Dynamic Request Scheduling Using Affinity in OS for multi-core Web Servers

COEN283 TERM PROJECT

Prajakta Patil

Rachana Deolikar

Shruti Saxena

Preface

Owing to more and more complex web applications and services, web performance often depends on server delays. This makes the performance of web services increasingly important. A web server requires some mechanism for the scheduling of requests, in order to reduce the mean response time and avoid bottlenecks. Our approach focuses on a scheduling algorithm which helps improve the mean service time of requests being handled in the web server by using affinity in operating systems.

Acknowledgements

We would like to thank Dr. Ming-Hwa Wang for granting us with the opportunity to work together on this project and for providing us with an environment to showcase our work and findings with our peers and classmates.

TABLE OF CONTENTS

1. Abstract

Any popular website is expected to deal with a large amount of requests simultaneously in a reasonable mean response time. The performance of web servers depends on the hardware performance and the processing strategy of requests. There are two kinds of requests at a web server i.e. static and dynamic. The processing of dynamic requests is more complex than static. This paper deals more with handling dynamic requests. The effective utilization of multiple cores without causing unnecessary delays is a crucial task. In this paper we deal with a scheduling algorithm that takes into account the multi core architecture of the web server and accordingly schedules requests on different cores.

2. INTRODUCTION

2.1 OBJECTIVE

The importance of utilizing multiple cores in a webserver is key to achieving an efficient web server with a quick response time. The objective in our paper is to efficiently use the multiple cores through multithreading and core affinity. We intend to implement a modified version of the Dynamic Request Scheduling Model based on the paper on Dynamic Requests Scheduling Model in Multi-Core Web Servers.

2.2 PROBLEM

With the advent of multiple core systems, web servers have begun to adopt CPU's with multiple cores in order to increase their performance. In such a web server if there are multiple threads the operating system usually assigns them to different cores to achieve this betterment in performance, but the key aspect to remember while doing so is that threads often share data. This shared data problem might lead to cache incoherence which degrades the performance instead of improving it. This is known as the ping pong affect. Our paper tries to deal with this issue by using affinity to cores while issuing threads.

2.3 WHY THIS IS A PROJECT RELATED TO THIS CLASS

In our project we will utilize the concept of multithreading to handle multiple requests and will ensure cache coherence by using affinity to cores.

2.4 OUR APPROACH COMPARED TO THE PAPER

There a different types of dynamic requests to a web server. Because the classes of dynamic requests are limited the paper uses an offline log to create a look up table and classify incoming requests. The offline log contains the mean service times and a frequency distribution of the different types of requests. The problem with this approach could be that the offline log is static and thus the weight calculation happens once. The weight of the queues that hold the requests is dependent on this frequency which can change in a practical situation. Our approach takes this possible variation in frequencies by continuously updating the look up table while assigning the incoming requests to the respective queue. We call this the modified DRWS algorithm.

2.5 STATEMENT OF THE PROBLEM

Our goal is to implement a dynamic request scheduling model in web servers and reduce mean service time of these requests.

2.6 SCOPE OF INVESTIGATION

Our scope is to achieve the following:

1. Simulate a web server.
2. Apply the dynamic request scheduling model on the web server and our modified algorithm (MDRWS) on the server.
3. Compare the results of using a static lookup table versus dynamically updating the look up table and classifying the incoming requests.

3.THEORETICAL BASES AND LITERATURE REVIEW

3.1 DEFINITION OF PROBLEM

Typical scheduling algorithms implemented on web servers such as First Come First Serve and Shortest Job First does not consider multiple cores on a system. Most web servers have multiple cores. If multiple cores are not taken into consideration the OS will assign threads to incoming requests, and these threads (even if they have shared data) can be assigned to different cores. This leads to a possibility of cache incoherence and slower performance of the web server. This is known as the Ping Pong affect. The DRWS algorithm takes into account the distribution of dynamic requests service times as well as exploits the multi core architecture of web servers by means of core affinity and avoids the ping pong affect. It is based on Weighted Fair Queuing.

3.2 THEORETICAL BACKGROUND OF THE PROBLEM

The DRWS algorithm is based on weighted fair queueing. The paper we are implementing will identify the requests based on the type and insert them into queues depending on the type. These queues are given weights according to the request frequency. Depending on the weights of the queue they are assigned a certain proportion of the core. Lastly, these requests are processed.

3.3 RELATED RESEARCH TO SOLVE THE PROBLEM

Our solution is based on the algorithms proposed in the following research paper.

1. Dynamic Request Scheduling Model in Multi Core Web Servers.

2. Scheduling Strategy to Improve Response Time for Web Applications.

Most web servers use FCFS or SJF for scheduling. In FCFS the requests are handled according to their arrival time. This results in a long average waiting time. To overcome this problem sometimes SJF strategy is applied where the shortest job is handled before longer jobs. In order to implement this strategy we need to know the length of this jobs beforehand. This is done by maintaining an offline log. SJF has the benefits of reducing the overall wait time. But both these approaches don't utilize the multiple core structure of a web server to its full extent. Even though FCFS is a starvation free technique it doesn't take into account multiple cores.

3.4 ADVANTAGES & DISADVANTAGES OF THOSE RESEARCHES

The drawbacks of FCFS and SJF are mentioned in section 3.3. The advantages of using DRWS is given below:

a) ADVANTAGES OF DYNAMIC REQUEST SCHEDULING MODEL

1. DRWS performs better compared to FCFS in terms of mean response time. It is very close to the results seen in SJF but better than SJF. The overall mean response time of FCFS is the least, which makes DRWS better among the three scheduling algorithms.
2. Since DRWS exploits the multi core nature of web servers, if the number of threads are increased in the thread pool the mean service time drops more sharply in case of DRWS than in the case of FCFS and SJF.
3. Lastly, threads that share data as we know, are assigned to the same processing core. This avoids the Ping Pong affect. If the shared data size is increased, DRWS performs the same which means, the mean service time does not increase due to this, but in FCFS and SJF is the shared data size is increased, the mean service time increases due to cache incoherence.

3.5 OUR SOLUTION TO THE PROBLEM

The modification that we add to DRWS is including a methodology to continuously update the lookup table which contains information about the incoming requests. In doing so, we will test the web server with different types of workloads and compare its performance when the frequency of certain request types suddenly spike to high or drop to a low count. In order to do so we will make use of Java's histogram and sliding time window reservoir classes to store the most recent information of the incoming requests. This will help is calculating the weights of the queue. Thus, in our solution the weights of the different queues will be dynamic and not static as is the case in the original research.

3.6 WHERE OUR SOLUTION DIFFERS FROM OTHERS AND IS BETTER

Our solution takes into consideration the following aspects:

1. Multiple core architecture of the web server by using affinity to cores.
2. The log file contains information about the frequency distribution of the requests and their mean service times. The researched paper uses an offline log and creates a static lookup table using the log, whereas in our approach we continuously update lookup table to contain the most recent information regarding the frequencies and mean service times such the weight assigned to a queue is most accurate.

3.6 WHY OUR SOLUTION IS BETTER

Same as above

4. HYPOTHESIS OR GOALS

4.1 HYPOTHESIS

Our goal is to achieve a quicker response time compared to DRWS by dynamically updating the lookup table and assigning the requests instead of using a static lookup table.

5. METHODOLOGY

5.1 HOW TO GENERATE/COLLECT INPUT DATA

We will make use of a tool called Apache JMeter to test performance on dynamic requests. We will use it to simulate a varied load on the web server to test its performance under different types of dynamic requests.

5.2 HOW TO SOLVE THE PROBLEM and ALGORITHM DESIGN

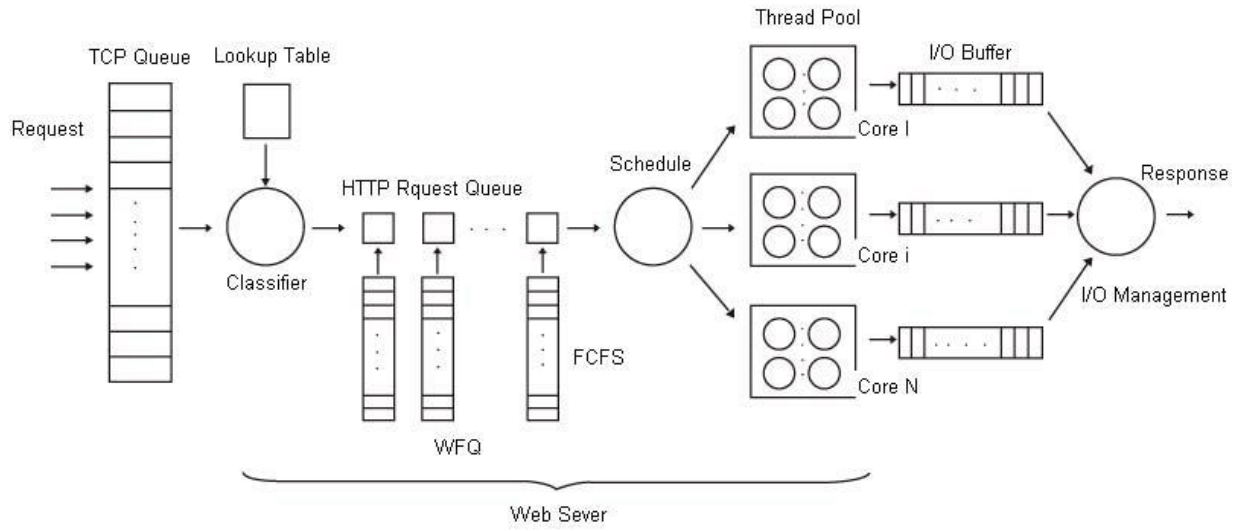


Fig-1

As shown in figure 1, the flow of the algorithm is depicted. Following are the steps that we will follow to implement this approach.

5.3 ALGORITHM DESIGN

Step 1: The initial look up table is taken from an offline log that contains information about the frequencies of request as visiting times. Similarly the service times are also pre populated from the offline log.

Step 2: Repeat steps 3-6 every T seconds. We make use of the sliding window implementation of the reservoir class in java in order to record the incoming requests in every T seconds. This method records the nature of the past T seconds on the web server and thus updates the lookup table every T seconds.

Steps 3 till 6 are used to calculate the scheduling parameters.

Step 3: Calculate the weight of the request queue

The request frequency and the mean service time C_i and T_i respectively are taken from the lookup table and are used to calculate the weight Wi for queue i:

$$W_i = C_i * T_i$$

Step 4: The weight of each queue decides the number of threads that will be assigned to each request queue from the thread pool. The thread pool contains a fixed number of threads N.

λ_i is the number of threads assigned to queue i , from the N threads in the thread pool.

Let W be the total weight of all the queues i.e. $W = W_1 + W_2 + \ldots\ldots + W_M$, where M is the total number of queues.

Where , $\lambda_i = \left(\frac{Wi}{W} * N\right)$

Step 5: In order to avoid the ping pong affect we will make sure that one type of request queue is only assigned to one core, although a core may service multiple queues. Thus, we now calculate the number of threads that will be assigned to a core. We implement a predefined thread allocation strategy to allocate threads from the pool to each processing core in order to achieve load balancing between cores.

Step 6: After the scheduling parameter calculation, the threads are issued to each request and these requests are processed. The requested content is then generated which is sent to the I/O buffer. This is the response of the webserver to each incoming requests.

Step 7: End

5.4 LANGUAGE USED

We will implement our algorithm using Java.

5.5 TOOLS USED

Java IDE, Apache JMeter.

5.6 HOW TO GENERATE OUTPUT

Our program will simulate a web server and the output will be graphs of the web server's performance based on different criterion.

