package com.twitter.search.earlybird.common.userupdates;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Arrays;

import java.util.Iterator;

import java.util.List;

import java.util.NoSuchElementException;

import java.util.Optional;

import java.util.Spliterator;

import java.util.Spliterators;

import java.util.concurrent.TimeUnit;

import java.util.function.Predicate;

import java.util.stream.Collectors;

import java.util.stream.Stream;

import java.util.stream.StreamSupport;

import javax.annotation.Nullable;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.hdfs.HdfsConfiguration;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common\_internal.hadoop.HdfsUtils;

import com.twitter.scalding.DateRange;

import com.twitter.scalding.Hours;

import com.twitter.scalding.RichDate;

import com.twitter.search.user\_table.sources.MostRecentGoodSafetyUserStateSource;

import com.twitter.search.common.indexing.thriftjava.SafetyUserState;

import com.twitter.search.common.util.io.LzoThriftBlockFileReader;

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

import com.twitter.util.Duration;

import com.twitter.util.Time;

/\*\*

\* Builds a user table from a user safety snapshot on HDFS.

\*/

public class UserTableBuilderFromSnapshot {

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

private static final int MAX\_DAYS\_TO\_CHECK = 7;

public static final String DATA\_DIR = "user\_states";

public static final String METADATA\_DIR = "last\_updated\_ms";

private final String snapshotBaseDir;

private String snapshotDataPath;

private String snapshotMetaDataPath;

private UserTable userTable;

private long nsfwCount;

private long antisocialCount;

private long isProtectedCount;

public UserTableBuilderFromSnapshot() {

snapshotBaseDir =

EarlybirdConfig.getString(EarlybirdConfig.USER\_SNAPSHOT\_BASE\_DIR, null);

LOG.info("Configured user snapshot directory: " + snapshotBaseDir);

}

private static final class UserUpdate {

public final long userId;

@Nullable public final Boolean antisocial;

@Nullable public final Boolean nsfw;

@Nullable public final Boolean isProtected;

private UserUpdate(long userId,

@Nullable Boolean antisocial,

@Nullable Boolean nsfw,

@Nullable Boolean isProtected) {

this.userId = userId;

this.antisocial = antisocial;

this.nsfw = nsfw;

this.isProtected = isProtected;

}

public static UserUpdate fromUserState(SafetyUserState safetyUserState) {

long userId = safetyUserState.getUserID();

@Nullable Boolean antisocial = null;

@Nullable Boolean nsfw = null;

@Nullable Boolean isProtected = null;

if (safetyUserState.isIsAntisocial()) {

antisocial = true;

}

if (safetyUserState.isIsNsfw()) {

nsfw = true;

}

if (safetyUserState.isSetIsProtected() && safetyUserState.isIsProtected()) {

isProtected = true;

}

return new UserUpdate(userId, antisocial, nsfw, isProtected);

}

}

/\*\*

\* Builds a user table from an HDFS user snapshot.

\* @return The table, or nothing if something went wrong.

\*/

public Optional<UserTable> build(Predicate<Long> userFilter) {

userTable = UserTable.newTableWithDefaultCapacityAndPredicate(userFilter);

nsfwCount = 0;

antisocialCount = 0;

isProtectedCount = 0;

if (snapshotBaseDir == null || snapshotBaseDir.isEmpty()) {

LOG.info("No snapshot directory. Can't build user table.");

return Optional.empty();

}

LOG.info("Starting to build user table.");

Stream<UserUpdate> stream = null;

try {

setSnapshotPath();

stream = getUserUpdates();

stream.forEach(this::insertUser);

} catch (IOException e) {

LOG.error("IOException while building table: {}", e.getMessage(), e);

return Optional.empty();

} finally {

if (stream != null) {

stream.close();

