package com.twitter.search.earlybird;

import java.net.InetAddress;

import java.net.InetSocketAddress;

import java.util.concurrent.atomic.AtomicLong;

import javax.annotation.concurrent.GuardedBy;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Maps;

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.Watcher;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.zookeeper.ServerSet;

import com.twitter.common.zookeeper.ZooKeeperClient;

import com.twitter.common\_internal.zookeeper.TwitterServerSet;

import com.twitter.search.common.config.Config;

import com.twitter.search.common.database.DatabaseConfig;

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.util.zookeeper.ZooKeeperProxy;

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

import com.twitter.search.earlybird.common.config.EarlybirdProperty;

import com.twitter.search.earlybird.config.TierConfig;

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

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

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

public class EarlybirdServerSetManager implements ServerSetMember {

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

// How many times this earlybird joined/left its partition's server set

@VisibleForTesting

protected final SearchCounter leaveServerSetCounter;

@VisibleForTesting

protected final SearchCounter joinServerSetCounter;

private final ZooKeeperProxy discoveryZKClient;

private final SearchLongGauge inServerSetGauge;

private final PartitionConfig partitionConfig;

private final int port;

private final String serverSetNamePrefix;

@VisibleForTesting

protected final SearchLongGauge connectedToZooKeeper;

private final Object endpointStatusLock = new Object();

@GuardedBy("endpointStatusLock")

private ServerSet.EndpointStatus endpointStatus = null;

private boolean inServerSetForServiceProxy = false;

public EarlybirdServerSetManager(

SearchStatsReceiver searchStatsReceiver,

ZooKeeperProxy discoveryZKClient,

final PartitionConfig partitionConfig,

int port,

String serverSetNamePrefix) {

this.discoveryZKClient = discoveryZKClient;

this.partitionConfig = partitionConfig;

this.port = port;

this.serverSetNamePrefix = serverSetNamePrefix;

// Export serverset related stats

Preconditions.checkNotNull(searchStatsReceiver);

this.joinServerSetCounter = searchStatsReceiver.getCounter(

serverSetNamePrefix + "join\_server\_set\_count");

this.leaveServerSetCounter = searchStatsReceiver.getCounter(

serverSetNamePrefix + "leave\_server\_set\_count");

// Create a new stat based on the partition number for hosts-in-partition aggregation.

// The value of the stat is dependent on whether the server is in the serverset so that the

// aggregate stat reflects the number serving traffic instead of the live process count.

AtomicLong sharedInServerSetStatus = new AtomicLong();

this.inServerSetGauge = searchStatsReceiver.getLongGauge(

serverSetNamePrefix + "is\_in\_server\_set", sharedInServerSetStatus);

this.connectedToZooKeeper = searchStatsReceiver.getLongGauge(

serverSetNamePrefix + "connected\_to\_zookeeper");

searchStatsReceiver.getLongGauge(

serverSetNamePrefix + "member\_of\_partition\_" + partitionConfig.getIndexingHashPartitionID(),

sharedInServerSetStatus);

this.discoveryZKClient.registerExpirationHandler(() -> connectedToZooKeeper.set(0));

this.discoveryZKClient.register(event -> {

if (event.getType() == Watcher.Event.EventType.None

&& event.getState() == Watcher.Event.KeeperState.SyncConnected) {

connectedToZooKeeper.set(1);

}

});

}

/\*\*

\* Join ServerSet and update endpointStatus.

\* This will allow Earlybird consumers, e.g. Blender, to detect when an

\* Earlybird goes online and offline.

\* @param username

\*/

@Override

public void joinServerSet(String username) throws ServerSet.UpdateException {

joinServerSetCounter.increment();

synchronized (endpointStatusLock) {

LOG.info("Joining {} ServerSet (instructed by: {}) ...", serverSetNamePrefix, username);

if (endpointStatus != null) {

LOG.warn("Already in ServerSet. Nothing done.");

throw new AlreadyInServerSetUpdateException("Already in ServerSet. Nothing done.");

}

try {

TwitterServerSet.Service service = getServerSetService();

ServerSet serverSet = discoveryZKClient.createServerSet(service);

endpointStatus = serverSet.join(

new InetSocketAddress(InetAddress.getLocalHost().getHostName(), port),

Maps.newHashMap(),

partitionConfig.getHostPositionWithinHashPartition());

inServerSetGauge.set(1);

String path = service.getPath();

EarlybirdStatus.recordEarlybirdEvent("Joined " + serverSetNamePrefix + " ServerSet " + path

+ " (instructed by: " + username + ")");

LOG.info("Successfully joined {} ServerSet {} (instructed by: {})",

serverSetNamePrefix, path, username);

} catch (Exception e) {

endpointStatus = null;

String message = "Failed to join " + serverSetNamePrefix + " ServerSet of partition "

+ partitionConfig.getIndexingHashPartitionID();

LOG.error(message, e);

throw new ServerSet.UpdateException(message, e);

}

}

}

/\*\*

\* Takes this Earlybird out of its registered ServerSet.

