> ids) throws QueryParserException {

if (DeciderUtil.isAvailableForRandomRecipient(

decider, USE\_MULTI\_TERM\_DISJUNCTION\_FOR\_LIKED\_BY\_USER\_IDS\_DECIDER\_KEY)) {

// rewrite LikedByUserIdFilter64 to a multi\_term\_disjuntion query

return createMultiTermDisjunctionQueryForLikedByUserIds(ids);

} else {

// rewrite LikedByUserIdFilter64 to a disjunction of multiple liked\_by\_user\_ids query

return createDisjunctionQueryForLikedByUserIds(ids);

}

}

/\*\*

\* Returns the Lucene query visitor that should be applied to the original request.

\*

\* @param fieldWeightMapOverride The per-field weight overrides.

\*/

@VisibleForTesting

public EarlybirdLuceneQueryVisitor getLuceneVisitor(

Map<String, Double> fieldWeightMapOverride) {

String clusterName = cluster.getNameForStats();

// Iff in relevance mode \_and\_ intepreteSinceId is false, we turn off since\_id

// operator by using LuceneRelevanceQueryVisitor.

if (searchQuery.getRankingMode() == ThriftSearchRankingMode.RELEVANCE

&& searchQuery.getRelevanceOptions() != null

&& !searchQuery.getRelevanceOptions().isInterpretSinceId()) {

// hack! reset top level since id, which is the same thing LuceneRelevanceVisitor

// is doing.

idTimeRanges = null;

return new LuceneRelevanceQueryVisitor(

schemaSnapshot,

queryCacheManager,

segmentManager.getUserTable(),

segmentManager.getUserScrubGeoMap(),

terminationTracker,

FieldWeightDefault.overrideFieldWeightMap(

schemaSnapshot.getFieldWeightMap(),

dropBadFieldWeightOverrides(fieldWeightMapOverride, decider, clusterName)),

MAPPABLE\_FIELD\_MAP,

multiSegmentTermDictionaryManager,

decider,

cluster,

queryTimeoutFactory.createQueryTimeout(

request, terminationTracker, clock));

} else {

return new EarlybirdLuceneQueryVisitor(

schemaSnapshot,

queryCacheManager,

segmentManager.getUserTable(),

segmentManager.getUserScrubGeoMap(),

terminationTracker,

FieldWeightDefault.overrideFieldWeightMap(

schemaSnapshot.getFieldWeightMap(),

dropBadFieldWeightOverrides(fieldWeightMapOverride, decider, clusterName)),

MAPPABLE\_FIELD\_MAP,

multiSegmentTermDictionaryManager,

decider,

cluster,

queryTimeoutFactory.createQueryTimeout(

request, terminationTracker, clock));

}

}

private void prepareFacetResults(ThriftFacetResults thriftFacetResults,

EarlybirdLuceneSearcher.FacetSearchResults hits,

FacetCountState<ThriftFacetFieldResults> facetCountState,

Set<Long> userIDWhitelist,

byte debugMode) throws IOException {

for (FacetRankingModule rankingModule : FacetRankingModule.REGISTERED\_RANKING\_MODULES) {

rankingModule.prepareResults(hits, facetCountState);

}

Map<Term, ThriftFacetCount> allFacetResults = new HashMap<>();

Iterator<FacetCountState.FacetFieldResults<ThriftFacetFieldResults>> fieldResultsIterator =

facetCountState.getFacetFieldResultsIterator();

while (fieldResultsIterator.hasNext()) {

FacetCountState.FacetFieldResults<ThriftFacetFieldResults> facetFieldResults =

fieldResultsIterator.next();

if (facetFieldResults.results == null) {

// return empty resultset for this facet

List<ThriftFacetCount> emptyList = new ArrayList<>();

facetFieldResults.results = new ThriftFacetFieldResults(emptyList, 0);

}

thriftFacetResults.putToFacetFields(facetFieldResults.facetName,

facetFieldResults.results);

Schema.FieldInfo field = schemaSnapshot.getFacetFieldByFacetName(

facetFieldResults.facetName);

for (ThriftFacetCount result : facetFieldResults.results.topFacets) {

if (result.facetLabel != null) {

allFacetResults.put(new Term(field.getName(), result.facetLabel), result);

} else {

LOG.warn("Null facetLabel, field: {}, result: {}", field.getName(), result);

}

}

}

searcher.fillFacetResultMetadata(allFacetResults, schemaSnapshot, debugMode);

if (userIDWhitelist != null) {

for (ThriftFacetCount facetCount : allFacetResults.values()) {

ThriftFacetCountMetadata metadata = facetCount.getMetadata();

if (metadata != null) {

metadata.setDontFilterUser(userIDWhitelist.contains(metadata.getTwitterUserId()));

