package com.twitter.search.common.util.ml.models\_manager;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.UncheckedIOException;

import java.util.Collections;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Optional;

import java.util.Set;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

import java.util.function.Function;

import java.util.function.Supplier;

import java.util.stream.Collectors;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.base.Strings;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Sets;

import com.google.common.util.concurrent.ThreadFactoryBuilder;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.yaml.snakeyaml.Yaml;

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

import com.twitter.search.common.file.FileUtils;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.metrics.SearchLongGauge;

/\*\*

\* Loads models from HDFS and provides an interface for reloading them periodically.

\*

\* There are 2 possible ways of detecting the active models:

\*

\* - DirectorySupplier: Uses all the subdirectories of a base path

\* - ConfigSupplier: Gets the list from from a configuration file

\*

\* Models can be updated or added. Depending on the selected method, existing models can be removed

\* if they are no longer active.

\*/

public abstract class BaseModelsManager<T> implements Runnable {

private static final Logger LOG = LoggerFactory.getLogger(BaseModelsManager.class);

protected final Map<String, Long> lastModifiedMsByModel = new ConcurrentHashMap<>();

protected final Map<String, T> loadedModels = new ConcurrentHashMap<>();

protected final Supplier<Map<String, AbstractFile>> activeModelsSupplier;

protected Map<String, T> prevLoadedModels = new ConcurrentHashMap<>();

// This flag determines whether models are unloaded immediately when they're removed from

// activeModelsSupplier. If false, old models stay in memory until the process is restarted.

// This may be useful to safely change model configuration without restarting.

protected final boolean shouldUnloadInactiveModels;

protected final SearchLongGauge numModels;

protected final SearchCounter numErrors;

protected final SearchLongGauge lastLoadedMs;

protected Supplier<Boolean> shouldServeModels;

protected Supplier<Boolean> shouldLoadModels;

public BaseModelsManager(

Supplier<Map<String, AbstractFile>> activeModelsSupplier,

boolean shouldUnloadInactiveModels,

String statsPrefix

) {

this(

activeModelsSupplier,

shouldUnloadInactiveModels,

statsPrefix,

() -> true,

() -> true

);

}

public BaseModelsManager(

Supplier<Map<String, AbstractFile>> activeModelsSupplier,

boolean shouldUnloadInactiveModels,

String statsPrefix,

Supplier<Boolean> shouldServeModels,

Supplier<Boolean> shouldLoadModels

) {

this.activeModelsSupplier = activeModelsSupplier;

this.shouldUnloadInactiveModels = shouldUnloadInactiveModels;

this.shouldServeModels = shouldServeModels;

this.shouldLoadModels = shouldLoadModels;

numModels = SearchLongGauge.export(

String.format("model\_loader\_%s\_num\_models", statsPrefix));

numErrors = SearchCounter.export(

String.format("model\_loader\_%s\_num\_errors", statsPrefix));

lastLoadedMs = SearchLongGauge.export(

String.format("model\_loader\_%s\_last\_loaded\_timestamp\_ms", statsPrefix));

}

/\*\*

\* Retrieves a particular model.

\*/

public Optional<T> getModel(String name) {

if (shouldServeModels.get()) {

return Optional.ofNullable(loadedModels.get(name));

} else {

return Optional.empty();

}

}

/\*\*

\* Reads a model instance from the directory file instance.

\*

\* @param modelBaseDir AbstractFile instance representing the directory.

\* @return Model instance parsed from the directory.

\*/

public abstract T readModelFromDirectory(AbstractFile modelBaseDir) throws Exception;

/\*\*

\* Cleans up any resources used by the model instance.

\* This method is called after removing the model from the in-memory map.

\* Sub-classes can provide custom overridden implementation as required.

\*

\* @param unloadedModel Model instance that would be unloaded from the manager.

\*/

protected void cleanUpUnloadedModel(T unloadedModel) { }

@Override

public void run() {

// Get available models, either from the config file or by listing the base directory

final Map<String, AbstractFile> modelPathsFromConfig;

if (!shouldLoadModels.get()) {

LOG.info("Loading models is currently disabled.");

return;

}

modelPathsFromConfig = activeModelsSupplier.get();

for (Map.Entry<String, AbstractFile> nameAndPath : modelPathsFromConfig.entrySet()) {

String modelName = nameAndPath.getKey();

try {

AbstractFile modelDirectory = nameAndPath.getValue();

if (!modelDirectory.exists() && loadedModels.containsKey(modelName)) {

LOG.warn("Loaded model '{}' no longer exists at HDFS path {}, keeping loaded version; "

+ "replace directory in HDFS to update model.", modelName, modelDirectory);

continue;

}

long previousModifiedTimestamp = lastModifiedMsByModel.getOrDefault(modelName, 0L);

long lastModifiedMs = modelDirectory.getLastModified();

if (previousModifiedTimestamp == lastModifiedMs) {

continue;

