

# **LOAD BALANCING OF WEB SERVERS USING FUZZY INFERENCE**

**Rodney Keenan**

**Bachelor of Science (Honours)**

**Monash University**

**Supervisor: Associate Professor Bin Qiu**

## **ABSTRACT**

In order to deal with the increasing loads being placed on web sites, clusters can be used. Servers can be added to support the existing servers in the cluster, and any changes in cluster configuration appear transparent to the client. This requires a method to balance the traffic load amongst the servers in the cluster. Various techniques have been developed and applied, each have their own limitations and are relatively simple. Efficient load balancing can be achieved by an effective measure of server power and dynamic load.

Round robin distributes incoming client jobs to servers in a sequential manner until it reaches the last server, when it starts again at the first server. Least connection gives the client job to the server with the least number of open connections (number of client jobs). Weighted distribution gives jobs to servers based on their 'weight' or power. These three techniques were developed and implemented within a simulation. The simulation was also used to test a fuzzy inference system that relies on fuzzy logic to choose a server for the next client job based on two inputs: queue length and CPU power.

Using the main performance measures of response time and job rate, the fuzzy inference method was found to be the most stable across a number of different traffic models, providing a low average response time and a high job rate. This can be attributed to its reliance on queue length to deal with changes in network load and the ability to choose more powerful servers when the load across the cluster is similar.

# Table Of Contents

1 INTRODUCTION .....	6
1.1 Server Clusters & the Internet .....	6
1.2 Load Distribution Across Servers .....	6
1.2.1 Classification of Load Balancing .....	7
1.2.2 Static and Dynamic Load Balancing .....	7
1.3 Important Factors in Load Balancing .....	7
1.4 Methods of Load Distribution .....	8
1.5 Fuzzy Logic .....	8
1.5.1 Fuzzy Logic Applications in Load Distribution .....	9
1.6 Thesis Outline .....	9
2 RESEARCH APPROACH .....	11
2.1 Project Description .....	11
2.2 Design of simulation .....	11
2.2.1 Assumptions made .....	11
2.2.2 Network Configuration .....	12
2.2.3 Traffic Models .....	12
2.3 Other possible methods .....	13
2.3.1 Implementation on server cluster .....	13
2.4 Project Implementation .....	13
2.4.1 Components .....	13
2.4.1.1 Traffic Model 1 .....	14
2.4.1.2 Traffic Model 2 .....	14
2.4.1.3 Traffic Model 3 .....	15
2.4.2 Simulation operation .....	15
2.4.2.1 Round Robin .....	16
2.4.2.2 Least Connection .....	16
2.4.2.3 Weighted Distribution .....	16
2.4.2.4 Fuzzy Inference System .....	16
2.4.2.5 Results outputted .....	16
2.4.3 Implementation of the Fuzzy Inference System .....	17
2.4.3.1 Inputs .....	17
2.4.3.2 Output .....	20
2.4.3.3 Rules .....	20
3 RESULTS AND DISCUSSION .....	22
3.1 Server Configurations .....	22
3.1.1 Same power .....	24
3.1.2 Increasing power .....	25
3.1.3 Random .....	27
3.2 Number of Servers .....	29
3.2.1 Two servers .....	29
3.2.2 Eight servers .....	31
3.2.3 Sixteen servers .....	33
3.3 Traffic Patterns .....	35
3.3.1 Traffic Pattern 1 – Near constant arrivals .....	35
3.3.1.1 Base Traffic Pattern 1 .....	35
3.3.1.2 Low variability in job arrivals .....	38
3.3.1.3 High variability in job arrivals .....	40
3.3.1.4 Low job arrival rate .....	42
3.3.1.5 High job arrival rate .....	44

3.3.2 Traffic Pattern 2 – Burst in arrival rates .....	46
3.3.2.1 Base Traffic Pattern 2 .....	46
3.3.2.2 Small chance of burst .....	49
3.3.2.3 High chance of burst .....	51
3.3.2.4 Short bursts .....	53
3.3.2.5 Long bursts .....	55
3.3.3 Traffic Pattern 3 – Burst in job requirements .....	57
3.3.3.1 Base Traffic Pattern 3 .....	57
3.3.3.2 Small chance of burst .....	60
3.3.3.3 High chance of burst .....	62
3.3.3.4 Small bursts .....	64
3.3.3.5 Large bursts .....	66
3.4 Performance Results .....	68
4 CONCLUSIONS .....	69
4.1 Future Work .....	69
5 BIBLIOGRAPHY .....	70

## Tables and Figures

Table 1- Same Power Server Configuration .....	24
Table 2 - Increasing Power Server Configuration.....	25
Table 3 - Random Power Server Configuration.....	27
Table 4 - Two server configuration.....	29
Table 5 - Eight Server Configuration.....	31
Table 6 - Sixteen Servers Configuration.....	33
Table 7 - Fixed configuration for traffic comparison.....	35
Figure 1 - Server Cluster Configuration.....	12
Figure 2 - Overview of the fuzzy inference system .....	18
Figure 3 - Queue length membership functions.....	19
Figure 4 - CPU power membership functions.....	19
Figure 5 - Output functions for fuzzy inference system.....	20
Figure 6 - Rules for fuzzy inference system .....	21
Figure 7- Traffic Condition 1 (Different Server Configurations) .....	23
Figure 8 - Results (Same power).....	24
Figure 9 - Results (Increasing Power).....	26
Figure 10 - Results (Random Power).....	28
Figure 11 - Results (two servers) .....	30
Figure 12 - Results (Eight Servers).....	32
Figure 13 - Results (Sixteen Servers).....	34
Figure 14 - Base Traffic Pattern 1 .....	36
Figure 15 - Results (Base Traffic Pattern 1) .....	37
Figure 16 – Traffic Pattern 1 (Low variability in arrival rates) .....	38
Figure 17 - Results (Low variability in arrival rates).....	39
Figure 18 - Traffic Pattern 1 (High variability in arrival rates) .....	40
Figure 19 - Results (High variability in arrival rates) .....	41
Figure 20 - Traffic Pattern 1 (low arrival rates).....	42
Figure 21 - Results (Low arrival rates) .....	43
Figure 22 - Traffic Pattern 1 (high arrival rate) .....	44
Figure 23 - Results (High arrival rates).....	45
Figure 24 – Base Traffic Pattern 2 .....	47
Figure 25 - Results (Base traffic pattern 2).....	48
Figure 26 - Traffic Patter 2 (small chance of burst).....	49
Figure 27 – Results (small chance of burst).....	50
Figure 28 - Traffic Pattern 2 (high chance of burst) .....	51
Figure 29 - Results (high chance of burst) .....	52
Figure 30 - Traffic Pattern 2 (short bursts) .....	53
Figure 31 - Results (short bursts) .....	54
Figure 32 - Traffic Pattern 2 (long bursts) .....	55
Figure 33 - Results (long bursts).....	56
Figure 34 - Base Traffic Pattern 3 .....	58
Figure 35 - Results (base Traffic Pattern 3).....	59
Figure 36 - Traffic Pattern 3 (small chance of burst).....	60
Figure 37 - Results (small chance of burst).....	61
Figure 38 – Traffic Pattern 3 (high chance of burst).....	62
Figure 39 - Results (high chance of burst) .....	63
Figure 40 - Traffic Pattern 3 (small burst size) .....	64
Figure 41 - Results (small burst size).....	65

Figure 42 - Traffic Pattern 3 (large burst size)..... 66  
Figure 43 - Results (large burst size) ..... 67

# 1 INTRODUCTION

With the rapid growth and use of the Internet, the technology used in running the world's largest network has had to develop at a similar pace. One of the main problems faced when hosting a web site or offering a service on the Internet is the potential for large amounts of client traffic to clog up web servers, resulting in a drop in the Quality of Service (QOS) and response times. One option is to upgrade the servers with more powerful machines, but this comes at a considerable cost.

A practical solution lies in the use of 'server clusters' (groups of servers connected and working together). The methods of evenly distributing the client traffic amongst the cluster are known as 'load balancing'. In general, "load balancing tries to ensure that every processor in the system does almost the same amount of work at any point of time"[20]. This method is not an exact science, and there are many different approaches and ideas on what makes an effective algorithm. The uncertainty inherent in trying to predict the loads and availability of the servers can be taken into account with the mathematical tool of fuzzy logic [19].

## 1.1 Server Clusters & the Internet

By joining a group of servers together to act as one single unit, considerable benefits can arise. Servers can be added or removed depending on the amount of web traffic that is present. This scalability is the main advantage of clustering [1, 2].

A load balancer can be added to the cluster to manage the incoming traffic (a centralized policy) or a load distribution algorithm can be placed on each server in the cluster, so that web traffic is redirected through the cluster links (a distributed policy) [3]. Centralized policies reduce the total traffic amongst the cluster but may create a bottleneck at the central load balancer [3]. On the other hand, a distributed policy can speed up the decision making process, but requires a lot of communication overhead between servers in the cluster [3].

## 1.2 Load Distribution Across Servers

Once a server cluster is created, there must be an algorithm for the distribution of traffic amongst the servers.

A load balancing algorithm can be based on 3 policies [19].

1. Information policy – specifies amount of load information made to, and the way this information is distributed among, the job placement decision-makers.
2. Transfer policy – determines the suitability of a process for task relocation; whether or not a task is eligible to be transferred to a node. This is mainly applicable in parallel processing distributed systems, but can be applied when deciding whether a not a node can accept the job on offer.
3. Placement policy – decides the node to which a job should be transferred. This refers to the actual implementation of the algorithm, taking into account the information and transfer policy decisions.

## 1.2.1 Classification of Load Balancing

Once the details of the load balancing algorithm are decided, the next step is to determine whether the traffic flow can be classified as either sender or receiver initiated [3, 4, 5, 8, 20].

A task is sender initiated if the local host or source node determines where a task is to be executed. This is most commonly used with a centralised policy, where the load balancer gives jobs to the servers. A task is receiver initiated if the node decides which jobs at different sources it will process. A receiver initiated form of job completion is usually implemented with a distributed policy, where each server chooses to run jobs that are being held at other servers.

Receiver initiated algorithms outperform sender initiated algorithms using the same information. Most server-initiative algorithms do not allow a server to become idle when there are jobs waiting in the system [5].

## 1.2.2 Static and Dynamic Load Balancing

Load balancing algorithms can also be grouped into static or dynamic algorithms [2, 3, 6]. Static algorithms use knowledge of the existing attributes of the servers to make distribution decisions. Processor power, total memory capacity and maximum queue length have all proven useful in generating static allocation algorithms [19]. A static algorithm may also include past information about the average behaviour of the cluster. However, any past overall information will not help in overcoming the uncertainty and sudden bursts in traffic that occur throughout the Internet.

However, with dynamic algorithms the load balancer uses knowledge of the current state of the nodes to make distribution decisions, such as server queue length, remaining processor power or remaining memory [2]. Adaptive algorithms are a special case of dynamic algorithms. They adapt their activities by changing (at run time) their parameters, or even their policies, to suit changing system states [6].

While dynamic methods are proven to offer a better prediction of server load and therefore a more effective load balancer, [4, 5, 6, 7] the major disadvantage of dynamic load balancing is the runtime overhead incurred [3]. When more factors are taken into consideration, the runtime overhead of the load balancing algorithm increases.

## 1.3 Important Factors in Load Balancing

Whether the algorithm is static or dynamic, the ‘load index’<sup>1</sup> should be an accurate prediction of the performance of a task if it is executed at a particular server and should correlate with task response time [6]. Algorithms with little overhead will be achieved by effective load index measures and efficient load distribution methods [3].

Simple load balancing algorithms that use a small amount of state information in simple ways yield dramatic performance improvement relative to the no load sharing case [8].

---

<sup>1</sup> A load index is a value which represents the power or resources that a server has. It can be used to compare servers in a cluster.

A number of factors can be used in evaluating the load index of a server [7, 9, 10, 11]

- Number of open connections on a server
- Maximum number of connections a server can have
- Time left in completing current job
- Processor's current usage/total power
- Memory usage/total memory
- Size of incoming request
- Speed of network
- Relative processing speeds of servers in the cluster
- Number of servers in cluster

Some of the previously mentioned factors can be determined statically; others are evaluated at run-time dynamically. The choice of load index has a considerable effect on load distribution performance, and the most effective measurement is the queue length (or number of connections the server has) [12, 19].

Load distribution and the use of a load index improve upon methods like Round robin (mentioned in section 5), no matter how simple the index [2, 12, 14]. Even when all the computers in a group are equally powerful and have equally heavy workloads over time, Linvy and Melman [13] have shown that without load distribution, at least one computer is likely to be idle while other computers are heavily loaded because of statistical fluctuations in the arrival of tasks to computers and server response time requirements.

## 1.4 Methods of Load Distribution

How the load index and the specific factors are used depends on the method of load distribution implemented. The following are commonly used [2, 14, 15, 26]:

- Round robin – jobs are allocated to servers from server1 through to the last server, then back to server1.
- Least connection – the incoming job is given to the server with the least number of open connections.
- Weighted Distribution – jobs are given to servers in proportion to their weight(or power).
- Weighted Least Connection – servers are allocated jobs at a rate based on their weight and on how many connections they have open.

The last three can be implemented statically or dynamically, with the weight and queue length being a fixed amount for a static approach, or as periodic updated values for a dynamic approach [6, 14].

## 1.5 Fuzzy Logic

Fuzzy logic was developed by Zadeh [16] and represents a form of mathematical logic. Values between 0 and 1 represent uncertainty in decision-making. 0 indicates a false value while 1 indicates a true value. So within a fuzzy set a value  $x$  is not restricted by the values 0 or 1, but from the real interval  $\{0;1\}$  [23]. Terms in the fuzzy set are given linguistic variables. Values

such as big, small, tall, short are mapped as functions on a graph. The value of the linguistic variable indicates the strength of the term. A sample can belong to more than one term in a fuzzy set (and this is where the modelling of uncertainty arises).

The uncertainty in predicting future loads on servers makes fuzzy inference a natural tool in the development of a load balancing algorithm [19]. Fuzzy logic has had a limited application in load distribution, although recent work shows a change in trend.

### **1.5.1 Fuzzy Logic Applications in Load Distribution**

Zalinda, [17] applied fuzzy inference to choose robots based on their reliability (which is a difficult thing to measure). The system balances and monitors the work-load presented to the robots – the scheduling of work could change when the shop conditions change or machines breakdown or are removed for maintenance. (much like a load balancer filters the traffic to its servers). Fuzzy logic was used to model the reliability and efficiency of the robots under the various conditions.

Shaout [18] used fuzzy load balancing for a small database driven network and proved to have reduced wait times and total times, although overall, the average run time per job was higher than a non-fuzzy approach. The fuzzy system worked especially well for high priority traffic (traffic that demanded a quick response time).

