package com.twitter.ann.service.query\_server.common

import com.twitter.ann.common.IndexOutputFile

import com.twitter.ann.hnsw.HnswCommon.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.logging.Logger

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

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

import com.twitter.search.common.file.PathUtils

import com.twitter.util.Try

import java.io.IOException

import java.util.concurrent.atomic.AtomicReference

import scala.collection.JavaConverters.\_

import scala.math.Ordering.comparatorToOrdering

abstract class IndexPathProvider {

def provideIndexPath(rootPath: AbstractFile, group: Boolean = false): Try[AbstractFile]

def provideIndexPathWithGroups(rootPath: AbstractFile): Try[Seq[AbstractFile]]

}

abstract class BaseIndexPathProvider extends IndexPathProvider {

protected val minIndexSizeBytes: Long

protected val maxIndexSizeBytes: Long

protected val statsReceiver: StatsReceiver

protected val log: Logger

private val invalidPathCounter = statsReceiver.counter("invalid\_index")

private val failToLocateDirectoryCounter = statsReceiver.counter("find\_latest\_path\_fail")

private val successProvidePathCounter = statsReceiver.counter("provide\_path\_success")

private val latestGroupCount = new AtomicReference(0f)

private val latestIndexTimestamp = new AtomicReference(0f)

private val latestValidIndexTimestamp = new AtomicReference(0f)

private val INDEX\_METADATA\_FILE = "ANN\_INDEX\_METADATA"

private val latestIndexGauge = statsReceiver.addGauge("latest\_index\_timestamp")(

latestIndexTimestamp.get()

)

private val latestValidIndexGauge = statsReceiver.addGauge("latest\_valid\_index\_timestamp")(

latestValidIndexTimestamp.get()

)

private val latestGroupCountGauge = statsReceiver.addGauge("latest\_group\_count")(

latestGroupCount.get()

)

private val latestTimeStampDirectoryFilter = new AbstractFile.Filter {

/\*\* Determines which files should be accepted when listing a directory. \*/

override def accept(file: AbstractFile): Boolean = {

val name = file.getName

PathUtils.TIMESTAMP\_PATTERN.matcher(name).matches()

}

}

private def findLatestTimeStampValidSuccessDirectory(

path: AbstractFile,

group: Boolean

): AbstractFile = {

log.info(s"Calling findLatestTimeStampValidSuccessDirectory with ${path.getPath}")

// Get all the timestamp directories

val dateDirs = path.listFiles(latestTimeStampDirectoryFilter).asScala.toSeq

if (dateDirs.nonEmpty) {

// Validate the indexes

val latestValidPath = {

if (group) {

// For grouped, check all the individual group indexes and stop as soon as a valid index

// is found.

dateDirs

.sorted(comparatorToOrdering(PathUtils.NEWEST\_FIRST\_COMPARATOR)).find(file => {

val indexMetadataFile = file.getChild(INDEX\_METADATA\_FILE)

val indexes = file.listFiles().asScala.filter(\_.isDirectory)

val isValid = if (indexMetadataFile.exists()) {

// Metadata file exists. Check the number of groups and verify the index is

// complete

val indexMetadata = new IndexOutputFile(indexMetadataFile).loadIndexMetadata()

if (indexMetadata.numGroups.get != indexes.size) {

log.info(

s"Grouped index ${file.getPath} should have ${indexMetadata.numGroups.get} groups but had ${indexes.size}")

}

indexMetadata.numGroups.get == indexes.size

} else {

// True if the file doesn't exist. This is to make this change backwards

// compatible for clients using the old version of the dataflow job

true

}

isValid && indexes.forall(index => {

index.hasSuccessFile && isValidIndex(index) && QueryServerUtil

.isValidIndexDirSize(index, minIndexSizeBytes, maxIndexSizeBytes)

})

})

} else {

// For non-grouped, find the first valid index.

dateDirs

.sorted(comparatorToOrdering(PathUtils.NEWEST\_FIRST\_COMPARATOR)).find(file => {

file.hasSuccessFile && QueryServerUtil

.isValidIndexDirSize(file, minIndexSizeBytes, maxIndexSizeBytes)

})

}

}

if (latestValidPath.nonEmpty) {

// Log the results

successProvidePathCounter.incr()

if (group) {

latestGroupCount.set(latestValidPath.get.listFiles().asScala.count(\_.isDirectory))

log.info(

s"findLatestTimeStampValidSuccessDirectory latestValidPath ${latestValidPath.get.getPath} and number of groups $latestGroupCount")

} else {

val latestValidPathSize =

latestValidPath.get.listFiles(true).asScala.map(\_.getSizeInBytes).sum

log.info(

s"findLatestTimeStampValidSuccessDirectory latestValidPath ${latestValidPath.get.getPath} and size $latestValidPathSize")

}

return latestValidPath.get

}

}

// Fail if no index or no valid index.

failToLocateDirectoryCounter.incr()

throw new IOException(s"Cannot find any valid directory with SUCCESS file at ${path.getName}")

}

def isValidIndex(index: AbstractFile): Boolean

override def provideIndexPath(

rootPath: AbstractFile,

group: Boolean = false

): Try[AbstractFile] = {

Try {

val latestValidPath = findLatestTimeStampValidSuccessDirectory(rootPath, group)

if (!group) {

val latestPath = PathUtils.findLatestTimeStampSuccessDirectory(rootPath)

// since latestValidPath does not throw exception, latestPath must exist

assert(latestPath.isPresent)

val latestPathSize = latestPath.get.listFiles(true).asScala.map(\_.getSizeInBytes).sum

log.info(s"provideIndexPath latestPath ${latestPath

.get()

.getPath} and size $latestPathSize")

latestIndexTimestamp.set(latestPath.get().getName.toFloat)

// latest directory is not valid, update counter for alerts

if (latestPath.get() != latestValidPath) {

invalidPathCounter.incr()

}

} else {

latestIndexTimestamp.set(latestValidPath.getName.toFloat)

}

latestValidIndexTimestamp.set(latestValidPath.getName.toFloat)

latestValidPath

}

}

override def provideIndexPathWithGroups(

rootPath: AbstractFile

): Try[Seq[AbstractFile]] = {

val latestValidPath = provideIndexPath(rootPath, true)

latestValidPath.map { path =>

path

.listFiles(new Filter {

override def accept(file: AbstractFile): Boolean =

file.isDirectory && file.hasSuccessFile

}).asScala.toSeq

}

}

}

case class ValidatedIndexPathProvider(

override val minIndexSizeBytes: Long,

override val maxIndexSizeBytes: Long,

override val statsReceiver: StatsReceiver)

extends BaseIndexPathProvider {

override val log = Logger.get("ValidatedIndexPathProvider")

override def isValidIndex(dir: AbstractFile): Boolean = {

isValidHnswIndex(dir)

}

}