}

LOG.info("Starting to load model. name={} path={}", modelName, modelDirectory.getPath());

T model = Preconditions.checkNotNull(readModelFromDirectory(modelDirectory));

LOG.info("Model initialized: {}. Last modified: {} ({})",

modelName, lastModifiedMs, new Date(lastModifiedMs));

T previousModel = loadedModels.put(modelName, model);

lastModifiedMsByModel.put(modelName, lastModifiedMs);

if (previousModel != null) {

cleanUpUnloadedModel(previousModel);

}

} catch (Exception e) {

numErrors.increment();

LOG.error("Error initializing model: {}", modelName, e);

}

}

// Remove any currently loaded models not present in the latest list

if (shouldUnloadInactiveModels) {

Set<String> inactiveModels =

Sets.difference(loadedModels.keySet(), modelPathsFromConfig.keySet()).immutableCopy();

for (String modelName : inactiveModels) {

T modelToUnload = loadedModels.get(modelName);

loadedModels.remove(modelName);

if (modelToUnload != null) {

// We could have an inactive model key without a model (value) if the

// initial readModelFromDirectory failed for the model entry.

// Checking for null to avoid exception.

cleanUpUnloadedModel(modelToUnload);

}

LOG.info("Unloaded model that is no longer active: {}", modelName);

}

}

if (!prevLoadedModels.keySet().equals(loadedModels.keySet())) {

LOG.info("Finished loading models: {}", loadedModels.keySet());

}

prevLoadedModels = loadedModels;

numModels.set(loadedModels.size());

lastLoadedMs.set(System.currentTimeMillis());

}

/\*\*

\* Schedules the loader to run periodically.

\* @param period Period between executions

\* @param timeUnit The time unit the period parameter.

\*/

public final void scheduleAtFixedRate(

long period, TimeUnit timeUnit, String builderThreadName) {

Executors.newSingleThreadScheduledExecutor(

new ThreadFactoryBuilder()

.setDaemon(true)

.setNameFormat(builderThreadName)

.build())

.scheduleAtFixedRate(this, 0, period, timeUnit);

}

/\*\*

\* Gets the active list of models from the subdirectories in a base directory.

\*

\* Each model is identified by the name of the subdirectory.

\*/

@VisibleForTesting

public static class DirectorySupplier implements Supplier<Map<String, AbstractFile>> {

private static final Logger LOG = LoggerFactory.getLogger(DirectorySupplier.class);

private final AbstractFile baseDir;

public DirectorySupplier(AbstractFile baseDir) {

this.baseDir = baseDir;

}

@Override

public Map<String, AbstractFile> get() {

try {

LOG.info("Loading models from the directories in: {}", baseDir.getPath());

List<AbstractFile> modelDirs =

ImmutableList.copyOf(baseDir.listFiles(AbstractFile.IS\_DIRECTORY));

LOG.info("Found {} model directories: {}", modelDirs.size(), modelDirs);

return modelDirs.stream()

.collect(Collectors.toMap(

AbstractFile::getName,

Function.identity()

));

} catch (IOException e) {

throw new UncheckedIOException(e);

}

}

}

/\*\*

\* Gets the active list of models by reading a YAML config file.

\*

\* The keys are the model names, the values are dictionaries with a single entry for the path

\* of the model in HDFS (without the HDFS name node prefix). For example:

\*

\* model\_a:

\* path: /path/to/model\_a

\* model\_b:

\* path: /path/to/model\_b

\*

\*/

@VisibleForTesting

public static class ConfigSupplier implements Supplier<Map<String, AbstractFile>> {

private final AbstractFile configFile;

public ConfigSupplier(AbstractFile configFile) {

this.configFile = configFile;

}

@SuppressWarnings("unchecked")

@Override

public Map<String, AbstractFile> get() {

try (BufferedReader configReader = configFile.getCharSource().openBufferedStream()) {

Yaml yamlParser = new Yaml();

//noinspection unchecked

Map<String, Map<String, String>> config =

(Map<String, Map<String, String>>) yamlParser.load(configReader);

if (config == null || config.isEmpty()) {

return Collections.emptyMap();

}

Map<String, AbstractFile> modelPaths = new HashMap<>();

for (Map.Entry<String, Map<String, String>> nameAndConfig : config.entrySet()) {

String path = Strings.emptyToNull(nameAndConfig.getValue().get("path"));

Preconditions.checkNotNull(path, "Missing path for model: %s", nameAndConfig.getKey());

modelPaths.put(nameAndConfig.getKey(), FileUtils.getHdfsFileHandle(path));

}

return modelPaths;

} catch (IOException e) {

throw new UncheckedIOException(e);

}

}

}

}