Dierkes [23] realised the uncertainty that was present in all load balancing algorithms and developed a fuzzy set to estimate the impact of the algorithm's estimation of the current cluster state. By using Mamdam's fuzzy inference [27] in his decision algorithm, Dierkes was able to map attributes such as queue length onto a fuzzy set with three terms (small medium and large) over specific ranges. In another paper [24], Dierkes based load balancing decision making on a set of possible actions, a set of goals with priorities, and an effect matrix. The effect matrix showed the effect of the actions on the goals (whether the action either distracts or supports a goal). The application of fuzzy sets taken from the matrix significantly improved system performance, with a bias towards achieving the goals given weight in the effect matrix.

Park [25] developed a system that was said to reflect “the imprecision in state information and makes scheduling decisions based on fuzzy logic”. Fuzzy states were used to model uncertainty, mainly in the system load and message propagation measurements. Each scheduler had two functions, threshold estimation and decision making. The two thresholds indicated the amount of variation between its current load and the servers around it; and the amount of variation between its current load and its external load (what it was working on and what it was yet to work on). The algorithm achieved an improved mean response time over conventional algorithms as well as a low constant message overhead cost.

## **1.6 Thesis Outline**

The purpose of this research is to develop and implement three common load balancing methods (round robin, least connection and weighted distribution) as well as a novel fuzzy inference system. The design and operation of the simulation is described below, followed by an explanation of the fuzzy inference system. The performance results for each algorithm are displayed and compared under a number of different cluster configurations and traffic conditions. A general discussion of the individual algorithms is then offered, followed by a

conclusion on the success of the fuzzy method and the future direction that the research could take.

## **2 RESEARCH APPROACH**

The main emphasis of this research was to compare a developed fuzzy inference model to existing load balancing techniques. This was achieved by using a simulation to compare the performance of these existing techniques to the fuzzy algorithm under a number of different traffic conditions.

### **2.1 Project Description**

A simulation was developed and implemented in Matlab 5.3, a programming language similar to C with a number of statistical and graphical enhancements, as well as a fuzzy inference system development tool. The simulation works by generating one of three possible traffic models. Each algorithm reads in this traffic during a set time period and outputs results based on a number of different performance measures. These results are gathered and displayed in the form of a graph where they can be compared and the algorithms can be evaluated. The design and operation of the simulation is given in 2.2 (Design of Simulation), and a detailed description of the implementation of specific components is given in 2.4 (Project Implementation).

### **2.2 Design of simulation**

Three different traffic models were designed with the purpose of testing the load balancing methods in a variety of conditions. Each traffic model ran for a specified number of time loops (no units were used). At each time increment a number of jobs were generated. This number refers to the amount of jobs arriving at the load balancer at one time instant.

Every job arriving at the load balancer was given three attributes: CPU cycle time, memory needed and hard drive access time. Consequently, each server in the cluster was also given three attributes: CPU speed, memory size and hard drive speed. Once the traffic was generated, each algorithm was tested. The algorithms only differed in the method they used to decide which server should get the incoming client job. When all the jobs for a time increment were given to the server cluster, the servers worked on the jobs in their memory exactly the same way for all algorithms.

Initially a job is placed in the CPU queue of a server and allocated memory. Once the CPU has worked on the job, it is moved into the hard drive access queue. When all hard drive access time is carried out, the job is given back to the client. For a more detailed explanation, please see 2.4.2 (simulation operation).

At each time increment (for each algorithm), all servers were polled for performance data, such as CPU use, memory use, jobs completed, number of jobs being processed, etc. Additionally, each individual job was given a time stamp to record how long it took the job to be completed (from the time it was given to the server). This is a measure of response time. More information is given under heading 2.4.2.5 (results outputted).

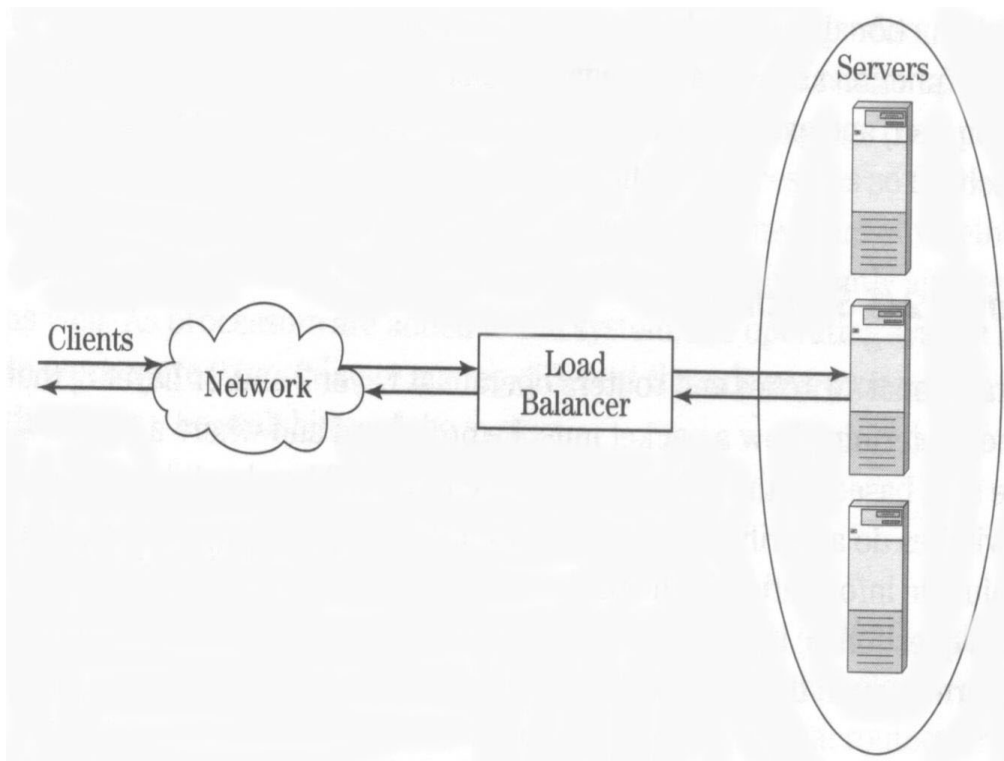
#### **2.2.1 Assumptions made**

In order to achieve what was proposed in this thesis, a number of assumptions and simplifications had to be made.

- The modelling of realistic web traffic and computer operation is an extremely complex task. The simulation relies on performance comparisons between the algorithms, not on accurate real world results.
- Since the running of the simulation is computationally intensive the standard time increment size used is 2000 loops of web traffic. Investigation has shown that an increase in the length of a simulation run does not affect the results significantly.
- The speed or efficiency of the algorithm is not taken into account when using the performance results. Dynamic methods like least connection and fuzzy inference will have a greater run time and this should be taken into consideration when evaluating the results.
- Any delays caused by the load balancer are considered negligible and not factored into the results.

## 2.2.2 Network Configuration

The basic configuration used by the simulation is a front-end load balancer that is connected to the server cluster and used as a gateway between the servers and the client traffic. This load balancer is dedicated to distributing the client requests to the servers (see Figure 1).



**Figure 1 - Server Cluster Configuration**

The role of the load balancer is to read in the client traffic generated at a time instant and to distribute it to the servers. This is implemented in the simulation by four different functions, each using the four load balancing techniques investigated in this research (see 2.4.1 for an explanation).

## 2.2.3 Traffic Models

Traffic generation is carried out by three separate files:

1. traffic\_generator1.m: generates a smooth arrival rate traffic and randomly distributed job requirements
2. traffic\_generator2.m: generates bursts in arrival rate traffic with randomly distributed job requirements
3. traffic\_generator3.m: generates a smooth arrival rate with random bursts in individual job requirements.

These traffic models are controlled by parameters than can be altered to affect things such as the chance of a burst happening, the variability in arrival rates, the size of a burst, etc. This enabled the algorithms to be compared against each other in a number of different traffic conditions. Each individual file is discussed in 2.4.1 (Components).

## 2.3 Other possible methods

### 2.3.1 Implementation on server cluster

As previously noted, the simulation makes a number of assumptions and simplifications. In order to get solid, verifiable results, the testing and implementation could have been carried out on an actual web server cluster. This removes all inaccuracies and gives a realistic measure of performance. However this method had a few drawbacks:

- It would have been very difficult to gain access to a server cluster being used as a web server, especially when it would be used to test something as operationally critical as load balancing.
- It lacked flexibility in testing. The simulation allowed an easy modification of server number or configuration, as well as on-demand generation of particular types of traffic.
- A system level implementation would have been difficult to develop and maintain, as it would have required recompiling and tweaking of low-level code. The high level language used by Matlab was easy to modify and debug.

## 2.4 Project Implementation

The simulation can be separated into two parts.

1. Generation of web traffic
2. Reading, distribution and processing of the web traffic

Traffic is generated by one of three different traffic models. This is written to a file called 'load\_traffic.m'. This file first specifies how many time loops the simulation runs for, and then lists the web traffic for each time loop. This file is read by 'main.m', which reads the file four times. Firstly it carries out round robin distribution, then least connection, then weighted distribution, then fuzzy inference.

### 2.4.1 Components

The simulation uses 'main.m' to call the following functions:

roundrobin()	gives jobs to servers in a sequential manner.
leastconnection()	gives jobs to server with minimum queue length (combination of CPU and hard drive queue)
weightedconnection()	gives jobs to servers in proportion to their CPU powers
fuzzy_inference()	uses the fuzzy inference system to choose the best server for a client job

For each time loop and algorithm, 'main.m' also calls these functions:

schedulejobs()	records memory and CPU use, carries out CPU work, updates response time values.
updateload()	moves jobs from CPU queue to hard drive queue if they have finished CPU use.
accesshd()	works on jobs in the hard drive queue, records response times for finished jobs.

The contents of 'load\_traffic.m' are dependent on the choice of traffic model and the parameters used within those models.

traffic\_generator1.m, traffic\_generator2.m & traffic\_generator3.m create the web traffic which drives the simulation.

### 2.4.1.1 Traffic Model 1

Traffic Model 1 is guided by the following parameters

- time\_loops: determines the length of simulation run. This has been fixed at 2000 for all testing.
- job\_range\_mean: average number of jobs arriving per time loop. Normally distributed. By default was set at 8.
- job\_range\_sd: standard deviation of job arrival. By default was set at 1.
- mean1: average CPU cycles needed by a job. Randomly distributed (poisson). By default was set at 500.
- mean2: average memory needed by a job (in Kbytes). Randomly distributed (poisson). By default was set at 2000.
- mean3: average hard drive access time needed by a job. Randomly distributed (poisson). By default was set at 50.

### 2.4.1.2 Traffic Model 2

Traffic Model 2 uses all the parameters as Traffic Model 1 (and the defaults) as well as these additional values:

- burst\_range: indicates the chance of a burst happening. For eg, a value of 500 indicates that around 1 in every 500 time loops will cause a burst in arrival rates.

If a burst occurs, the following four values are used:

- burst\_base1: the minimum job burst period (in time loops)
- burst\_range1: the possible range of the job burst period (on top of the minimum burst period). For eg, a burst\_base1 = 5 and burst\_range1 = 30, gives a possible period of 5–35. Note: these values are used when a burst in arrival rates is triggered.
- burst\_base2: the minimum jump in arrival rates if a burst occurs.
- burst\_range2: the possible range of the arrival rate burst on top of the base.

Traffic Model 2 keeps the same random distribution in job requirements as Traffic Model 1, but attempts to simulate the fluctuations in web traffic that can occur during peak times.

### 2.4.1.3 Traffic Model 3

Traffic Model 3 contains all the parameters that Traffic Model 1 has (and the defaults) as well as these additional values.

- range1: chance of a burst in a job's CPU cycle time occurring
- range2: chance of a burst in a job's memory requirements occurring
- range3: chance of a burst in a job's hard drive access needs occurring

For eg, if range1 = 1000, then approximately 1 in a thousand jobs will require a lot more CPU cycle time than any other jobs. The burst is also influenced by the following variables:

- et: multiplying factor for a CPU cycle time burst
- em: multiplying factor for a memory requirement burst
- eh: multiplying factor for a hard drive access burst

These factors are randomly distributed around a mean. For eg, if the value of et is 10, then a burst in CPU cycle time will be approximately 10 times a normal jobs CPU time.

Traffic Model 3 keeps the relatively smooth arrival rate of Traffic Model 1, but creates bursts in job requirements that are representative of large file downloads, network synchronisation, cgi script execution, etc.

## 2.4.2 Simulation operation

The generated traffic file 'load\_traffic.m' is read by 'main.m' to determine how many time loops there are. Then four simulation runs are carried out, each differing only in the algorithm used for client traffic distribution. In all simulations, jobs that are given to a server are first placed in the server's CPU queue and allocated memory. All jobs in the CPU queue are processed by an equal amount. For eg, if there are three jobs in a CPU queue and each have a required cycle time of 500, then a server with a CPU power of 900 would be able to complete 300 cycles for each job in one time loop. If there was only one job with 500 cycles needed, then the CPU would complete all 500 cycles and move the job to the hard drive queue.

Once a job has finished its CPU cycle time, it gets moved to the hard drive access queue. Unlike the CPU queue, which spreads its power equally over all jobs, the hard drive queue processes one job at a time and is a first in last out (FIFO) system. A computer with a hard drive speed of 250 and eight jobs in its queue (each with access of 50 needed), would be able to finish the first five jobs in one time loop. The rest get moved up to the front of the queue and wait until the next time increment.

When a job is finished in the hard drive queue, it is deleted (in effect it is sent back to the client). This also means the memory can be deallocated. If a server does not have enough memory to handle all the jobs in its CPU and hard drive queue, an overload occurs and a paging delay is introduced. This slows down the computer to approximately 67% of its original power.

Note that the two different queuing systems can get congested in two different ways. If the CPU has a lot of jobs to work on, its cycle time will be spread thinly and jobs will take a long time to be completed. This implementation was chosen due to the multitasking nature of a CPU. If the hard drive queue contains too many jobs then the queue will grow in size because the hard drive can only finish a certain amount of jobs per time loop. This aspect of the system is not confined to a hard drive, it could be used to represent any sort of network resource that requires sequential processing.

### **2.4.2.1 Round Robin**

For each time loop, there will be a number of jobs arriving that need to be distributed to the servers in the cluster. `roundrobin()` works by setting a counter to server 1, then incrementing it for each job it gives to a server. When the counter is greater than the number of servers, it is reset to 1.

### **2.4.2.2 Least Connection**

When deciding which server should get the next job, `leastconnection()` starts by randomly selecting a server out of the cluster. It then looks at the total queue lengths of all servers, and when it finds a server with a smaller queue length, it chooses that server as the chosen server. After looping through all the servers, the chosen server will be the one with the minimum queue length. This process is completed for every client job.

### **2.4.2.3 Weighted Distribution**

`weightedconnection()` works by gathering the CPU speed of all the servers and calculating their relative ratios. For eg, if server 1 = 2000 and server 2 = 1000, then the ratio will be 2:1. For each time loop, the servers will be given jobs in proportion to their CPU speeds.