}

}

}

}

private void prepareTermStatisticsResults(

ThriftTermStatisticsResults termStatistics,

TermStatisticsCollector.TermStatisticsSearchResults hits,

byte debugMode) throws IOException {

termStatistics.setBinIds(hits.binIds);

termStatistics.setHistogramSettings(termStatisticsRequest.getHistogramSettings());

termStatistics.setTermResults(hits.results);

setTermStatisticsDebugInfo(hits.getTermStatisticsDebugInfo());

if (hits.lastCompleteBinId != -1) {

termStatistics.setMinCompleteBinId(hits.lastCompleteBinId);

} else {

SearchRateCounter.export(String.format(

"term\_stats\_%s\_unset\_min\_complete\_bin\_id", request.getClientId())).increment();

}

if (idTimeRanges != null

&& idTimeRanges.getUntilTimeExclusive().isPresent()

&& hits.getMinSearchedTime() > idTimeRanges.getUntilTimeExclusive().get()) {

SearchRateCounter.export(String.format(

"term\_stats\_%s\_min\_searched\_time\_after\_until\_time", request.getClientId())).increment();

}

searcher.fillTermStatsMetadata(termStatistics, schemaSnapshot, debugMode);

}

private EarlybirdResponse respondSuccess(

ThriftSearchResults thriftSearchResults,

ThriftFacetResults thriftFacetResults,

ThriftTermStatisticsResults termStatisticResults,

@Nonnull EarlyTerminationInfo earlyTerminationState,

@Nonnull SearchResultsInfo searchResultsInfo) {

Preconditions.checkNotNull(earlyTerminationState);

Preconditions.checkNotNull(searchResultsInfo);

exportEarlyTerminationStats(earlyTerminationState);

EarlybirdResponse response =

newResponse(EarlybirdResponseCode.SUCCESS, request.getDebugMode() > 0);

response.setEarlyTerminationInfo(earlyTerminationState);

response.setNumSearchedSegments(searchResultsInfo.getNumSearchedSegments());

if (thriftSearchResults != null) {

// Nullcast check is only used when parsed query is available: if there is no parsed query,

// we would not add possible exclude nullcast filter.

if (parsedQuery != null && !parsedQueryAllowNullcast) {

logAndIncrementStatsIfNullcastInResults(thriftSearchResults);

}

response.setSearchResults(thriftSearchResults);

} else {

RESPONSE\_HAS\_NO\_THRIFT\_SEARCH\_RESULTS.increment();

}

if (thriftFacetResults != null) {

response.setFacetResults(thriftFacetResults);

}

if (termStatisticResults != null) {

response.setTermStatisticsResults(termStatisticResults);

}

appendFeatureSchemaIfNeeded(response);

appendLikedByUserIdsIfNeeded(response);

return response;

}

private void exportEarlyTerminationStats(@Nonnull EarlyTerminationInfo earlyTerminationState) {

if (earlyTerminationState.isSetEarlyTerminationReason()) {

SearchRateCounter.export(String.format("early\_termination\_%s\_%s",

ClientIdUtil.formatClientId(request.getClientId()),

earlyTerminationState.getEarlyTerminationReason())).increment();

SearchRateCounter.export(String.format("early\_termination\_%s\_%s",

ClientIdUtil.formatClientIdAndRequestType(

request.getClientId(), queryMode.name().toLowerCase()),

earlyTerminationState.getEarlyTerminationReason())).increment();

}

}

/\*\*

\* Builds a rank -> userId map for liked\_by\_user\_id queries that request hit attribution, and

\* appends the resulting map to the response.

\*/

private void appendLikedByUserIdsIfNeeded(EarlybirdResponse response) {

// Check if user asked for likedByUserIds list in response

ThriftSearchRelevanceOptions resultRelevanceOptions =

request.getSearchQuery().getRelevanceOptions();

if ((resultRelevanceOptions == null)

|| !resultRelevanceOptions.isCollectFieldHitAttributions()) {

return;

}

// Make sure we have results in response and hit attribution helper is set up correctly

if (!response.isSetSearchResults() || hitAttributeHelper == null) {

return;

}

// Get rank to node map

Map<com.twitter.search.queryparser.query.Query, Integer> nodeToRankMap =

Preconditions.checkNotNull(hitAttributeHelper.getNodeToRankMap());

Map<com.twitter.search.queryparser.query.Query, List<Integer>> expandedNodeToRankMap =

Preconditions.checkNotNull(hitAttributeHelper.getExpandedNodeToRankMap());

