package com.twitter.search.earlybird.archive;

import java.io.IOException;

import java.util.Calendar;

import java.util.Date;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.commons.io.IOUtils;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.partitioning.base.Segment;

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

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

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

public final class ArchiveHDFSUtils {

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

private static final Pattern SEGMENT\_NAME\_PATTERN =

Pattern.compile("\_start\_([0-9]+)\_p\_([0-9]+)\_of\_([0-9]+)\_([0-9]{14}+)\_");

private static final int MATCHER\_GROUP\_END\_DATE = 4;

private ArchiveHDFSUtils() {

}

/\*\*

\* Check if a given segment already has its indices built on hdfs.

\* @return true if the indices exist on hdfs; otherwise, false.

\*/

public static boolean hasSegmentIndicesOnHDFS(SegmentSyncConfig sync, SegmentInfo segment) {

LOG.info("checking segment on hdfs: " + segment

+ " enabled: " + sync.isSegmentLoadFromHdfsEnabled());

FileSystem fs = null;

try {

fs = HdfsUtil.getHdfsFileSystem();

String hdfsBaseDirPrefix = segment.getSyncInfo()

.getHdfsSyncDirPrefix();

FileStatus[] statuses = fs.globStatus(new Path(hdfsBaseDirPrefix));

return statuses != null && statuses.length > 0;

} catch (IOException ex) {

LOG.error("Failed checking segment on hdfs: " + segment, ex);

return false;

} finally {

IOUtils.closeQuietly(fs);

}

}

/\*\*

\* Delete the segment index directories on the HDFS. If 'deleteCurrentDir' is true, the

\* index directory with the end date matching 'segment' will be deleted. If 'deleteOlderDirs',

\* the index directories with the end date earlier than the the segment enddate will be deleted.

\*

\*/

public static void deleteHdfsSegmentDir(SegmentSyncConfig sync, SegmentInfo segment,

boolean deleteCurrentDir, boolean deleteOlderDirs) {

FileSystem fs = null;

try {

fs = HdfsUtil.getHdfsFileSystem();

String hdfsFlushDir = segment.getSyncInfo().getHdfsFlushDir();

String hdfsBaseDirPrefix = segment.getSyncInfo()

.getHdfsSyncDirPrefix();

String endDateStr = extractEndDate(hdfsBaseDirPrefix);

if (endDateStr != null) {

hdfsBaseDirPrefix = hdfsBaseDirPrefix.replace(endDateStr, "\*");

}

String[] hdfsDirs = {segment.getSyncInfo().getHdfsTempFlushDir(),

hdfsBaseDirPrefix};

for (String hdfsDir : hdfsDirs) {

FileStatus[] statuses = fs.globStatus(new Path(hdfsDir));

if (statuses != null && statuses.length > 0) {

for (FileStatus status : statuses) {

if (status.getPath().toString().endsWith(hdfsFlushDir)) {

if (deleteCurrentDir) {

fs.delete(status.getPath(), true);

LOG.info("Deleted segment: " + status.getPath());

}

} else {

if (deleteOlderDirs) {

fs.delete(status.getPath(), true);

LOG.info("Deleted segment: " + status.getPath());

}

}

}

}

}

} catch (IOException e) {

LOG.error("Error delete Segment Dir :" + segment, e);

} finally {

IOUtils.closeQuietly(fs);

}

}

/\*\*

\* Given a segment, check if there is any indices built on HDFS; if yes, return the end date

\* of the index built on HDFS; otherwise, return null.

\*/

public static Date getSegmentEndDateOnHdfs(SegmentSyncConfig sync, SegmentInfo segment) {

if (sync.isSegmentLoadFromHdfsEnabled()) {

LOG.info("About to check segment on hdfs: " + segment

+ " enabled: " + sync.isSegmentLoadFromHdfsEnabled());

FileSystem fs = null;

try {

String hdfsBaseDirPrefix = segment.getSyncInfo()

.getHdfsSyncDirPrefix();

String endDateStr = extractEndDate(hdfsBaseDirPrefix);

if (endDateStr == null) {

return null;

}

hdfsBaseDirPrefix = hdfsBaseDirPrefix.replace(endDateStr, "\*");

fs = HdfsUtil.getHdfsFileSystem();

FileStatus[] statuses = fs.globStatus(new Path(hdfsBaseDirPrefix));

if (statuses != null && statuses.length > 0) {

Path hdfsSyncPath = statuses[statuses.length - 1].getPath();

String hdfsSyncPathName = hdfsSyncPath.getName();

endDateStr = extractEndDate(hdfsSyncPathName);

return Segment.getSegmentEndDate(endDateStr);

}

} catch (Exception ex) {

LOG.error("Failed getting segment from hdfs: " + segment, ex);

return null;

} finally {

IOUtils.closeQuietly(fs);

}

}

return null;

}

private static String extractEndDate(String segmentDirPattern) {

Matcher matcher = SEGMENT\_NAME\_PATTERN.matcher(segmentDirPattern);

if (!matcher.find()) {

return null;

}

try {

return matcher.group(MATCHER\_GROUP\_END\_DATE);

} catch (IllegalStateException e) {

LOG.error("Match operation failed: " + segmentDirPattern, e);

return null;

} catch (IndexOutOfBoundsException e) {

LOG.error(" No group in the pattern with the given index : " + segmentDirPattern, e);

return null;

}

}

/\*\*

\* Converts the given date to a path, using the given separator. For example, if the sate is

\* January 5, 2019, and the separator is "/", this method will return "2019/01/05".

\*/

public static String dateToPath(Date date, String separator) {

StringBuilder builder = new StringBuilder();

Calendar cal = Calendar.getInstance();

cal.setTime(date);

builder.append(cal.get(Calendar.YEAR))

.append(separator)

.append(padding(cal.get(Calendar.MONTH) + 1, 2))

.append(separator)

.append(padding(cal.get(Calendar.DAY\_OF\_MONTH), 2));

return builder.toString();

}

private static String padding(int value, int len) {

return String.format("%0" + len + "d", value);

}

}