package com.twitter.search.earlybird.querycache;

import java.util.Collection;

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.base.Stopwatch;

import com.google.common.collect.Lists;

import com.google.common.primitives.Longs;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.quantity.Amount;

import com.twitter.common.quantity.Time;

import com.twitter.common.util.Clock;

import com.twitter.decider.Decider;

import com.twitter.search.common.concurrent.ScheduledExecutorServiceFactory;

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

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

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

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

import com.twitter.search.earlybird.EarlybirdIndexConfig;

import com.twitter.search.earlybird.EarlybirdStatus;

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

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

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

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

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

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

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

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

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

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

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

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

/\*\*

\* Main class to manage Earlybird's QueryCache.

\*

\* Initialize the QueryCache and new segments are notified to the QueryCache subsystem

\* through this class.

\*

\* This class is thread-safe when calling methods that modify the list of tasks that

\* we're executing or when we need to traverse all tasks and check something. The way

\* thread-safety is achieved here right now is through making methods synchronized.

\*/

public class QueryCacheManager implements SegmentUpdateListener {

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

private static final Amount<Long, Time> ZERO\_SECONDS = Amount.of(0L, Time.SECONDS);

private final boolean enabled = EarlybirdConfig.getBool("querycache", false);

// segments are removed from SegmentInfoMap lazily, and there may be a wait time.

// So, beware that there's short period of time where there's more segments than

// maxEnabledSegments.

private final int maxEnabledSegments;

private final UserTable userTable;

private final UserScrubGeoMap userScrubGeoMap;

private final EarlybirdIndexConfig indexConfig;

private QueryCacheUpdater updater;

private final Map<String, QueryCacheFilter> filters;

private final ScheduledExecutorServiceFactory updaterScheduledExecutorServiceFactory;

private final SearchStatsReceiver searchStatsReceiver;

private static final SearchLongGauge NUM\_CACHE\_ENTRY\_STAT =

SearchLongGauge.export("querycache\_num\_entries");

private static final SearchCounter NUM\_UPDATE\_SEGMENTS\_CALLS =

SearchCounter.export("querycache\_num\_update\_segments\_calls");

private volatile boolean didSetup = false;

private final EarlybirdSearcherStats searcherStats;

private final Decider decider;

private final CriticalExceptionHandler criticalExceptionHandler;

private final Clock clock;

public QueryCacheManager(

QueryCacheConfig config,

EarlybirdIndexConfig indexConfig,

int maxEnabledSegments,

UserTable userTable,

UserScrubGeoMap userScrubGeoMap,

ScheduledExecutorServiceFactory updaterScheduledExecutorServiceFactory,

SearchStatsReceiver searchStatsReceiver,

EarlybirdSearcherStats searcherStats,

Decider decider,

CriticalExceptionHandler criticalExceptionHandler,

Clock clock) {

Preconditions.checkArgument(maxEnabledSegments > 0);

QueryCacheConfig queryCacheConfig = config;

if (queryCacheConfig == null) {

queryCacheConfig = new QueryCacheConfig(searchStatsReceiver);

}

this.indexConfig = indexConfig;

this.maxEnabledSegments = maxEnabledSegments;

this.userTable = userTable;

this.userScrubGeoMap = userScrubGeoMap;

this.updaterScheduledExecutorServiceFactory = updaterScheduledExecutorServiceFactory;

this.searchStatsReceiver = searchStatsReceiver;

this.searcherStats = searcherStats;

this.filters = new HashMap<>();

this.decider = decider;

this.criticalExceptionHandler = criticalExceptionHandler;

this.clock = clock;

for (QueryCacheFilter filter : queryCacheConfig.filters()) {

filters.put(filter.getFilterName(), filter);

}

NUM\_CACHE\_ENTRY\_STAT.set(filters.size());

}

public EarlybirdIndexConfig getIndexConfig() {

return indexConfig;