\*

\* @throws ServerSet.UpdateException if there was a problem leaving the ServerSet,

\* or if this Earlybird is already not in a ServerSet.

\* @param username

\*/

@Override

public void leaveServerSet(String username) throws ServerSet.UpdateException {

leaveServerSetCounter.increment();

synchronized (endpointStatusLock) {

LOG.info("Leaving {} ServerSet (instructed by: {}) ...", serverSetNamePrefix, username);

if (endpointStatus == null) {

String message = "Not in a ServerSet. Nothing done.";

LOG.warn(message);

throw new NotInServerSetUpdateException(message);

}

endpointStatus.leave();

endpointStatus = null;

inServerSetGauge.set(0);

EarlybirdStatus.recordEarlybirdEvent("Left " + serverSetNamePrefix

+ " ServerSet (instructed by: " + username + ")");

LOG.info("Successfully left {} ServerSet. (instructed by: {})",

serverSetNamePrefix, username);

}

}

@Override

public int getNumberOfServerSetMembers()

throws InterruptedException, ZooKeeperClient.ZooKeeperConnectionException, KeeperException {

String path = getServerSetService().getPath();

return discoveryZKClient.getNumberOfServerSetMembers(path);

}

/\*\*

\* Determines if this earlybird is in the server set.

\*/

@Override

public boolean isInServerSet() {

synchronized (endpointStatusLock) {

return endpointStatus != null;

}

}

/\*\*

\* Returns the server set that this earlybird should join.

\*/

public String getServerSetIdentifier() {

TwitterServerSet.Service service = getServerSetService();

return String.format("/cluster/local/%s/%s/%s",

service.getRole(),

service.getEnv(),

service.getName());

}

private TwitterServerSet.Service getServerSetService() {

// If the tier name is 'all' then it treat it as an untiered EB cluster

// and do not add the tier component into the ZK path it registers under.

String tierZKPathComponent = "";

if (!TierConfig.DEFAULT\_TIER\_NAME.equalsIgnoreCase(partitionConfig.getTierName())) {

tierZKPathComponent = "/" + partitionConfig.getTierName();

}

if (EarlybirdConfig.isAurora()) {

// ROLE, EARYLBIRD\_NAME, and ENV properties are required on Aurora, thus will be set here

return new TwitterServerSet.Service(

EarlybirdProperty.ROLE.get(),

EarlybirdProperty.ENV.get(),

getServerSetPath(EarlybirdProperty.EARLYBIRD\_NAME.get() + tierZKPathComponent));

} else {

return new TwitterServerSet.Service(

DatabaseConfig.getZooKeeperRole(),

Config.getEnvironment(),

getServerSetPath("earlybird" + tierZKPathComponent));

}

}

private String getServerSetPath(String earlybirdName) {

return String.format("%s%s/hash\_partition\_%d", serverSetNamePrefix, earlybirdName,

partitionConfig.getIndexingHashPartitionID());

}

/\*\*

\* Join ServerSet for ServiceProxy with a named admin port and with a zookeeper path that Service

\* Proxy can translate to a domain name label that is less than 64 characters (due to the size

\* limit for domain name labels described here: https://tools.ietf.org/html/rfc1035)

\* This will allow us to access Earlybirds that are not on mesos via ServiceProxy.

\*/

@Override

public void joinServerSetForServiceProxy() {

// This additional Zookeeper server set is only necessary for Archive Earlybirds which are

// running on bare metal hardware, so ensure that this method is never called for services

// on Aurora.

Preconditions.checkArgument(!EarlybirdConfig.isAurora(),

"Attempting to join server set for ServiceProxy on Earlybird running on Aurora");

LOG.info("Attempting to join ServerSet for ServiceProxy");

try {

TwitterServerSet.Service service = getServerSetForServiceProxyOnArchive();

ServerSet serverSet = discoveryZKClient.createServerSet(service);

String hostName = InetAddress.getLocalHost().getHostName();

int adminPort = EarlybirdConfig.getAdminPort();

serverSet.join(

new InetSocketAddress(hostName, port),

ImmutableMap.of("admin", new InetSocketAddress(hostName, adminPort)),

partitionConfig.getHostPositionWithinHashPartition());

String path = service.getPath();

LOG.info("Successfully joined ServerSet for ServiceProxy {}", path);

inServerSetForServiceProxy = true;

} catch (Exception e) {

String message = "Failed to join ServerSet for ServiceProxy of partition "

+ partitionConfig.getIndexingHashPartitionID();

LOG.warn(message, e);

}

}

@VisibleForTesting

protected TwitterServerSet.Service getServerSetForServiceProxyOnArchive() {

String serverSetPath = String.format("proxy/%s/p\_%d",

partitionConfig.getTierName(),

partitionConfig.getIndexingHashPartitionID());

return new TwitterServerSet.Service(

DatabaseConfig.getZooKeeperRole(),

Config.getEnvironment(),

serverSetPath);

}

@VisibleForTesting

protected boolean isInServerSetForServiceProxy() {

return inServerSetForServiceProxy;

}

}