Performance Engineers – David Quillen, Kristoffer Spencer

Hypothesis -

Nothing is wrong.

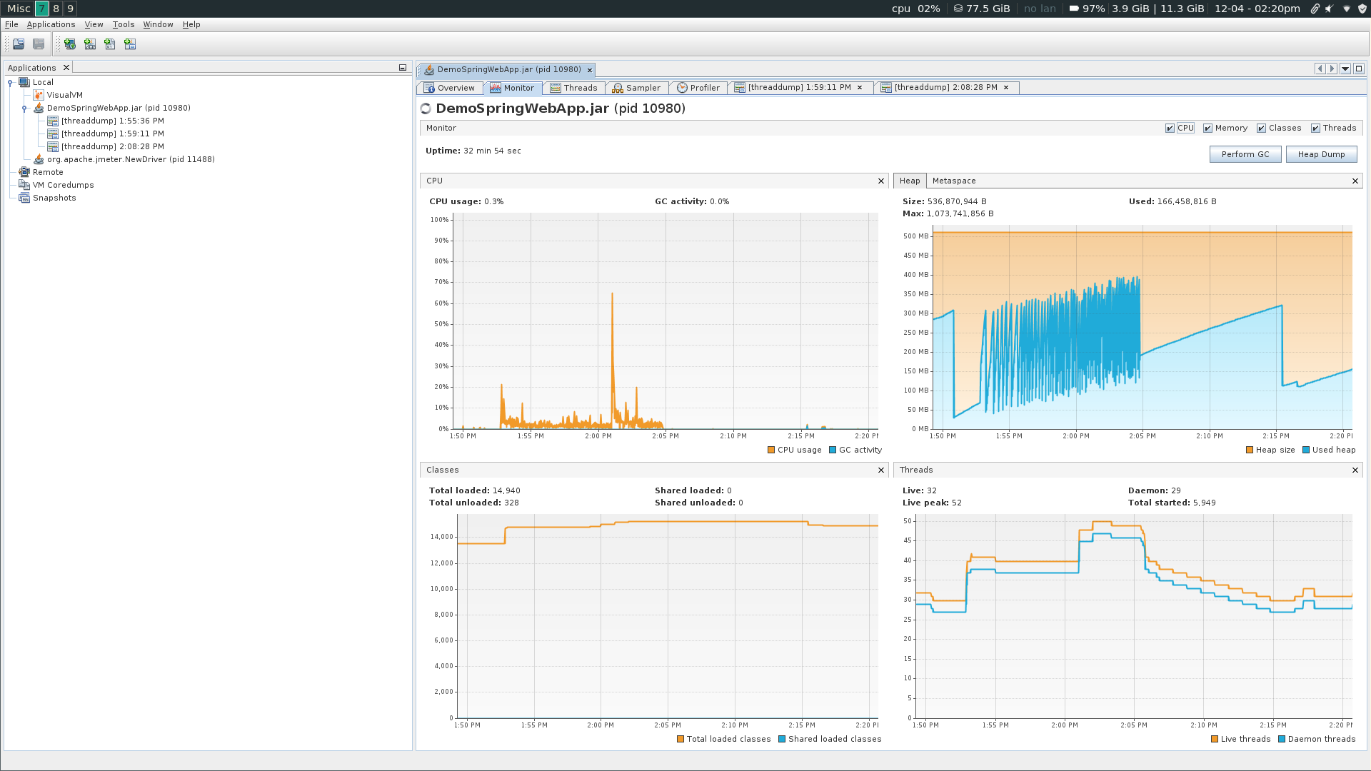
Suggestion -

run tests to ensure everything is fine.