// Build a rank to id map

ImmutableMap.Builder<Integer, Long> builder = ImmutableMap.builder();

for (com.twitter.search.queryparser.query.Query query : nodeToRankMap.keySet()) {

if (query instanceof SearchOperator) {

SearchOperator op = (SearchOperator) query;

if (expandedNodeToRankMap.containsKey(query)) {

// for multi\_term\_disjunction case

List<Integer> ranks = expandedNodeToRankMap.get(op);

Preconditions.checkArgument(op.getNumOperands() == ranks.size() + 1);

for (int i = 0; i < ranks.size(); ++i) {

builder.put(ranks.get(i), Long.valueOf(op.getOperands().get(i + 1)));

}

} else if (op.getOperatorType() == SearchOperator.Type.LIKED\_BY\_USER\_ID) {

// for liked\_by\_user\_id case

Preconditions.checkArgument(op.getAnnotationOf(Annotation.Type.NODE\_RANK).isPresent());

builder.put(

(Integer) op.getAnnotationOf(Annotation.Type.NODE\_RANK).get().getValue(),

Long.valueOf(op.getOperands().get(0)));

}

}

}

Map<Integer, Long> rankToIdMap = builder.build();

// Append liked\_by\_user\_id filed into result

for (ThriftSearchResult result : response.getSearchResults().getResults()) {

if (result.isSetMetadata()

&& result.getMetadata().isSetFieldHitAttribution()

&& result.getMetadata().getFieldHitAttribution().isSetHitMap()) {

List<Long> likedByUserIdList = Lists.newArrayList();

Map<Integer, FieldHitList> hitMap =

result.getMetadata().getFieldHitAttribution().getHitMap();

// iterate hit attributions

for (int rank : hitMap.keySet()) {

if (rankToIdMap.containsKey(rank)) {

likedByUserIdList.add(rankToIdMap.get(rank));

}

}

if (!result.getMetadata().isSetExtraMetadata()) {

result.getMetadata().setExtraMetadata(new ThriftSearchResultExtraMetadata());

}

result.getMetadata().getExtraMetadata().setLikedByUserIds(likedByUserIdList);

}

}

}

private void appendFeatureSchemaIfNeeded(EarlybirdResponse response) {

// Do not append the schema if the client didn't request it.

ThriftSearchResultMetadataOptions resultMetadataOptions =

request.getSearchQuery().getResultMetadataOptions();

if ((resultMetadataOptions == null) || !resultMetadataOptions.isReturnSearchResultFeatures()) {

return;

}

if (!response.isSetSearchResults()) {

return;

}

ThriftSearchFeatureSchema featureSchema = schemaSnapshot.getSearchFeatureSchema();

Preconditions.checkState(

featureSchema.isSetSchemaSpecifier(),

"The feature schema doesn't have a schema specifier set: {}", featureSchema);

// If the client has this schema, we only need to return the schema version.

// If the client doesn't have this schema, we need to return the schema entries too.

if (resultMetadataOptions.isSetFeatureSchemasAvailableInClient()

&& resultMetadataOptions.getFeatureSchemasAvailableInClient().contains(

featureSchema.getSchemaSpecifier())) {

CLIENT\_HAS\_FEATURE\_SCHEMA\_COUNTER.increment();

ThriftSearchFeatureSchema responseFeatureSchema = new ThriftSearchFeatureSchema();

responseFeatureSchema.setSchemaSpecifier(featureSchema.getSchemaSpecifier());

response.getSearchResults().setFeatureSchema(responseFeatureSchema);

} else {

CLIENT\_DOESNT\_HAVE\_FEATURE\_SCHEMA\_COUNTER.increment();

Preconditions.checkState(featureSchema.isSetEntries(),

"Entries are not set in the feature schema: " + featureSchema);

response.getSearchResults().setFeatureSchema(featureSchema);

}

}

private static long getQueryTimestamp(ThriftSearchQuery query) {

return query != null && query.isSetTimestampMsecs() ? query.getTimestampMsecs() : 0;

}

private static boolean includesCardField(ThriftSearchQuery searchQuery,

com.twitter.search.queryparser.query.Query query)

throws QueryParserException {

if (searchQuery.isSetRelevanceOptions()) {

ThriftSearchRelevanceOptions options = searchQuery.getRelevanceOptions();

if (options.isSetFieldWeightMapOverride()

&& (options.getFieldWeightMapOverride().containsKey(

EarlybirdFieldConstant.CARD\_TITLE\_FIELD.getFieldName())

|| options.getFieldWeightMapOverride()

.containsKey(EarlybirdFieldConstant.CARD\_DESCRIPTION\_FIELD.getFieldName()))) {

return true;

}

}

return query.accept(new DetectFieldAnnotationVisitor(ImmutableSet.of(

EarlybirdFieldConstant.CARD\_TITLE\_FIELD.getFieldName(),

EarlybirdFieldConstant.CARD\_DESCRIPTION\_FIELD.getFieldName())));

}

private static QueryMode getQueryMode(EarlybirdRequest request) {

if (request.isSetFacetRequest()) {

return QueryMode.FACETS;

} else if (request.isSetTermStatisticsRequest()) {

return QueryMode.TERM\_STATS;

}

// Recency mode until we determine otherwise.

