package com.twitter.search.earlybird.archive.segmentbuilder;

import java.io.IOException;

import java.util.Date;

import java.util.Optional;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

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.search.common.database.DatabaseConfig;

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

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

import com.twitter.search.earlybird.archive.DailyStatusBatches;

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

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

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

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

import com.twitter.util.Duration;

/\*\*

\* Coordinate between segment builders for scrubbing pipeline.

\* When segment builder is running, all of them will try to find a HDFS file indicating if data is

\* ready. If the file does not exist, only one of them will go through the files and see if

\* scrubbing pipeline has generated all data for this scrub gen.

\*

\* If the instance that got the lock found all data, it still exists, because otherwise we will

\* have one single segmentbuilder instance trying to build all segments, which is not what we want.

\* But if it exists, then the next time all segmentbuilder instances are scheduled, they will all

\* find the file, and will start building segments.

\*/

class SegmentBuilderCoordinator {

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

private static final Amount<Long, Time> ZK\_LOCK\_EXPIRATION\_MIN = Amount.of(5L, Time.MINUTES);

private static final String SEGMENT\_BUILDER\_SYNC\_NODE = "scrub\_gen\_data\_sync";

private static final String SEGMENT\_BUILDER\_SYNC\_ZK\_PATH =

EarlybirdProperty.ZK\_APP\_ROOT.get() + "/segment\_builder\_sync";

private static final String DATA\_FULLY\_BUILT\_FILE = "\_data\_fully\_built";

static final int FIRST\_INSTANCE = 0;

private static final long NON\_FIRST\_INSTANCE\_SLEEP\_BEFORE\_RETRY\_DURATION\_MS =

Duration.fromHours(1).inMillis();

private final ZooKeeperTryLockFactory zkTryLockFactory;

private final SegmentSyncConfig syncConfig;

private final Optional<Date> scrubGenDayOpt;

private final Optional<String> scrubGenOpt;

private final Clock clock;

SegmentBuilderCoordinator(

ZooKeeperTryLockFactory zkTryLockFactory, SegmentSyncConfig syncConfig, Clock clock) {

this.zkTryLockFactory = zkTryLockFactory;

this.syncConfig = syncConfig;

this.scrubGenOpt = syncConfig.getScrubGen();

this.scrubGenDayOpt = scrubGenOpt.map(ScrubGenUtil::parseScrubGenToDate);

this.clock = clock;

}

public boolean isScrubGenDataFullyBuilt(int instanceNumber) {

// Only segment builder that takes scrub gen should use isPartitioningOutputReady to coordinate

Preconditions.checkArgument(scrubGenDayOpt.isPresent());

final FileSystem hdfs;

try {

hdfs = HdfsUtil.getHdfsFileSystem();

} catch (IOException e) {

LOG.error("Could not create HDFS file system.", e);

return false;

}

return isScrubGenDataFullyBuilt(

instanceNumber,

scrubGenDayOpt.get(),

NON\_FIRST\_INSTANCE\_SLEEP\_BEFORE\_RETRY\_DURATION\_MS,

hdfs

);

}

@VisibleForTesting

boolean isScrubGenDataFullyBuilt(

int instanceNumber,

Date scrubGenDay,

long nonFirstInstanceSleepBeforeRetryDuration,

FileSystem hdfs) {

// Check if the scrub gen has been fully built file exists.

if (checkHaveScrubGenDataFullyBuiltFileOnHdfs(hdfs)) {

return true;

}

// If it doesn't exist, let first instance see if scrub gen has been fully built and create the

// file.

if (instanceNumber == FIRST\_INSTANCE) {

// We were missing some data on HDFS for this scrub gen in previous run,

// but we might've gotten more data in the meantime, check again.

// Only allow instance 0 to do this mainly for 2 reasons:

// 1) Since instances are scheduled in batches, it's possible that a instance from latter

// batch find the fully built file in hdfs and start processing. We end up doing work with

// only partial instances.

// 2) If we sleep before we release lock, it's hard to estimate how long a instance will

// be scheduled.

// For deterministic reason, we simplify a bit and only allow instance 0 to check and write

// data is fully build file to hdfs.

try {

checkIfScrubGenDataIsFullyBuilt(hdfs, scrubGenDay);

} catch (IOException e) {

LOG.error("Failed to grab lock and check scrub gen data.", e);

}

} else {

// for all other instances, sleep for a bit to give time for first instance to check if scrub

// gen has been fully built and create the file, then check again.

try {

LOG.info(

"Sleeping for {} ms before re-checking if scrub gen has been fully built file exists",

nonFirstInstanceSleepBeforeRetryDuration);

clock.waitFor(nonFirstInstanceSleepBeforeRetryDuration);

return checkHaveScrubGenDataFullyBuiltFileOnHdfs(hdfs);

} catch (InterruptedException e) {

LOG.warn("Interrupted when sleeping before re-checking if scrub gen has been fully built "

+ "file exists", e);

}

}

// if hasSuccessFileToHdfs returns false, then should always return false in the end.

// next run will find success file for this scrub gen and move forward.

return false;

}

private void checkIfScrubGenDataIsFullyBuilt(

FileSystem hdfs, Date scrubGenDay) throws IOException {

// Build the lock, try to acquire it, and check the data on HDFS

TryLock lock = zkTryLockFactory.createTryLock(

DatabaseConfig.getLocalHostname(),

SEGMENT\_BUILDER\_SYNC\_ZK\_PATH,

SEGMENT\_BUILDER\_SYNC\_NODE,

ZK\_LOCK\_EXPIRATION\_MIN);

Preconditions.checkState(scrubGenOpt.isPresent());

String scrubGen = scrubGenOpt.get();

lock.tryWithLock(() -> {

LOG.info(String.format(

"Obtained ZK lock to check if data for scrub gen %s is ready.", scrubGen));

final DailyStatusBatches directory =

new DailyStatusBatches(zkTryLockFactory, scrubGenDay);

if (directory.isScrubGenDataFullyBuilt(hdfs)

&& createScrubGenDataFullyBuiltFileOnHdfs(hdfs)) {

LOG.info(String.format("All data for scrub gen %s is ready.", scrubGen));

} else {

LOG.info(String.format("Data for scrub gen %s is not ready yet.", scrubGen));

}

});

}

private boolean createScrubGenDataFullyBuiltFileOnHdfs(FileSystem fs) {

Path path = getScrubGenDataFullyBuiltFilePath();

try {

fs.mkdirs(new Path(statusReadyHDFSPath()));

if (fs.createNewFile(path)) {

LOG.info("Successfully created file " + path + " on HDFS.");

return true;

} else {

LOG.warn("Failed to create file " + path + " on HDFS.");

}

} catch (IOException e) {

LOG.error("Failed to create file on HDFS " + path.toString(), e);

}

return false;

}

private boolean checkHaveScrubGenDataFullyBuiltFileOnHdfs(FileSystem fs) {

Path path = getScrubGenDataFullyBuiltFilePath();

try {

boolean ret = fs.exists(path);

LOG.info("Checking if file exists showing scrubgen is fully built.");

LOG.info("Path checked: {}, Exist check: {}", path, ret);

return ret;

} catch (IOException e) {

LOG.error("Failed to check file on HDFS " + path.toString(), e);

return false;

}

}

@VisibleForTesting

Path getScrubGenDataFullyBuiltFilePath() {

return new Path(statusReadyHDFSPath(), DATA\_FULLY\_BUILT\_FILE);

}

@VisibleForTesting

String statusReadyHDFSPath() {

return syncConfig.getHdfsSegmentSyncRootDir() + "/segment\_builder\_sync";

}

}