}

public UserScrubGeoMap getUserScrubGeoMap() {

return userScrubGeoMap;

}

/\*\* Setup all update tasks at once, should only be called after Earlybird has loaded/indexed all

\* segments during start-up

\*

\* Only the first call to the function has effect, subsequent calls are no-ops

\*/

public void setupTasksIfNeeded(SegmentManager segmentManager)

throws QueryParserException {

setupTasks(

segmentManager.getSegmentInfos(Filter.All, Order.OLD\_TO\_NEW),

segmentManager.getEarlybirdIndexConfig().getCluster());

}

@VisibleForTesting

synchronized void setupTasks(

Iterable<SegmentInfo> newSegments,

EarlybirdCluster earlybirdCluster) throws QueryParserException {

// Setup needs to be done only once after all index caught up.

if (didSetup) {

return;

}

LOG.info("Setting up {} query cache tasks", filters.values().size());

for (QueryCacheFilter filter : filters.values()) {

filter.setup(this, userTable, earlybirdCluster);

}

if (!enabled()) {

// Note that the definition of disabling the query caches here is "don't compute the caches".

// We still load the queries from the .yml, we still rewrite search queries to use

// cached queries. The reason we are choosing this definition is that it's somewhat simpler

// to implement (no need to turn off rewriting) and because we might get external queries that

// contain cached filters (they're listed in go/searchsyntax).

//

// If we need a stricter definition of turning off query caches, we can implement it too, or

// just tighten this one.

return;

}

Preconditions.checkState(updater == null);

updater = new QueryCacheUpdater(

filters.values(),

updaterScheduledExecutorServiceFactory,

userTable,

searchStatsReceiver,

searcherStats,

decider,

criticalExceptionHandler,

clock);

LOG.info("Finished setting up query cache updater.");

scheduleTasks(newSegments, false);

didSetup = true;

}

private void scheduleTasks(Iterable<SegmentInfo> segments, boolean isCurrent) {

List<SegmentInfo> sortedSegments = Lists.newArrayList(segments);

Collections.sort(sortedSegments, (o1, o2) -> {

// sort new to old (o2 and o1 are reversed here)

return Longs.compare(o2.getTimeSliceID(), o1.getTimeSliceID());

});

LOG.info("Scheduling tasks for {} segments.", sortedSegments.size());

for (int segmentIndex = 0; segmentIndex < sortedSegments.size(); ++segmentIndex) {

SegmentInfo segmentInfo = sortedSegments.get(segmentIndex);

if (segmentIndex == maxEnabledSegments) {

LOG.warn("Tried to add more segments than MaxEnabledSegments (" + maxEnabledSegments

+ "). Removed oldest segment " + segmentInfo.getTimeSliceID());

continue;

}

addQueryCacheTasksForSegment(segmentInfo, segmentIndex, !isCurrent);

}

}

/\*\*

\* Rebuilds the query cache for the given segment after it was optimized.

\*/

public synchronized void rebuildQueryCachesAfterSegmentOptimization(

SegmentInfo optimizedSegment) {

Preconditions.checkState(optimizedSegment.getIndexSegment().isOptimized(),

"Segment " + optimizedSegment.getSegmentName() + " is not optimized.");

if (!didSetup) {

// Once our indexing is current, we'll just start tasks for all segments, optimized or not.

// Before that event, we don't do anything query cache related.

LOG.info("Haven't done initial setup, returning.");

return;

}

LOG.info("Rebuilding query caches for optimized segment {}",

optimizedSegment.getSegmentName());

// The optimized segment should always be the 1st segment (the current segment has index 0).

Stopwatch stopwatch = Stopwatch.createStarted();

updater.removeAllTasksForSegment(optimizedSegment);

addQueryCacheTasksForSegment(optimizedSegment, 1, true);

while (!updater.allTasksRanForSegment(optimizedSegment)) {

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

// Ignore

}

}

LOG.info("Rebuilding all query caches for the optimized segment {} took {}.",

optimizedSegment.getSegmentName(), stopwatch);

}

/\*\*

\* Block until all the tasks inside this manager have ran at least once.

