package com.twitter.search.earlybird;

import java.io.File;

import java.io.IOException;

import java.util.Random;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.atomic.AtomicLong;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Charsets;

import org.apache.thrift.TException;

import org.apache.zookeeper.KeeperException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

import com.twitter.common.zookeeper.ZooKeeperClient;

import com.twitter.search.common.aurora.AuroraSchedulerClient;

import com.twitter.search.common.concurrent.ScheduledExecutorServiceFactory;

import com.twitter.search.common.decider.SearchDecider;

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

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

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

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

import com.twitter.search.common.schema.AnalyzerFactory;

import com.twitter.search.common.schema.DynamicSchema;

import com.twitter.search.common.schema.ImmutableSchema;

import com.twitter.search.common.schema.base.Schema;

import com.twitter.search.common.schema.thriftjava.ThriftSchema;

import com.twitter.search.common.util.ml.tensorflow\_engine.TensorflowModelsManager;

import com.twitter.search.common.util.thrift.ThriftUtils;

import com.twitter.search.common.util.zookeeper.ZooKeeperProxy;

import com.twitter.search.earlybird.common.NonPagingAssert;

import com.twitter.search.earlybird.exception.CriticalExceptionHandler;

import com.twitter.search.earlybird.ml.ScoringModelsManager;

import com.twitter.search.earlybird.partition.DynamicPartitionConfig;

import com.twitter.search.earlybird.partition.PartitionConfig;

import com.twitter.search.earlybird.partition.PartitionConfigLoader;

import com.twitter.search.earlybird.partition.PartitionConfigLoadingException;

import com.twitter.search.earlybird.util.OneTaskScheduledExecutorManager;

import com.twitter.search.earlybird.util.PeriodicActionParams;

import com.twitter.search.earlybird.util.ShutdownWaitTimeParams;

/\*\*

\* A class that keeps track of Earlybird state that may change while an Earlybird runs, and keeps

\* that state up to date. Currently keeps track of the current Earlybird schema and partition

\* configuration, and periodically updates them from Zookeeper. It also reloads periodically the

\* scoring models from HDFS.

\*/

public class UpdateableEarlybirdStateManager extends OneTaskScheduledExecutorManager {

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

public static final String SCHEMA\_SUFFIX = ".schema.v";

private static final String THREAD\_NAME\_PATTERN = "state\_update-%d";

private static final boolean THREAD\_IS\_DAEMON = true;

private static final long EXECUTOR\_SHUTDOWN\_WAIT\_SEC = 5;

private static final String DEFAULT\_ZK\_SCHEMA\_LOCATION =

"/twitter/search/production/earlybird/schema";

private static final String DEFAULT\_LOCAL\_SCHEMA\_LOCATION =

"/home/search/earlybird\_schema\_canary";

private static final long DEFAULT\_UPDATE\_PERIOD\_MILLIS =

TimeUnit.MINUTES.toMillis(30);

private static final String SCHEMA\_MAJOR\_VERSION\_NAME =

"schema\_major\_version";

private static final String SCHEMA\_MINOR\_VERSION\_NAME =

"schema\_minor\_version";

private static final String LAST\_SUCCESSFUL\_SCHEMA\_RELOAD\_TIME\_MILLIS\_NAME =

"last\_successful\_schema\_reload\_timestamp\_millis";

@VisibleForTesting

static final String FAIL\_TO\_LOAD\_SCHEMA\_COUNT\_NAME =

"fail\_to\_load\_schema\_count";

@VisibleForTesting

static final String HOST\_IS\_CANARY\_SCHEME = "host\_is\_canary\_schema";

@VisibleForTesting

static final String DID\_NOT\_FIND\_SCHEMA\_COUNT\_NAME =

"did\_not\_find\_schema\_count";

private static final String LAST\_SUCCESSFUL\_PARTITION\_CONFIG\_RELOAD\_TIME\_MILLIS\_NAME =

"last\_successful\_partition\_config\_reload\_timestamp\_millis";

