package com.twitter.search.earlybird.partition;

import java.util.concurrent.TimeUnit;

import com.google.common.annotations.VisibleForTesting;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import com.twitter.search.common.config.Config;

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

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

import com.twitter.search.earlybird.EarlybirdStatus;

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

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

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

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

import com.twitter.search.earlybird.segment.SegmentDataProvider;

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

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

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

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

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

/\*\*

\* PartitionManager is responsible for indexing data for a partition, including Tweets and Users.

\*/

public abstract class PartitionManager extends OneTaskScheduledExecutorManager {

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

private static final SearchCounter IGNORED\_EXCEPTIONS =

SearchCounter.export("partition\_manager\_ignored\_exceptions");

private static final String PARTITION\_MANAGER\_THREAD\_NAME = "PartitionManager";

private static final boolean THREAD\_IS\_DAEMON = true;

protected static final String INDEX\_CURRENT\_SEGMENT = "indexing the current segment";

protected static final String SETUP\_QUERY\_CACHE = "setting up query cache";

protected final SegmentManager segmentManager;

protected final QueryCacheManager queryCacheManager;

// Should be updated by info read from ZK

protected final DynamicPartitionConfig dynamicPartitionConfig;

private final SearchIndexingMetricSet searchIndexingMetricSet;

private boolean partitionManagerFirstLoop = true;

public PartitionManager(QueryCacheManager queryCacheManager,

SegmentManager segmentManager,

DynamicPartitionConfig dynamicPartitionConfig,

ScheduledExecutorServiceFactory executorServiceFactory,

SearchIndexingMetricSet searchIndexingMetricSet,

SearchStatsReceiver searchStatsReceiver,

CriticalExceptionHandler criticalExceptionHandler) {

super(

executorServiceFactory,

PARTITION\_MANAGER\_THREAD\_NAME,

THREAD\_IS\_DAEMON,

PeriodicActionParams.withFixedDelay(

EarlybirdConfig.getInt("time\_slice\_roll\_check\_interval\_ms", 500),

TimeUnit.MILLISECONDS),

ShutdownWaitTimeParams.indefinitely(),

searchStatsReceiver,

criticalExceptionHandler);

this.segmentManager = segmentManager;

this.queryCacheManager = queryCacheManager;

this.dynamicPartitionConfig = dynamicPartitionConfig;

this.searchIndexingMetricSet = searchIndexingMetricSet;

}

/\*\*

\* Runs the partition manager.

\*/

public final void runImpl() {

if (partitionManagerFirstLoop) {

try {

testHookBeforeStartUp();

startUp();

validateSegments();

segmentManager.logState("After startUp");

} catch (Throwable t) {

criticalExceptionHandler.handle(this, t);

shutDownIndexing();

throw new RuntimeException("PartitionManager unhandled exception, stopping scheduler", t);

}

}

try {

testHookAfterSleep();

indexingLoop(partitionManagerFirstLoop);

} catch (InterruptedException e) {

LOG.warn("PartitionManager thread interrupted, stoping scheduler", e);

shutDownIndexing();

throw new RuntimeException("PartitionManager thread interrupted", e);

} catch (Exception e) {

LOG.error("Exception in indexing PartitionManager loop", e);

IGNORED\_EXCEPTIONS.increment();

} catch (Throwable t) {

LOG.error("Unhandled exception in indexing PartitionManager loop", t);

criticalExceptionHandler.handle(this, t);

shutDownIndexing();

throw new RuntimeException("PartitionManager unhandled exception, stopping scheduler", t);

} finally {

partitionManagerFirstLoop = false;

}

}

/\*\*

\* Returns the SegmentDataProvider instance that will be used to fetch the information for all

\* segments.

\*/

public abstract SegmentDataProvider getSegmentDataProvider();

/\*\*

\* Starts up this partition manager.

\*/

protected abstract void startUp() throws Exception;

/\*\*

\* Runs one indexing iteration.

\*

\* @param firstLoop Determines if this is the first time the indexing loop is running.

\*/

protected abstract void indexingLoop(boolean firstLoop) throws Exception;

/\*\*

\* Shuts down all indexing.

\*/

protected abstract void shutDownIndexing();

@Override

public void shutdownComponent() {

shutDownIndexing();

}

/\*\*

\* Notifies all other threads that the partition manager has become current (ie. has indexed all

\* available events).

\*/

public void becomeCurrent() {

LOG.info("PartitionManager became current");

if (EarlybirdStatus.isStarting()) {

EarlybirdStatus.setStatus(EarlybirdStatusCode.CURRENT);

} else {

LOG.warn("Could not set statusCode to CURRENT from " + EarlybirdStatus.getStatusCode());

}

// Now that we're done starting up, set the query cache thread pool size to one.

queryCacheManager.setWorkerPoolSizeAfterStartup();

}

protected void setupQueryCacheIfNeeded() throws QueryParserException {

queryCacheManager.setupTasksIfNeeded(segmentManager);

}

// Only for tests, used for testing exception handling

private static TestHook testHookBeforeStartUp;

private static TestHook testHookAfterSleep;

private static void testHookBeforeStartUp() throws Exception {

if (Config.environmentIsTest() && testHookBeforeStartUp != null) {

testHookBeforeStartUp.run();

}

}

private static void testHookAfterSleep() throws Exception {

if (Config.environmentIsTest() && testHookAfterSleep != null) {

testHookAfterSleep.run();

}

}

@Override

protected void runOneIteration() {

try {

runImpl();

} catch (Throwable t) {

LOG.error("Unhandled exception in PartitionManager loop", t);

throw new RuntimeException(t.getMessage());

}

}

public SearchIndexingMetricSet getSearchIndexingMetricSet() {

return searchIndexingMetricSet;

}

/\*\*

\* Allows tests to run code before the partition manager starts up.

\*

\* @param testHook The code to run before the start up.

\*/

@VisibleForTesting

public static void setTestHookBeforeStartUp(TestHook testHook) {

if (Config.environmentIsTest()) {

testHookBeforeStartUp = testHook;

} else {

throw new RuntimeException("Trying to set startup test hook in non-test code!!");

}

}

/\*\*

\* Allows tests to run code before the indexing loop.

\*

\* @param testHook The code to run before the indexing loop.

\*/

@VisibleForTesting

public static void setTestHookAfterSleep(TestHook testHook) {

if (Config.environmentIsTest()) {

testHookAfterSleep = testHook;

} else {

throw new RuntimeException("Trying to set test hook in non-test code!!");

}

}

/\*\*

\* An interface that allows tests to run code at various points in the PartitionManager's

\* lyfecycle.

\*/

@VisibleForTesting

public interface TestHook {

/\*\*

\* Defines the code that should be run.

\*/

void run() throws Exception;

}

/\*\*

\* Allows tests to determine if this partition manager is all caught up.

\*

\* @return {@code true} if this partition manager is caught up, {@code false} otherwise.

\*/

@VisibleForTesting

public abstract boolean isCaughtUpForTests();

@VisibleForTesting

protected void validateSegments() throws EarlybirdStartupException {

// This is necessary because many tests rely on starting partition manager but not indexing any

// tweets. However, we do not want Earlybirds to start in production if they are not serving any

// tweets. (SEARCH-24238)

if (Config.environmentIsTest()) {

return;

}

validateSegmentsForNonTest();

}

@VisibleForTesting

protected void validateSegmentsForNonTest() throws EarlybirdStartupException {

// Subclasses can override this and provide additional checks.

if (segmentManager.getNumIndexedDocuments() == 0) {

throw new EarlybirdStartupException("Earlybird has zero indexed documents.");

}

}

}