package com.twitter.ann.common

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

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

import com.twitter.util.Future

import org.apache.beam.sdk.io.fs.ResourceId

import scala.collection.JavaConverters.\_

object ShardConstants {

val ShardPrefix = "shard\_"

}

/\*\*

\* Serialize shards to directory

\* @param shards: List of shards to serialize

\*/

class ShardedSerialization(

shards: Seq[Serialization])

extends Serialization {

override def toDirectory(directory: AbstractFile): Unit = {

toDirectory(new IndexOutputFile(directory))

}

override def toDirectory(directory: ResourceId): Unit = {

toDirectory(new IndexOutputFile(directory))

}

private def toDirectory(directory: IndexOutputFile): Unit = {

shards.indices.foreach { shardId =>

val shardDirectory = directory.createDirectory(ShardConstants.ShardPrefix + shardId)

val serialization = shards(shardId)

if (shardDirectory.isAbstractFile) {

serialization.toDirectory(shardDirectory.abstractFile)

} else {

serialization.toDirectory(shardDirectory.resourceId)

}

}

}

}

/\*\*

\* Deserialize directories containing index shards data to a composed queryable

\* @param deserializationFn function to deserialize a shard file to Queryable

\* @tparam T the id of the embeddings

\* @tparam P : Runtime params type

\* @tparam D: Distance metric type

\*/

class ComposedQueryableDeserialization[T, P <: RuntimeParams, D <: Distance[D]](

deserializationFn: (AbstractFile) => Queryable[T, P, D])

extends QueryableDeserialization[T, P, D, Queryable[T, P, D]] {

override def fromDirectory(directory: AbstractFile): Queryable[T, P, D] = {

val shardDirs = directory

.listFiles(new Filter {

override def accept(file: AbstractFile): Boolean =

file.getName.startsWith(ShardConstants.ShardPrefix)

})

.asScala

.toList

val indices = shardDirs

.map { shardDir =>

deserializationFn(shardDir)

}

new ComposedQueryable[T, P, D](indices)

}

}

class ShardedIndexBuilderWithSerialization[T, P <: RuntimeParams, D <: Distance[D]](

shardedIndex: ShardedAppendable[T, P, D],

shardedSerialization: ShardedSerialization)

extends Appendable[T, P, D]

with Serialization {

override def append(entity: EntityEmbedding[T]): Future[Unit] = {

shardedIndex.append(entity)

}

override def toDirectory(directory: AbstractFile): Unit = {

shardedSerialization.toDirectory(directory)

}

override def toDirectory(directory: ResourceId): Unit = {

shardedSerialization.toDirectory(directory)

}

override def toQueryable: Queryable[T, P, D] = {

shardedIndex.toQueryable

}

}