@VisibleForTesting

static final String FAIL\_TO\_LOAD\_PARTITION\_CONFIG\_COUNT\_NAME =

"fail\_to\_load\_partition\_config\_count";

@VisibleForTesting

static final String HOST\_IS\_IN\_LAYOUT\_STAT\_NAME = "host\_is\_in\_layout";

private static final String NOT\_IN\_LAYOUT\_SHUT\_DOWN\_ATTEMPTED\_NAME =

"not\_in\_layout\_shut\_down\_attempted";

private static final String SHUT\_DOWN\_EARLYBIRD\_WHEN\_NOT\_IN\_LAYOUT\_DECIDER\_KEY =

"shut\_down\_earlybird\_when\_not\_in\_layout";

private static final String NO\_SHUTDOWN\_WHEN\_NOT\_IN\_LAYOUT\_NAME =

"no\_shutdown\_when\_not\_in\_layout";

private final SearchLongGauge schemaMajorVersion;

private final SearchLongGauge schemaMinorVersion;

private final SearchLongGauge lastSuccessfulSchemaReloadTimeMillis;

private final SearchCounter failToLoadSchemaCount;

private final SearchLongGauge hostIsCanarySchema;

private final SearchCounter didNotFindSchemaCount;

private final SearchLongGauge lastSuccessfulPartitionConfigReloadTimeMillis;

private final SearchCounter failToLoadPartitionConfigCount;

private final SearchLongGauge hostIsInLayout;

private final SearchCounter notInLayoutShutDownAttemptedCount;

private final SearchLongGauge noShutdownWhenNotInLayoutGauge;

private final EarlybirdIndexConfig indexConfig;

private final DynamicPartitionConfig partitionConfig;

private final String schemaLocationOnLocal;

private final String schemaLocationOnZK;

private final ZooKeeperProxy zkClient;

private final AuroraSchedulerClient schedulerClient;

private final ScoringModelsManager scoringModelsManager;

private final TensorflowModelsManager tensorflowModelsManager;

private final SearchDecider searchDecider;

private final AtomicLong noShutdownWhenNotInLayout;

private EarlybirdServer earlybirdServer;

private Clock clock;

public UpdateableEarlybirdStateManager(

EarlybirdIndexConfig indexConfig,

DynamicPartitionConfig partitionConfig,

ZooKeeperProxy zooKeeperClient,

@Nullable AuroraSchedulerClient schedulerClient,

ScheduledExecutorServiceFactory executorServiceFactory,

ScoringModelsManager scoringModelsManager,

TensorflowModelsManager tensorflowModelsManager,

SearchStatsReceiver searchStatsReceiver,

SearchDecider searchDecider,

CriticalExceptionHandler criticalExceptionHandler,

Clock clock) {

this(

indexConfig,

partitionConfig,

DEFAULT\_LOCAL\_SCHEMA\_LOCATION,

DEFAULT\_ZK\_SCHEMA\_LOCATION,

DEFAULT\_UPDATE\_PERIOD\_MILLIS,

zooKeeperClient,

schedulerClient,

executorServiceFactory,

scoringModelsManager,

tensorflowModelsManager,

searchStatsReceiver,

searchDecider,

criticalExceptionHandler,

clock);

}

protected UpdateableEarlybirdStateManager(

EarlybirdIndexConfig indexConfig,

DynamicPartitionConfig partitionConfig,

String schemaLocationOnLocal,

String schemaLocationOnZK,

long updatePeriodMillis,

ZooKeeperProxy zkClient,

@Nullable AuroraSchedulerClient schedulerClient,

ScheduledExecutorServiceFactory executorServiceFactory,

ScoringModelsManager scoringModelsManager,

TensorflowModelsManager tensorflowModelsManager,

SearchStatsReceiver searchStatsReceiver,

SearchDecider searchDecider,

CriticalExceptionHandler criticalExceptionHandler,

Clock clock) {

super(

executorServiceFactory,

THREAD\_NAME\_PATTERN,

THREAD\_IS\_DAEMON,

PeriodicActionParams.withFixedDelay(

updatePeriodMillis,

TimeUnit.MILLISECONDS

),

new ShutdownWaitTimeParams(

EXECUTOR\_SHUTDOWN\_WAIT\_SEC,

TimeUnit.SECONDS

),

searchStatsReceiver,

criticalExceptionHandler);

