package com.twitter.search.earlybird.querycache;

import java.util.List;

import java.util.TreeMap;

import com.google.common.base.Preconditions;

import org.apache.lucene.search.Query;

import com.twitter.common.collections.Pair;

import com.twitter.common.quantity.Amount;

import com.twitter.common.quantity.Time;

import com.twitter.common.util.Clock;

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

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

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

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

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

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

import com.twitter.search.common.util.text.regex.Regex;

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

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

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

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

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

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

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

/\*\*

\* The definition of a QueryCache filter/entry, like the name of the filter, the query used

\* to populate the cache, update schedule, etc..

\*

\* Instances of this class are created by the YAML loader when loading the config file. Most

\* members are populated by YAML using setters through reflection.

\*/

public class QueryCacheFilter {

// Data structure type supported as cache result holder

public enum ResultSetType {

FixedBitSet,

SparseFixedBitSet

}

// Fields set directly from YML config file.

private String filterName; // unique name for cached filter

private String query; // serialized query string

private ResultSetType resultType;

private boolean cacheModeOnly;

private List<UpdateInterval> schedule;

private SearchCounter queries;

// Fields generated based on config (but not directly).

private volatile Pair<ThriftSearchQuery, Query> queryPair;

private TreeMap<Integer, UpdateInterval> scheduleMap; // tree map from index to interval

public class InvalidEntryException extends Exception {

public InvalidEntryException(String message) {

super("Filter [" + filterName + "]: " + message);

}

}

public static class UpdateInterval {

// Overrides \*all\* query cache update frequencies to be this value, in seconds.

private final int overrideSecondsForTests = EarlybirdConfig.getInt(

"override\_query\_cache\_update\_frequency", -1);

// Fields set directly from YML config file.

private int segment;

private long seconds;

public void setSegment(int segment) {

this.segment = segment;

}

/\*\*

\* Sets the update period in seconds. If the override\_query\_cache\_update\_frequency parameter is

\* specified in the earlybird configuration, its value is used instead (the value passed to this

\* method is ignored).

\*/

public void setSeconds(long seconds) {

if (overrideSecondsForTests != -1) {

this.seconds = overrideSecondsForTests;

} else {

this.seconds = seconds;

}

}

public int getSegment() {

return segment;

}

public long getSeconds() {

return seconds;

}

}

public void setFilterName(String filterName) throws InvalidEntryException {

sanityCheckFilterName(filterName);

this.filterName = filterName;

}

/\*\*

\* Sets the driving query for this query cache filter.

\*/

public void setQuery(String query) throws InvalidEntryException {

if (query == null || query.isEmpty()) {

throw new InvalidEntryException("Empty query string");

}

this.query = query;

}

/\*\*

\* Sets the type of the results that will be generated by this query cache filter.

\*/

public void setResultType(String resultType) throws InvalidEntryException {

if (ResultSetType.FixedBitSet.toString().equalsIgnoreCase(resultType)) {

this.resultType = ResultSetType.FixedBitSet;

} else if (ResultSetType.SparseFixedBitSet.toString().equalsIgnoreCase(resultType)) {

this.resultType = ResultSetType.SparseFixedBitSet;

} else {

throw new InvalidEntryException("Unregconized result type [" + resultType + "]");

}

}

public void setCacheModeOnly(boolean cacheModeOnly) {

this.cacheModeOnly = cacheModeOnly;

}

public void setSchedule(List<UpdateInterval> schedule)

throws QueryCacheFilter.InvalidEntryException {

sanityCheckSchedule(schedule);

this.schedule = schedule;

this.scheduleMap = createScheduleMap(schedule);

}

public void createQueryCounter(SearchStatsReceiver statsReceiver) {

queries = statsReceiver.getCounter("cached\_filter\_" + filterName + "\_queries");

}

public void incrementUsageStat() {

queries.increment();

}

public String getFilterName() {

return filterName;

}

public String getQueryString() {

return query;

}

// snakeyaml does not like a getter named getResultType() that does not return a string

public ResultSetType getResultSetType() {

return resultType;

}

public boolean getCacheModeOnly() {

return cacheModeOnly;

}

public Query getLuceneQuery() {

return queryPair.getSecond();

}

public ThriftSearchQuery getSearchQuery() {

return queryPair.getFirst();

}

/\*\*

\* Create a new {@link SearchRequestInfo} using {@link #queryPair}.

