package com.twitter.search.earlybird.partition;

import java.sql.Timestamp;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.time.Duration;

import java.util.Date;

import java.util.Optional;

import com.google.common.annotations.VisibleForTesting;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

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

import com.twitter.search.earlybird.EarlybirdStatus;

import com.twitter.search.earlybird.common.NonPagingAssert;

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

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

import com.twitter.search.earlybird.common.userupdates.UserScrubGeoMap;

import com.twitter.search.earlybird.common.userupdates.UserTableBuilderFromSnapshot;

import com.twitter.search.earlybird.common.userupdates.UserTable;

import com.twitter.search.earlybird.factory.EarlybirdIndexConfigUtil;

/\*\*

\* Indexer class responsible for getting the the {@link UserTable} and {@link UserScrubGeoMap}

\* indexed up until the current moment.

\*/

public class StartupUserEventIndexer {

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

private static final String LOAD\_USER\_UPDATE\_SNAPSHOT =

"loading user update snapshot";

private static final String INDEX\_ALL\_USER\_EVENTS =

"indexing all user events";

private static final NonPagingAssert FAILED\_USER\_TABLE\_HDFS\_LOAD

= new NonPagingAssert("failed\_user\_table\_hdfs\_load");

private static final long MAX\_RETRY\_MILLIS\_FOR\_SEEK\_TO\_TIMESTAMP =

Duration.ofMinutes(1).toMillis();

private static final long SLEEP\_MILLIS\_BETWEEN\_RETRIES\_FOR\_SEEK\_TO\_TIMESTAMP =

Duration.ofSeconds(1).toMillis();

private static final long MILLIS\_IN\_FOURTEEN\_DAYS = 1209600000;

private static final long MILLIS\_IN\_ONE\_DAY = 86400000;

private final SearchIndexingMetricSet searchIndexingMetricSet;

private final UserUpdatesStreamIndexer userUpdatesStreamIndexer;

private final UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer;

private final SegmentManager segmentManager;

private final Clock clock;

public StartupUserEventIndexer(

SearchIndexingMetricSet searchIndexingMetricSet,

UserUpdatesStreamIndexer userUpdatesStreamIndexer,

UserScrubGeoEventStreamIndexer userScrubGeoEventStreamIndexer,

SegmentManager segmentManager,

Clock clock) {

this.searchIndexingMetricSet = searchIndexingMetricSet;

this.userUpdatesStreamIndexer = userUpdatesStreamIndexer;

this.userScrubGeoEventStreamIndexer = userScrubGeoEventStreamIndexer;

this.segmentManager = segmentManager;

this.clock = clock;

}

/\*\*

\* Index all user events.

\*/

public void indexAllEvents() {

EarlybirdStatus.beginEvent(

INDEX\_ALL\_USER\_EVENTS, searchIndexingMetricSet.startupInUserEventIndexer);

indexUserUpdates();

if (EarlybirdConfig.consumeUserScrubGeoEvents()) {

indexUserScrubGeoEvents();

}

EarlybirdStatus.endEvent(

INDEX\_ALL\_USER\_EVENTS, searchIndexingMetricSet.startupInUserEventIndexer);

}

/\*\*

\* Index user updates until current.

\*/

public void indexUserUpdates() {

EarlybirdStatus.beginEvent(

LOAD\_USER\_UPDATE\_SNAPSHOT, searchIndexingMetricSet.startupInUserUpdates);

Optional<UserTable> userTable = buildUserTable();

if (userTable.isPresent()) {

segmentManager.getUserTable().setTable(userTable.get());

LOG.info("Set new user table.");

if (!seekToTimestampWithRetriesIfNecessary(

userTable.get().getLastRecordTimestamp(),

userUpdatesStreamIndexer)) {

LOG.error("User Updates stream indexer unable to seek to timestamp. "

+ "Will seek to beginning.");

userUpdatesStreamIndexer.seekToBeginning();

}

} else {

LOG.info("Failed to load user update snapshot. Will reindex user updates from scratch.");

FAILED\_USER\_TABLE\_HDFS\_LOAD.assertFailed();

userUpdatesStreamIndexer.seekToBeginning();

}

userUpdatesStreamIndexer.readRecordsUntilCurrent();

LOG.info("Finished catching up on user updates via Kafka");

EarlybirdStatus.endEvent(

LOAD\_USER\_UPDATE\_SNAPSHOT, searchIndexingMetricSet.startupInUserUpdates);

}

/\*\*

\* Index UserScrubGeoEvents until current.