QueryMode queryMode = QueryMode.RECENCY;

ThriftSearchQuery searchQuery = request.getSearchQuery();

if (searchQuery != null) {

switch (searchQuery.getRankingMode()) {

case RECENCY:

queryMode = QueryMode.RECENCY;

break;

case RELEVANCE:

queryMode = QueryMode.RELEVANCE;

break;

case TOPTWEETS:

queryMode = QueryMode.TOP\_TWEETS;

break;

default:

break;

}

}

if (searchQuery == null

|| !searchQuery.isSetSerializedQuery()

|| searchQuery.getSerializedQuery().isEmpty()) {

LOG.debug("Search query was empty, query mode was " + queryMode);

}

return queryMode;

}

private static <V> ImmutableMap<String, V> dropBadFieldWeightOverrides(

Map<String, V> map, Decider decider, String clusterName) {

if (map == null) {

return null;

}

FIELD\_WEIGHT\_OVERRIDE\_MAP\_NON\_NULL\_COUNT.increment();

ImmutableMap.Builder<String, V> builder = ImmutableMap.builder();

for (Map.Entry<String, V> entry : map.entrySet()) {

if (EarlybirdFieldConstant.CAMELCASE\_USER\_HANDLE\_FIELD.getFieldName().equals(entry.getKey())

&& !isAllowedCamelcaseUsernameFieldWeightOverride(decider, clusterName)) {

DROPPED\_CAMELCASE\_USERNAME\_FIELD\_WEIGHT\_OVERRIDE.increment();

} else if (EarlybirdFieldConstant.TOKENIZED\_USER\_NAME\_FIELD.getFieldName().equals(

entry.getKey())

&& !isAllowedTokenizedScreenNameFieldWeightOverride(decider, clusterName)) {

DROPPED\_TOKENIZED\_DISPLAY\_NAME\_FIELD\_WEIGHT\_OVERRIDE.increment();

} else {

builder.put(entry.getKey(), entry.getValue());

}

}

return builder.build();

}

private static boolean isAllowedCamelcaseUsernameFieldWeightOverride(

Decider decider, String clusterName) {

return DeciderUtil.isAvailableForRandomRecipient(decider,

ALLOW\_CAMELCASE\_USERNAME\_FIELD\_WEIGHT\_OVERRIDE\_DECIDER\_KEY\_PREFIX + clusterName);

}

private static boolean isAllowedTokenizedScreenNameFieldWeightOverride(

Decider decider, String clusterName) {

return DeciderUtil.isAvailableForRandomRecipient(decider,

ALLOW\_TOKENIZED\_DISPLAY\_NAME\_FIELD\_WEIGHT\_OVERRIDE\_DECIDER\_KEY\_PREFIX + clusterName);

}

private static com.twitter.search.queryparser.query.Query

createMultiTermDisjunctionQueryForLikedByUserIds(List<Long> ids) throws QueryParserException {

List<String> operands = new ArrayList<>(ids.size() + 1);

operands.add(EarlybirdFieldConstant.LIKED\_BY\_USER\_ID\_FIELD.getFieldName());

for (long id : ids) {

operands.add(String.valueOf(id));

}

return new SearchOperator(SearchOperator.Type.MULTI\_TERM\_DISJUNCTION, operands)

.simplify();

}

private static com.twitter.search.queryparser.query.Query createDisjunctionQueryForLikedByUserIds(

List<Long> ids) throws QueryParserException {

return new Disjunction(

ids.stream()

.map(id -> new SearchOperator(SearchOperator.Type.LIKED\_BY\_USER\_ID, id))

.collect(Collectors.toList()))

.simplify();

}

public com.twitter.search.queryparser.query.Query getParsedQuery() {

return parsedQuery;

}

/\*\*

\* Get the index fields that were queried after this searcher completed its job.

\* @return

\*/

public Set<String> getQueriedFields() {

return queriedFields;

}

public Query getLuceneQuery() {

return luceneQuery;

}

}