\*

\* @return a new {@link SearchRequestInfo}

\*/

public SearchRequestInfo createSearchRequestInfo() {

ThriftSearchQuery searchQuery = Preconditions.checkNotNull(queryPair.getFirst());

Query luceneQuery = Preconditions.checkNotNull(queryPair.getSecond());

return new SearchRequestInfo(

searchQuery, luceneQuery, new TerminationTracker(Clock.SYSTEM\_CLOCK));

}

public void setup(

QueryCacheManager queryCacheManager,

UserTable userTable,

EarlybirdCluster earlybirdCluster) throws QueryParserException {

createQuery(queryCacheManager, userTable, earlybirdCluster);

}

// index corresponds to 'segment' from the config file. this is the index of the

// segment, starting with the current segment (0) and counting backwards in time.

public Amount<Long, Time> getUpdateInterval(int index) {

long seconds = scheduleMap.floorEntry(index).getValue().getSeconds();

return Amount.of(seconds, Time.SECONDS);

}

private TreeMap<Integer, UpdateInterval> createScheduleMap(List<UpdateInterval> scheduleToUse) {

TreeMap<Integer, UpdateInterval> map = new TreeMap<>();

for (UpdateInterval interval : scheduleToUse) {

map.put(interval.segment, interval);

}

return map;

}

private void createQuery(

QueryCacheManager queryCacheManager,

UserTable userTable,

EarlybirdCluster earlybirdCluster) throws QueryParserException {

int maxSegmentSize = EarlybirdConfig.getMaxSegmentSize();

CollectorParams collectionParams = new CollectorParams();

collectionParams.setNumResultsToReturn(maxSegmentSize);

CollectorTerminationParams terminationParams = new CollectorTerminationParams();

terminationParams.setMaxHitsToProcess(maxSegmentSize);

collectionParams.setTerminationParams(terminationParams);

ThriftSearchQuery searchQuery = new ThriftSearchQuery();

searchQuery.setMaxHitsPerUser(maxSegmentSize);

searchQuery.setCollectorParams(collectionParams);

searchQuery.setSerializedQuery(query);

final SerializedQueryParser parser = new SerializedQueryParser(

EarlybirdConfig.getPenguinVersion());

Query luceneQuery = parser.parse(query).simplify().accept(

new EarlybirdLuceneQueryVisitor(

queryCacheManager.getIndexConfig().getSchema().getSchemaSnapshot(),

queryCacheManager,

userTable,

queryCacheManager.getUserScrubGeoMap(),

earlybirdCluster,

queryCacheManager.getDecider()));

if (luceneQuery == null) {

throw new QueryParserException("Unable to create lucene query from " + query);

}

queryPair = new Pair<>(searchQuery, luceneQuery);

}

private void sanityCheckFilterName(String filter) throws InvalidEntryException {

if (filter == null || filter.isEmpty()) {

throw new InvalidEntryException("Missing filter name");

}

if (Regex.FILTER\_NAME\_CHECK.matcher(filter).find()) {

throw new InvalidEntryException(

"Invalid character in filter name. Chars allowed [a-zA-Z\_0-9]");

}

}

private void sanityCheckSchedule(List<UpdateInterval> intervals)

throws InvalidEntryException {

// Make sure there's at least 1 interval defined

if (intervals == null || intervals.isEmpty()) {

throw new InvalidEntryException("No schedule defined");

}

// Make sure the first interval starts with segment 0

if (intervals.get(0).getSegment() != 0) {

throw new InvalidEntryException(

"The first interval in the schedule must start from segment 0");

}

// Make sure segments are defined in order, and no segment is defined more than twice

int prevSegment = intervals.get(0).getSegment();

for (int i = 1; i < intervals.size(); ++i) {

int currentSegment = intervals.get(i).getSegment();

if (prevSegment > currentSegment) {

throw new InvalidEntryException("Segment intervals out of order. Segment " + prevSegment

+ " is defined before segment " + currentSegment);

}

if (prevSegment == intervals.get(i).getSegment()) {

throw new InvalidEntryException("Segment " + prevSegment + " is defined twice");

}

prevSegment = currentSegment;

}

}

protected void sanityCheck() throws InvalidEntryException {

sanityCheckFilterName(filterName);

if (query == null || query.isEmpty()) {

throw new InvalidEntryException("Missing query");

}

if (resultType == null) {

throw new InvalidEntryException("Missing result type");

}

if (schedule == null || schedule.size() == 0) {

throw new InvalidEntryException("Missing update schedule");

}

if (scheduleMap == null || scheduleMap.size() == 0) {

throw new InvalidEntryException("Missing update schedule map");

}

}

@Override

public String toString() {

return "filterName: [" + getFilterName()

+ "] query: [" + getQueryString()

+ "] result type [" + getResultSetType()

+ "] schedule: " + schedule;

}

}