package com.twitter.search.feature\_update\_service.modules;

import java.util.concurrent.LinkedBlockingQueue;

import java.util.concurrent.ThreadPoolExecutor;

import java.util.concurrent.TimeUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.inject.Provides;

import com.google.inject.Singleton;

import com.twitter.inject.TwitterModule;

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

import com.twitter.search.feature\_update\_service.stats.FeatureUpdateStats;

import com.twitter.util.ExecutorServiceFuturePool;

import com.twitter.util.InterruptibleExecutorServiceFuturePool;

public class FuturePoolModule extends TwitterModule {

/\*\*

\* Provide future pool backed by executor service, with bounded thread pool and bounded backing

\* queue.

\*/

@Provides

@Singleton

public ExecutorServiceFuturePool futurePool() {

// These limits are based on service capacity estimates and testing on staging,

// attempting to give the pool as many resources as possible without overloading anything.

// 100-200 threads is manageable, and the 2000 queue size is based on a conservative upper

// limit that tasks in the queue take 1 MB each, meaning queue maxes out at 2 GB, which should

// be okay given 4 GB RAM with 3 GB reserved heap.

return createFuturePool(100, 200, 2000);

}

/\*\*

\* Create a future pool backed by executor service, with bounded thread pool and bounded backing

\* queue. ONLY VISIBILE FOR TESTING; don't invoke outside this class.

\*/

@VisibleForTesting

public static ExecutorServiceFuturePool createFuturePool(

int corePoolSize, int maximumPoolSize, int queueCapacity) {

final LinkedBlockingQueue<Runnable> queue = new LinkedBlockingQueue<>(queueCapacity);

ExecutorServiceFuturePool futurePool = new InterruptibleExecutorServiceFuturePool(

new ThreadPoolExecutor(

corePoolSize,

maximumPoolSize,

60L,

TimeUnit.SECONDS,

queue));

SearchCustomGauge.export(FeatureUpdateStats.PREFIX + "thread\_pool\_size",

futurePool::poolSize);

SearchCustomGauge.export(FeatureUpdateStats.PREFIX + "work\_queue\_size",

queue::size);

return futurePool;

}

}