5.7 HOW TO TEST AGAINST HYPOTHESIS

We will run two versions DRWS on the web server. One version uses a static lookup table and the other version, MDRWS uses a dynamic lookup table. This table is updated every T seconds by implementing the reservoir sliding window class in java. We will test the web server on different workloads and compare the results seen on DRWS and MDRWS.

6. IMPLEMENTATION:

6.1 CODE:

The code for the implementation of our web server is:

MyServer Class:

```
package project;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.LinkedList;
import java.util.Queue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.Executors;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

import net.openhft.affinity.AffinitySupport;
```

```
import net.openhft.affinity.IAffinity;


public  class MyServer{

        private BlockingQueue<Request> qFact = new LinkedBlockingQueue<Request>();
        private BlockingQueue<Request> qSqua = new LinkedBlockingQueue<Request>();
        private BlockingQueue<Request> qFibo = new LinkedBlockingQueue<Request>();
        private BlockingQueue<Request> qReve = new LinkedBlockingQueue<Request>();
        private long qFactCounter = 0;
        private long qSquaCounter = 0;
        private long qFiboCounter = 0;
        private long qReveCounter = 0;
        private double qFactWeight = 0.146;
        private double qSquaWeight = 0.243;
        private double qFiboWeight = 0.073;
        private double qReveWeight = 0.1825;
        public static final double mstFact = 0.73;
        public static final double mstFibo = 0.73;
        public static final double mstReve = 0.73;
        public static final double mstSqua = 0.54;
        // this scheduler can be given a classs that implelements runnaable.

        private static final ScheduledExecutorService scheduler =
                Executors.newScheduledThreadPool(1);


            public BlockingQueue<Request> getqFact() {
            return qFact;
        }
```

```
public void setqFact(BlockingQueue<Request> qFact) {
        this.qFact = qFact;
}


public BlockingQueue<Request> getqFibo() {
        return qFibo;
}


public void setqFibo(BlockingQueue<Request> qFibo) {
        this.qFibo = qFibo;
}

public BlockingQueue<Request> getqSqua() {
        return qSqua;
}


public void setqSqua(BlockingQueue<Request> qSqua) {
        this.qSqua = qSqua;
}


public BlockingQueue<Request> getqReve() {
        return qReve;
```

```
}


public void setqReve(BlockingQueue<Request> qReve) {
        this.qReve = qReve;
}


 public long getqFactCounter() {
        return qFactCounter;
}


public void setqFactCounter(long qFactCounter) {
        this.qFactCounter = qFactCounter;
}


public long getqSquaCounter() {
        return qSquaCounter;
}


public void setqSquaCounter(long qSquaCounter) {
        this.qSquaCounter = qSquaCounter;
}
```

```
public long getqFiboCounter() {
        return qFiboCounter;
}

public void setqFiboCounter(long qFiboCounter) {
        this.qFiboCounter = qFiboCounter;
}

public long getqReveCounter() {
        return qReveCounter;
}

public void setqReveCounter(long qReveCounter) {
        this.qReveCounter = qReveCounter;
}

public double getqFactWeight() {
        return qFactWeight;
}

public void setqFactWeight(double qFactWeight) {
        this.qFactWeight = qFactWeight;
}
```

```
public double getqSquaWeight() {
	return qSquaWeight;
}

public void setqSquaWeight(double qSquaWeight) {
	this.qSquaWeight = qSquaWeight;
}

public double getqFiboWeight() {
	return qFiboWeight;
}

public void setqFiboWeight(double qFiboWeight) {
	this.qFiboWeight = qFiboWeight;
}

public double getqReveWeight() {
	return qReveWeight;
}

public void setqReveWeight(double qReveWeight) {
	this.qReveWeight = qReveWeight;
}
```

```
private ServerSocket ss= null;
 private int port=0;

 public MyServer(int port, ServerSocket ss) throws IOException {
            this.port= port;
            this.ss=ss;

 }


 public ServerSocket getSs() {
            return ss;
 }

 public void setSs(ServerSocket ss) {
            this.ss = ss;
 }

 public int getPort() {
            return port;
 }

 public void setPort(int port) {
            this.port = port;
 }

 public void parseAndAssignQueue(Socket socket){
```

```
String clientRequest = "";
String clientRequestTemp;
try {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        //System.out.println("coming from client");
        int x = 0;
        while(!(clientRequestTemp= reader.readLine().trim()).equals("")){
            //System.out.println(clientRequestTemp);
            if(x== 0) clientRequest += clientRequestTemp + "\n";
            else continue;
            x++;
        }
        //System.out.println(clientRequest);
        //System.out.println("Return Value :" );
        String[] retval= clientRequest.split(" ", 0);
        String request= retval[1].substring(1);

        double ans=0;
        String method = request.substring(0, 4);
        String arg = request.substring(4, request.length());

        switch (method) {
        case "fact":
            Request req = new FactorialRequest(socket, arg);
            qFact.add(req);
            qFactCounter++;
            break;
```

```
                case "fibo":
                        req = new FibbonacciRequest(socket, arg);
                        qFibo.add(req);
                        qFiboCounter++;
                        break;

                case "squa":
                        req = new SquareRequest(socket, arg);
                        qSqua.add(req);
                        qSquaCounter++;
                        break;

                case "reve":
                        req = new ReverseStringRequest(socket, arg);
                        qReve.add(req);
                        qReveCounter++;
                        break;

                default:
                        System.out.println("404");
                        socket.close();
                       break;
                }

        }catch(IOException e){
                System.out.println(" some error from run menthod");}

}
```

```
public static void main(String[] args)  {

        ServerSocket ss  = null;
        int portNumber= 9990;
        MyServer myserver = null;
        try {
                ServerSocket socket = new ServerSocket(portNumber);
                myserver = new MyServer(portNumber, socket);
        } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
                return;
        }

        // we are pre creating threads and then assigning the queues to the created threads.
        //Let the thread pool have 10 threads. 30% fib and 70% to fact
        final int nThreads = 100;
        final int nCores = 8;
        ThreadPool tp = new ThreadPool(nThreads, nCores,  myserver);
        tp.setupThreads();

        ScheduledWeightUpdater updater = new ScheduledWeightUpdater(myserver, tp);
        //scheduler.scheduleAtFixedRate(updater, 10, 10, TimeUnit.SECONDS);
        // Shifting main thread to CPU1 since CPU0 should be left to OS.
        IAffinity impl = AffinitySupport.getAffinityImpl();
        impl.setAffinity(1 << 1);
        while(true){
                try{
                        Socket newSocket = myserver.getSs().accept();
```

```
                        myserver.parseAndAssignQueue(newSocket);
                    }catch(Exception e){
                        e.printStackTrace();
                        System.out.println("connection error");
                    }
                }
            }
        }
```

ServerThread Class:

```
package project;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Queue;
import java.util.concurrent.BlockingQueue;

import net.openhft.affinity.AffinitySupport;
import net.openhft.affinity.IAffinity;

public class ServerThread extends Thread {
```

```
private BlockingQueue<Request> myQueue;
private long affinityMask = 1;
private BlockingQueue<Request> updatedQueue;


public BlockingQueue<Request> getMyQueue() {
        return myQueue;
}
public void setMyQueue(BlockingQueue<Request> myQueueParam) {
        this.updatedQueue = myQueueParam;
}



public void setCoreAffinity(int coreId) {
        long mask = 1;
        this.affinityMask = mask << coreId;
}

public void run()  {

        // The calling thread should call setAffinity to set it's affinity to a particular core.
        handleInterrupt();

        Request r = null;

        while (true) {
                // Check if interrupt flag is set.
```

```
                    if(Thread.interrupted()) {
                            handleInterrupt();
                    }

                    try {
                            // take blocks if the queue is empty.
                            //this thread is always reading from one queue,
                            r = myQueue.take();
                    } catch (InterruptedException e1) {
                            //e1.printStackTrace();
                            // interrupted. Return.
                            handleInterrupt();
                            Thread.currentThread().isInterrupted(); // clear the interrupt flag.
                            continue;
                    }


                    String response = r.process();
                    BufferedWriter writer = null;
                    try{
                            writer = new BufferedWriter(new
OutputStreamWriter(r.getSocket().getOutputStream()));
                            //instead of simply sending the response on the screen, we will have to
send it
                            // as HTTP response
                            writer.write("HTTP/1.1 200 OK\r\n");
                            writer.write("\r\n");
                            writer.write("Response:"+ response);
                            writer.flush();
```

```
                    writer.close();


                } catch(Exception e) {
                    System.out.println("some probelm");
                } finally {
                    if (writer != null) {
                        try {
                            writer.close();
                        } catch (IOException e) {
                        }
                    }
                }
            }

        public void handleInterrupt() {
            // Two things to do when handling interrupts
            // 1. update core affinity (OpenHFT)
            IAffinity impl = AffinitySupport.getAffinityImpl();
            impl.setAffinity(this.affinityMask);

            //2. Update myqueue. This is safe because thread loop code is jumped to this handler
            this.myQueue = this.updatedQueue;
        }
}
```

ThreadPool Class:

```
package project;
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.concurrent.BlockingQueue;

public class ThreadPool
{
   private final int nThreads;
   private final ServerThread[] serverThreads;
   private MyServer sockServer;
   private int maxCores;
   private boolean isInitial = true;

   public ThreadPool(int nThreads, int maxCores, MyServer socServer)
   {
      this.nThreads = nThreads;
      this.sockServer = socServer;
      this.serverThreads = new ServerThread[nThreads+4];
      for(int i = 0 ; i < nThreads+4; i++) {
           this.serverThreads[i] = new ServerThread();
      }
      this.maxCores = maxCores;
   }

   public void setupThreads() {

           ArrayList<Double> weights = new ArrayList<>();
           weights.add(sockServer.getqFactWeight());
```

```
weights.add(sockServer.getqFiboWeight());
weights.add(sockServer.getqReveWeight());
weights.add(sockServer.getqSquaWeight());
Collections.sort(weights);

int cpuid = 0;

HashMap<BlockingQueue<Request>, Integer> queueToCore = new HashMap<>();
HashMap<BlockingQueue<Request>, Double> queueToWeight = new HashMap<>();
queueToWeight.put(sockServer.getqFact(), sockServer.getqFactWeight());
queueToWeight.put(sockServer.getqFibo(), sockServer.getqFiboWeight());
queueToWeight.put(sockServer.getqSqua(), sockServer.getqSquaWeight());
queueToWeight.put(sockServer.getqReve(), sockServer.getqReveWeight());


// Do roundrobin per group.
for (Double weight : weights) {
        BlockingQueue<Request> queueans = null;
        for(BlockingQueue<Request> queue: queueToWeight.keySet()) {
                Double w = queueToWeight.get(queue);
                if(Double.compare(weight, w) == 0){
                        queueans = queue;
                }
        }
        if(queueans !=null) {
                queueToWeight.remove(queueans);
        }
        //interleaving the q's assigned to the cores.
        queueToCore.put(queueans, cpuid++);
```

```
                if (cpuid > this.maxCores) {
                        cpuid = 0;
                }
        }

        double totalW = sockServer.getqFactWeight() + sockServer.getqFiboWeight() +
sockServer.getqReveWeight() + sockServer.getqSquaWeight();
        int lFact = (int) (sockServer.getqFactWeight()/totalW * nThreads);
        int lFibo = (int) (sockServer.getqFiboWeight()/totalW * nThreads);
        int lReve = (int) (sockServer.getqReveWeight()/totalW * nThreads);
        int lSqua = (int) (sockServer.getqSquaWeight()/totalW * nThreads);
        ///asign minimum one threaD to a q
        if (lFact == 0) {
                lFact = 1;
        }

        if (lFibo == 0) {
                lFibo = 1;
        }

        if (lReve == 0) {
                lReve = 1;
        }

        if (lSqua == 0) {
                lSqua = 1;
        }
    int i = 0;
        for(i = 0; i < lFact; i++) {
```

```
                serverThreads[i].setMyQueue(sockServer.getqFact());
                serverThreads[i].setCoreAffinity(queueToCore.get(sockServer.getqFact()));
        }

        System.out.println("Assigned " + lFact + " threads to queue Factorial on Core: " +
queueToCore.get(sockServer.getqFact()));
        int base = i;
        for(; i < base + lFibo; i++) {
                serverThreads[i].setMyQueue(sockServer.getqFibo());
                serverThreads[i].setCoreAffinity(queueToCore.get(sockServer.getqFibo()));
    }

        System.out.println("Assigned " + lFibo + " threads to queue Fibbonacci on Core: " +
queueToCore.get(sockServer.getqFibo()));
    base = i;
        for(; i < base + lReve; i++) {
                serverThreads[i].setMyQueue(sockServer.getqReve());
                serverThreads[i].setCoreAffinity(queueToCore.get(sockServer.getqReve()));
    }

        System.out.println("Assigned " + lReve + " threads to queue Reverse on Core: " +
queueToCore.get(sockServer.getqReve()));
    base = i;
        for(; i < base + lSqua; i++) {
                serverThreads[i].setMyQueue(sockServer.getqSqua());
                serverThreads[i].setCoreAffinity(queueToCore.get(sockServer.getqSqua()));
    }

        System.out.println("Assigned " + lSqua + " threads to queue Square on Core: " +
queueToCore.get(sockServer.getqSqua()));
```

```
        base = i;
                for(i=0 ; i < base ; i++) {
                        if(isInitial) {
                                serverThreads[i].start();
                        } else {
                                // if not inital, we just interrupt all threads so that they internally call
handle interrupts.
                                serverThreads[i].interrupt();
                        }
            }
                if (isInitial){
                        isInitial = false;
                }
     }
}
```