### **2.4.2.4 Fuzzy Inference System**

The function `fuzzy_inference()` uses the fuzzy inference system (FIS) 'balancer1.fis'. A list of all the servers, together with their CPU speeds and queue lengths is used as input to the FIS and the server with the minimum value outputted is chosen as the most appropriate server.

### **2.4.2.5 Results outputted**

The simulation recorded all aspects of the cluster's performance, including:

- CPU usage
- Memory use and overload frequency
- CPU queue length
- Hard drive queue length
- Total queue length
- Job rate

- Response time for jobs

The last two are the main performance metrics that were used to evaluate each algorithm. Job rate refers to how many client requests were completed by a server in one time loop. The number of time increments it takes for a job to be completed is called the response time.

Job rates for each server were recorded at every time increment. The average job rate for an algorithm was found by computing the average of all job rates across all servers and the maximum job rate was found by taking the highest of all the recorded job rates.

Each client job had an extra variable called 'time\_taken' (together with CPU cycle time, memory and hard drive access). This was set to 1 when a client job first entered the server's CPU queue and was incremented for every time loop until the job was finished in the hard drive queue. This was then written to a file and at the end of each simulation the average was calculated by finding the mean of all client jobs and the maximum response time was calculated by selecting the job with the highest response time.

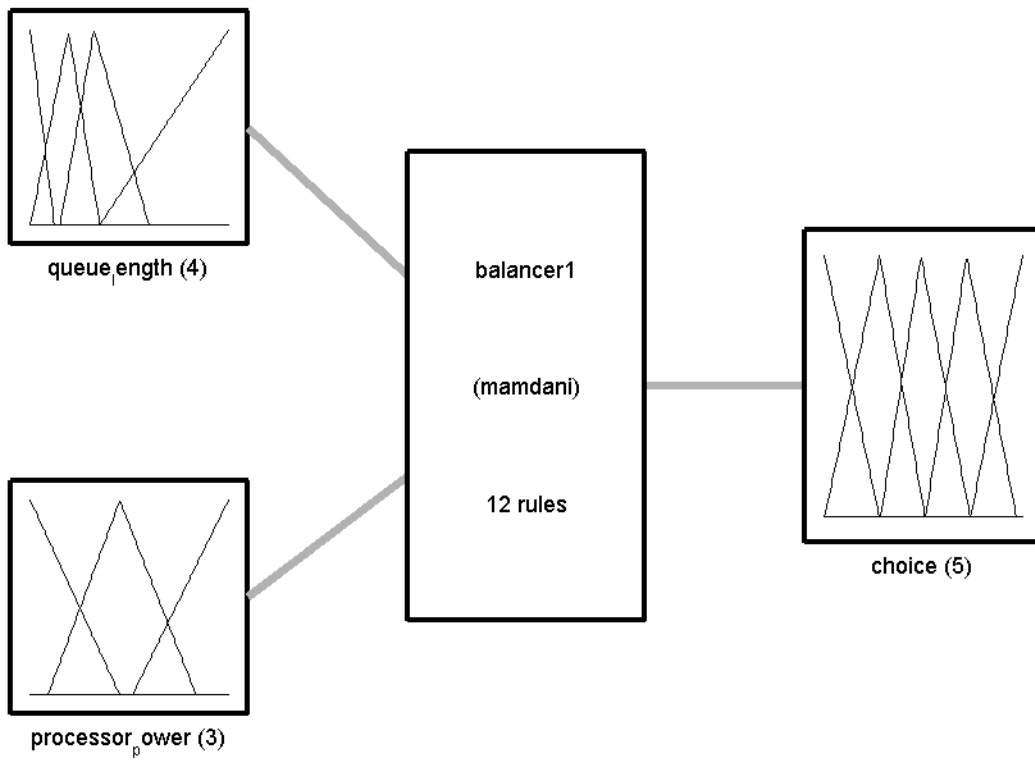
### 2.4.3 Implementation of the Fuzzy Inference System

Using Matlab's FIS toolbox, a system was created that took two inputs (CPU power and total queue length), applied a set of rules, and gave an output in the form of a numerical value. Each server was put through the system and the one with the minimum outputted value was chosen as the best server. Since fuzzy logic works by using a degree of belief in true and false questions, the question that was posed in this case was: is the current server the best choice for the next incoming client request? The server that agreed with this statement the most was chosen.

Figure 2 shows the structure of the fuzzy system. The queue length of the server and its CPU power are used as inputs and then a set of rules are applied and the output is mapped to a value based on those rules. This particular FIS implements the mamdani method of defuzzification that uses central weights to calculate output values.

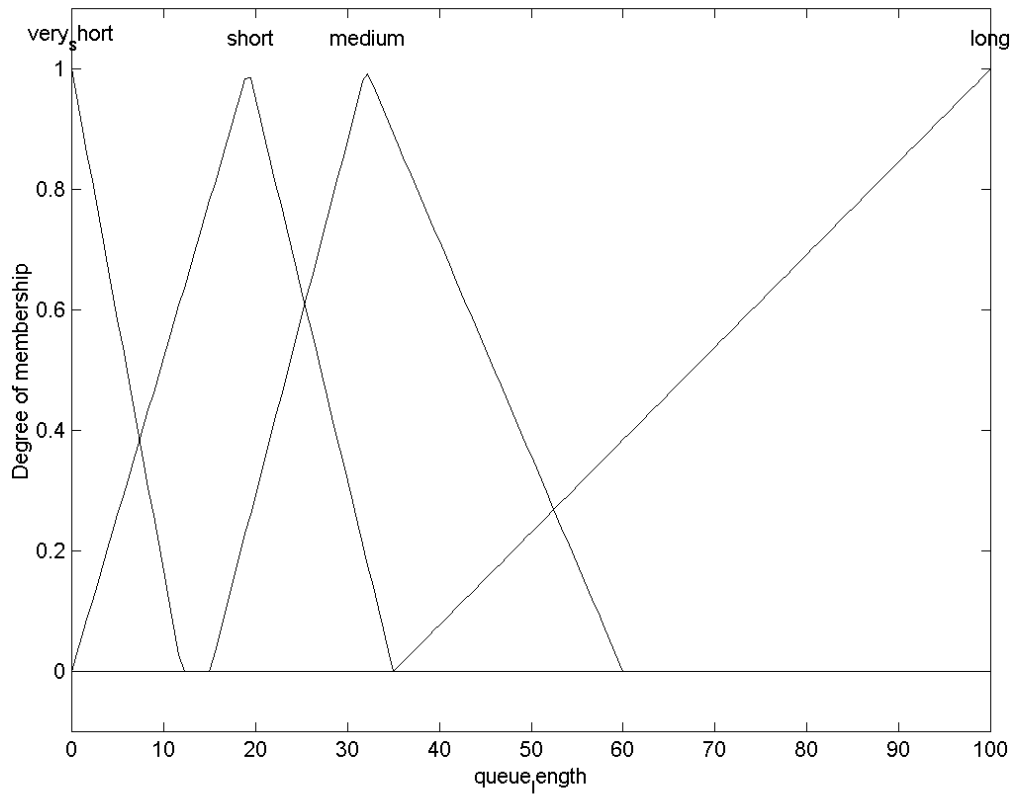
#### 2.4.3.1 Inputs

The two membership functions for the inputs that were used for the FIS are shown below in Figure 3 and Figure 4. The horizontal axis corresponds to the server attributes and the vertical axis gives the degree of membership (0 indicates no membership, 1 indicates total membership). For eg, a queue length of 40 has a 0.75 degree of membership to the function 'medium' and around 0.05 degree of membership to the function 'long'. So we can say that a queue length of 40 is mainly classified as 'medium'. Queue lengths greater than 100 are definitely 'long' because the degree of membership to the function 'long' is 1. The membership functions for CPU power operate in a similar way.

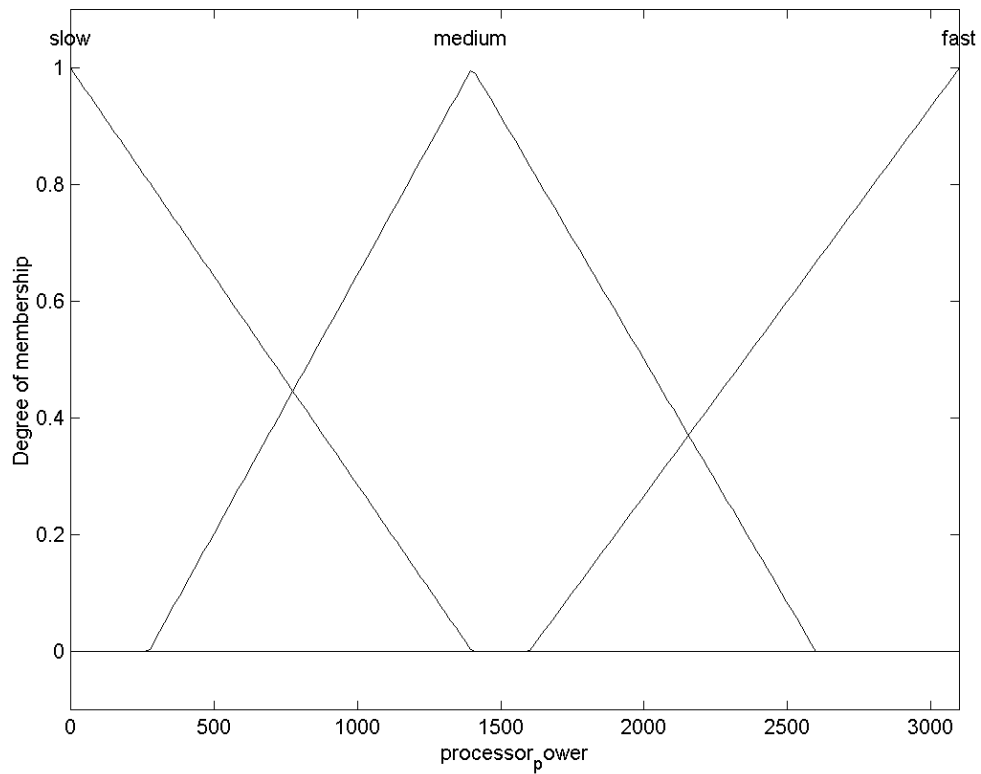


System balancer1: 2 inputs, 1 outputs, 12 rules

**Figure 2 - Overview of the fuzzy inference system**



**Figure 3 - Queue length membership functions**



**Figure 4 - CPU power membership functions**

### 2.4.3.2 Output

The output value is called ‘choice’ and has five membership functions: ‘excellent’, ‘very\_good’, ‘good’, ‘poor’ and ‘very\_poor’ (Figure 5). The rules of the fuzzy system determine which output(s) are chosen and the degree of membership to these output functions determines the value (on the horizontal axis) given to the server. The server with the minimum output value is chosen as the best server to receive the next incoming client job. The degree of membership to these output functions is determined by the degree of membership to the input functions and the strength of the applicable rules.

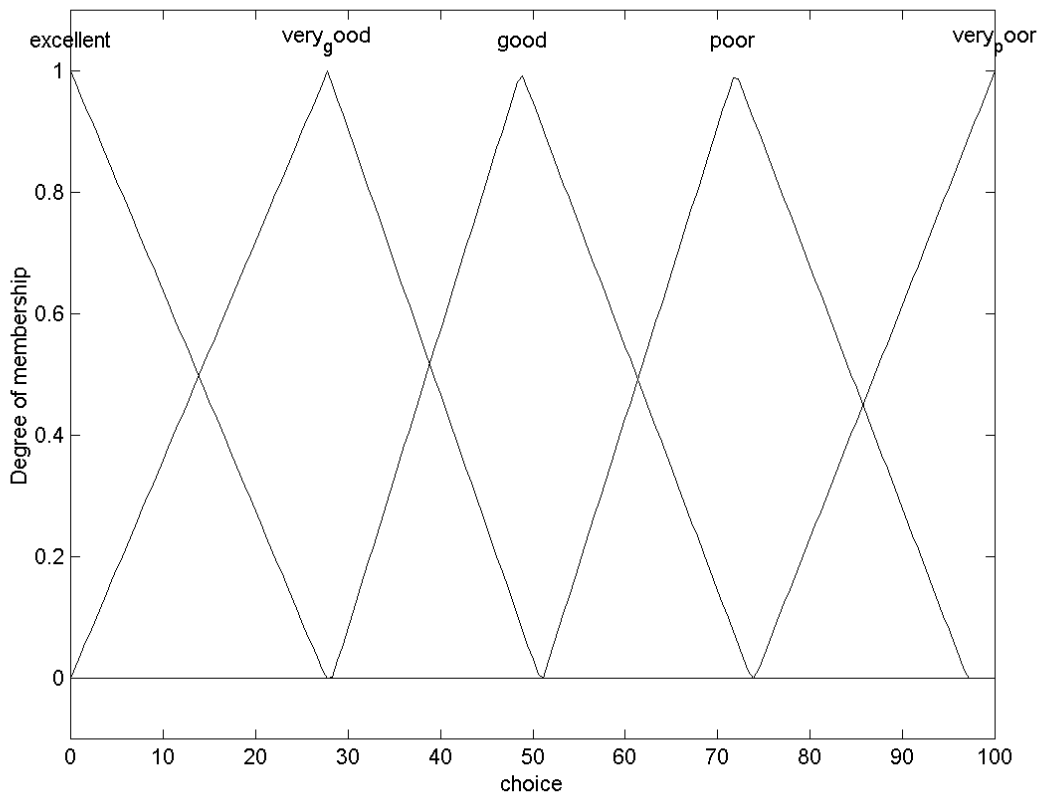


Figure 5 - Output functions for fuzzy inference system

### 2.4.3.3 Rules

Figure 6 shows the set of rules used in the FIS. The rules work by mapping each input to one or more of the input functions. For eg, a queue length of 20 is classified as both ‘short’ and ‘medium’. If the CPU power of that server is 1500, then the CPU is classified as ‘medium’ in speed. These inputs activate rule 5 and 8. The rules then get mapped to the output functions. Rule 5 maps to ‘very\_good’ and rule 8 maps to ‘good’.

Note the number at the end of the rules, which indicates the weight the rule is given. A server with a ‘very\_short’ queue length and a ‘fast’ processor will be an ‘excellent’ choice. But so will a server with a ‘very\_short’ queue length and a ‘slow’ processor. The weight given to the first case is greater than the other case. In other words, a ‘fast’ processor is a more excellent choice than a ‘slow’ processor if both queue lengths are similar. If a server activates more

than one rule then the outputs are averaged out based on the weight of the rule and the degree of membership to the input functions.

