package com.twitter.tweetypie

package backends

import com.twitter.servo.util.FutureArrow

import com.twitter.stitch.Stitch

import com.twitter.storage.client.manhattan.bijections.Bijections.\_

import com.twitter.storage.client.manhattan.kv.\_

import com.twitter.storage.client.manhattan.kv.impl.\_

import com.twitter.util.Time

/\*\*

\* Read and write the timestamp of the last delete\_location\_data request

\* for a user. This is used as a safeguard to prevent leaking geo data

\* with tweets that have not yet been scrubbed or were missed during the

\* geo scrubbing process.

\*/

object GeoScrubEventStore {

type GetGeoScrubTimestamp = UserId => Stitch[Option[Time]]

type SetGeoScrubTimestamp = FutureArrow[(UserId, Time), Unit]

private[this] val KeyDesc =

KeyDescriptor(

Component(LongInjection),

Component(LongInjection, StringInjection)

).withDataset("geo\_scrub")

private[this] val ValDesc = ValueDescriptor(LongInjection)

// This modulus determines how user ids get assigned to PKeys, and

// thus to shards within the MH cluster. The origin of the specific

// value has been lost to time, but it's important that we don't

// change it, or else the existing data will be inaccessible.

private[this] val PKeyModulus: Long = 25000L

private[this] def toKey(userId: Long) =

KeyDesc

.withPkey(userId % PKeyModulus)

.withLkey(userId, "\_last\_scrub")

def apply(client: ManhattanKVClient, config: Config, ctx: Backend.Context): GeoScrubEventStore = {

new GeoScrubEventStore {

val getGeoScrubTimestamp: UserId => Stitch[Option[Time]] = {

val endpoint = config.read.endpoint(client)

(userId: UserId) => {

endpoint

.get(toKey(userId), ValDesc)

.map(\_.map(value => Time.fromMilliseconds(value.contents)))

}

}

val setGeoScrubTimestamp: SetGeoScrubTimestamp = {

val endpoint = config.write.endpoint(client)

FutureArrow {

case (userId, timestamp) =>

val key = toKey(userId)

// Use the geo scrub timestamp as the MH entry timestamp. This

// ensures that whatever timestamp is highest will win any

// update races.

val value = ValDesc.withValue(timestamp.inMilliseconds, timestamp)

Stitch.run(endpoint.insert(key, value))

}

}

}

}

case class EndpointConfig(requestTimeout: Duration, maxRetryCount: Int) {

def endpoint(client: ManhattanKVClient): ManhattanKVEndpoint =

ManhattanKVEndpointBuilder(client)

.defaultMaxTimeout(requestTimeout)

.maxRetryCount(maxRetryCount)

.build()

}

case class Config(read: EndpointConfig, write: EndpointConfig)

}

trait GeoScrubEventStore {

import GeoScrubEventStore.\_

val getGeoScrubTimestamp: GetGeoScrubTimestamp

val setGeoScrubTimestamp: SetGeoScrubTimestamp

}