this.indexConfig = indexConfig;

this.partitionConfig = partitionConfig;

this.schemaLocationOnLocal = schemaLocationOnLocal;

this.schemaLocationOnZK = schemaLocationOnZK;

this.zkClient = zkClient;

this.schedulerClient = schedulerClient;

this.scoringModelsManager = scoringModelsManager;

this.searchDecider = searchDecider;

this.noShutdownWhenNotInLayout = new AtomicLong(0);

this.tensorflowModelsManager = tensorflowModelsManager;

this.clock = clock;

this.schemaMajorVersion = getSearchStatsReceiver().getLongGauge(

SCHEMA\_MAJOR\_VERSION\_NAME);

this.schemaMinorVersion = getSearchStatsReceiver().getLongGauge(

SCHEMA\_MINOR\_VERSION\_NAME);

this.lastSuccessfulSchemaReloadTimeMillis = getSearchStatsReceiver().getLongGauge(

LAST\_SUCCESSFUL\_SCHEMA\_RELOAD\_TIME\_MILLIS\_NAME);

this.failToLoadSchemaCount = getSearchStatsReceiver().getCounter(

FAIL\_TO\_LOAD\_SCHEMA\_COUNT\_NAME);

this.hostIsCanarySchema = getSearchStatsReceiver().getLongGauge(HOST\_IS\_CANARY\_SCHEME);

this.didNotFindSchemaCount = getSearchStatsReceiver().getCounter(

DID\_NOT\_FIND\_SCHEMA\_COUNT\_NAME);

this.lastSuccessfulPartitionConfigReloadTimeMillis = getSearchStatsReceiver().getLongGauge(

LAST\_SUCCESSFUL\_PARTITION\_CONFIG\_RELOAD\_TIME\_MILLIS\_NAME);

this.failToLoadPartitionConfigCount = getSearchStatsReceiver().getCounter(

FAIL\_TO\_LOAD\_PARTITION\_CONFIG\_COUNT\_NAME);

this.hostIsInLayout = getSearchStatsReceiver().getLongGauge(

HOST\_IS\_IN\_LAYOUT\_STAT\_NAME);

this.notInLayoutShutDownAttemptedCount = getSearchStatsReceiver().getCounter(

NOT\_IN\_LAYOUT\_SHUT\_DOWN\_ATTEMPTED\_NAME);

this.noShutdownWhenNotInLayoutGauge = getSearchStatsReceiver().getLongGauge(

NO\_SHUTDOWN\_WHEN\_NOT\_IN\_LAYOUT\_NAME, noShutdownWhenNotInLayout);

updateSchemaVersionStats(indexConfig.getSchema());

}

private void updateSchemaVersionStats(Schema schema) {

schemaMajorVersion.set(schema.getMajorVersionNumber());

schemaMinorVersion.set(schema.getMinorVersionNumber());

lastSuccessfulSchemaReloadTimeMillis.set(System.currentTimeMillis());

lastSuccessfulPartitionConfigReloadTimeMillis.set(System.currentTimeMillis());

hostIsInLayout.set(1);

}

private void updateSchemaVersionWithThriftSchema(ThriftSchema thriftSchema)

throws Schema.SchemaValidationException, DynamicSchema.SchemaUpdateException {

ImmutableSchema newSchema = new ImmutableSchema(

thriftSchema, new AnalyzerFactory(), indexConfig.getCluster().getNameForStats());

indexConfig.getSchema().updateSchema(newSchema);

tensorflowModelsManager.updateFeatureSchemaIdToMlIdMap(newSchema.getSearchFeatureSchema());

updateSchemaVersionStats(indexConfig.getSchema());

LOG.info("Schema updated. New Schema is: \n" + ThriftUtils.toTextFormatSafe(thriftSchema));

}

protected void updateSchema(ZooKeeperProxy zkClientToUse) {

// There are 3 cases:

// 1. Try to locate local schema file to canary, it might fail either because file not exist or

// ineligible versions.