1. If (queue\_length is very\_short) and (processor\_power is fast) then (choice is excellent) (1)
2. If (queue\_length is very\_short) and (processor\_power is medium) then (choice is excellent) (0.5)
3. If (queue\_length is very\_short) and (processor\_power is slow) then (choice is excellent) (0.2)
4. If (queue\_length is short) and (processor\_power is fast) then (choice is very\_good) (1)
5. If (queue\_length is short) and (processor\_power is medium) then (choice is very\_good) (0.5)
6. If (queue\_length is short) and (processor\_power is slow) then (choice is very\_good) (0.2)
7. If (queue\_length is medium) and (processor\_power is fast) then (choice is good) (1)
8. If (queue\_length is medium) and (processor\_power is medium) then (choice is good) (0.5)
9. If (queue\_length is medium) and (processor\_power is slow) then (choice is good) (0.2)
10. If (queue\_length is long) and (processor\_power is fast) then (choice is poor) (0.2)
11. If (queue\_length is long) and (processor\_power is medium) then (choice is poor) (0.5)
12. If (queue\_length is long) and (processor\_power is slow) then (choice is very\_poor) (1)

**Figure 6 - Rules for fuzzy inference system**

## 3 RESULTS AND DISCUSSION

By using the three different traffic models and a number of possible server configurations, it was possible to compare the performance of the individual algorithms across a range of conditions and set-ups. However, with infinitely many combinations of server configurations and different traffic parameters, it was necessary to have a default set-up which could be used as a base to compare all results to. The parameters were as follows:

Traffic Model	traffic condition 1
Time_loops:	2000
Job_range_mean:	8
Job_range_sd:	1
Mean1 (job CPU time):	500
Mean2 (job memory size):	2000
Mean3 (job hard drive time):	50

### Server Configuration

Number of servers:	4
Set-up:	Random (see 3.3 Server Configurations)

All comparisons were done by changing one of these variables. For eg, increasing the number of servers, increasing the job\_range\_sd, changing the traffic condition to 3 (while keeping the other parameters the same), etc. For situations where only the configurations of the servers changed, the traffic data used was exactly the same. This approach allowed a structured and meaningful comparison of results.

All results graphs show round robin as number 1 on the horizontal axis, then least connection as number 2, weighted distribution as number 3, then the FIS as number 4.

### 3.1 Server Configurations

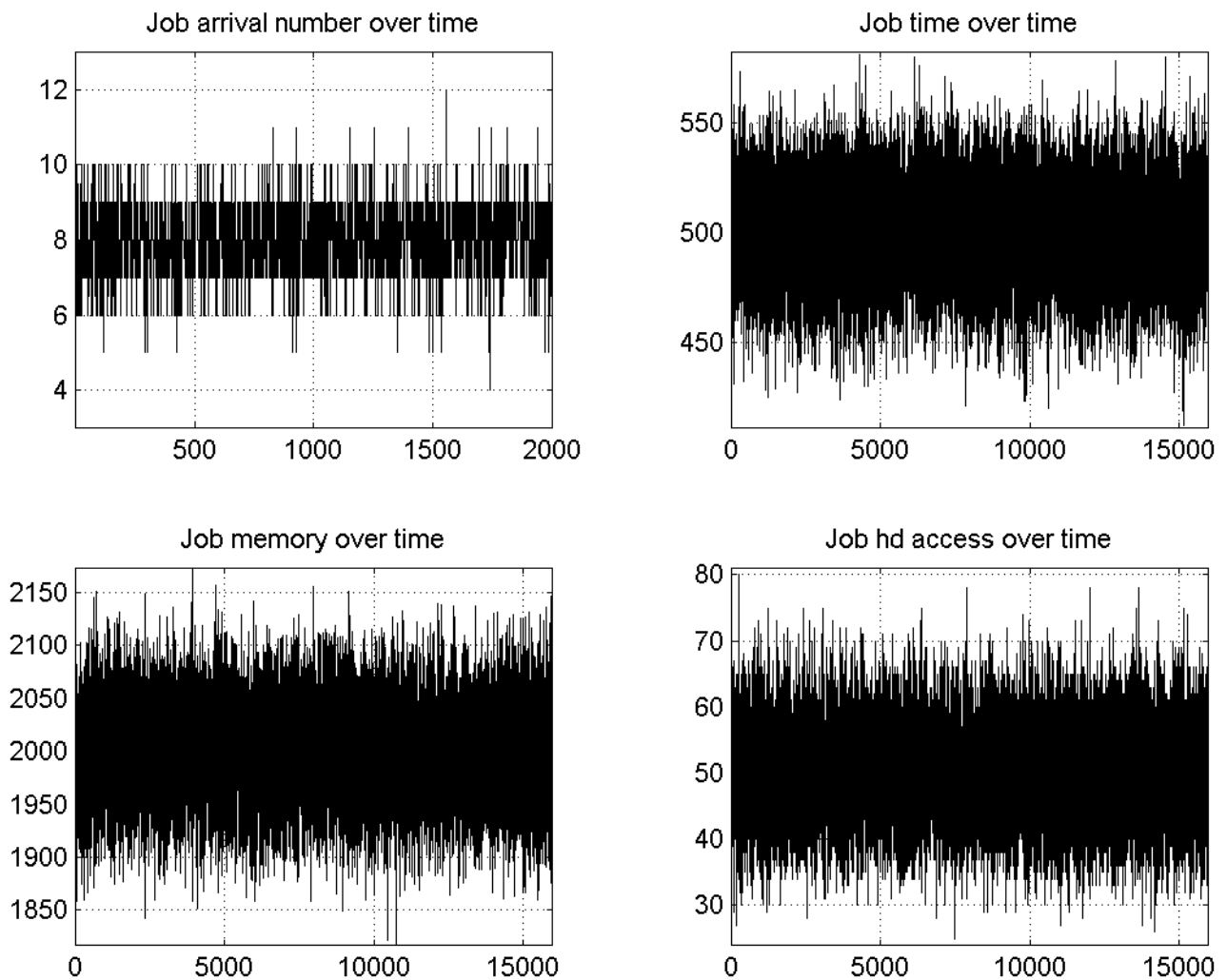
The configuration of the servers refers to the individual characteristics of each server. For this research, three possible configurations were chosen.

1. All servers are exactly the same power
2. Each server is more powerful than the previous server
3. Servers are given random powers (within reasonable bounds)

The ‘power’ of a server refers to the values of all its attributes: CPU power, memory size and hard drive speed. For comparisons between different configurations, the number of servers was kept at 4 and traffic condition 1 was used (as mentioned above). Traffic Condition 1 can be seen in

Figure 7. Arrival rates are fairly constant and job attributes are distributed randomly around a mean.

'Job arrival number over time' shows the arrival rate over the course of the simulation (2000 time loops). The other three graphs show the random distribution of the job attributes for all jobs (the horizontal axis referring to the job number).



**Figure 7- Traffic Condition 1 (Different Server Configurations)**

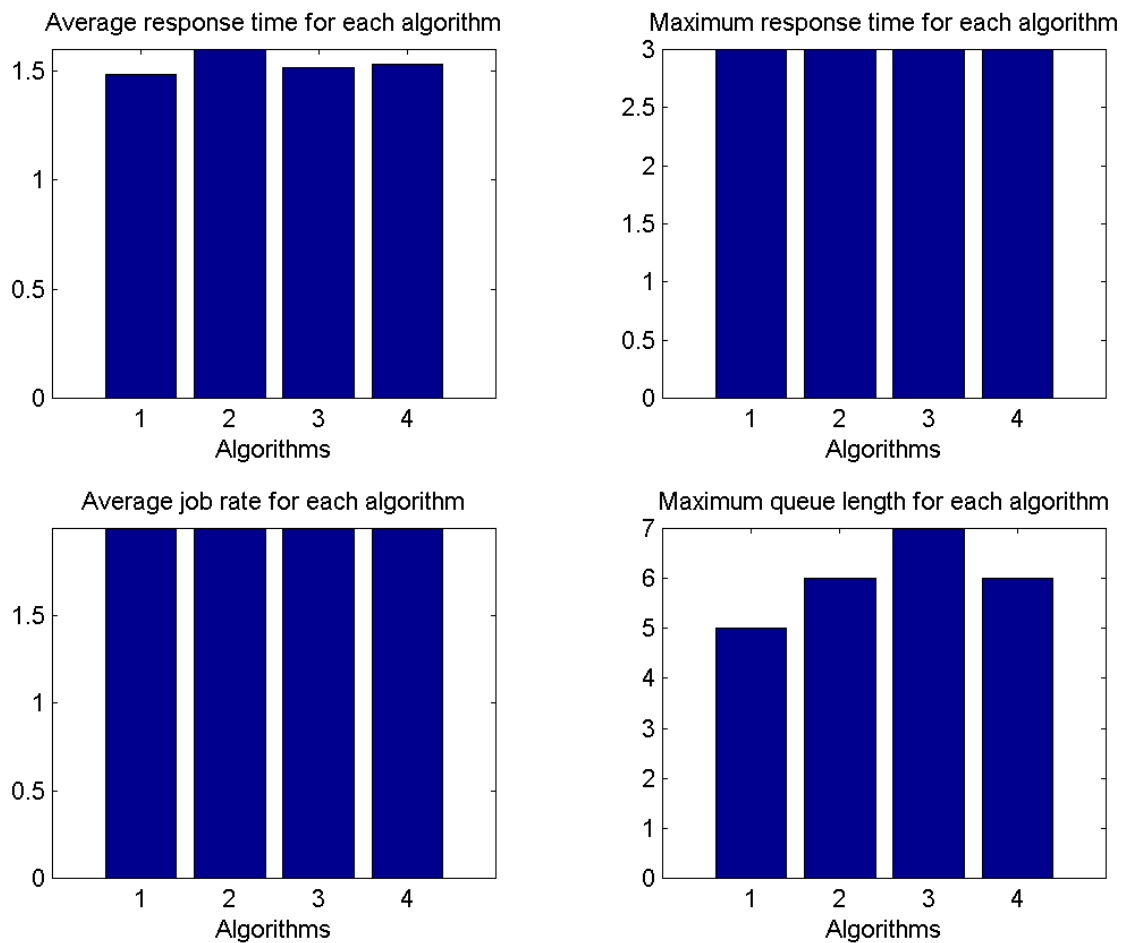
### 3.1.1 Same power

The configuration was as follows:

	CPU Power	Memory Size	Hard Drive Speed
Server 1	1500	256000	180
Server 2	1500	256000	180
Server 3	1500	256000	180
Server 4	1500	256000	180

**Table 1- Same Power Server Configuration**

The results for the simulation run are shown in Figure 8. It can be seen that all algorithms perform quite well in response time and job rate. Round robin and weighted distribution are at a distinct advantage when servers are the same power, because they both give jobs to the servers in an equal amount. Together with their static efficiency (especially round-robin), they would make an ideal choice in this cluster set-up. Round robin has actually outperformed all other algorithms in terms of response time because it does the best job of evenly distributing client traffic amongst servers that (in the long run) will produce very similar response times.



**Figure 8 - Results (Same power)**

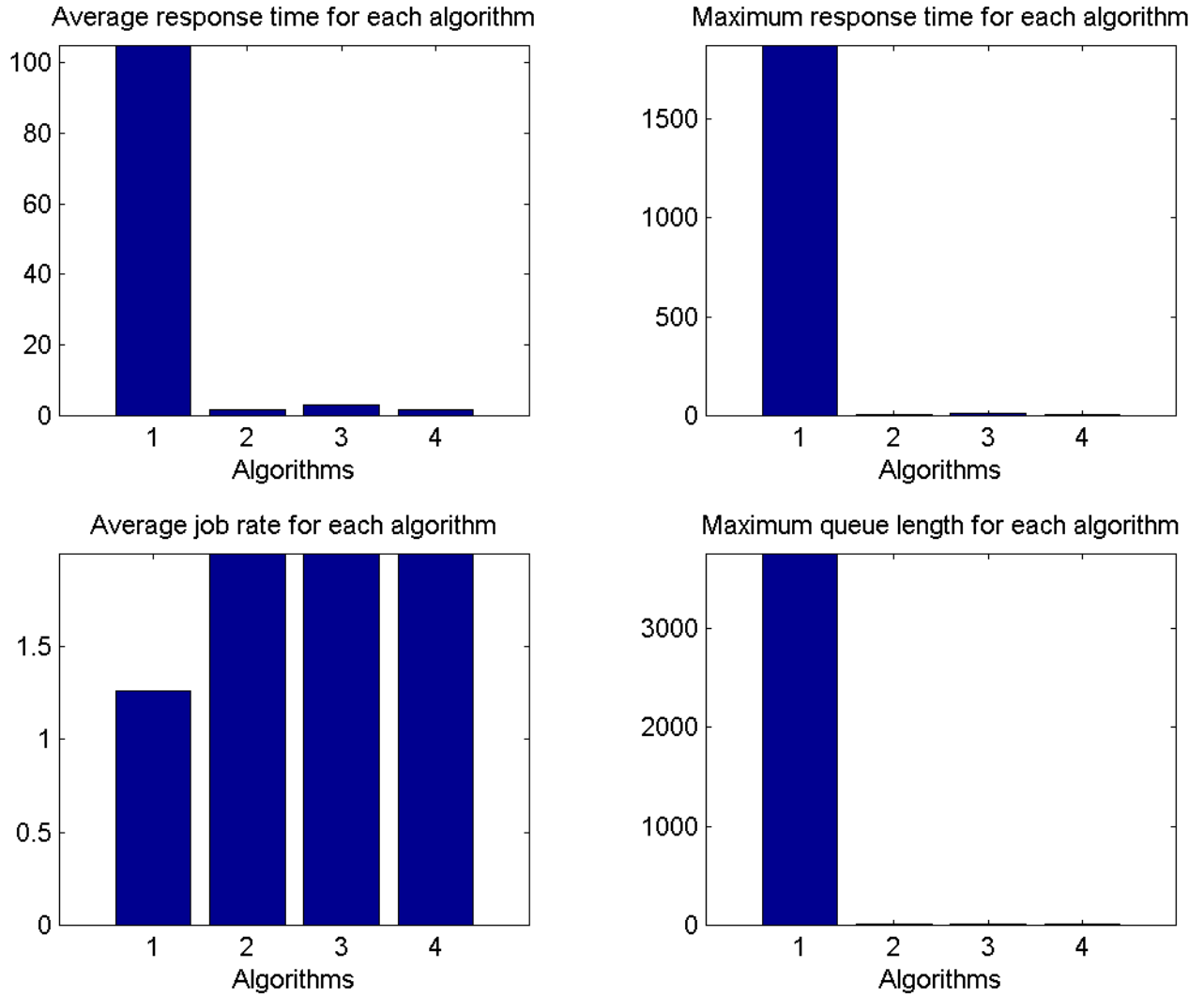
### 3.1.2 Increasing power

The configuration was as follows:

	CPU Power	Memory Size	Hard Drive Speed
Server 1	500	32000	90
Server 2	1000	64000	130
Server 3	1700	128000	190
Server 4	2500	256000	270

**Table 2 - Increasing Power Server Configuration**

The results of the simulation run are shown in Figure 9. We can clearly see the failure of round robin to take into consideration the individual powers of the servers, with server 1 and 2 receiving the same amount of jobs as server 3 and 4, even though the average load on 3 and 4 is less. Weighted distribution has a slightly higher average response time than least connection and the FIS, but the same average job rate. While the static nature of weighted distribution does not adapt to fluctuations in network load, it is generally very effective in situations where a servers CPU speed is a good indication of its power. In more complicated situations, CPU speed may not be the sole indicator of power and a more sophisticated weight measure should be adopted. Least connection delivers a slightly lower average response time of 1.66 time increments, compared to the 1.74 of the FIS.