ScheduledWeightUpdater Class:

```
package project;
public class ScheduledWeightUpdater implements Runnable {

        MyServer myServer;
                // These arrays contain last 5 minute snapshots

        public static long factV[] = new long[5];
        static long fiboV[] = new long[5];
        static long squaV[] = new long[5];
        static long reveV[] = new long[5];
        static int current = 0;
```

```
static double threshold = 0.03;
static ThreadPool threadPool;

public ScheduledWeightUpdater(MyServer myserver, ThreadPool tp) {
        this.myServer = myserver;
        threadPool = tp;
}


@Override
public void run() {
        try {
                run1();
        } catch (Throwable e) {

        }
}

public void run1() {
        System.out.println("Scheduled weight updater");
        // Get all frequencies.
        current++;
        if (current == 5) {
                current = 0;
        }

        // Take snapshot
        factV[current] = myServer.getqFactCounter();
        fiboV[current] = myServer.getqFiboCounter();
```

```
squaV[current] = myServer.getqSquaCounter();
reveV[current] = myServer.getqReveCounter();

double windowWeights[] = new double[4];

int last5 = current + 1;
if ( last5 == 5) {
        last5 = 0;
}

long factC = factV[current] - factV[last5];
long fiboC = fiboV[current] - fiboV[last5];
long reveC = reveV[current] - reveV[last5];
long squaC = squaV[current] - squaV[last5];

long sum = factC + fiboC + reveC + squaC;
if (sum == 0) return;
double factCi = factC / (double)sum;
double fiboCi = fiboC / (double)sum;
double squaCi = squaC / (double)sum;
double reveCi = reveC / (double)sum;

System.out.println ("Queue: Fact\t\t\tFibo\t\t\tSqua\t\t\tReve");
System.out.println("Count: " + factC + "\t\t\t " + fiboC + "\t\t\t " + squaC + "\t\t\t " +
reveC + " ");
System.out.println("Ci: " + factCi + "\t " + fiboCi + "\t " + squaCi + "\t " + reveCi + " ");
double factw = factCi * MyServer.mstFact;
double fibow = fiboCi * MyServer.mstFibo;
double squaw = squaCi * MyServer.mstSqua;
```

```
                double revew = reveCi * MyServer.mstReve;
                System.out.println("W: " + factw + "\t " + fibow + "\t " + squaw + "\t " + revew + " ");
                if (Math.abs(factw - myServer.getqFactWeight()) > threshold ||
                                Math.abs(revew - myServer.getqReveWeight()) > threshold ||
                                Math.abs(fibow - myServer.getqFiboWeight()) > threshold ||
                                Math.abs(squaw - myServer.getqSquaWeight()) > threshold) {
                        System.out.println("Updating weights");
                        myServer.setqFactWeight(factw);
                        myServer.setqFiboWeight(fibow);
                        myServer.setqReveWeight(revew);
                        myServer.setqSquaWeight(squaw);
                        System.out.println("Rebalancing threads and affinities.");
                        threadPool.setupThreads();
                }
        }
}
```

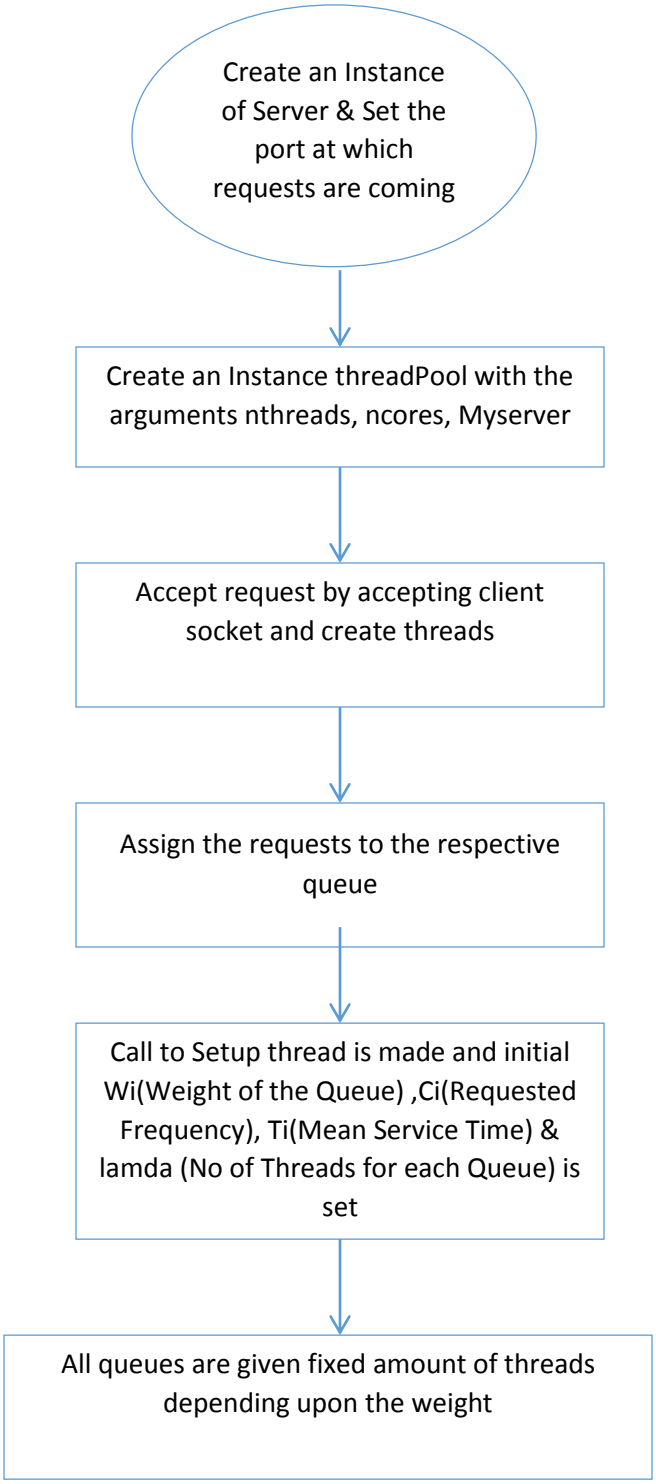
Request Class:

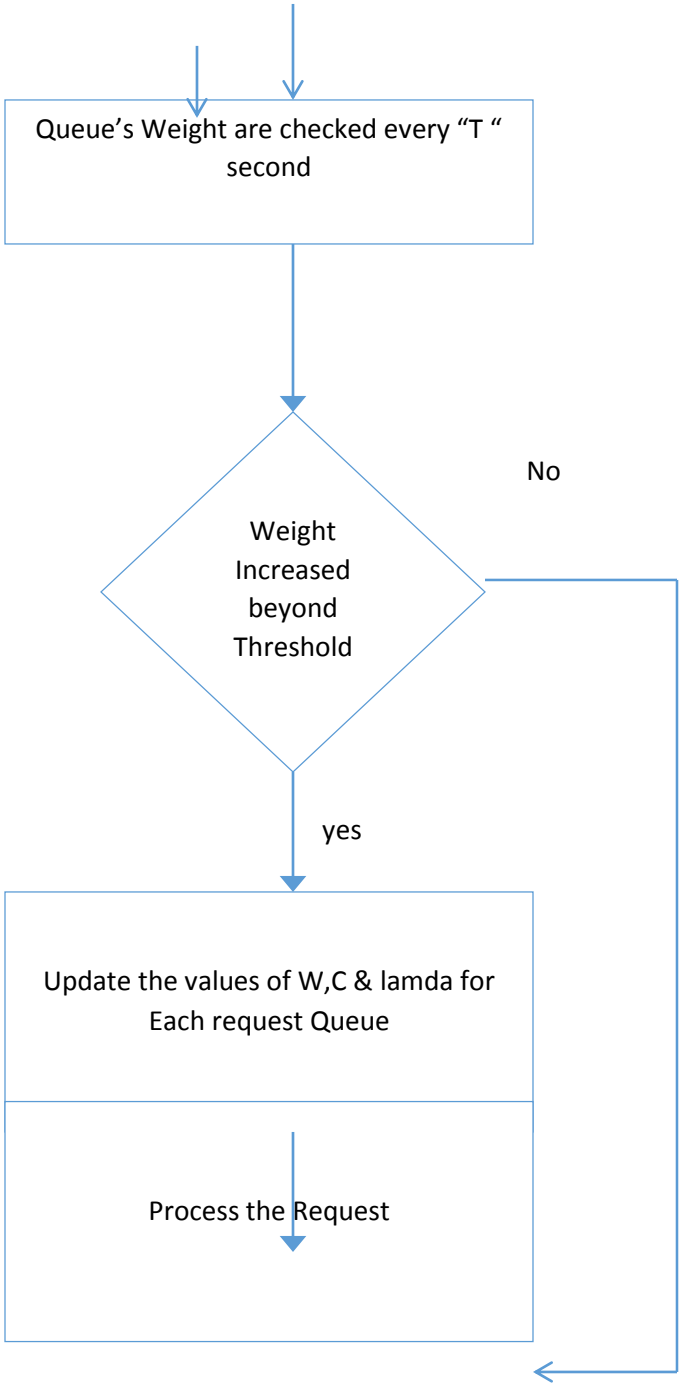
We have considered four types of requests coming to the web server. These are the FactorialRequest, FibonacciRequest, SuareRequest and the ReverseStringRequest classes. All these classes are inherited from the parent class Request which is given below:

```
package project;

import java.net.Socket;

abstract public class Request {
```

```
        //private static int  counter;

        private Socket socket;
        private String arg;

        public String getArg() {
                return arg;
        }
        public void setArg(String arg) {
                this.arg = arg;
        }
        public Socket getSocket() {
                return socket;
        }
        public void setSocket(Socket socket) {
                this.socket = socket;
        }

        public abstract String process();

        public Request(Socket s, String argParam){
                socket=s;
                arg=argParam;
        }
}
```

6.2 DESIGN DOCUMENT AND FLOWCHART:




Queue's Weight are checked every "T " second
Weight Increased beyond Threshold
No
yes
Update the values of W,C & lamda for Each request Queue
Process the Request