\*/

public void waitUntilAllQueryCachesAreBuilt() {

LOG.info("Waiting until all query caches are built...");

Stopwatch stopwatch = Stopwatch.createStarted();

while (!allTasksRan()) {

try {

Thread.sleep(1000);

} catch (InterruptedException ex) {

Thread.currentThread().interrupt();

}

}

LOG.info("Ran query cache tasks in: {}", stopwatch);

}

private void addQueryCacheTasksForSegment(

SegmentInfo segmentInfo, int segmentIndex, boolean scheduleImmediately) {

LOG.info("Adding query cache tasks for segment {}.", segmentInfo.getTimeSliceID());

double updateIntervalMultiplier =

EarlybirdConfig.getDouble("query\_cache\_update\_interval\_multiplier", 1.0);

for (QueryCacheFilter filter : filters.values()) {

Amount<Long, Time> updateIntervalFromConfig = filter.getUpdateInterval(segmentIndex);

Amount<Long, Time> updateInterval = Amount.of(

(long) (updateIntervalFromConfig.getValue() \* updateIntervalMultiplier),

updateIntervalFromConfig.getUnit());

Amount<Long, Time> initialDelay = scheduleImmediately ? ZERO\_SECONDS : updateInterval;

updater.addTask(filter, segmentInfo, updateInterval, initialDelay);

}

}

/\*\*

\* Notify QueryCacheManager of a new list of segments we currently have, so that cache tasks

\* can be updated.

\*

\* @param segments fresh list of all segments

\*

\* All existing tasks will be canceled/removed/destroyed, new tasks will be created for all

\* segments.

\*/

@Override

public synchronized void update(Collection<SegmentInfo> segments, String message) {

if (!enabled()) {

return;

}

// This manager is created right at the beginning of a startup. Before we set it up,

// we'll read tweets and create segments and therefore this method will be called.

// We don't want to start computing query caches during that time, so we just return.

if (!didSetup) {

return;

}

NUM\_UPDATE\_SEGMENTS\_CALLS.increment();

LOG.info("Rescheduling all query cache tasks ({}). Number of segments received = {}.",

message, segments.size());

updater.clearTasks(); // cancel and remove all scheduled tasks

// If Earlybird is still starting up, and we get a partition roll, don't delay rebuilding

// the query cache.

boolean isCurrent = EarlybirdStatus.getStatusCode() == EarlybirdStatusCode.CURRENT;

scheduleTasks(segments, isCurrent);

}

/\*\*

\* Determines if all query cache tasks ran at least once (even if they failed).

\*/

public synchronized boolean allTasksRan() {

return (!(enabled() && didSetup)) || updater.allTasksRan();

}

/\*\*

\* Determines if the query cache manager is enabled.

\*/

public boolean enabled() {

return enabled;

}

/\*\*

\* Returns the query cache filter with the given name.

\*/

public QueryCacheFilter getFilter(String filterName) {

return filters.get(filterName);

}

/\*\*

\* Shuts down the query cache manager.

\*/

public synchronized void shutdown() throws InterruptedException {

LOG.info("Shutting down QueryCacheManager");

if (updater != null) {

updater.shutdown();

updater = null;

}

didSetup = false; // needed for unit test

}

/\*\*

\* After startup, we want only one thread to update the query cache.

\*/

public void setWorkerPoolSizeAfterStartup() {

if (this.updater != null) {

this.updater.setWorkerPoolSizeAfterStartup();

}

}

public Decider getDecider() {

return this.decider;

}

//////////////////////////

// for unit tests only

//////////////////////////

QueryCacheUpdater getUpdaterForTest() {

return updater;

}

Map<String, QueryCacheFilter> getCacheMapForTest() {

return filters;

}

}