Verify -

Idling no tests, no HTTP requests

* already have a garbage collection 300MB to 25ish MB, heap use is already going back up
* 36 threads started, 29 live, 13 threads on monitor status, we believe most of this is the connection pool



ran all paths test

* Heap use spiked up, GCs every 30 seconds and going down
* Deadlock detected immediately by JVisualVM
* threads are being spawned and killed immediately
* spawned up over 1000 threads
* classes loaded had initial increase on start of test and has not significantly changed
* MetaSpace similar behavior to classes loaded

Canceled test

* threads are dying slowly
* slow climb in the heap has gotten much sharper -- probable memory leak
* Thread dump has large number of DarkMode threads

Conclusion - We were wrong. We should look in DarkModeServiceImpl for SupportDarkModeThread and EnableDarkModeThread.

Hypothesis -

inside DarkMode Service is a loop spawning threads leading to unnecessary memory usage causing increased garbage collection

Suggestion -

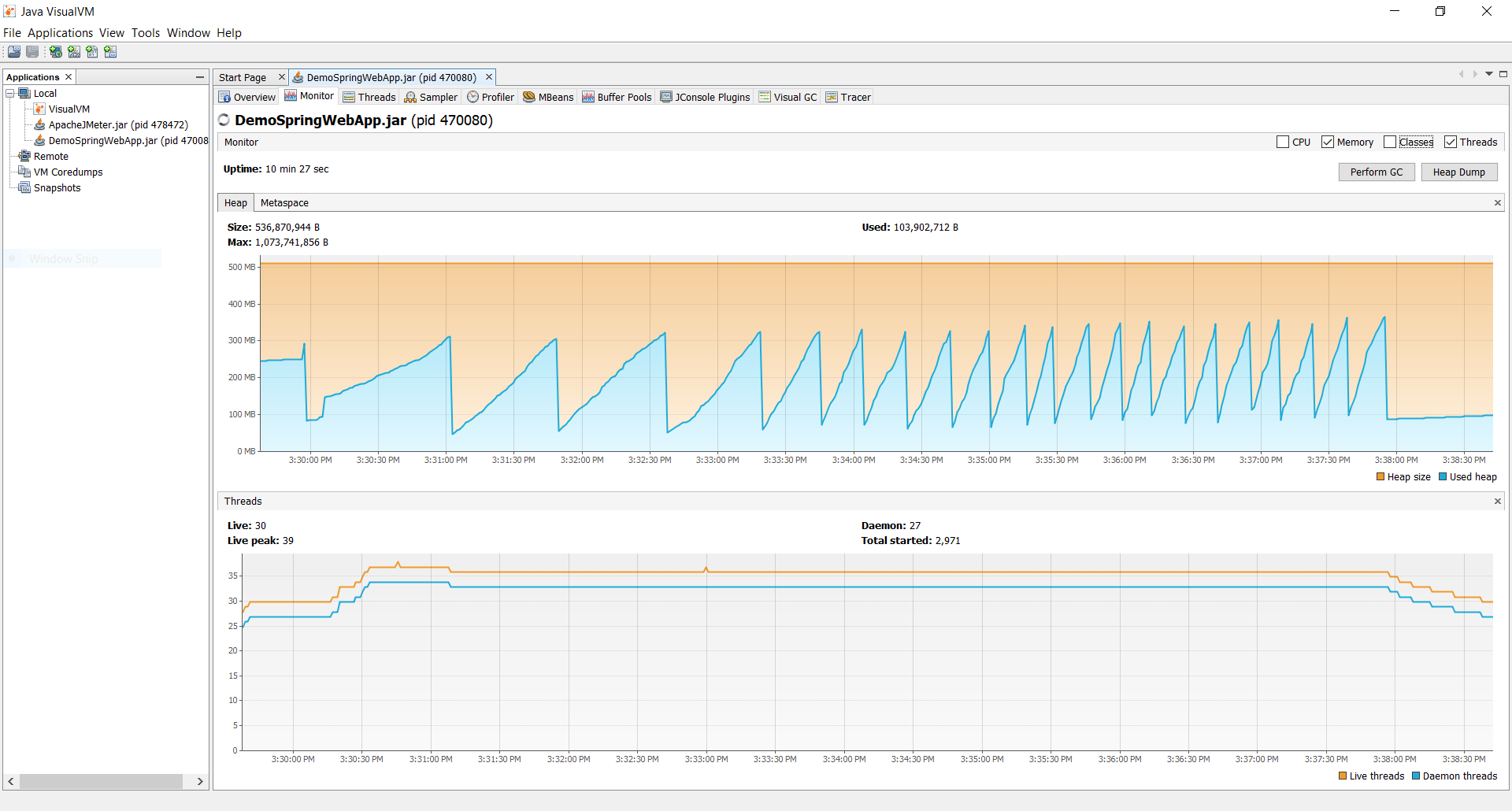
Checkout DarkModeServiceImpl for unnecessary loops and move thread creation outside of loop if possible