\*/

public void indexUserScrubGeoEvents() {

seekUserScrubGeoEventKafkaConsumer();

SearchTimer timer = new SearchTimer();

timer.start();

userScrubGeoEventStreamIndexer.readRecordsUntilCurrent();

timer.stop();

LOG.info("Finished catching up on user scrub geo events via Kafka");

LOG.info("UserScrubGeoMap contains {} users and finished in {} milliseconds",

segmentManager.getUserScrubGeoMap().getNumUsersInMap(), timer.getElapsed());

}

/\*\*

\* Seeks UserScrubGeoEventKafkaConsumer using timestamp derived from

\* getTimestampForUserScrubGeoEventKafkaConsumer().

\*/

@VisibleForTesting

public void seekUserScrubGeoEventKafkaConsumer() {

long seekTimestamp = getTimestampForUserScrubGeoEventKafkaConsumer();

if (seekTimestamp == -1) {

userScrubGeoEventStreamIndexer.seekToBeginning();

} else {

if (!seekToTimestampWithRetriesIfNecessary(seekTimestamp, userScrubGeoEventStreamIndexer)) {

LOG.error("User Scrub Geo stream indexer unable to seek to timestamp. "

+ "Will seek to beginning.");

userScrubGeoEventStreamIndexer.seekToBeginning();

}

}

}

/\*\*

\* Get timestamp to seek UserScrubGeoEventKafkaConsumer to.

\* @return

\*/

public long getTimestampForUserScrubGeoEventKafkaConsumer() {

if (EarlybirdIndexConfigUtil.isArchiveSearch()) {

return getTimestampForArchive();

} else {

return getTimestampForRealtime();

}

}

/\*\*

\* For archive: grab scrub gen from config file and convert date into a timestamp. Add buffer of

\* one day. We need all UserScrubGeoEvents since the date of the current scrub gen.

\*

\* See go/realtime-geo-filtering

\* @return

\*/

public long getTimestampForArchive() {

try {

String scrubGenString = EarlybirdProperty.EARLYBIRD\_SCRUB\_GEN.get();

DateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");

Date date = dateFormat.parse(scrubGenString);

return new Timestamp(date.getTime()).getTime() - MILLIS\_IN\_ONE\_DAY;

} catch (Exception e) {

LOG.error("Could not derive timestamp from scrub gen. "

+ "Will seek User Scrub Geo Kafka consumer to beginning of topic");

}

return -1;

}

/\*\*

\* For realtime/protected: Compute the timestamp 14 days from the current time. This will account

\* for all events that have occurred during the lifecylce of the current index.

\*

\* See go/realtime-geo-filtering

\*/

public long getTimestampForRealtime() {

return System.currentTimeMillis() - MILLIS\_IN\_FOURTEEN\_DAYS;

}

private boolean seekToTimestampWithRetriesIfNecessary(

long lastRecordTimestamp,

SimpleStreamIndexer streamIndexer) {

long initialTimeMillis = clock.nowMillis();

int numFailures = 0;

while (shouldTrySeekToTimestamp(initialTimeMillis, numFailures)) {

try {

streamIndexer.seekToTimestamp(lastRecordTimestamp);

LOG.info("Seeked consumer to timestamp {} after {} failures",

lastRecordTimestamp, numFailures);

return true;

} catch (Exception e) {

numFailures++;

LOG.info("Caught exception when seeking to timestamp. Num failures: {}. Exception: {}",

numFailures, e);

// Sleep before attempting to retry

try {

clock.waitFor(SLEEP\_MILLIS\_BETWEEN\_RETRIES\_FOR\_SEEK\_TO\_TIMESTAMP);

} catch (InterruptedException interruptedException) {

LOG.warn("Interrupted while sleeping between seekToTimestamp retries",

interruptedException);

// Preserve interrupt status.

Thread.currentThread().interrupt();

break;

}

}

}

// Failed to seek to timestamp

return false;

}

private boolean shouldTrySeekToTimestamp(long initialTimeMillis, int numFailures) {

if (numFailures == 0) {

// no attempts have been made yet, so we should try to seek to timestamp

return true;

} else {

return clock.nowMillis() - initialTimeMillis < MAX\_RETRY\_MILLIS\_FOR\_SEEK\_TO\_TIMESTAMP;

}

}

protected Optional<UserTable> buildUserTable() {

UserTableBuilderFromSnapshot builder = new UserTableBuilderFromSnapshot();

return builder.build(segmentManager.getUserTable().getUserIdFilter());

}

}