// 2. Canary local schema failed, lookup schema file from zookeeper.

// 3. Both local and zookeeper updates failed, we do not update schema. Either schema not exists

// in zookeeper, or this would happened after canary schema: we updated current schema but did

// not rollback after finished.

if (updateSchemaFromLocal()) {

LOG.info("Host is used for schema canary");

hostIsCanarySchema.set(1);

} else if (updateSchemaFromZooKeeper(zkClientToUse)) {

// Host is using schema file from zookeeper

hostIsCanarySchema.set(0);

} else {

// Schema update failed. Please check schema file exists on zookeeper and make sure

// rollback after canary. Current version: {}.{}

return;

}

}

private boolean updateSchemaFromLocal() {

ThriftSchema thriftSchema =

loadCanaryThriftSchemaFromLocal(getCanarySchemaFileOnLocal());

if (thriftSchema == null) {

// It is expected to not find a local schema file. The schema file only exists when the host

// is used as canary for schema updates

return false;

}

return updateSchemaFromThriftSchema(thriftSchema);

}

private boolean updateSchemaFromZooKeeper(ZooKeeperProxy zkClientToUse) {

ThriftSchema thriftSchema = loadThriftSchemaFromZooKeeper(zkClientToUse);

if (thriftSchema == null) {

// It is expected to usually not find a schema file on ZooKeeper; one is only uploaded if the

// schema changes after the package has been compiled. All the relevant error handling and

// logging is expected to be handled by loadThriftSchemaFromZooKeeper().

failToLoadSchemaCount.increment();

return false;

}

return updateSchemaFromThriftSchema(thriftSchema);

}

private boolean updateSchemaFromThriftSchema(ThriftSchema thriftSchema) {

Schema currentSchema = indexConfig.getSchema();

if (thriftSchema.getMajorVersionNumber() != currentSchema.getMajorVersionNumber()) {

LOG.warn(

"Major version updates are not allowed. Current major version {}, try to update to {}",

currentSchema.getMajorVersionNumber(), thriftSchema.getMajorVersionNumber());

return false;

}

if (thriftSchema.getMinorVersionNumber() > currentSchema.getMinorVersionNumber()) {

try {

updateSchemaVersionWithThriftSchema(thriftSchema);

} catch (Schema.SchemaValidationException | DynamicSchema.SchemaUpdateException e) {

LOG.warn("Exception while updating schema: ", e);

return false;

}

return true;

} else if (thriftSchema.getMinorVersionNumber() == currentSchema.getMinorVersionNumber()) {

LOG.info("Schema version to update is same as current one: {}.{}",

currentSchema.getMajorVersionNumber(), currentSchema.getMinorVersionNumber());

return true;

} else {

LOG.info("Found schema to update, but not eligible for dynamic update. "

+ "Current Version: {}.{}; Schema Version for updates: {}.{}",

currentSchema.getMajorVersionNumber(),

currentSchema.getMinorVersionNumber(),

thriftSchema.getMajorVersionNumber(),

thriftSchema.getMinorVersionNumber());

return false;

}

}

void updatePartitionConfig(@Nullable AuroraSchedulerClient schedulerClientToUse) {

try {

if (schedulerClientToUse == null) {

NonPagingAssert.assertFailed("aurora\_scheduler\_client\_is\_null");

throw new PartitionConfigLoadingException("AuroraSchedulerClient can not be null.");

}

PartitionConfig newPartitionConfig =

PartitionConfigLoader.getPartitionInfoForMesosConfig(schedulerClientToUse);

partitionConfig.setCurrentPartitionConfig(newPartitionConfig);

lastSuccessfulPartitionConfigReloadTimeMillis.set(System.currentTimeMillis());

hostIsInLayout.set(1);

} catch (PartitionConfigLoadingException e) {

// Do not change hostIsInLayout's value if we could not load the layout.