Verify -

checked out DarkModeServiceImpl commented out all implementation code in DarkModeServiceImpl,

looking over code, class spawns two threads, deadlocks both and does nothing.

idling looks the same



Starting test

garbage collection much less frequent but becoming quicker as we continue running the test,

far fewer live threads, still too many getting created and immediately killed

Tests Done

program resumed behavior as before tests

Conclusion -

not a loop, just our test hitting the EnableDarkMode button a lot.

better throughput because of lessened frequency of garbage collection

Code

package com.revature.service;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Service;

@Service

public class DarkModeServiceImpl implements DarkModeService {

    /\*private final Object lock1 = new Object();

    private final Object lock2 = new Object();

    \*/

    @Override

    public void enableDarkMode() {

        //enableDarkModeAsynchronously();

    }

    /\*

    private void enableDarkModeAsynchronously() {

        new Thread(() -> {

            Thread t1 = new EnableDarkModeThread();

            Thread t2 = new SupportDarkModeThread();

            t1.start();

            t2.start();

        }).start();

    }

    private class EnableDarkModeThread extends Thread {

        private final Logger logger = LoggerFactory.getLogger(getClass());

        @Override

        public void run() {

            synchronized(lock1) {

                logger.info("Enabling Dark Mode...");

                try {

                    Thread.sleep(10);

                } catch (InterruptedException e) {

                }

                logger.info("Waiting for second lock...");

                synchronized(lock2) {

                    logger.info("Dark Mode Enabled");

                }

            }

        }

    }

    private class SupportDarkModeThread extends Thread {

        private final Logger logger = LoggerFactory.getLogger(getClass());

        @Override

        public void run() {

            synchronized(lock2) {

                logger.info("Attempting to enable Dark Mode support...");

                try {

                    Thread.sleep(10);

                } catch (InterruptedException e) {

                }

                logger.info("Waiting for second lock...");

                synchronized(lock1) {

                    logger.info("Dark Mode Enabled");

                }

            }

        }

    }

    \*/

    public static void main(String[] args) {

        //new DarkModeServiceImpl().enableDarkMode();

    }

}

Hypothesis -

http-nio threads are dying and being remade, we are losing http-nio threads slowing down our application.

Suggestion -

Check thread pool to see if threads are getting returned properly

Verify

recorded flight from JMC, examined threads, http-nio threads are tomcat threads. There is nothing we can do here.

Hypothesis -

Threads are being created and killed immediately, this seems like unnecessary memory allocation and an opening for a memory leak.

Suggestion -

We should run each test separately to figure out where the threads are being created.

Verify -

The threads are redirects; no change necessary.