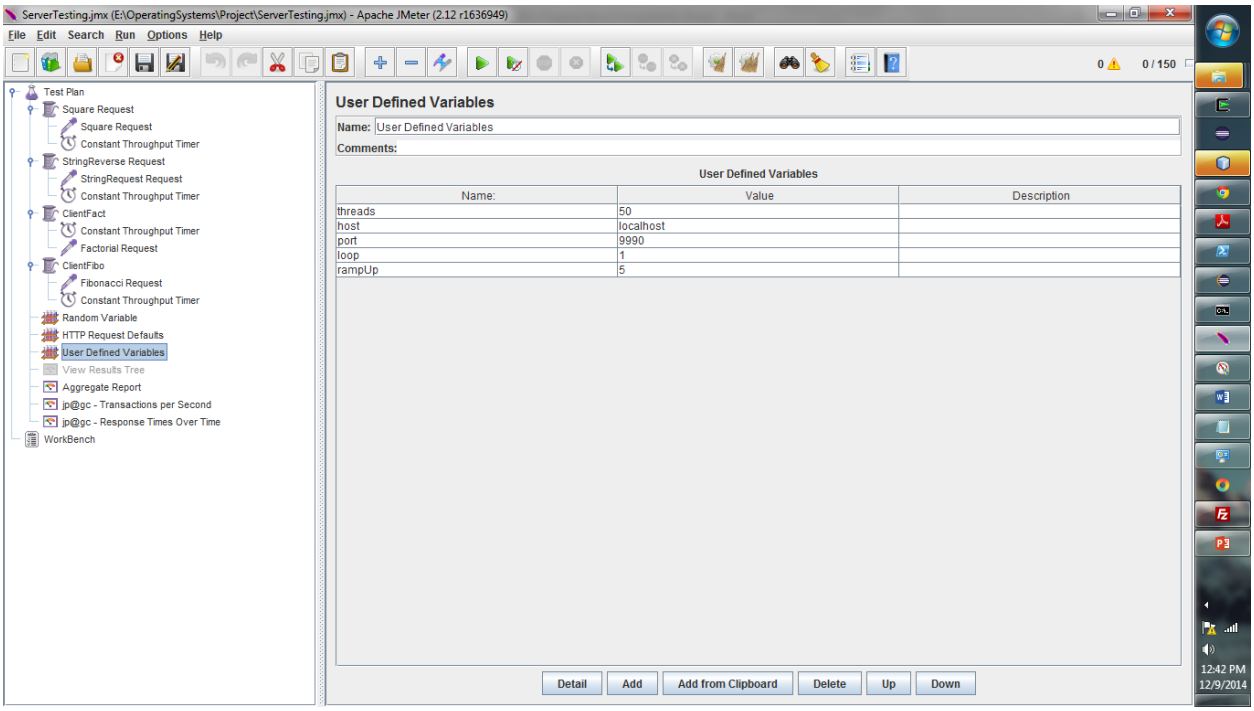
7. DATA ANALYSIS AND OUTPUT GENERATION:

The web server was scheduled using DRWS and the modified DRWS. In order to test the webserver we used a jmeter test plan. This test plan represents four classes of requests that we defined in our program. We picked the following functions:

1. Calculation of the Fibonacci sequence for a randomly generated number.
2. Calculation of the factorial for a randomly generated number.
3. Calculation of the square for a randomly generated number.
4. Calculation of the reverse of a randomly generate String.

On jmeter we can change the number of such requests that are given to the webserver and study the output in terms of an Aggregate report and Response over Time.

The snapshots of the test plan is given below:



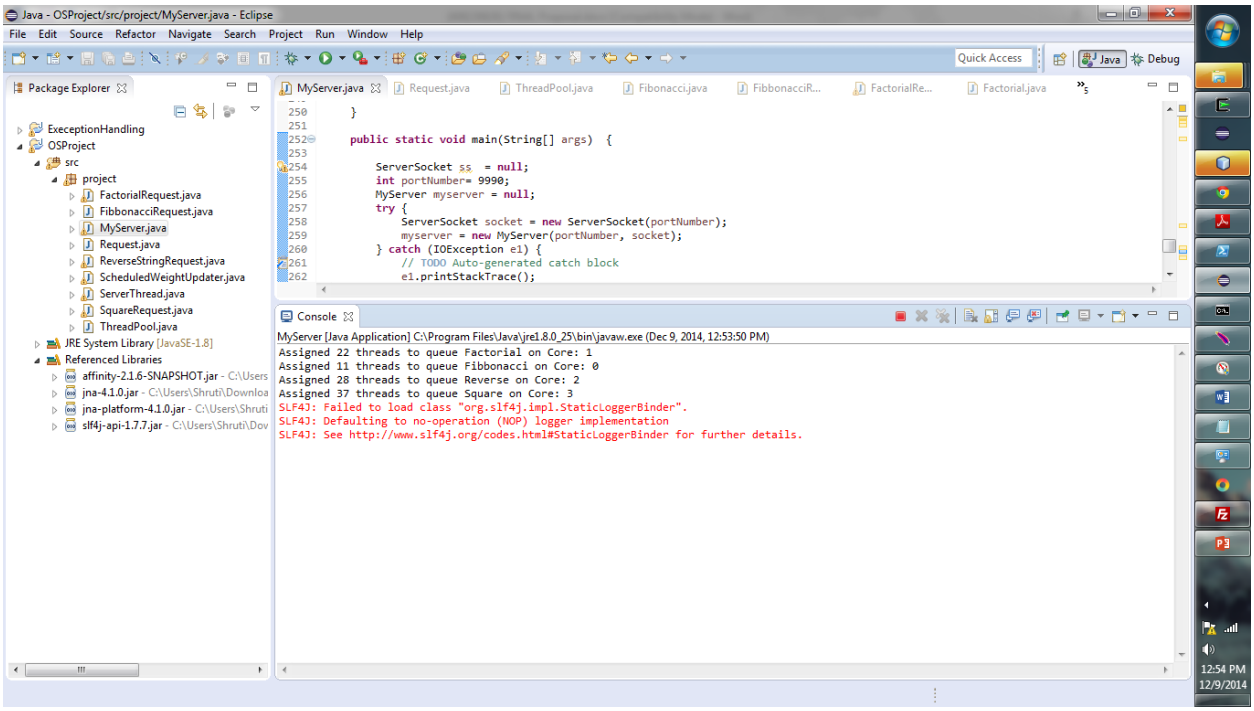
Initially we assumed the values of the request frequencies, mean service times, weight of the queues and threads assigned to each queue as the following:

The summation of all the frequencies is 1, and the total number of threads (we can change this) is 100. These are the values in the offline log which are used to assign weights to the queues

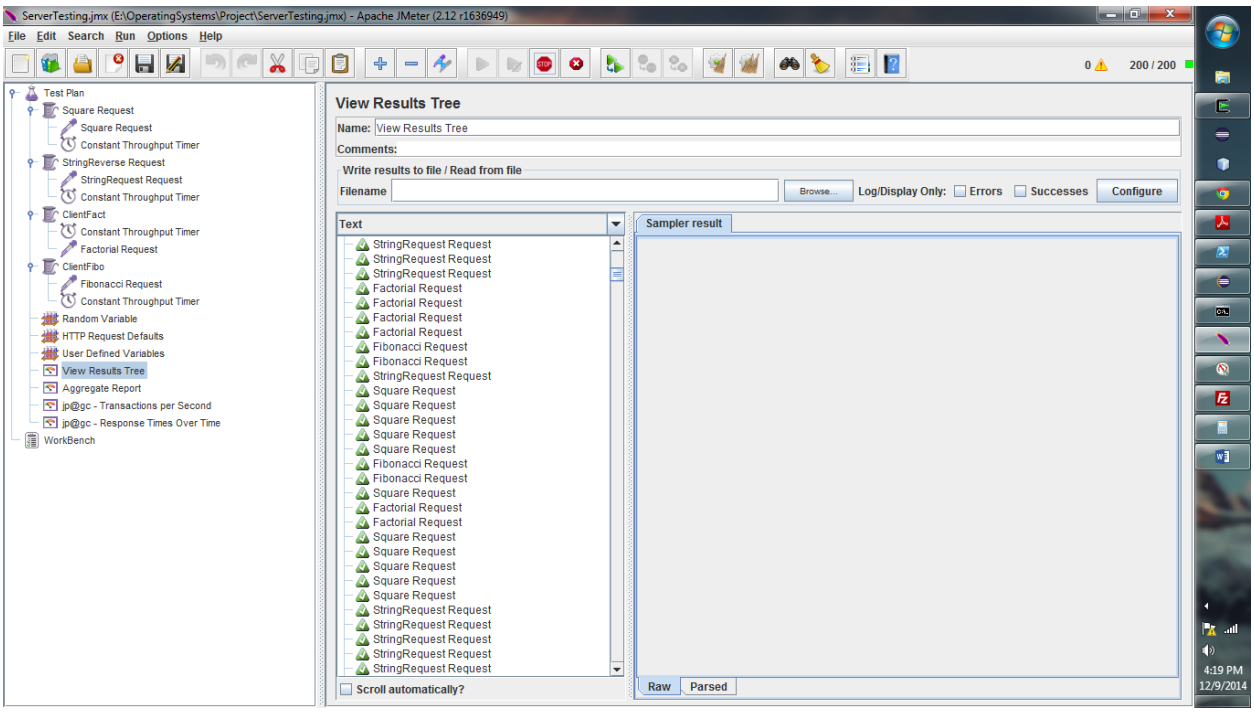
	Request Factorial	Request Fibonacci	Request Square	Request String Reverse
Request Frequency (C_i)	0.2	0.1	0.45	0.25
Request Mean Service Time (T_i)	0.73 ms	0.73 ms	0.54 ms	0.73 ms
Weight of queue (W_i)	0.146	0.073	0.243	0.1825
Threads Assigned to each Queue	23	11	38	28

and then threads to the queues.

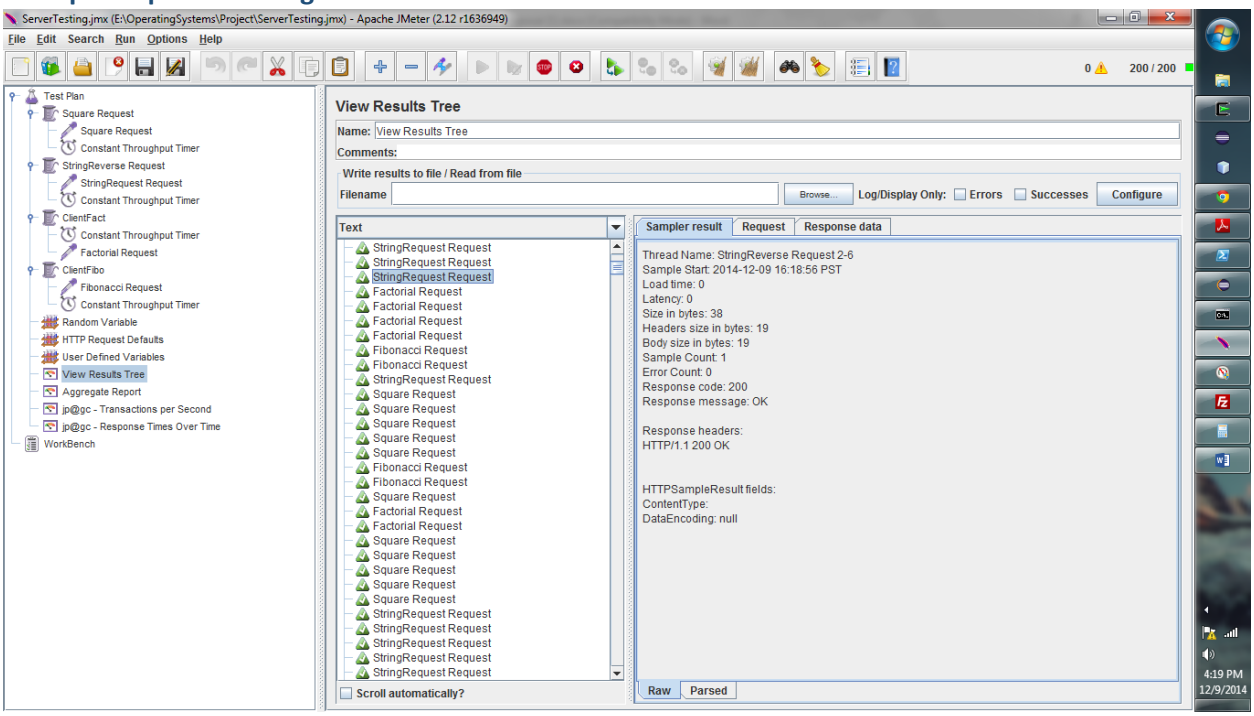
On running the webserver on the original algorithm, without changing the weights and simply using the above offline log.

