package com.twitter.search.earlybird.util;

import java.util.Optional;

import java.util.Random;

import java.util.concurrent.atomic.AtomicBoolean;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.base.Stopwatch;

import org.apache.zookeeper.KeeperException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.base.ExceptionalFunction;

import com.twitter.common.quantity.Amount;

import com.twitter.common.quantity.Time;

import com.twitter.common.zookeeper.ServerSet;

import com.twitter.common.zookeeper.ZooKeeperClient;

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

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

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

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

import com.twitter.search.common.util.zktrylock.TryLock;

import com.twitter.search.common.util.zktrylock.ZooKeeperTryLockFactory;

import com.twitter.search.earlybird.ServerSetMember;

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

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

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

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

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

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

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

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

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

/\*\*

\* Utility class for executing tasks on Earlybirds that need to be coordinated across replicas

\* on the same hash partition.

\* Can be used for things like coordinating optimization on the same timeslice.

\* When enabled, a try-lock will be taken out in zookeeper while the task is performed.

\* The action will attempt to leave the partition's server set. If the attempt fails, the action

\* is aborted.

\*/

public class CoordinatedEarlybirdAction implements CoordinatedEarlybirdActionInterface {

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

private static final Boolean COORDINATED\_ACTION\_FLAG = Boolean.TRUE;

private static final Boolean NOT\_COORDINATED\_ACTION\_FLAG = Boolean.FALSE;

private final String actionName;

private final DynamicPartitionConfig dynamicPartitionConfig;

@Nullable private final ServerSetMember serverSetMember;

private final ZooKeeperTryLockFactory zooKeeperTryLockFactory;

// Whether this action should be coordinated through zookeeper in the first place (could be

// config'ed off).

// If the action is coordinated, this earlybird will leave its server set when performing the

// coordinated action.

private final AtomicBoolean shouldSynchronize;

// Whether this action should ensure that there are enough replicas in the serverset (defined by

// maxAllowedReplicasNotInServerSet) before leaving the serverset.

private final boolean checkNumReplicasInServerSet;

// If this many (or more) servers have left the partition, we cannot perform a coordinated action

private final int maxAllowedReplicasNotInServerSet;

// How long to lock out all other replicas in this hash partition for.

// Should be some small multiple of how long the action is expected to take, to allow for longer

// running cases.

private final long zkLockExpirationTimeMinutes;

// Prefix for the zookeeper lock used when coordinating daily updates.

// Full name should include the hash partition number.

private final String zkLockNamePrefix;

// If we're unable to re-join this earlybird's server set during coordinated updates,

// how many times to retry.

private final int joinServerSetRetries;

// How long to sleep between retries if unable to job back into server set.

private final int joinServerSetRetrySleepMillis;

// How long to sleep between leaving the serverset and executing the action

private final int sleepAfterLeaveServerSetMillis;

// How many times a this action was called within a lock block.

private final SearchCounter numCoordinatedFunctionCalls;

private final SearchCounter numCoordinatedLeaveServersetCalls;

private final CriticalExceptionHandler criticalExceptionHandler;

private final SegmentSyncConfig segmentSyncConfig;

/\*\*

\* Create a CoordinatedEarlybirdAction.

\*

\* @param actionName the name to be used for logging and the prefix for config options.

\* @param dynamicPartitionConfig maintains the current partitioning configuration for this

\* earlybird. Used mainly to determine the hash partition of this earlybird.

\* @param serverSetMember the server that this action is running on. To be used to leaving and

\* rejoining the server's server set.

\*/

public CoordinatedEarlybirdAction(

ZooKeeperTryLockFactory zooKeeperTryLockFactory,

String actionName,

DynamicPartitionConfig dynamicPartitionConfig,

@Nullable ServerSetMember serverSetMember,

CriticalExceptionHandler criticalExceptionHandler,

SegmentSyncConfig segmentSyncConfig) {

this.actionName = actionName;

this.dynamicPartitionConfig = dynamicPartitionConfig;

this.serverSetMember = serverSetMember;

this.criticalExceptionHandler = criticalExceptionHandler;

this.segmentSyncConfig = segmentSyncConfig;

this.zooKeeperTryLockFactory = zooKeeperTryLockFactory;

if (serverSetMember == null) {

Preconditions.checkState(Config.environmentIsTest(),

"Should only have a null server in tests");

}

this.shouldSynchronize = new AtomicBoolean(

EarlybirdConfig.getBool(actionName + "\_should\_synchronize", false));