Hypothesis -

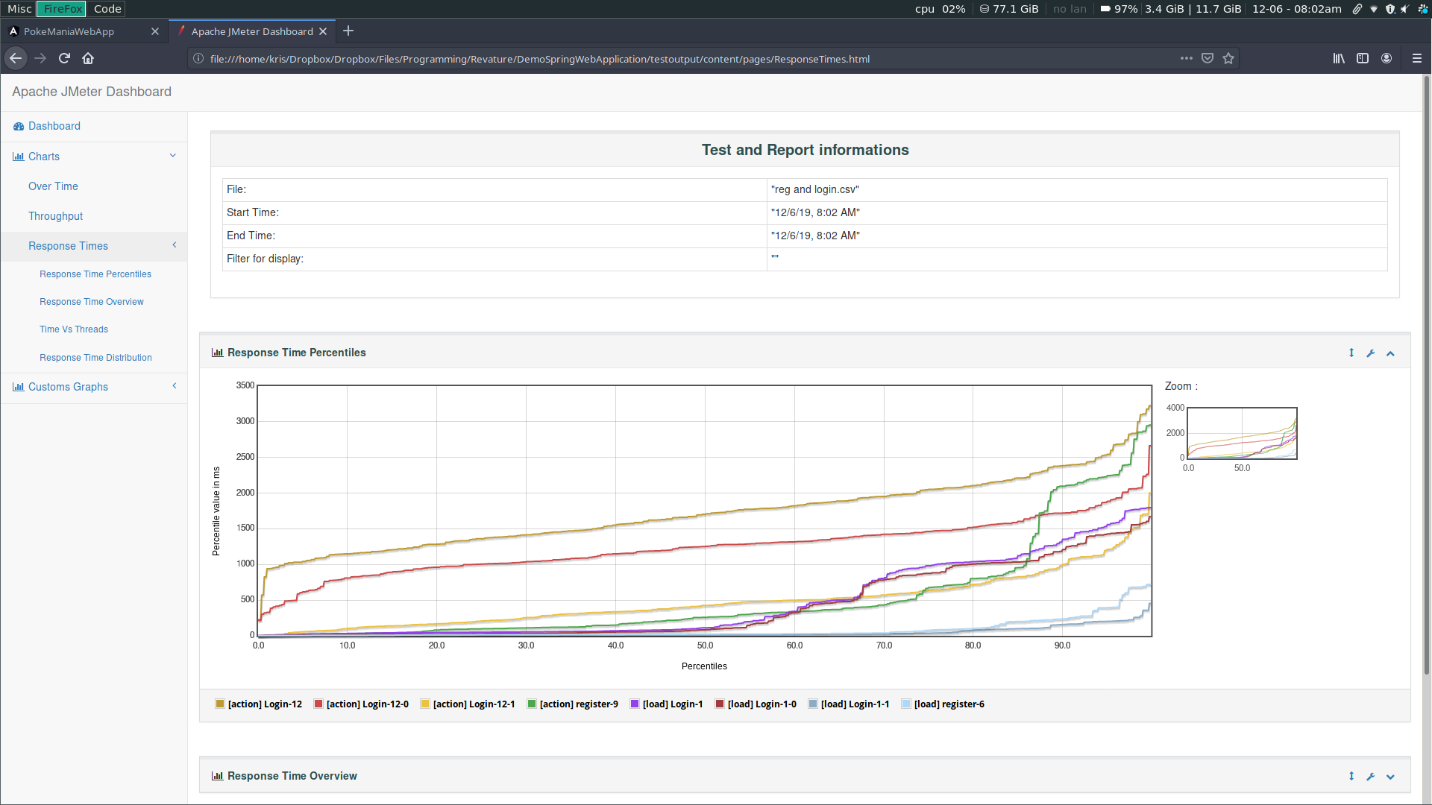
JMC flight uncovered possible ConnectionPool being remade each time it’s called in TodoServiceImpl.FetchCurrentUserTodoTask.call()

