package com.twitter.search.earlybird;

import java.util.concurrent.TimeUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.collect.Lists;

import org.apache.thrift.protocol.TCompactProtocol;

import com.twitter.finagle.ThriftMux;

import com.twitter.finagle.builder.ClientBuilder;

import com.twitter.finagle.builder.ClientConfig.Yes;

import com.twitter.finagle.mtls.client.MtlsThriftMuxClient;

import com.twitter.finagle.stats.StatsReceiver;

import com.twitter.finagle.thrift.ClientId;

import com.twitter.finagle.thrift.ThriftClientRequest;

import com.twitter.finagle.zipkin.thrift.ZipkinTracer;

import com.twitter.search.common.dark.DarkProxy;

import com.twitter.search.common.dark.ResolverProxy;

import com.twitter.search.common.dark.ServerSetResolver;

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

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

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

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

import com.twitter.util.Duration;

public class EarlybirdDarkProxy {

private static final String WARM\_UP\_DECIDER\_KEY\_PREFIX = "warmup\_";

private static final int DARK\_REQUESTS\_TOTAL\_REQUEST\_TIMEOUT\_MS =

EarlybirdConfig.getInt("dark\_requests\_total\_request\_timeout\_ms", 800);

private static final int DARK\_REQUESTS\_INDIVIDUAL\_REQUEST\_TIMEOUT\_MS =

EarlybirdConfig.getInt("dark\_requests\_individual\_request\_timeout\_ms", 800);

private static final int DARK\_REQUESTS\_CONNECT\_TIMEOUT\_MS =

EarlybirdConfig.getInt("dark\_requests\_connect\_timeout\_ms", 500);

private static final int DARK\_REQUESTS\_NUM\_RETRIES =

EarlybirdConfig.getInt("dark\_requests\_num\_retries", 1);

private static final String DARK\_REQUESTS\_FINAGLE\_CLIENT\_ID =

EarlybirdConfig.getString("dark\_requests\_finagle\_client\_id", "earlybird\_warmup");

private final DarkProxy<ThriftClientRequest, byte[]> darkProxy;

public EarlybirdDarkProxy(SearchDecider searchDecider,

StatsReceiver statsReceiver,

EarlybirdServerSetManager earlybirdServerSetManager,

EarlybirdWarmUpManager earlybirdWarmUpManager,

String clusterName) {

darkProxy = newDarkProxy(searchDecider,

statsReceiver,

earlybirdServerSetManager,

earlybirdWarmUpManager,

clusterName);

}

public DarkProxy<ThriftClientRequest, byte[]> getDarkProxy() {

return darkProxy;

}

@VisibleForTesting

protected DarkProxy<ThriftClientRequest, byte[]> newDarkProxy(

SearchDecider searchDecider,

StatsReceiver statsReceiver,

EarlybirdServerSetManager earlybirdServerSetManager,

final EarlybirdWarmUpManager earlybirdWarmUpManager,

String clusterName) {

ResolverProxy resolverProxy = new ResolverProxy();

ServerSetResolver.SelfServerSetResolver selfServerSetResolver =

new ServerSetResolver.SelfServerSetResolver(

earlybirdServerSetManager.getServerSetIdentifier(), resolverProxy);

selfServerSetResolver.init();

final String clusterNameForDeciderKey = clusterName.toLowerCase().replaceAll("-", "\_");

final String warmUpServerSetIdentifier = earlybirdWarmUpManager.getServerSetIdentifier();

DarkProxy newDarkProxy = new DarkProxy<ThriftClientRequest, byte[]>(

selfServerSetResolver,

newClientBuilder(statsReceiver),

resolverProxy,

searchDecider,

Lists.newArrayList(warmUpServerSetIdentifier),

new BytesToThriftFilter(),

statsReceiver) {

@Override

protected String getServicePathDeciderKey(String servicePath) {

if (warmUpServerSetIdentifier.equals(servicePath)) {

return WARM\_UP\_DECIDER\_KEY\_PREFIX + clusterNameForDeciderKey;

}

return clusterNameForDeciderKey;

}

};

newDarkProxy.init();

return newDarkProxy;

}

private ClientBuilder<ThriftClientRequest, byte[], ?, Yes, Yes> newClientBuilder(

StatsReceiver statsReceiver) {

return ClientBuilder.get()

.daemon(true)

.timeout(Duration.apply(DARK\_REQUESTS\_TOTAL\_REQUEST\_TIMEOUT\_MS, TimeUnit.MILLISECONDS))

.requestTimeout(

Duration.apply(DARK\_REQUESTS\_INDIVIDUAL\_REQUEST\_TIMEOUT\_MS, TimeUnit.MILLISECONDS))

.tcpConnectTimeout(Duration.apply(DARK\_REQUESTS\_CONNECT\_TIMEOUT\_MS, TimeUnit.MILLISECONDS))

.retries(DARK\_REQUESTS\_NUM\_RETRIES)

.reportTo(statsReceiver)

.tracer(ZipkinTracer.mk(statsReceiver))

.stack(new MtlsThriftMuxClient(

ThriftMux.client())

.withMutualTls(EarlybirdProperty.getServiceIdentifier())

.withProtocolFactory(new TCompactProtocol.Factory())

.withClientId(new ClientId(DARK\_REQUESTS\_FINAGLE\_CLIENT\_ID)));

}

}