// Export whether or not synchronization is enabled as a stat

SearchCustomGauge.export(

actionName + "\_should\_synchronize", () -> shouldSynchronize.get() ? 1 : 0);

this.checkNumReplicasInServerSet = EarlybirdProperty.CHECK\_NUM\_REPLICAS\_IN\_SERVER\_SET.get();

int numReplicas =

dynamicPartitionConfig.getCurrentPartitionConfig().getNumReplicasInHashPartition();

this.maxAllowedReplicasNotInServerSet =

EarlybirdProperty.MAX\_ALLOWED\_REPLICAS\_NOT\_IN\_SERVER\_SET.get(numReplicas);

this.zkLockExpirationTimeMinutes =

EarlybirdConfig.getLong(actionName + "\_lock\_expiration\_time\_minutes", 60L);

this.zkLockNamePrefix = actionName + "\_for\_hash\_partition\_";

this.joinServerSetRetries =

EarlybirdConfig.getInt(actionName + "\_join\_server\_set\_retries", 20);

this.joinServerSetRetrySleepMillis =

EarlybirdConfig.getInt(actionName + "\_join\_server\_retry\_sleep\_millis", 2000);

this.sleepAfterLeaveServerSetMillis =

EarlybirdConfig.getInt("coordinated\_action\_sleep\_after\_leave\_server\_set\_millis", 30000);

this.numCoordinatedFunctionCalls = SearchCounter.export(actionName + "\_num\_coordinated\_calls");

this.numCoordinatedLeaveServersetCalls =

SearchCounter.export(actionName + "\_num\_coordinated\_leave\_serverset\_calls");

if (this.checkNumReplicasInServerSet) {

LOG.info(

"Coordinate action config ({}): allowedNotIn: {}, current number of replicas: {}, "

+ "synchronization enabled: {}, checkNumReplicasInServerSet enabled: {}",

actionName,

maxAllowedReplicasNotInServerSet,

dynamicPartitionConfig.getCurrentPartitionConfig().getNumReplicasInHashPartition(),

shouldSynchronize,

this.checkNumReplicasInServerSet);

} else {

LOG.info(

"Coordinate action config ({}): synchronization enabled: {}, "

+ "checkNumReplicasInServerSet enabled: {}",

actionName,

shouldSynchronize,

this.checkNumReplicasInServerSet);

}

}

@Override

public <E extends Exception> boolean execute(

String description,

ExceptionalFunction<Boolean, Boolean, E> function)

throws E, CoordinatedEarlybirdActionLockFailed {

if (this.shouldSynchronize.get()) {

return executeWithCoordination(description, function);

} else {

return function.apply(NOT\_COORDINATED\_ACTION\_FLAG);

}

}

enum LeaveServerSetResult {

SUCCESS,

FAILURE,

NOT\_IN\_SERVER\_SET,

NO\_SERVER\_SET\_MEMBER

}

private LeaveServerSetResult leaveServerSet() {

LOG.info("Leaving serving server set for " + actionName);

try {

serverSetMember.leaveServerSet("CoordinatedAction: " + actionName);

return LeaveServerSetResult.SUCCESS;

} catch (ServerSet.UpdateException ex) {

if (ex instanceof NotInServerSetUpdateException) {

LOG.info("No need to leave; already out of server set during: "

+ actionName, ex);

return LeaveServerSetResult.NOT\_IN\_SERVER\_SET;

} else {

LOG.warn("Unable to leave server set during: " + actionName, ex);

return LeaveServerSetResult.FAILURE;

}

}

}

private LeaveServerSetResult maybeLeaveServerSet() {

if (serverSetMember != null) {

if (serverSetMember.isInServerSet()) {

if (!checkNumReplicasInServerSet) {

return leaveServerSet();

} else {

PartitionConfig curPartitionConfig = dynamicPartitionConfig.getCurrentPartitionConfig();

final int minNumServers =

curPartitionConfig.getNumReplicasInHashPartition() - maxAllowedReplicasNotInServerSet;

Optional<Integer> numServerSetMembers = getNumberOfServerSetMembers();

LOG.info("Checking number of replicas before leaving server set for " + actionName

+ ". Number of members is: " + numServerSetMembers + " minMembers: " + minNumServers);

if (numServerSetMembers.isPresent() && numServerSetMembers.get() > minNumServers) {

return leaveServerSet();

} else {

LOG.warn("Not leaving server set during: " + actionName);

return LeaveServerSetResult.FAILURE;

}

}

} else {

LOG.info("Not in server set, no need to leave it.");

return LeaveServerSetResult.NOT\_IN\_SERVER\_SET;

}

}

return LeaveServerSetResult.NO\_SERVER\_SET\_MEMBER;

}

private <E extends Exception> boolean executeWithCoordination(

final String description,

final ExceptionalFunction<Boolean, Boolean, E> function)

throws E, CoordinatedEarlybirdActionLockFailed {

PartitionConfig curPartitionConfig = dynamicPartitionConfig.getCurrentPartitionConfig();

TryLock lock = zooKeeperTryLockFactory.createTryLock(

DatabaseConfig.getLocalHostname(),

segmentSyncConfig.getZooKeeperSyncFullPath(),

zkLockNamePrefix

+ curPartitionConfig.getIndexingHashPartitionID(),

Amount.of(zkLockExpirationTimeMinutes, Time.MINUTES)

);

final AtomicBoolean success = new AtomicBoolean(false);

boolean gotLock = lock.tryWithLock(() -> {

Stopwatch actionTiming = Stopwatch.createStarted();

LeaveServerSetResult leftServerSet = maybeLeaveServerSet();

if (leftServerSet == LeaveServerSetResult.FAILURE) {

LOG.info("Failed to leave the server set, will not execute action.");

return;

}

LOG.info("maybeLeaveServerSet returned: {}", leftServerSet);

// Sleep for a short time to give the server some time to finish requests that it is currently

// executing and allow roots some time to register that this host has left the server set.

