package com.twitter.search.earlybird\_root;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.finagle.Service;

import com.twitter.finagle.SimpleFilter;

import com.twitter.finagle.client.BackupRequestFilter;

import com.twitter.finagle.service.ResponseClassifier;

import com.twitter.finagle.service.RetryBudgets;

import com.twitter.finagle.stats.StatsReceiver;

import com.twitter.finagle.util.DefaultTimer;

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

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

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

import com.twitter.search.earlybird.thrift.EarlybirdRequest;

import com.twitter.search.earlybird.thrift.EarlybirdResponse;

import com.twitter.util.Future;

import com.twitter.util.tunable.Tunable;

public class ClientBackupFilter extends SimpleFilter<EarlybirdRequest, EarlybirdResponse> {

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

private final Map<String, BackupRequestFilter<EarlybirdRequest, EarlybirdResponse>>

clientBackupFilters = new ConcurrentHashMap<>();

private final boolean sendInterupts = false;

private final String statPrefix;

private final Tunable.Mutable<Object> maxExtraLoad;

private final StatsReceiver statsReceiver;

private final SearchDecider decider;

private final String backupRequestPrecentExtraLoadDecider;

private final int minSendBackupAfterMs = 1;

public ClientBackupFilter(String serviceName,

String statPrefix,

StatsReceiver statsReceiver,

SearchDecider decider) {

this.statPrefix = statPrefix;

this.backupRequestPrecentExtraLoadDecider = serviceName + "\_backup\_request\_percent\_extra\_load";

this.decider = decider;

this.maxExtraLoad = Tunable.mutable("backup\_tunable", getMaxExtraLoadFromDecider());

this.statsReceiver = statsReceiver;

SearchCustomGauge.export(serviceName + "\_backup\_request\_factor",

() -> (maxExtraLoad.apply().isDefined()) ? (double) maxExtraLoad.apply().get() : -1);

}

private double getMaxExtraLoadFromDecider() {

return ((double) decider.getAvailability(backupRequestPrecentExtraLoadDecider)) / 100 / 100;

}

private BackupRequestFilter<EarlybirdRequest, EarlybirdResponse> backupFilter(String client) {

return new BackupRequestFilter<EarlybirdRequest, EarlybirdResponse>(

maxExtraLoad,

sendInterupts,

minSendBackupAfterMs,

ResponseClassifier.Default(),

RetryBudgets.newRetryBudget(),

statsReceiver.scope(statPrefix, client, "backup\_filter"),

DefaultTimer.getInstance(),

client);

}

private void updateMaxExtraLoadIfNecessary() {

double maxExtraLoadDeciderValue = getMaxExtraLoadFromDecider();

if (maxExtraLoad.apply().isDefined()

&& !maxExtraLoad.apply().get().equals(maxExtraLoadDeciderValue)) {

LOG.info("Updating maxExtraLoad from {} to {}",

maxExtraLoad.apply().get(),

maxExtraLoadDeciderValue);

maxExtraLoad.set(maxExtraLoadDeciderValue);

}

}

@Override

public Future<EarlybirdResponse> apply(EarlybirdRequest request,

Service<EarlybirdRequest, EarlybirdResponse> service) {

updateMaxExtraLoadIfNecessary();

String clientID = ClientIdUtil.getClientIdFromRequest(request);

BackupRequestFilter<EarlybirdRequest, EarlybirdResponse> filter =

clientBackupFilters.computeIfAbsent(clientID, this::backupFilter);

return filter

.andThen(service)

.apply(request);

}

}