**Figure 9 - Results (Increasing Power)**

### 3.1.3 Random

The configuration was as follows:

	CPU Power	Memory Size	Hard Drive Speed
Server 1	2500	256000	200
Server 2	1000	128000	250
Server 3	3000	512000	130
Server 4	500	64000	120

**Table 3 - Random Power Server Configuration**

The results for this simulation are shown in Figure 10. Once again, round robin has performed poorly, offering an average response time of above 60 time periods, and a maximum response time of over 1500. Compared to the last simulation configuration, the power of these servers were not as easily predicted by their CPU speed, and consequently the weighted distribution method did not perform as well. The average response time for the algorithm has jumped from 2.97 in the previous run to 12.52 under the configuration shown in Table 3. This can be compared with a decrease in average response time from both least connection and fuzzy inference. For this run, the average response time of least-connection was 1.62 and the fuzzy method was 1.55. Note that weighted distribution has maintained a job rate close to least connection and fuzzy inference, due to its tendency to direct jobs towards more powerful servers that can achieve higher job rates.

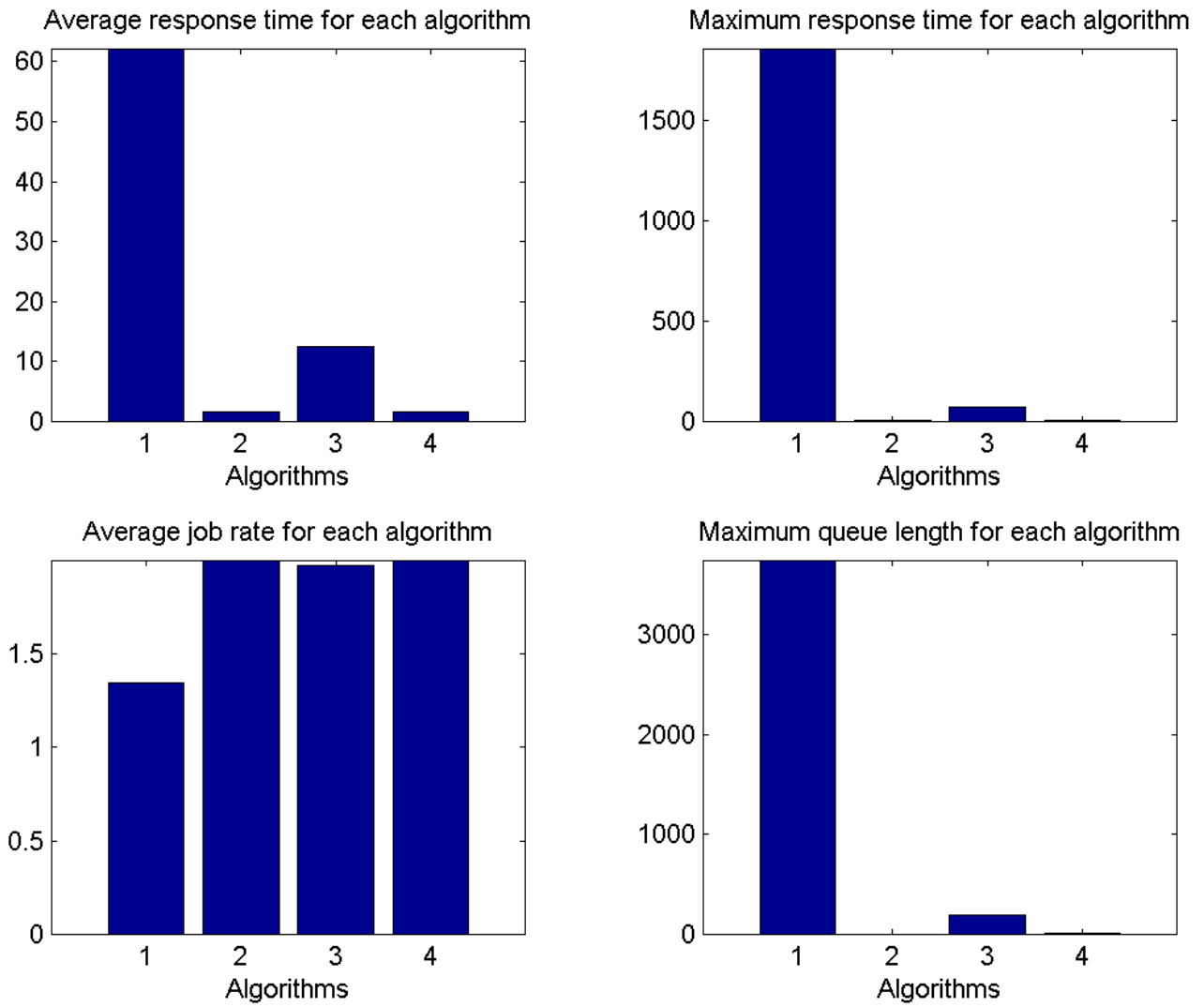


Figure 10 - Results (Random Power)

## 3.2 Number of Servers

This section gives the performance results for different numbers of servers in the cluster. All server statistics are of a random configuration. The case of four servers had already been tested under 3.1.3, due to the configurations being exactly the same. The same traffic data (generated by Traffic Condition 1 in Figure 7) was used throughout this section.

### 3.2.1 Two servers

Configuration was as follows:

	CPU Power	Memory Size	Hard Drive Speed
Server 1	2500	256000	200
Server 2	1000	128000	250

**Table 4 - Two server configuration**

Figure 11 shows the results of the simulation run. Surprisingly, round robin outperforms all algorithms in terms of average response time and average job rate, although is the worst performed with maximum response time. Compare this to 3.3.1.4 (Figure 22). In both cases the servers have been highly overloaded and round robin has performed considerably well. It seems that in situations of constant overload with growing queue sizes, a method that ensures job arrivals are split up equally is very effective.

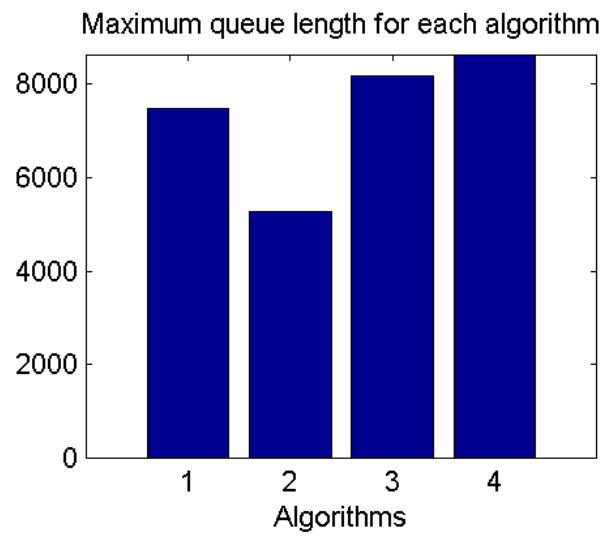
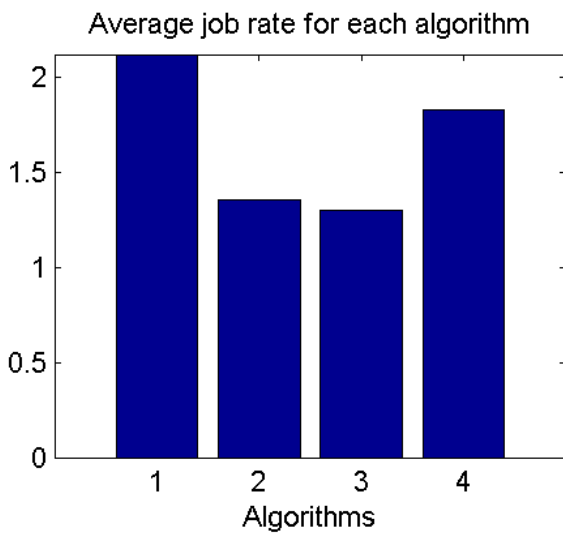
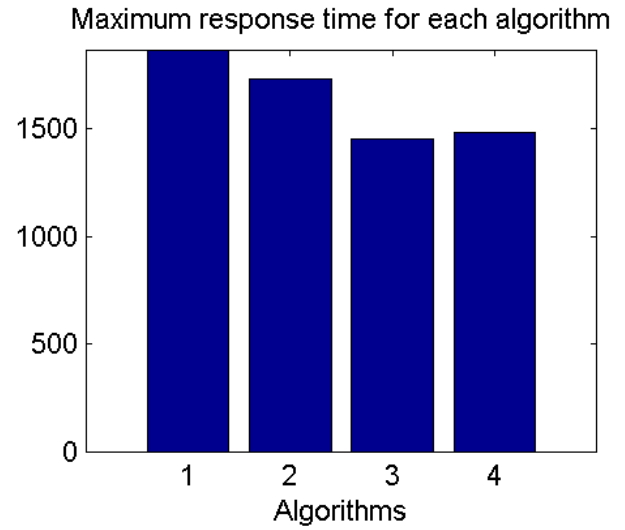
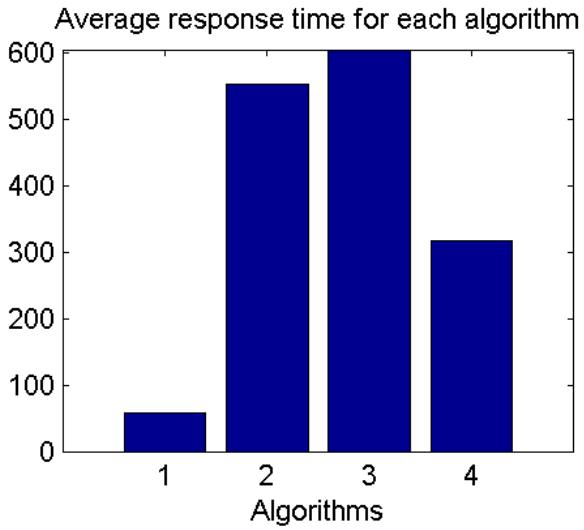


Figure 11 - Results (two servers)

### 3.2.2 Eight servers

The configuration was as follows:

	CPU Power	Memory Size	Hard Drive Speed
Server 1	2500	256000	200
Server 2	1000	128000	250
Server 3	3000	512000	130
Server 4	500	64000	120
Server 5	1700	128000	200
Server 6	2200	256000	250
Server 7	1100	64000	80
Server 8	800	32000	170

**Table 5 - Eight Server Configuration**

The results for the eight server configuration are shown below in Figure 12. When comparing the results from 3.1.3 (4 server random configuration), we can see that response times have dropped for all servers, with the round robin average coming down from over 60 to about 15. The average response times for the other three algorithms have all come down, with fuzzy inference the lowest at 1.06 time loops. Weighted distribution recorded an average response time of 1.45. So increasing the number of servers has decreased the differences in response times between algorithms. Also note that the job rates are very similar across all algorithms and have decreased. This is because each server is getting less client traffic (since there are more servers available) and they cannot complete as many jobs per time loop.

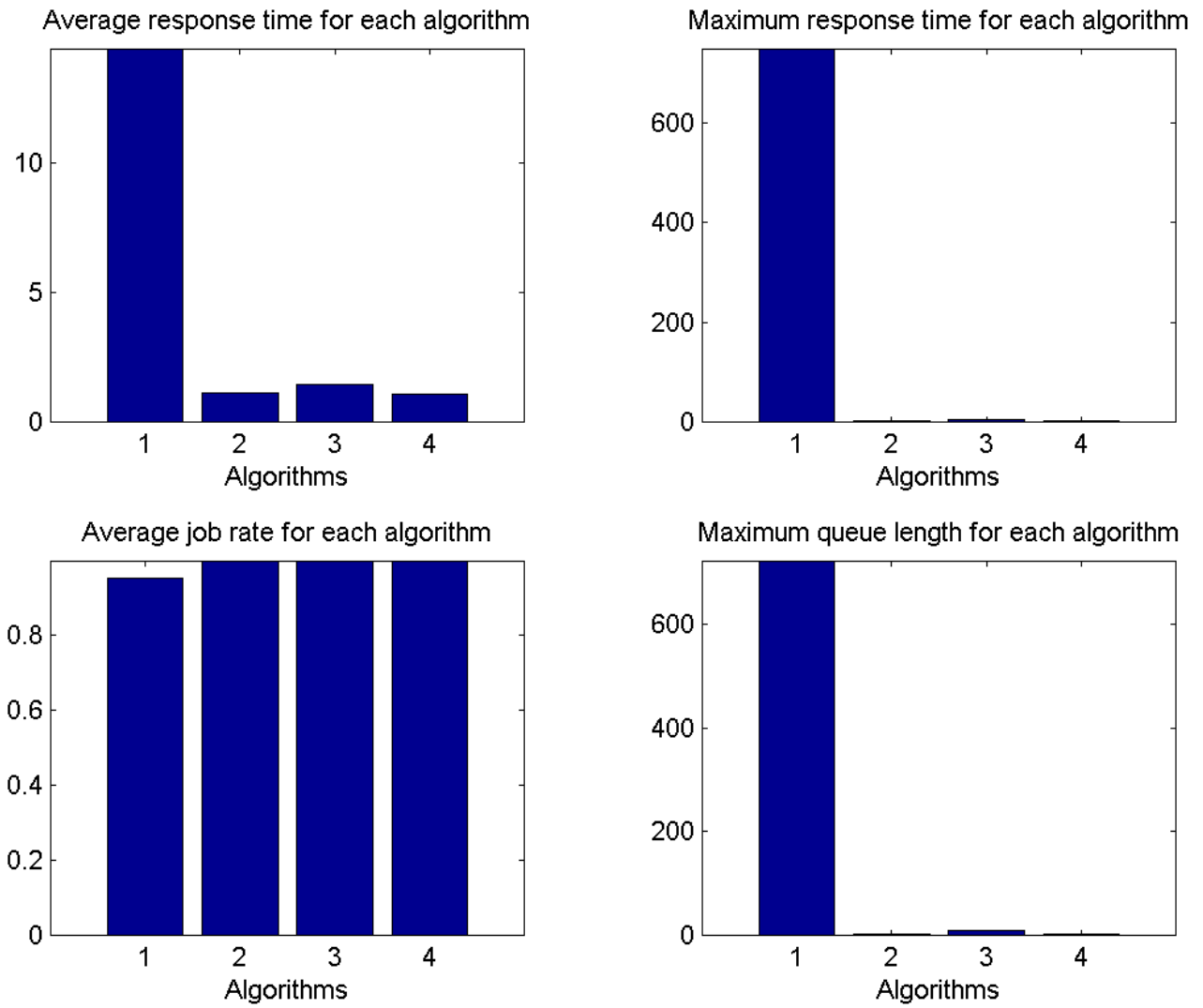


Figure 12 - Results (Eight Servers)

### 3.2.3 Sixteen servers

The configuration was as follows:

	CPU Power	Memory Size	Hard Drive Speed
Server 1	2500	256000	200
Server 2	1000	128000	250
Server 3	3000	512000	130
Server 4	500	64000	120
Server 5	1700	128000	200
Server 6	2200	256000	250
Server 7	1100	64000	80
Server 8	800	32000	170
Server 9	2800	512000	240
Server 10	500	64000	100
Server 11	1800	256000	150
Server 12	600	128000	200
Server 13	800	256000	250
Server 14	1700	256000	200
Server 15	2200	128000	150
Server 16	900	128000	90

**Table 6 - Sixteen Servers Configuration**

The simulation results are displayed below in Figure 13. The response times have again been reduced, as have the job rates. Again the lowest was achieved by the FIS. It is interesting to see that weighted distribution has a higher average response time and maximum queue length than the others. This is due to the fact that an average arrival rate of 8 per time loop distributed amongst 16 servers tends to be randomised slightly. In other words, there are not enough jobs arriving per time loop to satisfy every server's proportion that weighted distribution determines. Round robin keeps track of which server it gave the last job to, so it ensures an equal distribution throughout the simulation. The ability of the FIS system to distinguish on the basis of power when servers have similar queue length proves superior to least connection, which only uses queue length.