// If we didn't do this and the coordinated action included a full GC, then latency and error

// rate at the root layer would spike higher at the time of the GC. SEARCH-35456

try {

Thread.sleep(sleepAfterLeaveServerSetMillis);

} catch (InterruptedException ex) {

Thread.currentThread().interrupt();

}

LOG.info(actionName + " synchronization action for " + description);

try {

numCoordinatedFunctionCalls.increment();

numCoordinatedLeaveServersetCalls.increment();

Boolean successValue = function.apply(COORDINATED\_ACTION\_FLAG);

success.set(successValue);

} finally {

if (leftServerSet == LeaveServerSetResult.SUCCESS) {

joinServerSet();

}

LOG.info("{} synchronization action for {} completed after {}, success: {}",

actionName,

description,

actionTiming,

success.get());

}

});

if (!gotLock) {

String errorMsg = actionName + ": Failed to get zk indexing lock for " + description;

LOG.info(errorMsg);

throw new CoordinatedEarlybirdActionLockFailed(errorMsg);

}

return success.get();

}

@Override

public void retryActionUntilRan(String description, Runnable action) {

Random random = new Random(System.currentTimeMillis());

boolean actionExecuted = false;

int attempts = 0;

while (!actionExecuted) {

try {

attempts++;

actionExecuted = this.execute(description, isCoordinated -> {

action.run();

return true;

});

} catch (CoordinatedEarlybirdActionLockFailed ex) {

}

if (!actionExecuted) {

// Variable sleep amount. The reason for the random sleeps

// is so that across multiple earlybirds this doesn't get

// executed in some sequence that depends on something else

// like maybe deploy times. It might be easier to catch possible

// problems if implicit orderings like this are not introduced.

long msToSleep = (10 + random.nextInt(5)) \* 1000L;

try {

Thread.sleep(msToSleep);

} catch (InterruptedException ex) {

LOG.info("Interrupted while trying to execute");

Thread.currentThread().interrupt();

}

} else {

LOG.info("Executed {} after {} attempts", actionName, attempts);

}

}

}

/\*\*

\* Gets the current number of servers in this server's server set.

\* @return absent Optional if we encountered an exception getting the number of hosts.

\*/

private Optional<Integer> getNumberOfServerSetMembers() {

try {

return serverSetMember != null ? Optional.of(serverSetMember.getNumberOfServerSetMembers())

: Optional.empty();

} catch (InterruptedException ex) {

LOG.warn("Action " + actionName + " was interrupted.", ex);

Thread.currentThread().interrupt();

return Optional.empty();

} catch (ZooKeeperClient.ZooKeeperConnectionException | KeeperException ex) {

LOG.warn("Exception during " + actionName, ex);

return Optional.empty();

}

}

/\*\*

\* After a coordinated action, join back this earlybird's server set with retries

\* and sleeps in between.

\*/

private void joinServerSet() {

Preconditions.checkNotNull(serverSetMember);

boolean joined = false;

for (int i = 0; i < joinServerSetRetries; i++) {

try {

serverSetMember.joinServerSet("CoordinatedAction: " + actionName);

joined = true;

break;

} catch (AlreadyInServerSetUpdateException ex) {

// Most likely leaving the server set failed

joined = true;

break;

} catch (ServerSet.UpdateException ex) {

LOG.warn("Unable to join server set after " + actionName + " on attempt "

+ i, ex);

if (i < (joinServerSetRetries - 1)) {

try {

Thread.sleep(joinServerSetRetrySleepMillis);

} catch (InterruptedException e) {

LOG.warn("Interrupted while waiting to join back server set for: " + actionName);

// Preserve interrupt status.

Thread.currentThread().interrupt();

break;

}

}

}

}

if (!joined) {

String message = String.format(

"Unable to join server set after %s, setting fatal flag.",

actionName);

EarlybirdException exception = new EarlybirdException(message);

LOG.error(message, exception);

criticalExceptionHandler.handle(this, exception);

}

}

@Override

public boolean setShouldSynchronize(boolean shouldSynchronizeParam) {

boolean oldValue = this.shouldSynchronize.getAndSet(shouldSynchronizeParam);

LOG.info("Updated shouldSynchronize for: " + actionName + " from " + oldValue

+ " to " + shouldSynchronizeParam);

return oldValue;

}

@Override

@VisibleForTesting

public long getNumCoordinatedFunctionCalls() {

return this.numCoordinatedFunctionCalls.get();

}

@Override

@VisibleForTesting

public long getNumCoordinatedLeaveServersetCalls() {

return this.numCoordinatedLeaveServersetCalls.get();

}

}