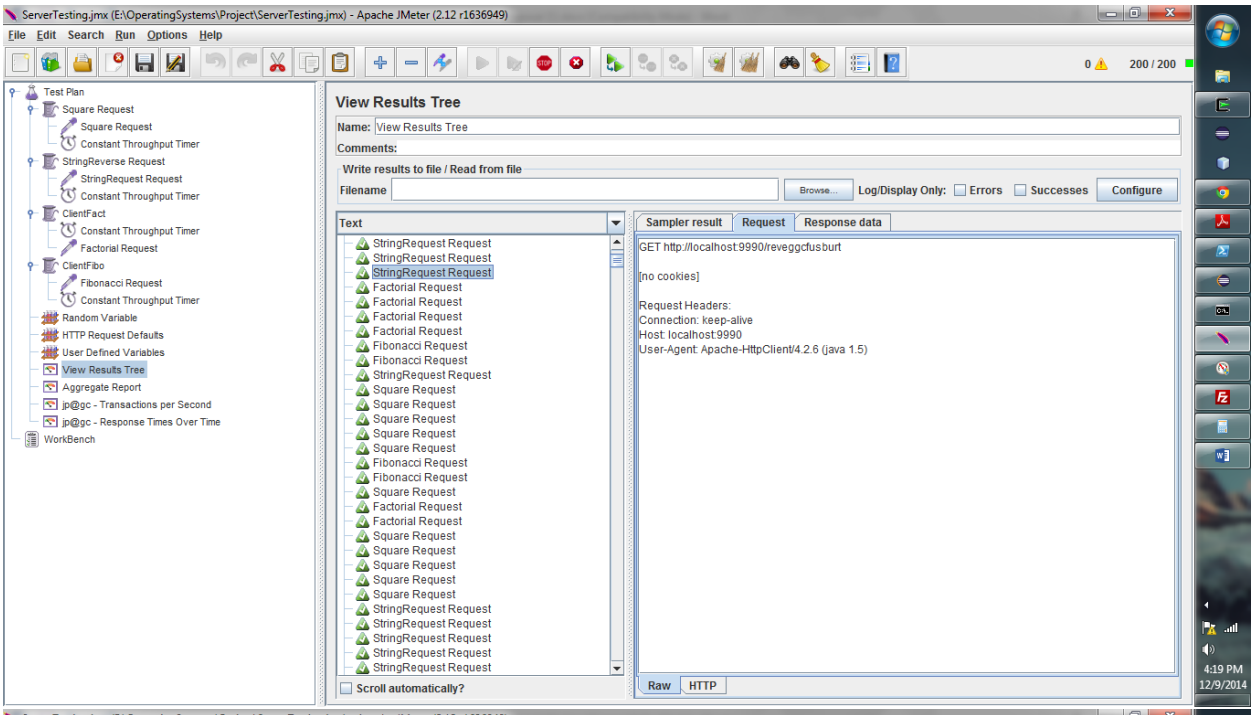


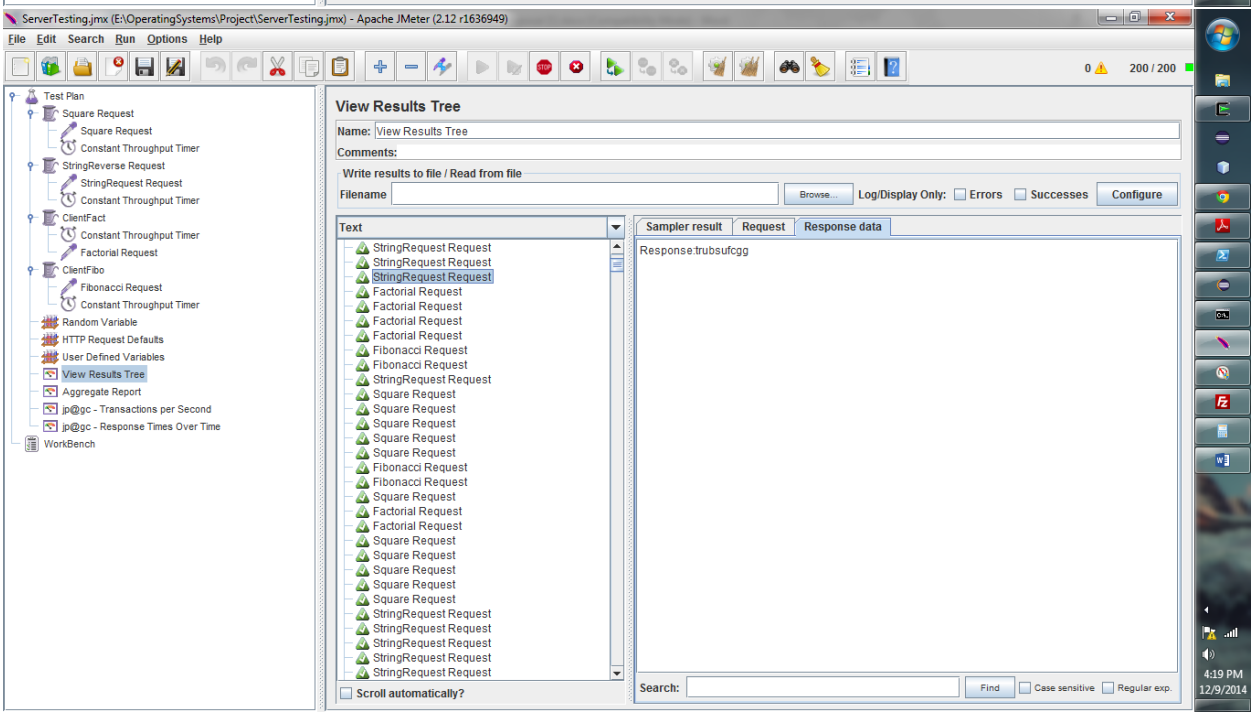
Jmeter response:



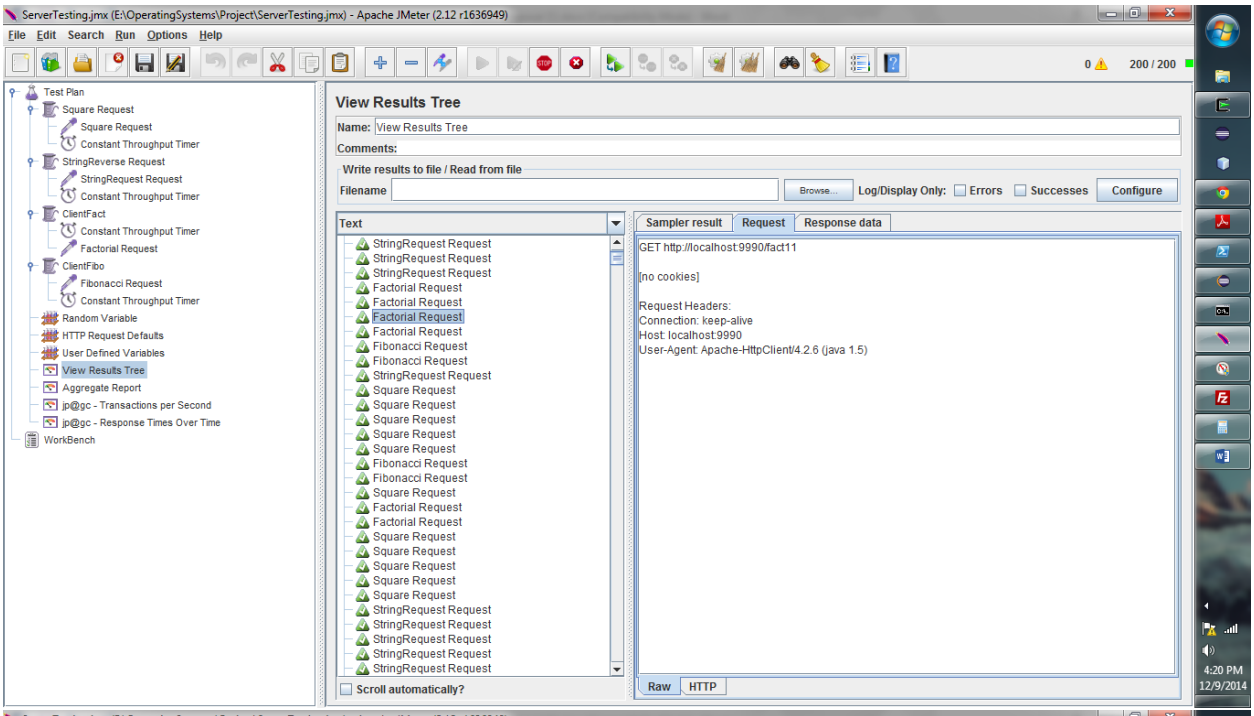
Example request of string reverse:

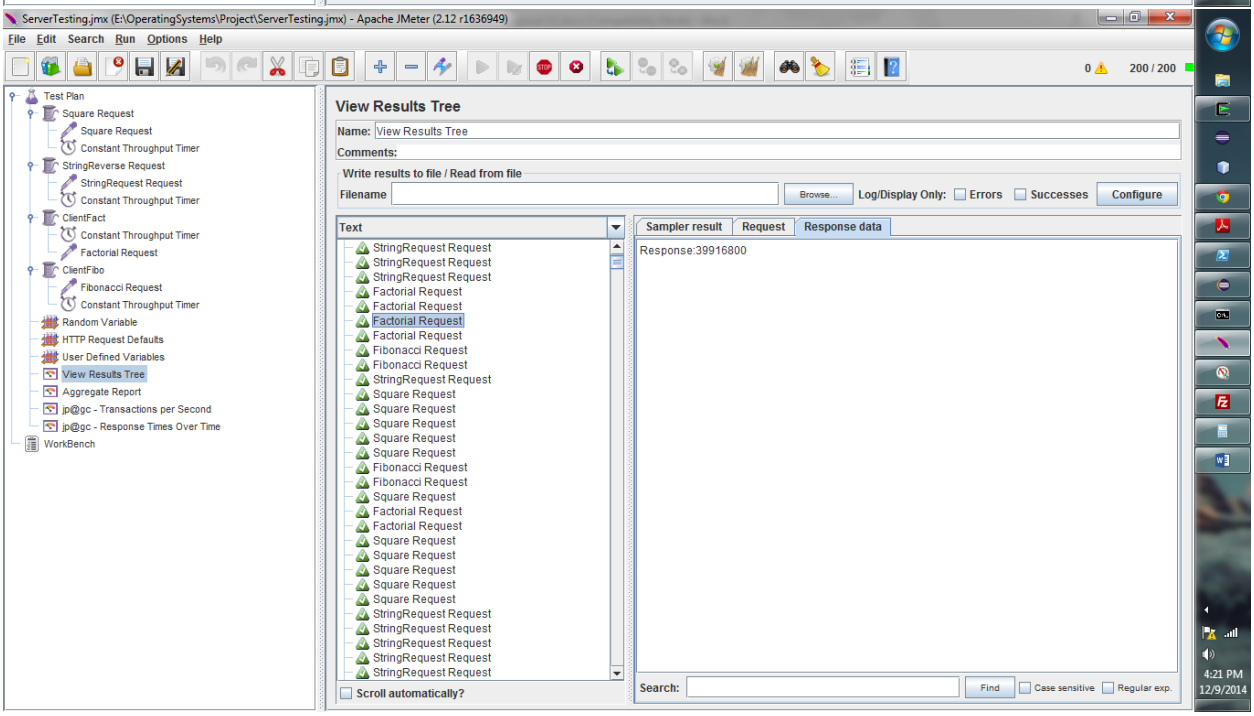


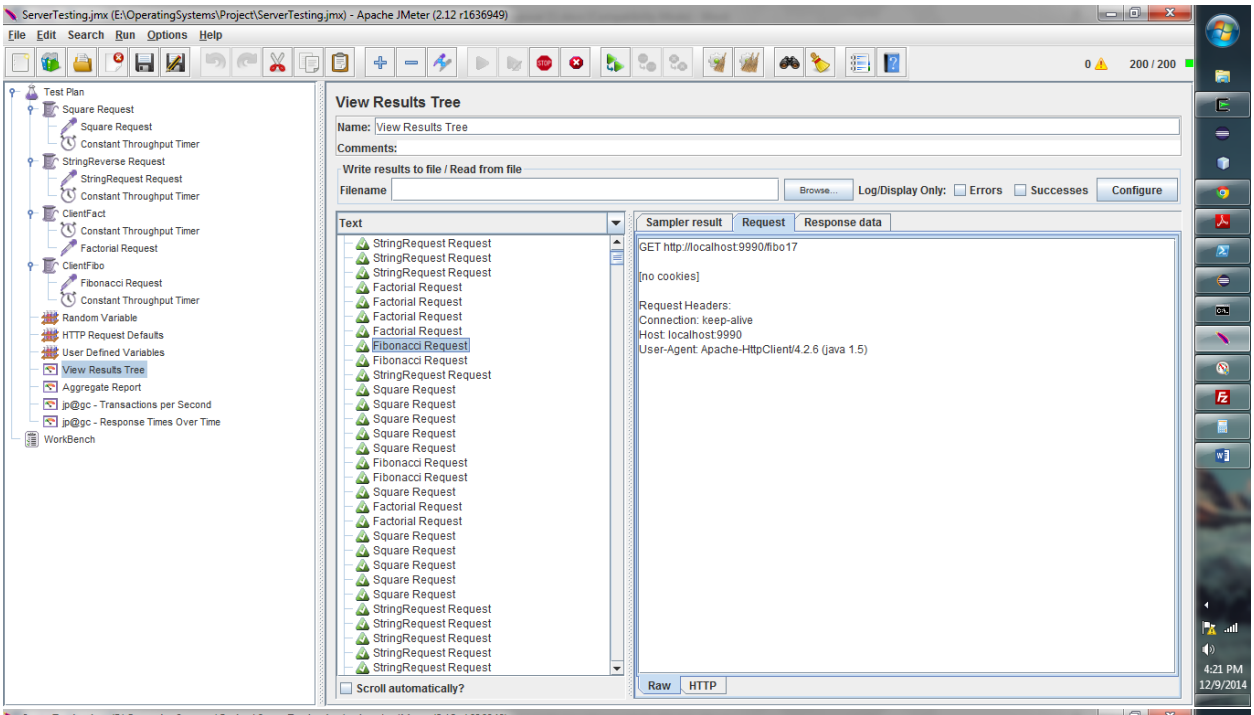

ServerTesting.jmx (E:\OperatingSystems\Project\ServerTesting.jmx) - Apache JMeter (2.12 r1636949)
File Edit Search Run Options Help
0 200 / 200
Test Plan
Square Request
Square Request
Constant Throughput Timer
StringReverse Request
StringRequest Request
Constant Throughput Timer
ClientFact
Constant Throughput Timer
Factorial Request
ClientFibo
Fibonacci Request
Constant Throughput Timer
Random Variable
HTTP Request Defaults
User Defined Variables
View Results Tree
Aggregate Report
jp@gc - Transactions per Second
jp@gc - Response Times Over Time
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Text
StringRequest Request
Factorial Request
Fibonacci Request
Square Request
Scroll automatically?
Sampler result
Request
Response data
GET http://localhost:9990/reveggcfusburt
[no cookies]
Request Headers:
Connection: keep-alive
Host: localhost:9990
User-Agent: Apache-HttpClient/4.2.6 (java 1.5)
Raw
HTTP
4:19 PM
12/9/2014

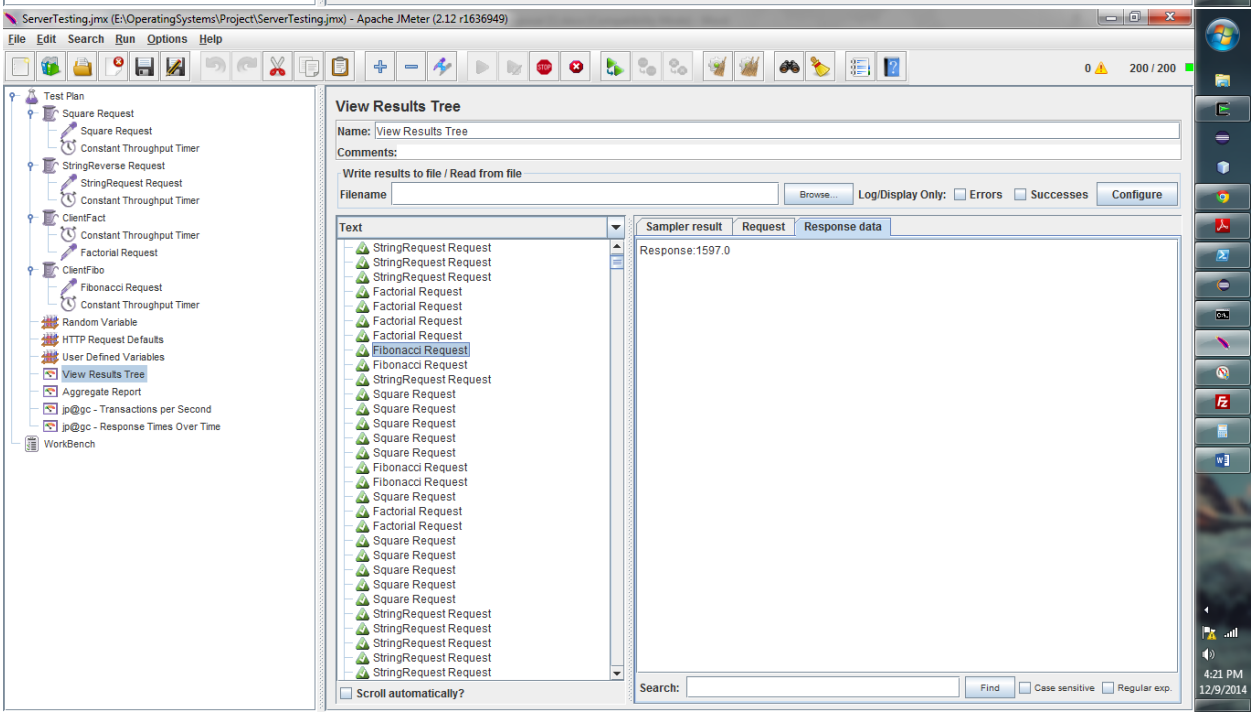

ServerTesting.jmx (E:\OperatingSystems\Project\ServerTesting.jmx) - Apache JMeter (2.12 r1636949)
File Edit Search Run Options Help
0 200 / 200
Test Plan
Square Request
Square Request
Constant Throughput Timer
StringReverse Request
StringRequest Request
Constant Throughput Timer
ClientFact
Constant Throughput Timer
Factorial Request
ClientFibo
Fibonacci Request
Constant Throughput Timer
Random Variable
HTTP Request Defaults
User Defined Variables
View Results Tree
Aggregate Report
jp@gc - Transactions per Second
jp@gc - Response Times Over Time
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Text
StringRequest Request
Factorial Request
Fibonacci Request
Square Request
Scroll automatically?
Sampler result
Request
Response data
Response:trubsufcgg
Search:
Find
Case sensitive
Regular exp.
4:19 PM
12/9/2014

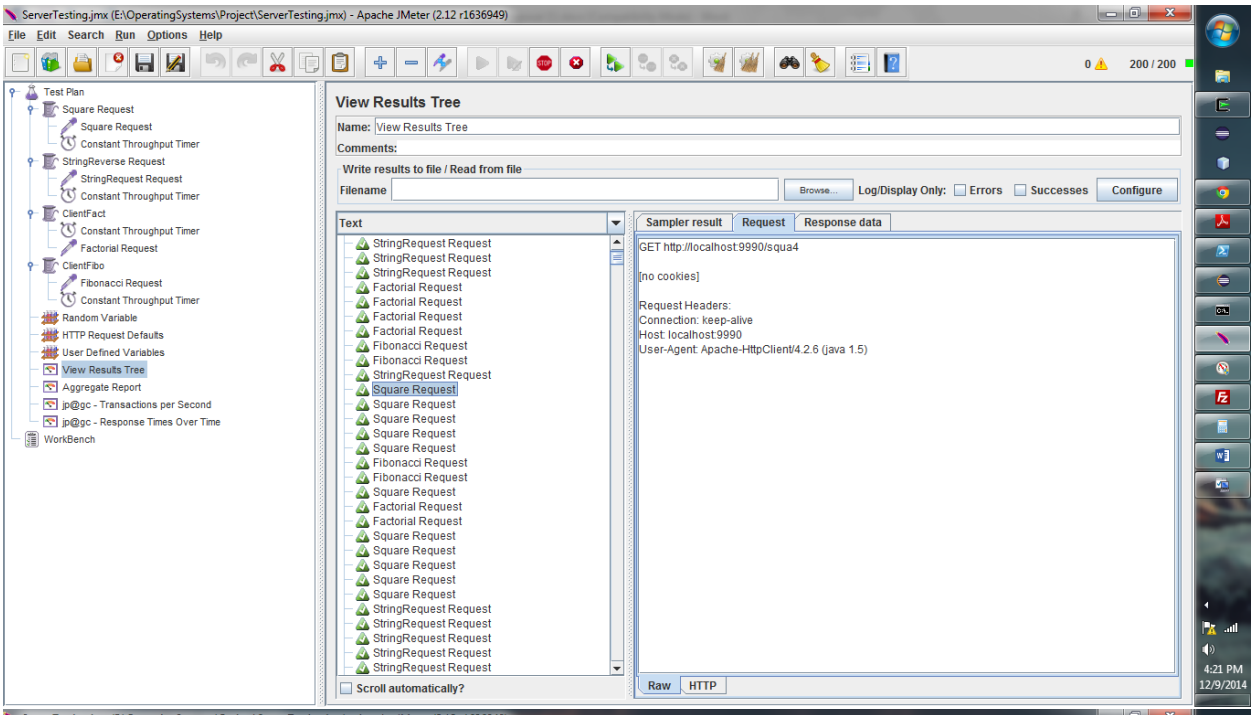
Similarly example request of type Factorial, Fibonacci, Sqaure:

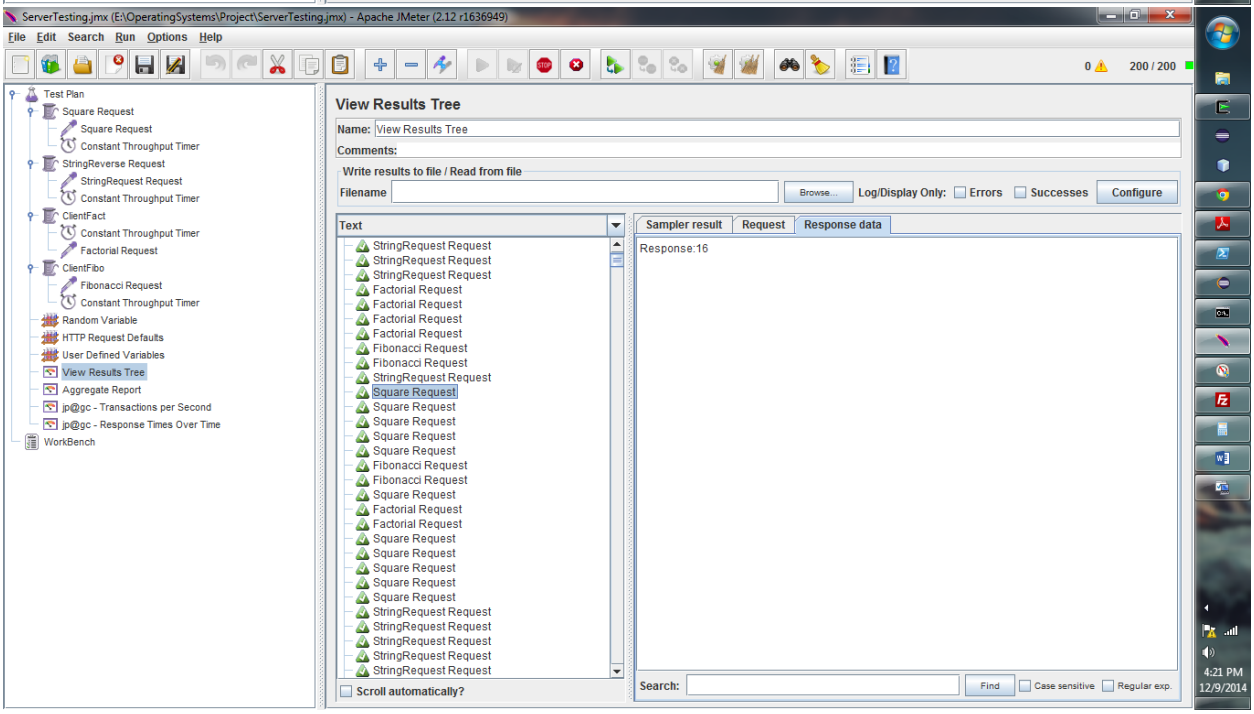




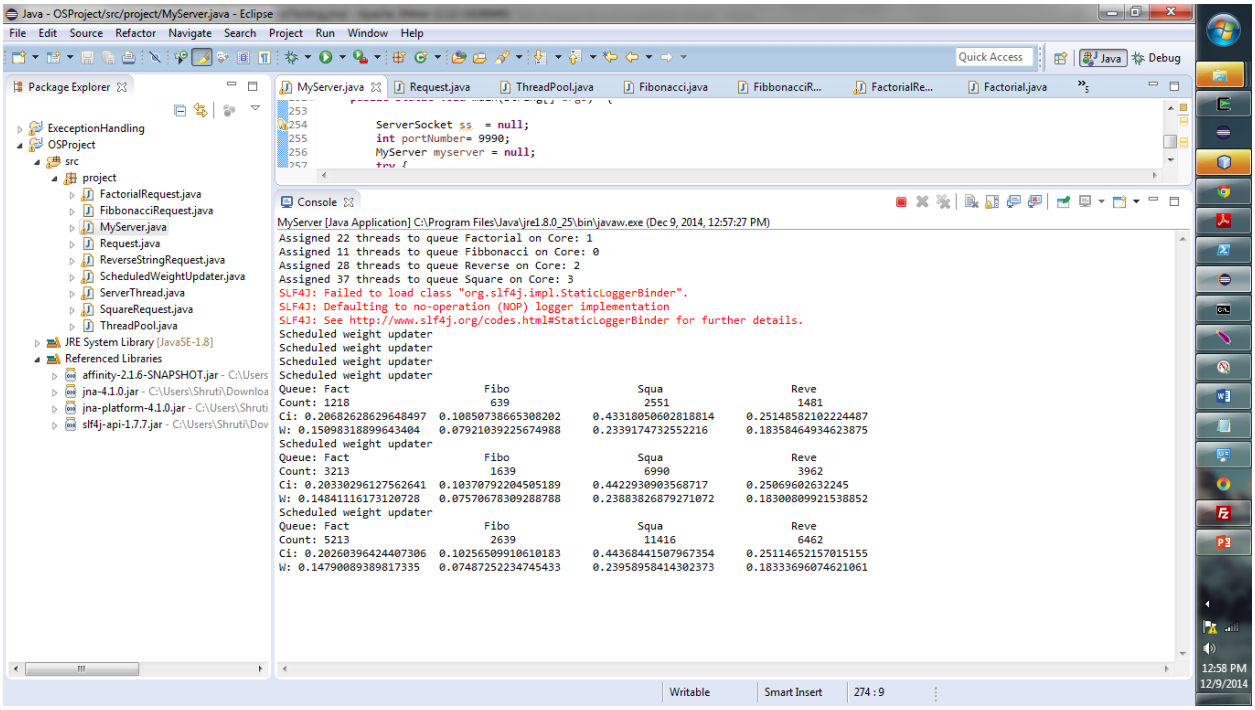

ServerTesting.jmx (E:\OperatingSystems\Project\ServerTesting.jmx) - Apache JMeter (2.12 r1636949)
File Edit Search Run Options Help
0 200 / 200
Test Plan
Square Request
Square Request
Constant Throughput Timer
StringReverse Request
StringRequest Request
Constant Throughput Timer
ClientFact
Constant Throughput Timer
Factorial Request
ClientFibo
Fibonacci Request
Constant Throughput Timer
Random Variable
HTTP Request Defaults
User Defined Variables
View Results Tree
Aggregate Report
jp@gc - Transactions per Second
jp@gc - Response Times Over Time
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Text
StringRequest Request
Fibonacci Request
Square Request
Factorial Request
Sampler result
Request
Response data
GET http://localhost:9990/fibo17
[no cookies]
Request Headers:
Connection: keep-alive
Host: localhost:9990
User-Agent: Apache-HttpClient/4.2.6 (java 1.5)
Scroll automatically?
Raw
HTTP
4:21 PM
12/9/2014


ServerTesting.jmx (E:\OperatingSystems\Project\ServerTesting.jmx) - Apache JMeter (2.12 r1636949)
File Edit Search Run Options Help
0 200 / 200
Test Plan
Square Request
Square Request
Constant Throughput Timer
StringReverse Request
StringRequest Request
Constant Throughput Timer
ClientFact
Constant Throughput Timer
Factorial Request
ClientFibo
Fibonacci Request
Constant Throughput Timer
Random Variable
HTTP Request Defaults
User Defined Variables
View Results Tree
Aggregate Report
jp@gc - Transactions per Second
jp@gc - Response Times Over Time
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Text
StringRequest Request
Fibonacci Request
Square Request
Factorial Request
Sampler result
Request
Response data
Response:1597.0
Search:
Find
Case sensitive
Regular exp.
Scroll automatically?
4:21 PM
12/9/2014

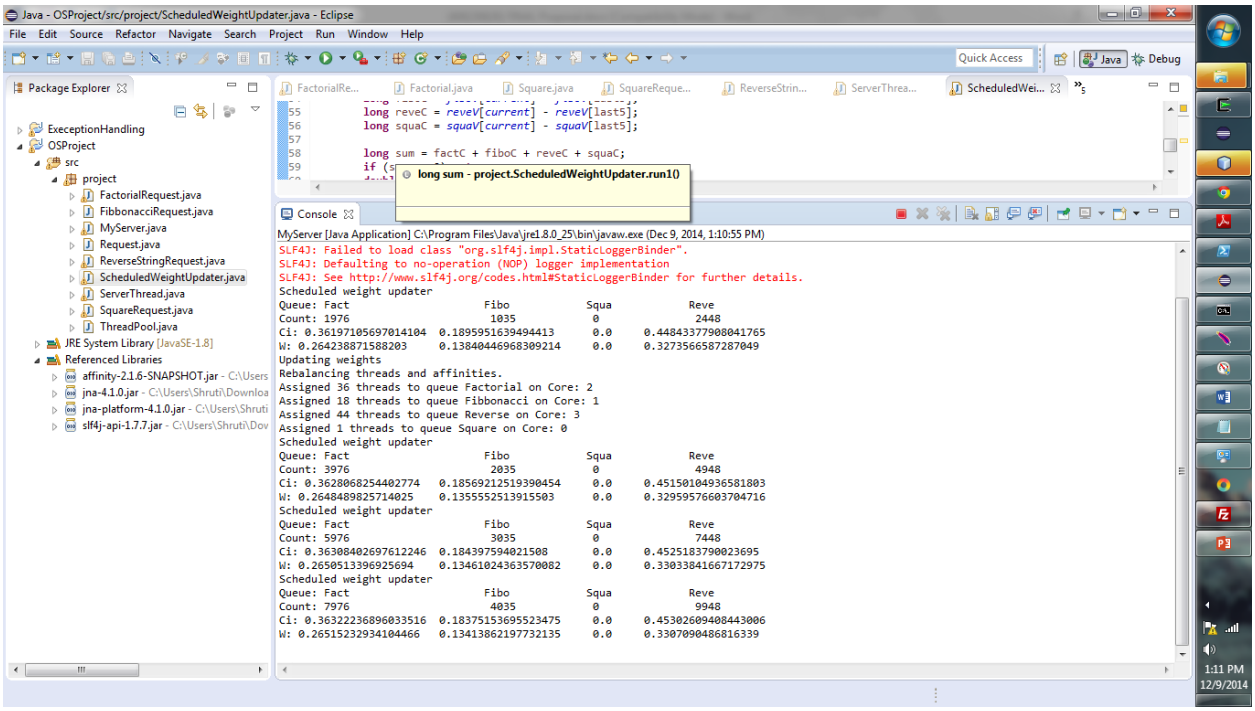

ServerTesting.jmx (E:\OperatingSystems\Project\ServerTesting.jmx) - Apache JMeter (2.12 r1636949)
File Edit Search Run Options Help
0 200 / 200
Test Plan
Square Request
Square Request
Constant Throughput Timer
StringReverse Request
StringRequest Request
Constant Throughput Timer
ClientFact
Constant Throughput Timer
Factorial Request
ClientFibo
Fibonacci Request
Constant Throughput Timer
Random Variable
HTTP Request Defaults
User Defined Variables
View Results Tree
Aggregate Report
jp@gc - Transactions per Second
jp@gc - Response Times Over Time
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Text
StringRequest Request
Square Request
Factorial Request
Fibonacci Request
Scroll automatically?
Sampler result
Request
Response data
GET http://localhost:9990/squa4
[no cookies]
Request Headers:
Connection: keep-alive
Host: localhost:9990
User-Agent: Apache-HttpClient/4.2.6 (java 1.5)
Raw
HTTP
4:21 PM
12/9/2014


ServerTesting.jmx (E:\OperatingSystems\Project\ServerTesting.jmx) - Apache JMeter (2.12 r1636949)
File Edit Search Run Options Help
0 200 / 200
Test Plan
Square Request
Square Request
Constant Throughput Timer
StringReverse Request
StringRequest Request
Constant Throughput Timer
ClientFact
Constant Throughput Timer
Factorial Request
ClientFibo
Fibonacci Request
Constant Throughput Timer
Random Variable
HTTP Request Defaults
User Defined Variables
View Results Tree
Aggregate Report
jp@gc - Transactions per Second
jp@gc - Response Times Over Time
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Text
StringRequest Request
Square Request
Factorial Request
Fibonacci Request
Scroll automatically?
Sampler result
Request
Response data
Response:16
Search:
Find
Case sensitive
Regular exp.
4:21 PM
12/9/2014