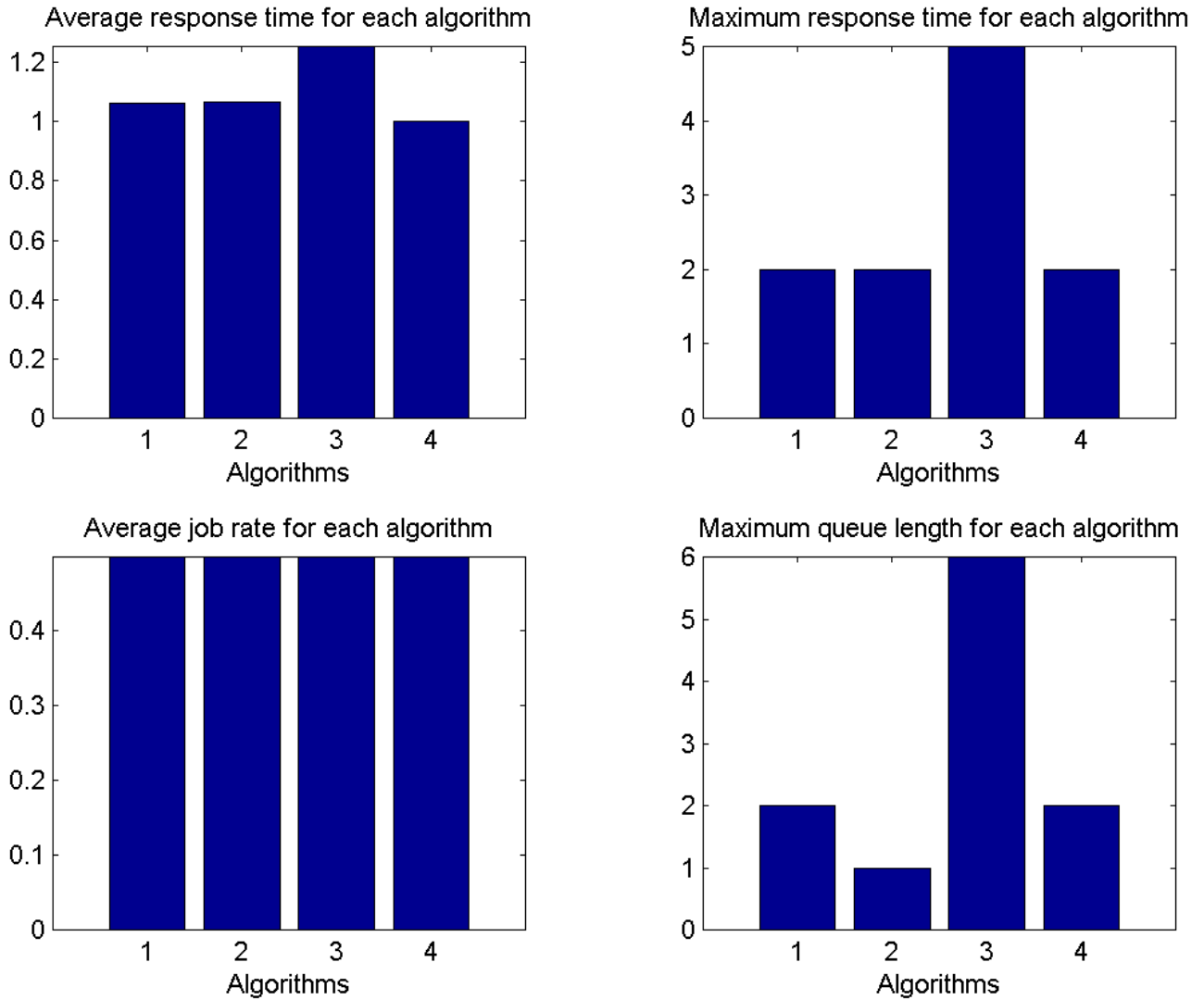


Figure 13 - Results (Sixteen Servers)

### 3.3 Traffic Patterns

For comparison under different traffic conditions, the server number and configuration was kept constant and were as follows:

	CPU Power	Memory Size	Hard Drive Speed
Server 1	2500	256000	200
Server 2	1000	128000	250
Server 3	3000	512000	130
Server 4	500	64000	120

**Table 7 - Fixed configuration for traffic comparison**

In this section, the parameters described in 2.4.1 (Components) are altered and the results are compared. When comparing the effects of changing parameters within a traffic model, a base traffic configuration is used. This is described and displayed at the start of the section for each traffic model.

#### 3.3.1 Traffic Pattern 1 – Near constant arrivals

##### 3.3.1.1 Base Traffic Pattern 1

The base model for traffic pattern 1 was as follows:

```

time_loops:           2000
job_range_mean:      8
job_range_sd:        1
mean1 (job CPU time): 500
mean2 (job memory size): 2000
mean3 (job hard drive time): 50

```

This model is displayed in Figure 14. This is the same traffic pattern that was used to compare different server configurations and numbers (section 3.1 and 3.2). The results of a simulation run using the configuration in Table 7 is given in Figure 15. The main things to note are the similar performance of least connection and fuzzy inference, as well as the failure of weighted distribution to adapt to changing cluster loads, even in the relatively smooth arrival rate generated. Job rates for the last three algorithms are all similar, showing that the strength of weighted distribution is in its ability to achieve a high job rate for the servers. This is because the server's CPU is a good (but not perfect) indicator of how many jobs it can complete per time loop. The performance of round robin is poor, particularly in the areas of response time. This is caused by the long queue lengths it allows to form at the weaker servers.

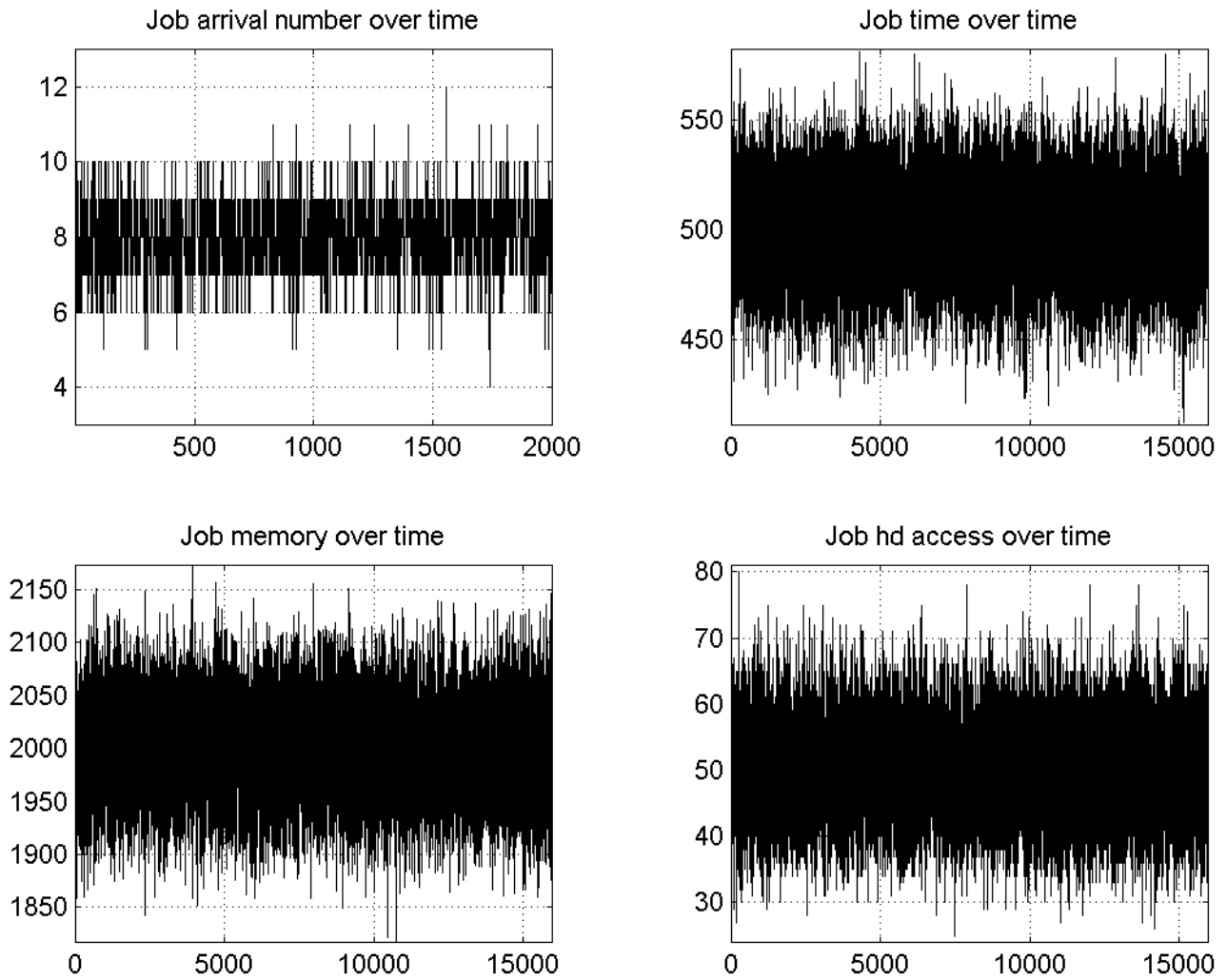


Figure 14 - Base Traffic Pattern 1

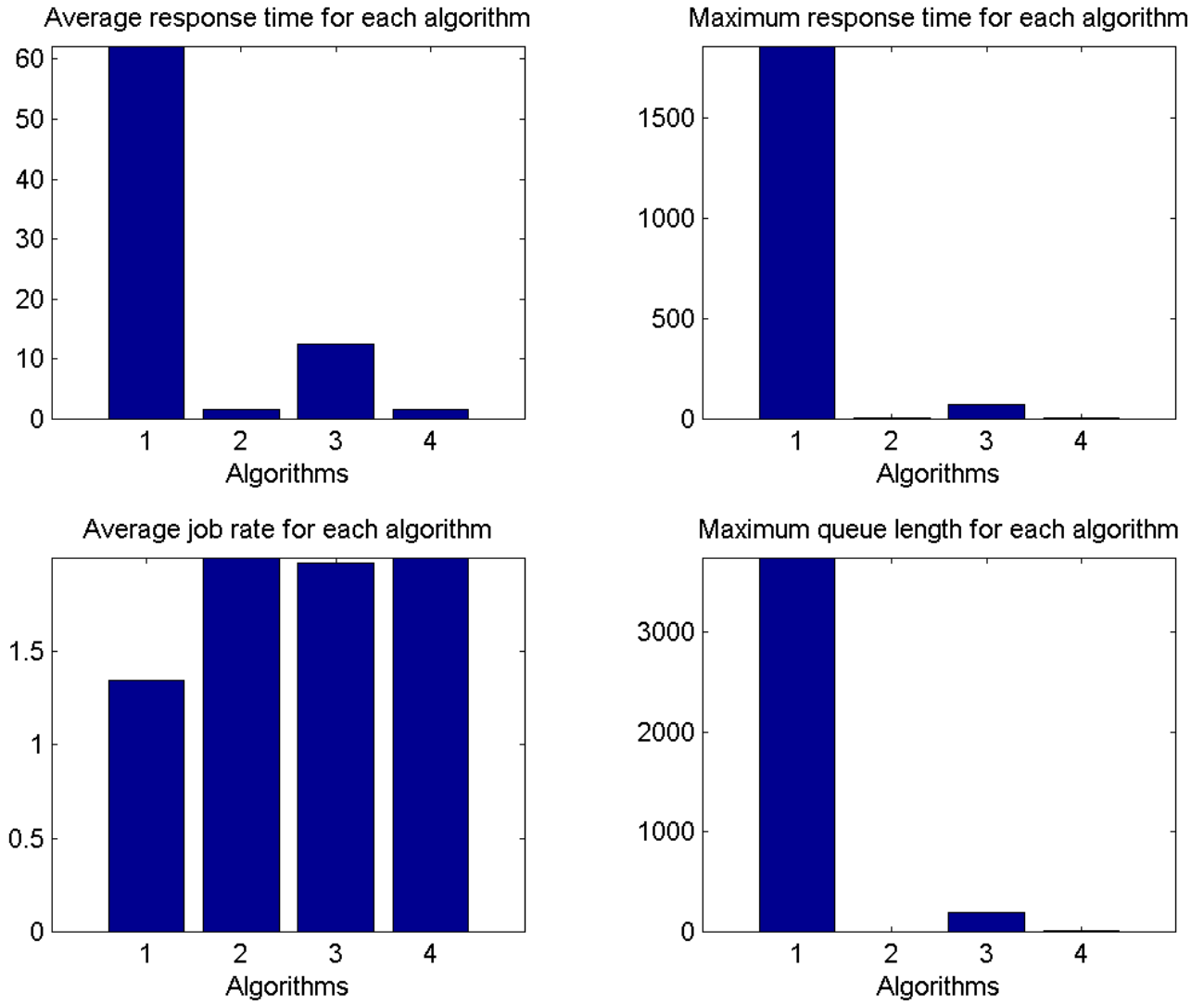


Figure 15 - Results (Base Traffic Pattern 1)

### 3.3.1.2 Low variability in job arrivals

For this simulation, the value `job_range_sd` was lowered to 0.4 (compared to the base traffic value of 1) to monitor the effect of making the traffic close to constant in terms of arrival rates. The traffic pattern is shown in Figure 16 and the results are shown in Figure 17. The average response time for weighted distribution has risen by approximately 0.7 while the least-connection and fuzzy inference methods have dropped by about 0.1 each. The round-robin algorithm has risen by about 25 (compared to the base traffic model). There doesn't seem to be any significant effect on the job-rate, since the average arrival rate is still the same and the servers have to deal with approximately the same amount of traffic. The changes in results could be due to the difference in the randomly generated traffic.

