package com.twitter.ann.faiss

import com.twitter.ann.common.Distance

import com.twitter.ann.common.MemoizedInEpochs

import com.twitter.ann.common.Metric

import com.twitter.ann.common.Task

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.search.common.file.AbstractFile

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.Time

import com.twitter.util.Try

import com.twitter.util.logging.Logging

import java.util.concurrent.atomic.AtomicReference

object HourlyShardedIndex {

def loadIndex[T, D <: Distance[D]](

dimension: Int,

metric: Metric[D],

directory: AbstractFile,

shardsToLoad: Int,

shardWatchInterval: Duration,

lookbackInterval: Int,

statsReceiver: StatsReceiver

): HourlyShardedIndex[T, D] = {

new HourlyShardedIndex[T, D](

metric,

dimension,

directory,

shardsToLoad,

shardWatchInterval,

lookbackInterval,

statsReceiver)

}

}

class HourlyShardedIndex[T, D <: Distance[D]](

outerMetric: Metric[D],

outerDimension: Int,

directory: AbstractFile,

shardsToLoad: Int,

shardWatchInterval: Duration,

lookbackInterval: Int,

override protected val statsReceiver: StatsReceiver)

extends QueryableIndexAdapter[T, D]

with Logging

with Task {

// QueryableIndexAdapter

protected val metric: Metric[D] = outerMetric

protected val dimension: Int = outerDimension

protected def index: Index = {

castedIndex.get()

}

// Task trait

protected def task(): Future[Unit] = Future.value(reloadShards())

protected def taskInterval: Duration = shardWatchInterval

private def loadIndex(directory: AbstractFile): Try[Index] =

Try(QueryableIndexAdapter.loadJavaIndex(directory))

private val shardsCache = new MemoizedInEpochs[AbstractFile, Index](loadIndex)

// Destroying original index invalidate casted index. Keep a reference to both.

private val originalIndex = new AtomicReference[IndexShards]()

private val castedIndex = new AtomicReference[Index]()

private def reloadShards(): Unit = {

val freshDirectories =

HourlyDirectoryWithSuccessFileListing.listHourlyIndexDirectories(

directory,

Time.now,

shardsToLoad,

lookbackInterval)

if (shardsCache.currentEpochKeys == freshDirectories.toSet) {

info("Not reloading shards, as they're exactly same")

} else {

val shards = shardsCache.epoch(freshDirectories)

val indexShards = new IndexShards(dimension, false, false)

for (shard <- shards) {

indexShards.add\_shard(shard)

}

replaceIndex(() => {

castedIndex.set(swigfaiss.upcast\_IndexShards(indexShards))

originalIndex.set(indexShards)

})

// Potentially it's time to drop huge native index from memory, ask for GC

System.gc()

}

require(castedIndex.get() != null, "Failed to find any shards during startup")

}

}