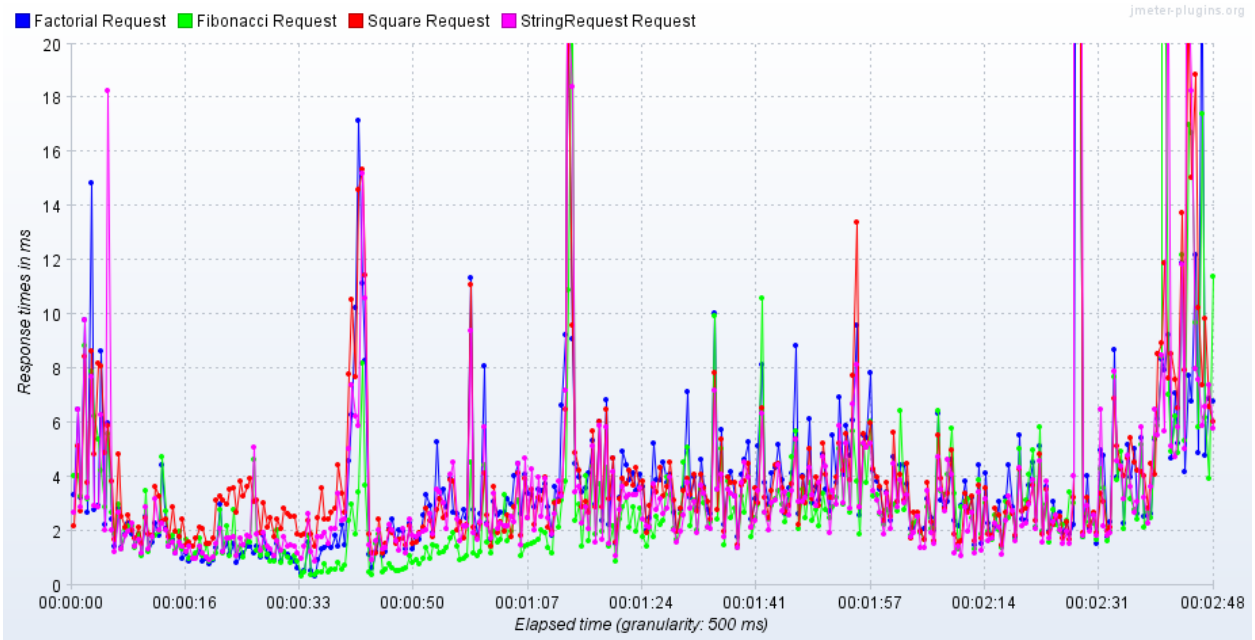
On running the webserver on the modified algorithm, without changing the weights and simply using the above offline log.



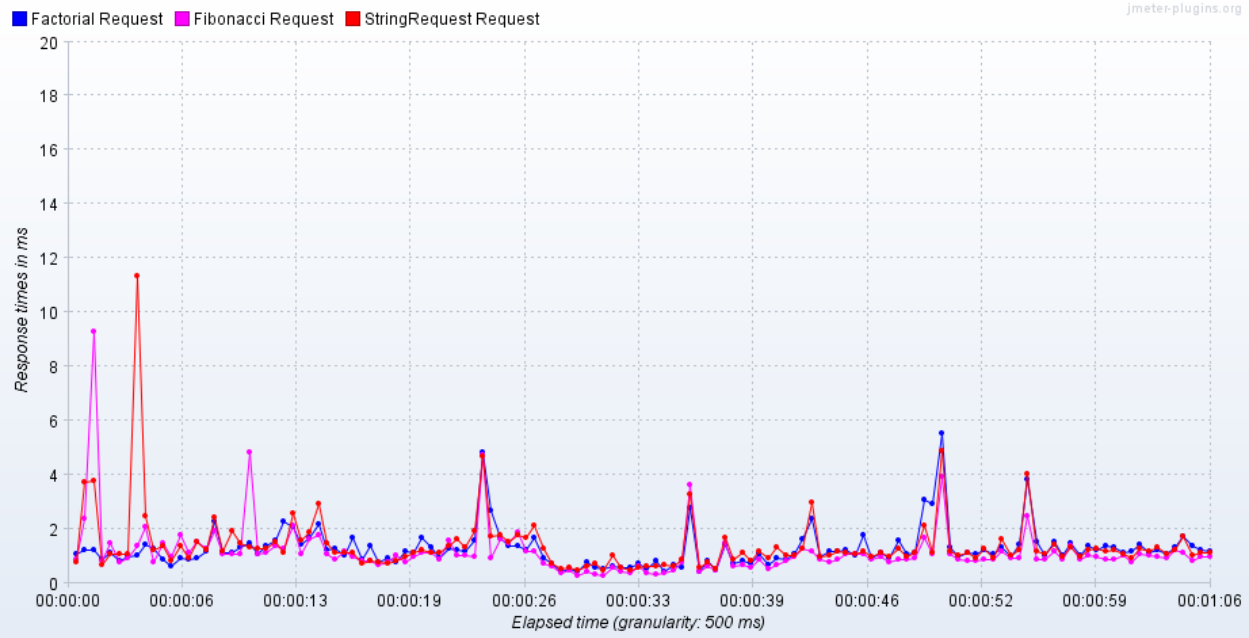
On running the webserver on the modified algorithm, and stopping requests of type square which causes changing the weights.



While running the version where the weights are not updated , i.e. the DRWS given in the paper we referred the mean response time is around 4-5 ms as shown below:



The mean response time when we run the Modified DRWS the mean response time decreases and is now, around 1.5 ms.



8. CONCLUSIONS AND RECOMMENDATIONS:

After running the modified version of the scheduling algorithm we conclude that:

The thread assignment changes if the weights of the queues change. As the weights of the queues are dependent on the frequencies of the incoming requests, we assign a higher weight to the queue that has more entries. These threads are assigned to cores according to the request types. We make sure that there is core affinity for each kind of thread by setting its affinity to the desired core when we instantiate the threads. This ensure that no two threads that share data or belong to the same queue are assigned on different cores, thereby preventing the ping pong affect mentioned earlier.

1. If there occurs a case where the requests of a certain type suddenly drop to zero, all the threads that were assigned to that queue are then transferred to work on a different queue. This is done over a few seconds depending on the size of the window we have used to record the most “t” recent weights.
2. We have also ensured that, depending on the weights of the queue, we achieve load balancing by assigning the queues after the sorting their weights.
3. We can change the number of cores depending on the system from 2-8 and test how the setting the affinity of a thread, reflects in the web server’s operation.
4. We used jmeter, which is in effective tool that simulates the requests we designed in our java class.
5. We have effectively implemented the paper under consideration on Dynamic Requests Scheduling Model in Multi-core Web Servers.

9. BIBLIOGRAPHY

[1] Multi-Core from Intel-Products and Platforms. http://www.intel.com/multicore/products.htm, 2006
[2] AMD Multi-Core Products. http://multicore.amd.com/en/Products/, 2006.
[3] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: a 32-way multithreaded sparc processor," IEEE Micro, vol. 25, pp. 21–29, 2005.
[4] P. M. Gorder, "Multicore processors for science and engineering," IEEE CS and the AIP, vol. 9, pp. 3-7, 2007.
[5] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger, "A hybrid real-time scheduling approach for large-scale multicore platforms," 19th Euromicro Conference on Real-Time Systems (ECRTS'07), IEEE, pp. 247-258, July 2007.
[6] Van der Mei RD, R. Hariharan, P.K. Reeser, "Web Server Performance Modeling," Telecommunication Systems, vol 16, pp.361-378, 2001.
[7] Guohua You, Ying Zhao, "Application optimization research in multicore system," unpublished
[8] Enrique Hernández-Orallo, Joan Vila-Carbó, "Web server performance analysis using histogram workload models," Computer Networks, vol 53, pp. 2727-2739, 2009
[9] Q.Zhang, A.Riska, W.Sun, E.Smirni, G.Ciardo, "Workload-aware load balancing for clustered web servers," IEEE Transactionson Parallel and Distributed Systems, vol 16, pp.219 233, 2005
[10] Saeed Sharifian, Seyed A. Motamedi, Mohammad K. Akbari, "A content-based load balancing algorithm with admission control for cluster webservers," Future Generation Computer Systems, vol 24, pp. 775-787, 2008
[11] Apache.The Apache Software Foundation. http://www.apache.org
[12] M.E. Crovella, R. Frangioso, M.Harchol-Balter, "Connection scheduling in web servers," Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, pp. 243-254, 11-14 October 1999.
[13] L. Cherkasova, "Scheduling strategy to improve response time for web applications," In Proceedings on High Performance Computing and Networking, vol 1401, pp.305–314, April 1998.
[14] Rachid El Abdouni Khayari, "A new scheduling algorithm for WWW, validation and comparison by trace-driven simulation," Society for Computer Simulation, 2004
[15] A. Chonka, W. Zhou, K. Knapp, and Y. Xiang, "Protecting information systems from DDoS attack using multi-core methodology," Proceedings of the IEEE 8th International Conference on Computer and Information Technology, pp. 270–275, 2008.
[16] Y. Lu, J. Tang, J. Zhao, and X. Li, "A case study for monitoringoriented programming in multi-core architecture," IWMSE 08 Proceedings of the 1st international workshop on Multicore software engineering, pp. 47-52, May 2008.
[17] R. Islam, W. Zhou, Y. Xiang, and A. N. Mahmood, "Spam filtering for network traffic security on a multi-core environment," Concurrency Computat.: Pract. Exper., vol.21, pp. 1307-1320, 2009.
[18] A. Chonka, S. K. Chong, W. Zhou, and Y. Xiang, "Multi-core defense system (MSDS) for protecting computer infrastructure against DDoS attacks," Proceedings of the 2008 Ninth International Conference on Parallel and Distributed Computing, pp. 503-508, Dec. 2008.
[19] A. Chonka, W. Zhou, L. Ngo, and Y. Xiang, "Ubiquitous multicore(UM) methodology for

multimedia," International Journal of Multimedia and Ubiquitous Engineering, vol. 4, pp.145-156, April 2009.
[20] M. Andreolini, M. Colajanni, M. Nuccio, "Scalability of content-aware server switches for cluster-based web information systems," Proc.of 12th International World Wide Web Conf. , May 2003.
[21] Rachid El Abdouni Khayari, "Class-based weighted fair queueing: Validation and comparison by trace-driven simulation," International Journal of Communication Systems, vol 18, pp. 975-994, 2005
[22] S. C. Borst, O. J. Boxma, R. Nunez Queija, "Heavy tails: The effect of the service discipline," In Computer Performance Evaluation-Modelling Techniques and Tools, vol.2324, pp.1–30, 2002.
[23] S. C. Borst, O. J. Boxma, R. Nunez Queija, A. P. Zwart, "The impact of the service discipline on delay asymptotics," Performance Evaluation, vol. 54(2), pp.175–206, 2003.
[24] M. Crovella, A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," IEEE–ACM Transactionson Networking, vol.5(6), pp.835–846, 1997;.
[25] El Abdouni Khayari, R. Sadre, B. Haverkort, "The pseudo-self-similar traffic model: application and validation," Performance Evaluation, vol. 56(1-4), pp.3-22, 2004.
[26] W. Willinger, M. Taqqu, A.Erramilli, "A bibliographical guide to self similar traffic and performance modeling for modern high-speed networks," In Stochastic Networks: Theory and Applications, pp.339–366, 1996.