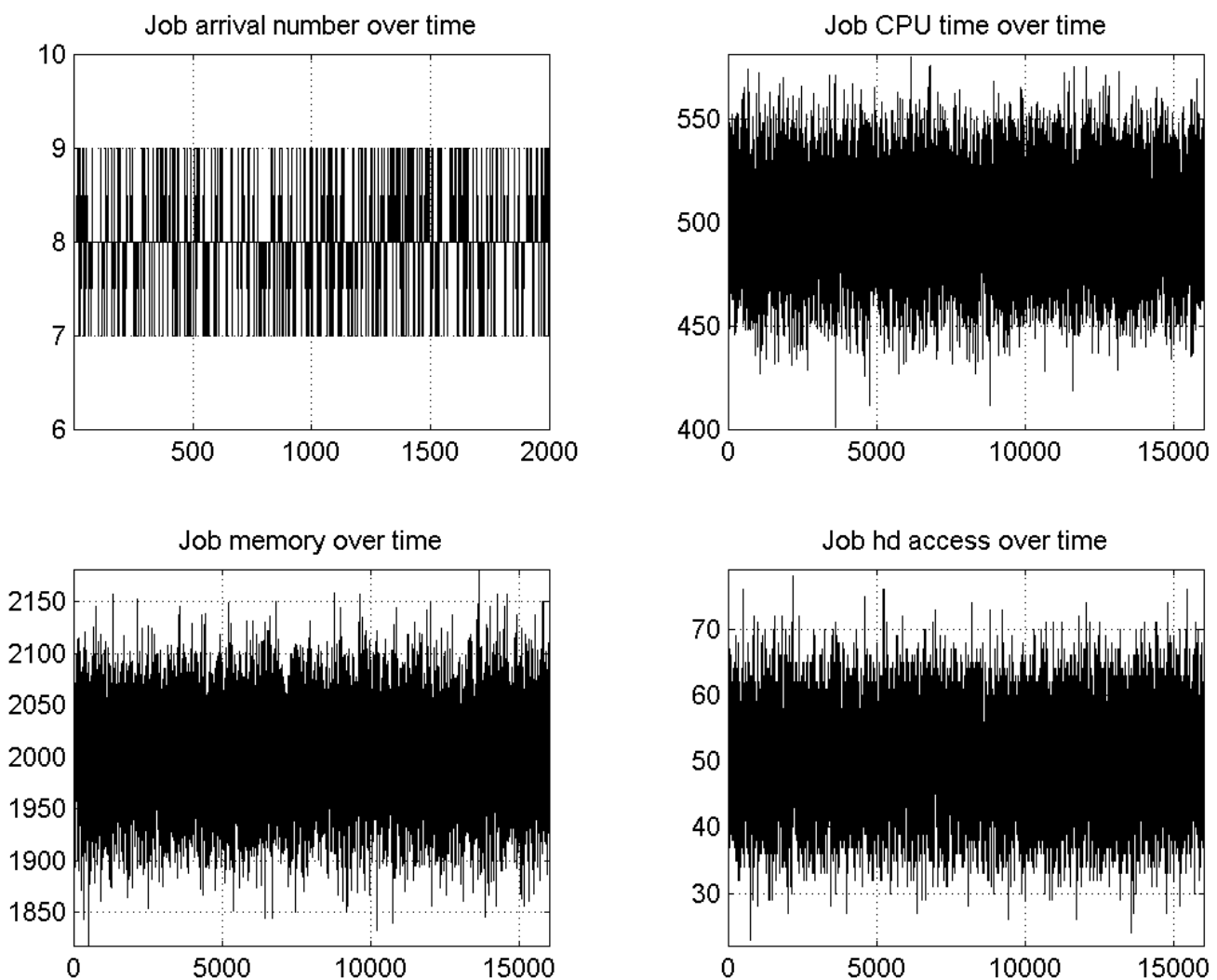


Figure 16 – Traffic Pattern 1 (Low variability in arrival rates)

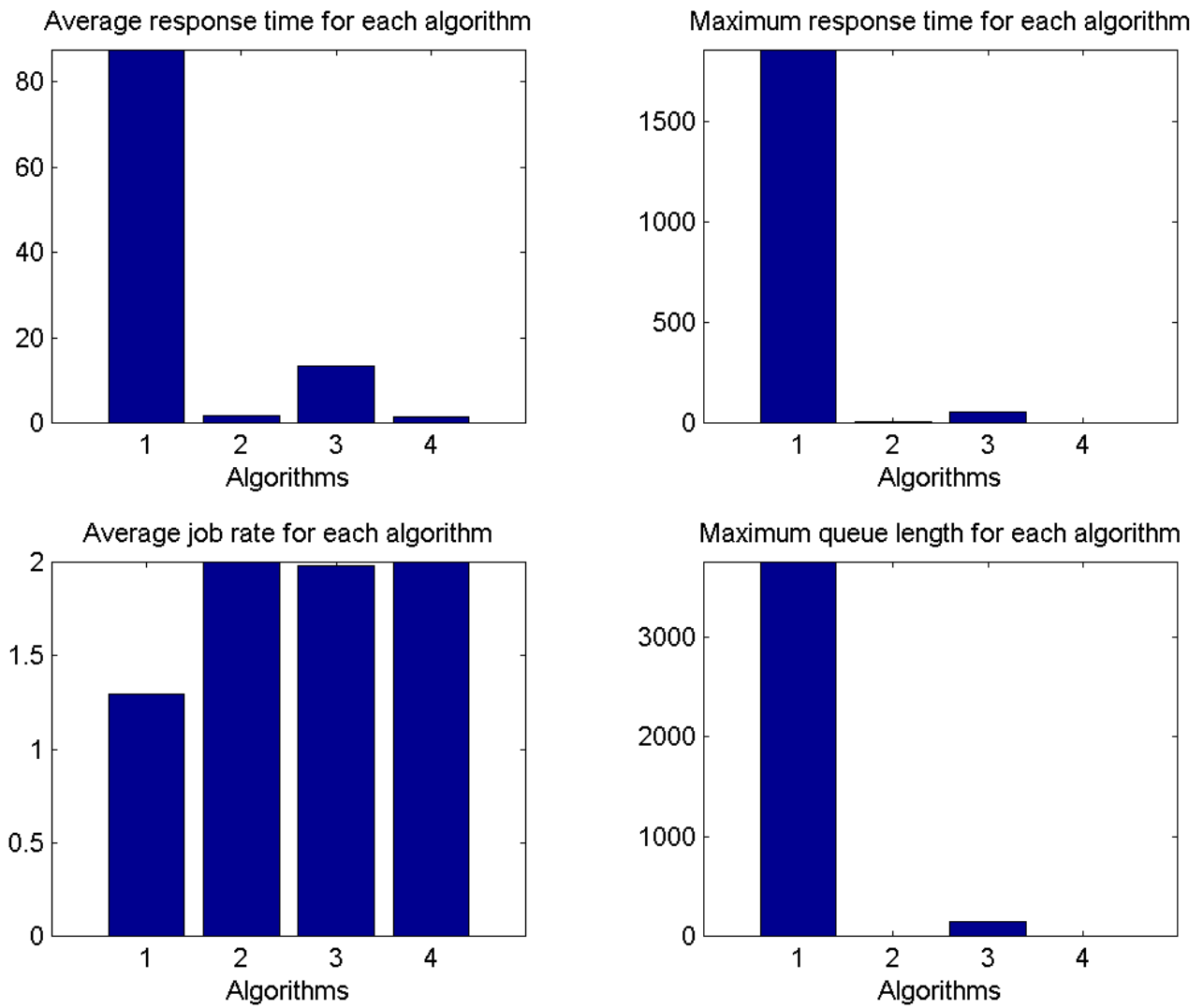


Figure 17 - Results (Low variability in arrival rates)

### 3.3.1.3 High variability in job arrivals

This time the `job_range_sd` has been increased from 1 to 2.3. The traffic model and performance results are shown in Figure 18 and Figure 19. The basic performance comparisons from 3.3.1.1 are still applicable, although the values have changed. All average response times have increased. Weighted distribution's average response time has increased from 13.2 with a `job_range_sd` of 0.4, to 28.99 when `job_range_sd` was 2.3. Compare this to an increase from 1.5 to 2.1 for least-connection and 1.4 to 2.1 for fuzzy inference. In general, increasing the variance of arrival rates seems to increase the response times (including the maximum). The job-rate stays fairly constant for different variance in arrival rates.

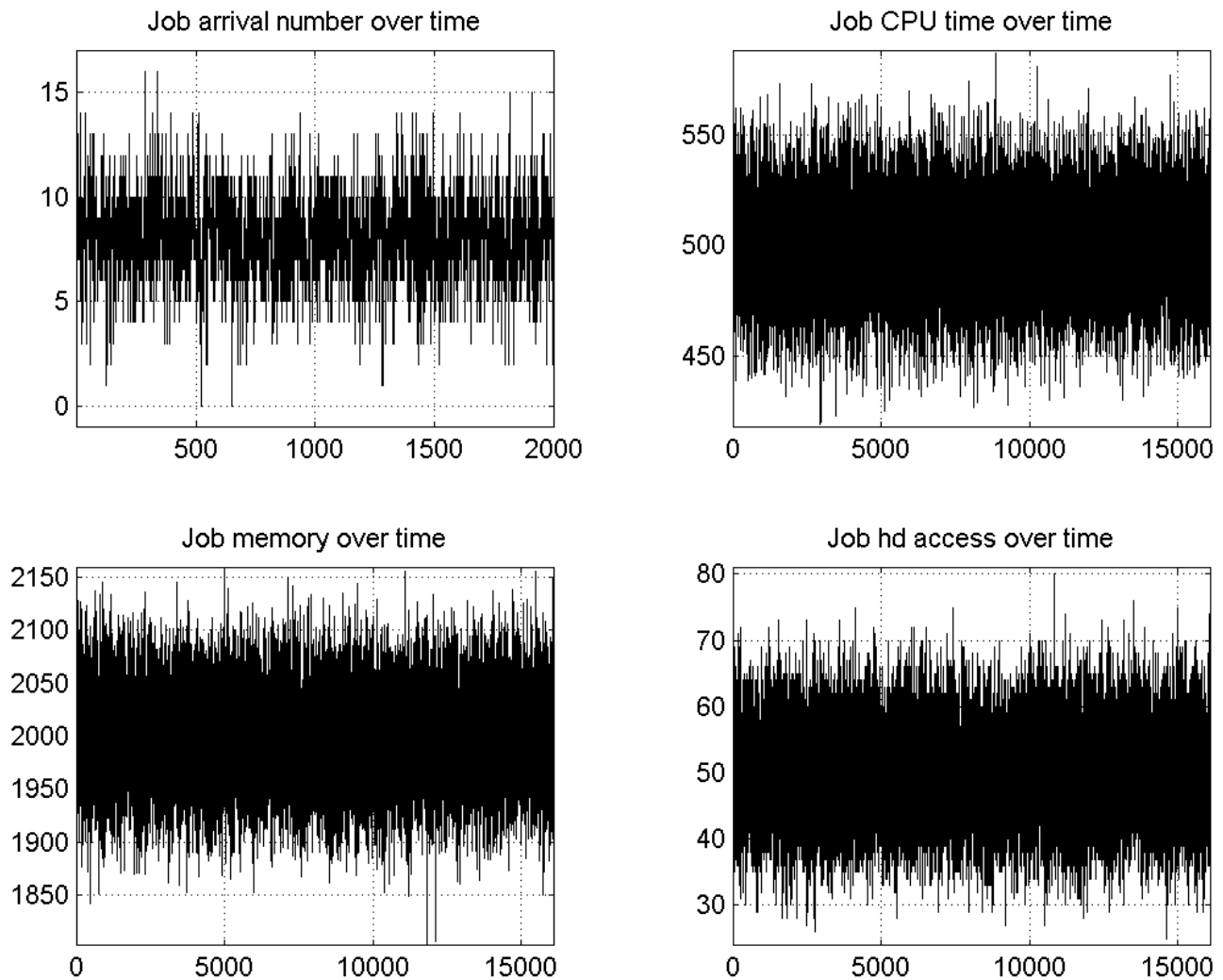


Figure 18 - Traffic Pattern 1 (High variability in arrival rates)