Suggestion -

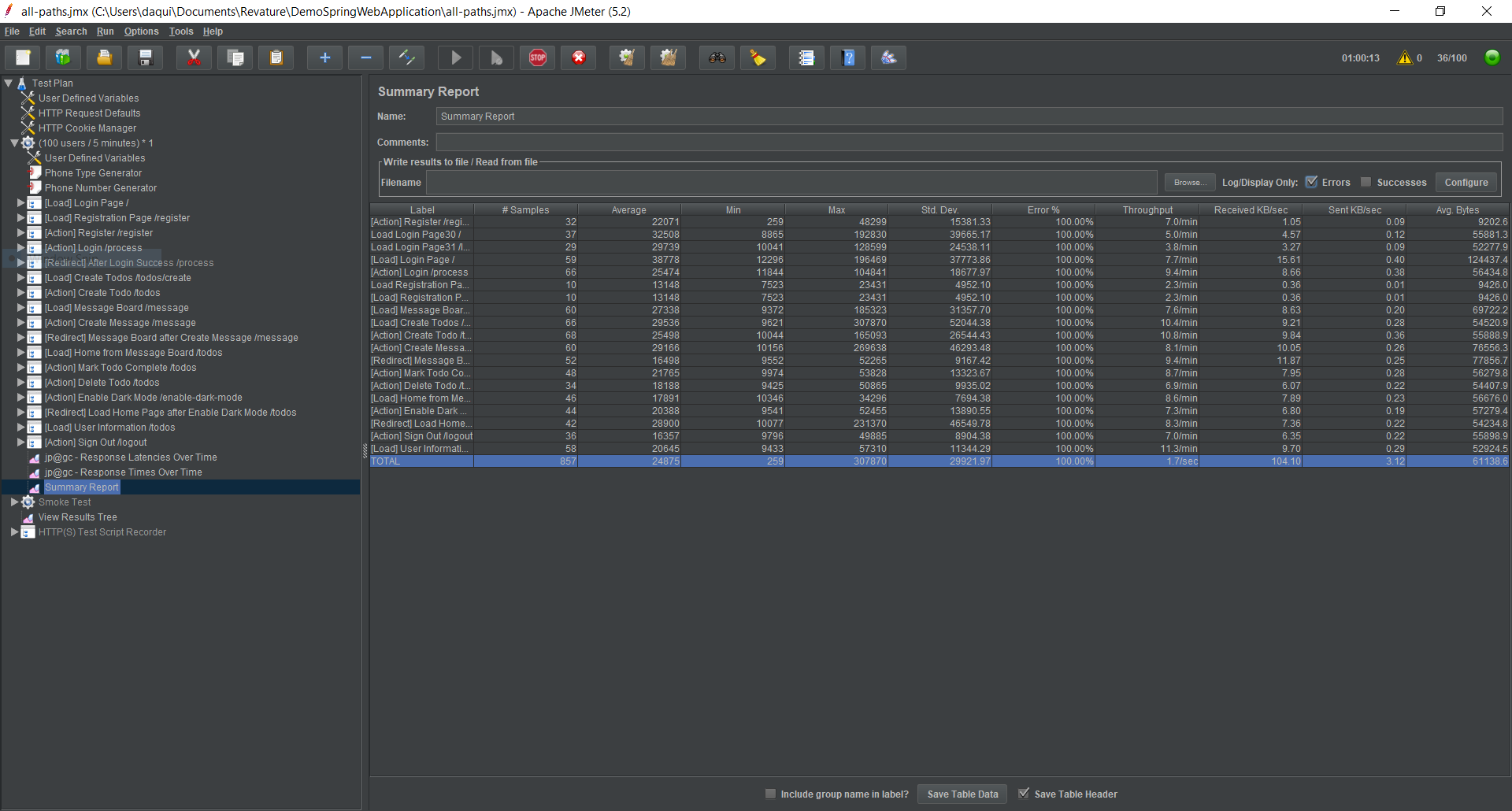
check TodoServiceImpl for a wrong implementation, commented out thread pool creation.

Verify -

testing showed that commenting out the thread pool slowed down the loading of the homepage by about 0.5 seconds average.



can't tell what the TodoServiceImpl is doing, suggest that the code writer rewrites the code.

Removed any changes to the code.  
  


This is our most recent run’s statistics.