package com.twitter.search.earlybird.config;

import java.util.Date;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.twitter.common.util.Clock;

import com.twitter.search.common.partitioning.snowflakeparser.SnowflakeIdParser;

/\*\*

\* The start or end boundary of a tier's serving range.

\* This is used to add since\_id and max\_id operators onto search queries.

\*/

public class TierServingBoundaryEndPoint {

@VisibleForTesting

public static final String INFERRED\_FROM\_DATA\_RANGE = "inferred\_from\_data\_range";

public static final String RELATIVE\_TO\_CURRENT\_TIME\_MS = "relative\_to\_current\_time\_ms";

// Either offsetToCurrentTimeMillis is set or (absoluteTweetId and timeBoundarySecondsFromEpoch)

// are set.

@Nullable

private final Long offsetToCurrentTimeMillis;

@Nullable

private final Long absoluteTweetId;

@Nullable

private final Long timeBoundarySecondsFromEpoch;

private final Clock clock;

TierServingBoundaryEndPoint(Long absoluteTweetId,

Long timeBoundarySecondsFromEpoch,

Long offsetToCurrentTimeMillis,

Clock clock) {

this.offsetToCurrentTimeMillis = offsetToCurrentTimeMillis;

this.absoluteTweetId = absoluteTweetId;

this.timeBoundarySecondsFromEpoch = timeBoundarySecondsFromEpoch;

this.clock = clock;

}

/\*\*

\* Parse the boundary string and construct a TierServingBoundaryEndPoint instance.

\* @param boundaryString boundary configuration string. Valid values are:

\* <li>

\* "inferred\_from\_data\_range" infers serving range from data range. This only works after

\* Nov 2010 when Twitter switched to snowflake IDs.

\* This is the default value.

\* </li>

\* <li>

\* "absolute\_tweet\_id\_and\_timestamp\_millis:id:timestamp" a tweet ID/timestamp is given

\* explicitly as the serving range

\* boundary.

\* </li>

\* <li>

\* "relative\_to\_current\_time\_ms:offset" adds offset onto current timestamp in millis to

\* compute serving range.

\* </li>

\*

\* @param boundaryDate the data boundary. This is used in conjunction with

\* inferred\_from\_data\_date to determine the serving boundary.

\* @param clock Clock used to obtain current time, when relative\_to\_current\_time\_ms is used.

\* Tests pass in a FakeClock.

\*/

public static TierServingBoundaryEndPoint newTierServingBoundaryEndPoint(String boundaryString,

Date boundaryDate,

Clock clock) {

if (boundaryString == null || boundaryString.trim().equals(

INFERRED\_FROM\_DATA\_RANGE)) {

return inferBoundaryFromDataRange(boundaryDate, clock);

} else if (boundaryString.trim().startsWith(RELATIVE\_TO\_CURRENT\_TIME\_MS)) {

return getRelativeBoundary(boundaryString, clock);

} else {

throw new IllegalStateException("Cannot parse serving range string: " + boundaryString);

}

}

private static TierServingBoundaryEndPoint inferBoundaryFromDataRange(Date boundaryDate,

Clock clock) {

// infer from data range

// handle default start date and end date, in case the dates are not specified in the config

if (boundaryDate.equals(TierConfig.DEFAULT\_TIER\_START\_DATE)) {

return new TierServingBoundaryEndPoint(

-1L, TierConfig.DEFAULT\_TIER\_START\_DATE.getTime() / 1000, null, clock);

} else if (boundaryDate.equals(TierConfig.DEFAULT\_TIER\_END\_DATE)) {

return new TierServingBoundaryEndPoint(

Long.MAX\_VALUE, TierConfig.DEFAULT\_TIER\_END\_DATE.getTime() / 1000, null, clock);

} else {

// convert data start / end dates into since / max ID.

long boundaryTimeMillis = boundaryDate.getTime();

if (!SnowflakeIdParser.isUsableSnowflakeTimestamp(boundaryTimeMillis)) {

throw new IllegalStateException("Serving time range can not be determined, because "

+ boundaryDate + " is before Twitter switched to snowflake tweet IDs.");

}

// Earlybird since\_id is inclusive and max\_id is exclusive. We substract 1 here.

// Consider example:

// full0: 5000 (inclusive) - 6000 (exclusive)

// full1: 6000 (inclusive) - 7000 (exclusive)

// For tier full0, we should use max\_id 5999 instead of 6000.

// For tier full1, we should use since\_id 5999 instead of 6000.

// Hence we substract 1 here.

long adjustedTweetId =

SnowflakeIdParser.generateValidStatusId(boundaryTimeMillis, 0) - 1;

Preconditions.checkState(adjustedTweetId >= 0, "boundary tweet ID must be non-negative");

return new TierServingBoundaryEndPoint(

adjustedTweetId, boundaryTimeMillis / 1000, null, clock);

}

}

private static TierServingBoundaryEndPoint getRelativeBoundary(String boundaryString,

Clock clock) {

// An offset relative to current time is given

String[] parts = boundaryString.split(":");

Preconditions.checkState(parts.length == 2);

long offset = Long.parseLong(parts[1]);

return new TierServingBoundaryEndPoint(null, null, offset, clock);

}

/\*\*

\* Returns the tweet ID for this tier boundary. If the tier boundary was created using a tweet ID,

\* that tweet ID is returned. Otherwise, a tweet ID is derived from the time boundary.

\*/

@VisibleForTesting

public long getBoundaryTweetId() {

// If absoluteTweetId is available, use it.

if (absoluteTweetId != null) {

return absoluteTweetId;

} else {

Preconditions.checkNotNull(offsetToCurrentTimeMillis);

long boundaryTime = clock.nowMillis() + offsetToCurrentTimeMillis;

return SnowflakeIdParser.generateValidStatusId(boundaryTime, 0);

}

}

/\*\*

\* Returns the time boundary for this tier boundary, in seconds since epoch.

\*/

public long getBoundaryTimeSecondsFromEpoch() {

if (timeBoundarySecondsFromEpoch != null) {

return timeBoundarySecondsFromEpoch;

} else {

Preconditions.checkNotNull(offsetToCurrentTimeMillis);

return (clock.nowMillis() + offsetToCurrentTimeMillis) / 1000;

}

}

}