}

}

LOG.info("Built user table with {} users, {} nsfw, {} antisocial and {} protected.",

userTable.getNumUsersInTable(),

nsfwCount,

antisocialCount,

isProtectedCount);

try {

userTable.setLastRecordTimestamp(readTimestampOfLastSeenUpdateFromSnapshot());

} catch (IOException e) {

LOG.error("IOException reading timestamp of last update: {}", e.getMessage(), e);

return Optional.empty();

}

LOG.info("Setting last record timestamp to {}.", userTable.getLastRecordTimestamp());

return Optional.of(userTable);

}

private void setSnapshotPath() {

snapshotDataPath =

new MostRecentGoodSafetyUserStateSource(

snapshotBaseDir,

DATA\_DIR,

METADATA\_DIR,

DateRange.apply(

RichDate.now().$minus(Hours.apply(MAX\_DAYS\_TO\_CHECK \* 24)),

RichDate.now())

).partitionHdfsPaths(new HdfsConfiguration())

.\_1()

.head()

.replaceAll("\\\*$", "");

snapshotMetaDataPath = snapshotDataPath.replace(DATA\_DIR, METADATA\_DIR);

LOG.info("Snapshot data path: {}", snapshotDataPath);

LOG.info("Snapshot metadata path: {}", snapshotMetaDataPath);

}

private Stream<UserUpdate> getUserUpdates() throws IOException {

FileSystem fs = FileSystem.get(new Configuration());

List<String> lzoFiles =

Arrays.stream(fs.listStatus(new Path(snapshotDataPath),

path -> path.getName().startsWith("part-")))

.map(fileStatus -> Path.getPathWithoutSchemeAndAuthority(fileStatus.getPath())

.toString())

.collect(Collectors.toList());

final LzoThriftBlockFileReader<SafetyUserState> thriftReader =

new LzoThriftBlockFileReader<>(lzoFiles, SafetyUserState.class, null);

Iterator<UserUpdate> iter = new Iterator<UserUpdate>() {

private SafetyUserState next;

@Override

public boolean hasNext() {

if (next != null) {

return true;

}

do {

try {

next = thriftReader.readNext();

} catch (IOException e) {

throw new RuntimeException(e);

}

} while (next == null && !thriftReader.isExhausted());

return next != null;

}

@Override

public UserUpdate next() {

if (next != null || hasNext()) {

UserUpdate userUpdate = UserUpdate.fromUserState(next);

next = null;

return userUpdate;

}

throw new NoSuchElementException();

}

};

return StreamSupport

.stream(

Spliterators.spliteratorUnknownSize(iter, Spliterator.ORDERED | Spliterator.NONNULL),

false)

.onClose(thriftReader::stop);

}

private long readTimestampOfLastSeenUpdateFromSnapshot() throws IOException {

String timestampFile = snapshotMetaDataPath + "part-00000";

BufferedReader buffer = new BufferedReader(new InputStreamReader(

HdfsUtils.getInputStreamSupplier(timestampFile).openStream()));

long timestampMillis = Long.parseLong(buffer.readLine());

LOG.info("read timestamp {} from HDFS:{}", timestampMillis, timestampFile);

Time time = Time.fromMilliseconds(timestampMillis)

.minus(Duration.fromTimeUnit(10, TimeUnit.MINUTES));

return time.inMilliseconds();

}

private void insertUser(UserUpdate userUpdate) {

if (userUpdate == null) {

return;

}

if (userUpdate.antisocial != null) {

userTable.set(

userUpdate.userId,

UserTable.ANTISOCIAL\_BIT,

userUpdate.antisocial);

antisocialCount++;

}

if (userUpdate.nsfw != null) {

userTable.set(

userUpdate.userId,

UserTable.NSFW\_BIT,

userUpdate.nsfw);

nsfwCount++;

}

if (userUpdate.isProtected != null) {

userTable.set(

userUpdate.userId,

UserTable.IS\_PROTECTED\_BIT,

userUpdate.isProtected);

isProtectedCount++;

}

}

}