LOG.warn("Failed to load partition config from ZooKeeper.", e);

failToLoadPartitionConfigCount.increment();

}

}

@Nullable

private ThriftSchema loadCanaryThriftSchemaFromLocal(LocalFile schemaFile) {

String schemaString;

if (!schemaFile.getFile().exists()) {

return null;

}

try {

schemaString = schemaFile.getCharSource().read();

} catch (IOException e) {

LOG.warn("Fail to read from local schema file.");

return null;

}

ThriftSchema thriftSchema = new ThriftSchema();

try {

ThriftUtils.fromTextFormat(schemaString, thriftSchema);

return thriftSchema;

} catch (TException e) {

LOG.warn("Unable to deserialize ThriftSchema loaded locally from {}.\n{}",

schemaFile.getName(), e);

return null;

}

}

@Nullable

private ThriftSchema loadThriftSchemaFromZooKeeper(ZooKeeperProxy zkClientToUse) {

String schemaPathOnZk = getFullSchemaPathOnZK();

byte[] rawBytes;

try {

rawBytes = zkClientToUse.getData(schemaPathOnZk, false, null);

} catch (KeeperException.NoNodeException e) {

didNotFindSchemaCount.increment();

return null;

} catch (KeeperException e) {

LOG.warn("Exception while loading schema from ZK at {}.\n{}", schemaPathOnZk, e);

return null;

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

LOG.warn("Interrupted while loading schema from ZK at {}.\n{}", schemaPathOnZk, e);

return null;

} catch (ZooKeeperClient.ZooKeeperConnectionException e) {

LOG.warn("Exception while loading schema from ZK at {}.\n{}", schemaPathOnZk, e);

return null;

}

if (rawBytes == null) {

LOG.warn("Got null schema from ZooKeeper at {}.", schemaPathOnZk);

return null;

}

String schemaString = new String(rawBytes, Charsets.UTF\_8);

ThriftSchema thriftSchema = new ThriftSchema();

try {

ThriftUtils.fromTextFormat(schemaString, thriftSchema);

return thriftSchema;

} catch (TException e) {

LOG.warn("Unable to deserialize ThriftSchema loaded from ZK at {}.\n{}", schemaPathOnZk, e);

return null;

}

}

@VisibleForTesting

protected String getSchemaFileName() {

return indexConfig.getCluster().name().toLowerCase()

+ UpdateableEarlybirdStateManager.SCHEMA\_SUFFIX

+ indexConfig.getSchema().getMajorVersionNumber();

}

@VisibleForTesting

protected String getFullSchemaPathOnZK() {

return String.format("%s/%s", schemaLocationOnZK, getSchemaFileName());

}

LocalFile getCanarySchemaFileOnLocal() {

String canarySchemaFilePath =

String.format("%s/%s", schemaLocationOnLocal, getSchemaFileName());

return new LocalFile(new File(canarySchemaFilePath));

}

void setNoShutdownWhenNotInLayout(boolean noShutdown) {

noShutdownWhenNotInLayout.set(noShutdown ? 1 : 0);

}

@Override

protected void runOneIteration() {

updateSchema(zkClient);

updatePartitionConfig(schedulerClient);

LOG.info("Reloading models.");

scoringModelsManager.reload();

tensorflowModelsManager.run();

Random random = new Random();

try {

// We had an issue where HDFS operations were blocking, so reloading these models

// was finishing at the same time on each instance and after that every time an instance

// was reloading models, it was happening at the same time. This caused issues with HDFS

// load. We now place a "guard" waiting time after each reload so that the execution time

// on every instance is different and these calls can't easily sync to the same point in time.

int sleepSeconds = random.nextInt(30 \* 60);

LOG.info("Sleeping for {} seconds", sleepSeconds);

clock.waitFor(sleepSeconds \* 1000);

} catch (InterruptedException ex) {

LOG.info("Interrupted while sleeping");

}

}

public void setEarlybirdServer(EarlybirdServer earlybirdServer) {

this.earlybirdServer = earlybirdServer;

}

}