package com.twitter.search.earlybird.archive;

import java.io.IOException;

import java.util.Date;

import java.util.Map;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.collect.Maps;

import com.google.gson.Gson;

import com.google.gson.JsonParseException;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

/\*\*

\* Represents a day's worth of statuses (tweets) for multiple hash partitions.

\*

\* Note that what this class contains is not the data, but metadata.

\*

\* A day of tweets will come from:

\* - A scrubgen, if it has happened before the scrubgen date.

\* - Our daily jobs pipeline, if it has happened after that.

\*

\* This class checks the \_SUCCESS file exists in the "statuses" subdirectory and extracts the status

\* count, min status id and max status id.

\*/

public class DailyStatusBatch implements Comparable<DailyStatusBatch> {

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

public static final long EMPTY\_BATCH\_STATUS\_ID = -1;

private static final String PARTITION\_FORMAT = "p\_%d\_of\_%d";

private static final String SUCCESS\_FILE\_NAME = "\_SUCCESS";

private final Map<Integer, PartitionedBatch> hashPartitionToStatuses = Maps.newHashMap();

private final Date date;

private final int numHashPartitions;

private final boolean hasSuccessFiles;

public DailyStatusBatch(Date date, int numHashPartitions, Path statusPath, FileSystem hdfs) {

this.date = date;

this.numHashPartitions = numHashPartitions;

this.hasSuccessFiles = checkForSuccessFile(hdfs, date, statusPath);

}

public Date getDate() {

return date;

}

/\*\*

\* Check for the presence of the \_SUCCESS file for the given day's path on HDFS for the statuses

\* field group.

\*/

private boolean checkForSuccessFile(FileSystem hdfs, Date inputDate, Path statusPath) {

Path dayPath = new Path(statusPath, ArchiveHDFSUtils.dateToPath(inputDate, "/"));

Path successFilePath = new Path(dayPath, SUCCESS\_FILE\_NAME);

try {

return hdfs.getFileStatus(successFilePath).isFile();

} catch (IOException e) {

LOG.error("Could not verify existence of the \_SUCCESS file. Assuming it doesn't exist.", e);

}

return false;

}

/\*\*

\* Loads the data for this day for the given partition.

\*/

public PartitionedBatch addPartition(FileSystem hdfs, Path dayPath, int hashPartitionID)

throws IOException {

String partitionDir = String.format(PARTITION\_FORMAT, hashPartitionID, numHashPartitions);

Path path = new Path(dayPath, partitionDir);

PartitionedBatch batch =

new PartitionedBatch(path, hashPartitionID, numHashPartitions, date);

batch.load(hdfs);

hashPartitionToStatuses.put(hashPartitionID, batch);

return batch;

}

public PartitionedBatch getPartition(int hashPartitionID) {

return hashPartitionToStatuses.get(hashPartitionID);

}

/\*\*

\* Returns the greatest status count in all partitions belonging to this batch.

\*/

public int getMaxPerPartitionStatusCount() {

int maxPerPartitionStatusCount = 0;

for (PartitionedBatch batch : hashPartitionToStatuses.values()) {

maxPerPartitionStatusCount = Math.max(batch.getStatusCount(), maxPerPartitionStatusCount);

}

return maxPerPartitionStatusCount;

}

public int getNumHashPartitions() {

return numHashPartitions;

}

@VisibleForTesting

boolean hasSuccessFiles() {

return hasSuccessFiles;

}

/\*\*

\* Returns true if the \_status\_counts files could be found in each

\* hash partition subfolder that belongs to this timeslice

\* AND the \_SUCCESS file can be found at the root folder for day

\*/

public boolean isValid() {

// make sure we have data for all hash partitions

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

PartitionedBatch day = hashPartitionToStatuses.get(i);

if (day == null || !day.hasStatusCount() || day.isDisallowedEmptyPartition()) {

return false;

}

}

return hasSuccessFiles;

}

@Override

public String toString() {

StringBuilder builder = new StringBuilder();

builder.append("DailyStatusBatch[date=").append(date)

.append(",valid=").append(isValid())

.append(",hasSuccessFiles=").append(hasSuccessFiles)

.append(",numHashPartitions=").append(numHashPartitions)

.append("]:\n");

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

builder.append('\t').append(hashPartitionToStatuses.get(i).toString()).append('\n');

}

return builder.toString();

}

@Override

public int compareTo(DailyStatusBatch o) {

return date.compareTo(o.date);

}

/\*\*

\* Serialize DailyStatusBatch to a json string.

\*/

public String serializeToJson() {

return serializeToJson(new Gson());

}

@VisibleForTesting

String serializeToJson(Gson gson) {

return gson.toJson(this);

}

/\*\*

\* Given a json string, parse its fields and construct a daily status batch.

\* @param batchStr the json string representation of a daily status batch.

\* @return the daily status batch constructed; if the string is of invalid format, null will be

\* returned.

\*/

static DailyStatusBatch deserializeFromJson(String batchStr) {

try {

return new Gson().fromJson(batchStr, DailyStatusBatch.class);

} catch (JsonParseException e) {

LOG.error("Error parsing json string: " + batchStr, e);

return null;

}

}

}