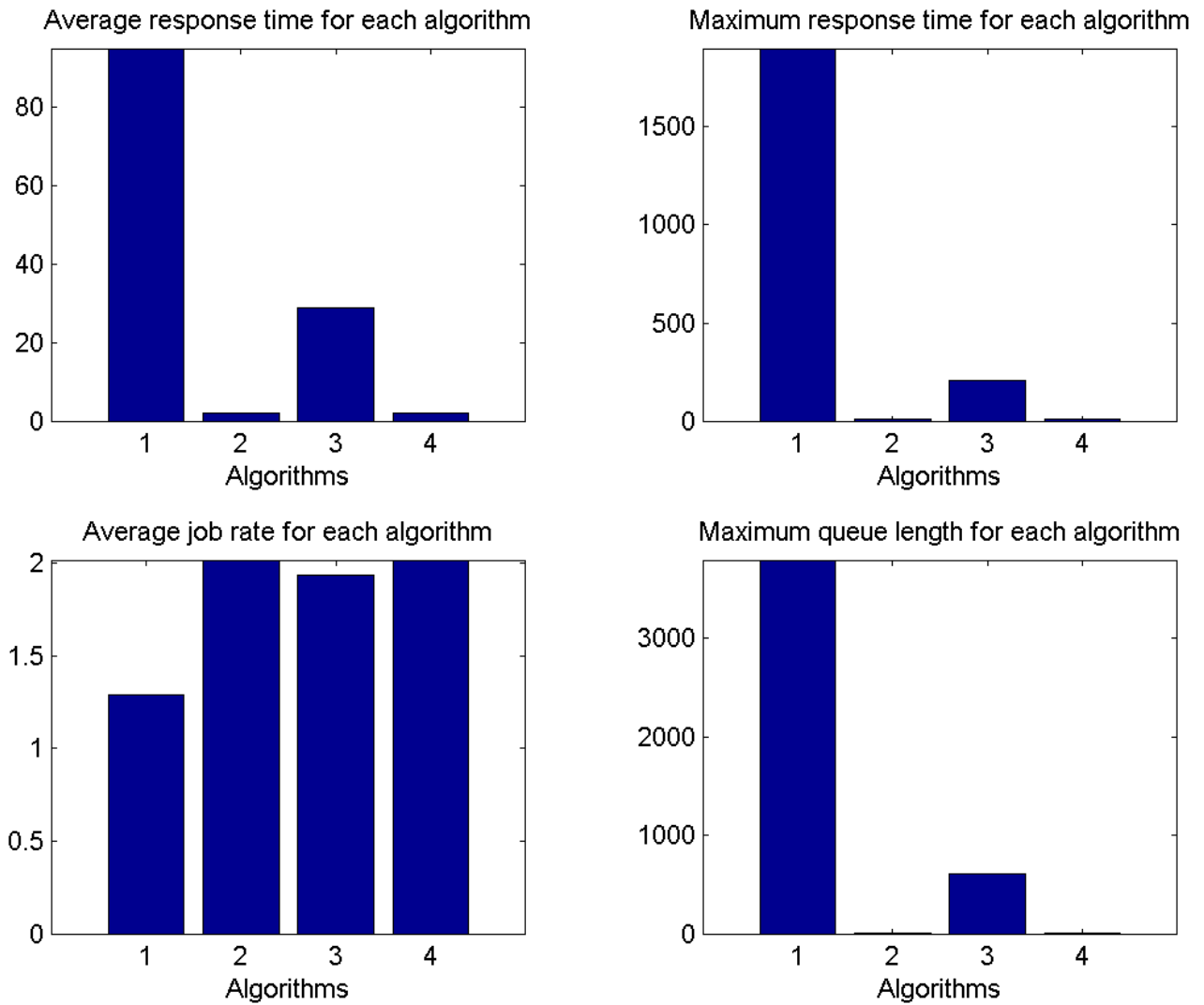


Figure 19 - Results (High variability in arrival rates)

### 3.3.1.4 Low job arrival rate

For this simulation run the `job_range_mean` was lowered to an average of 4 per time loop. This change is reflected in the traffic pattern shown in Figure 20. Figure 21 shows an improvement in all response times and a drop in job rates. Decreasing the arrival rate has a similar effect to increasing the number of servers. The response time of round robin has dropped from 62 in the base condition to 44 in this case. Weighted distribution drops from an average response time of 12.5 down to 1.3, only slightly higher than least connection and the FIS method.

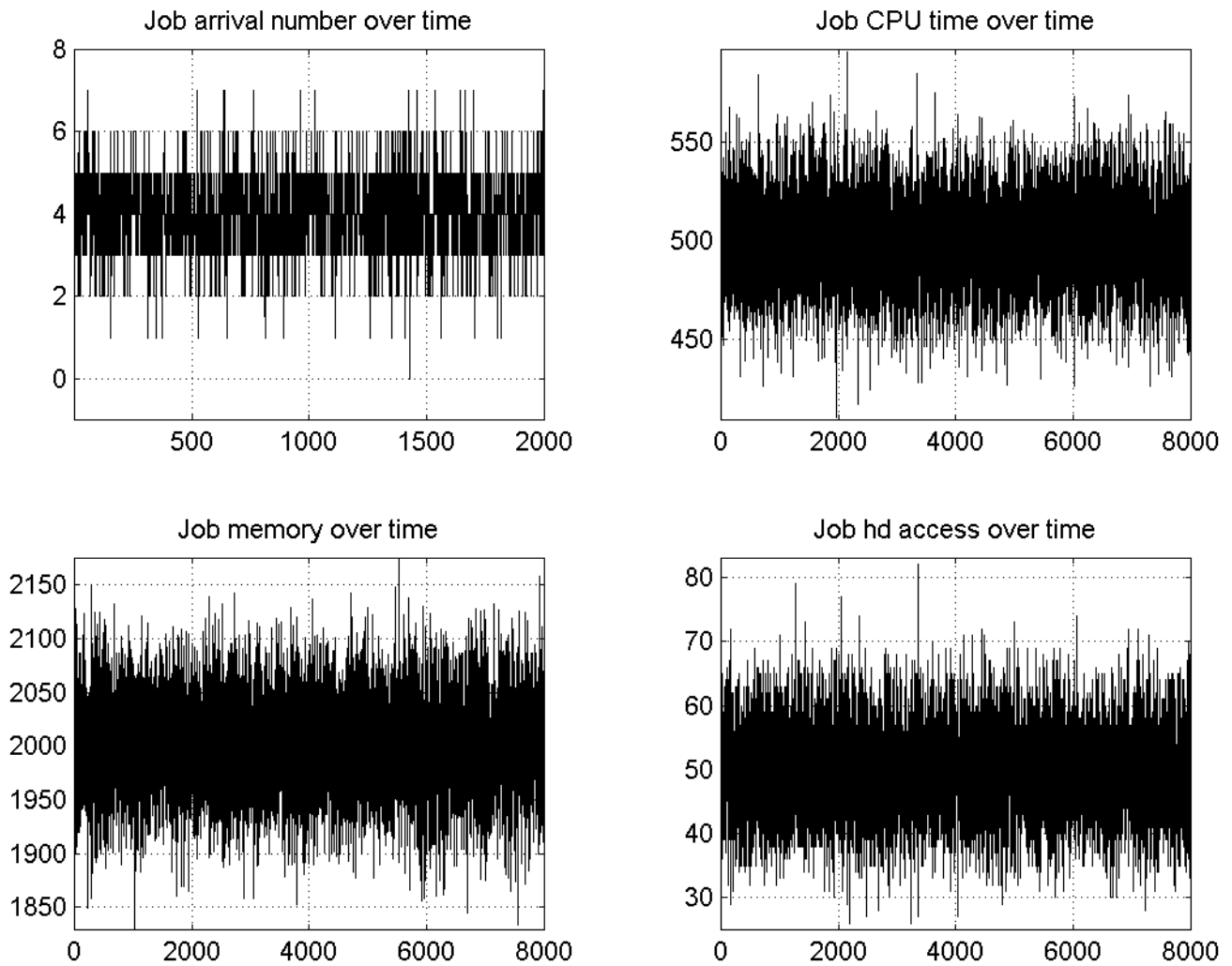


Figure 20 - Traffic Pattern 1 (low arrival rates)

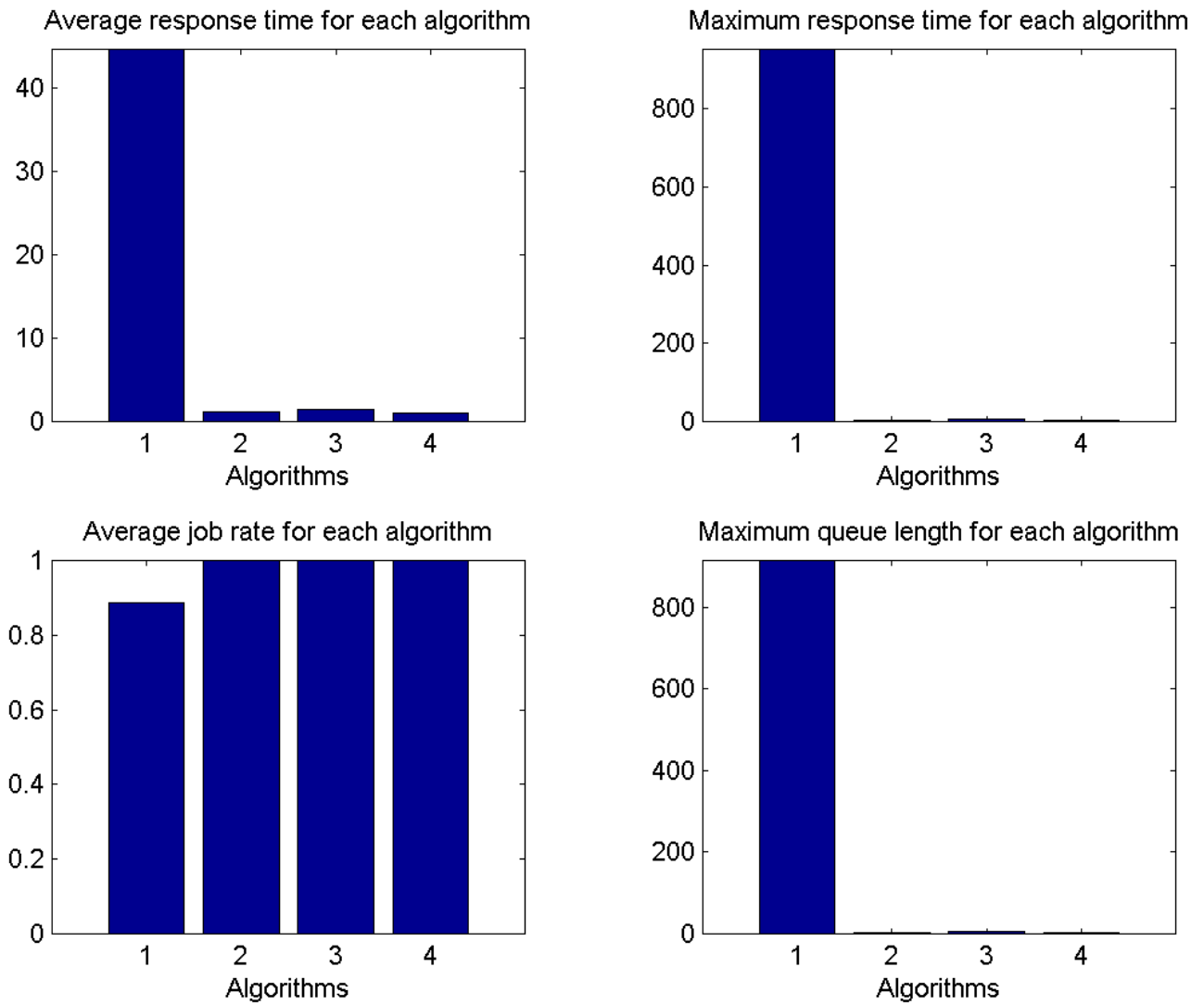


Figure 21 - Results (Low arrival rates)

### 3.3.1.5 High job arrival rate

The next two figures give the traffic pattern and performance results for Traffic Model 1 with a job\_range\_mean of 20, 12 more than the base job\_range\_mean. This traffic condition creates an overload of the server cluster for all algorithms. The average response time of least-connection rises from 1.6 to 658, weighted distribution offers the worst performance at 777, and the FIS does better than both with 296. This arrival rate mean has overloaded the server cluster and consequently a blowout in response times. Job rates have also dropped, but this time it is due to the fact that servers were slowed down by large queues, not because there weren't enough jobs to complete. The FIS clearly outperforms all other methods in terms of response times, making it a good choice for long periods of high traffic arrivals. While its maximum queue length is the largest, the ability to take server power and load into consideration gives it the edge when dealing with heavy traffic.

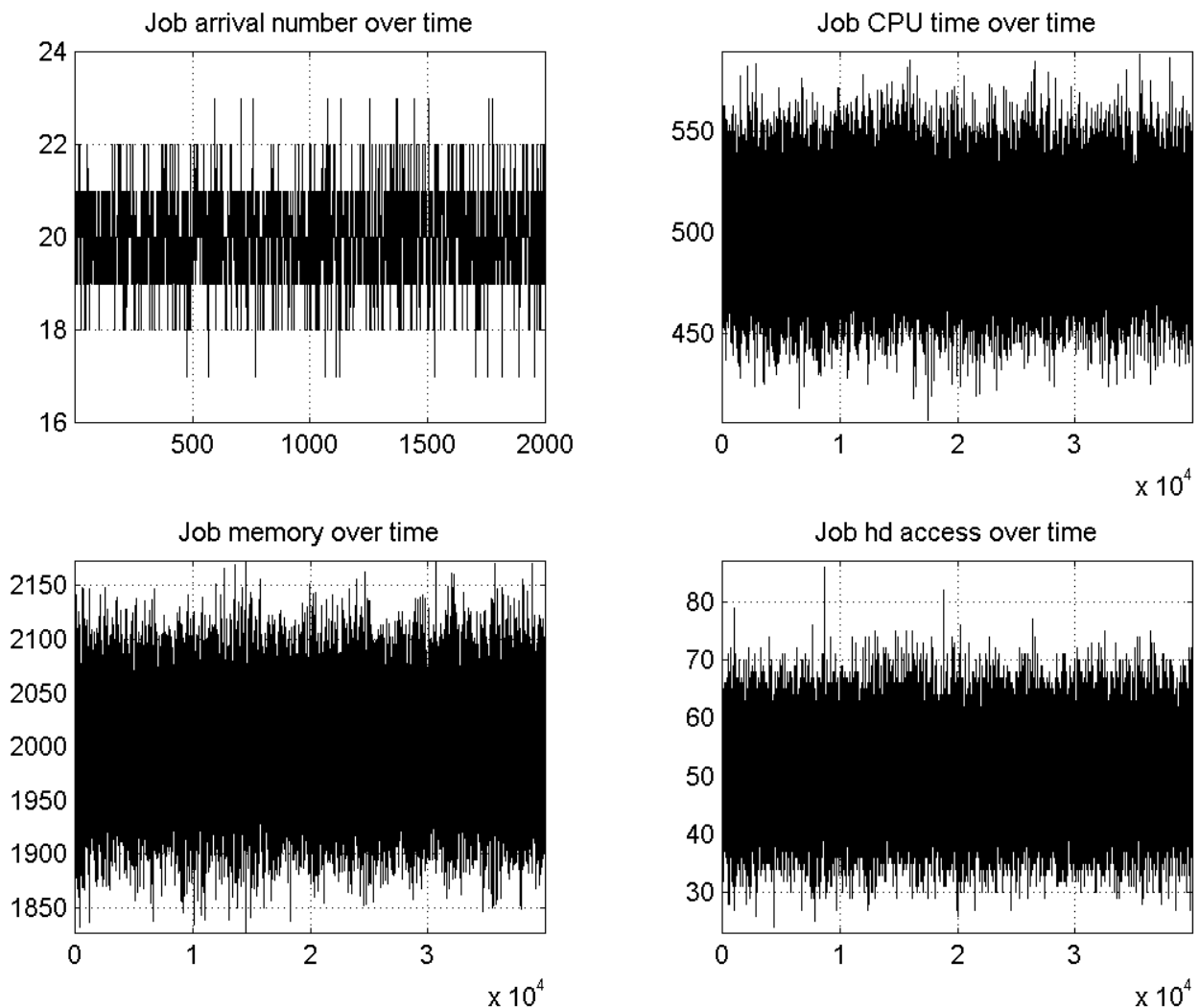


Figure 22 - Traffic Pattern 1 (high arrival rate)

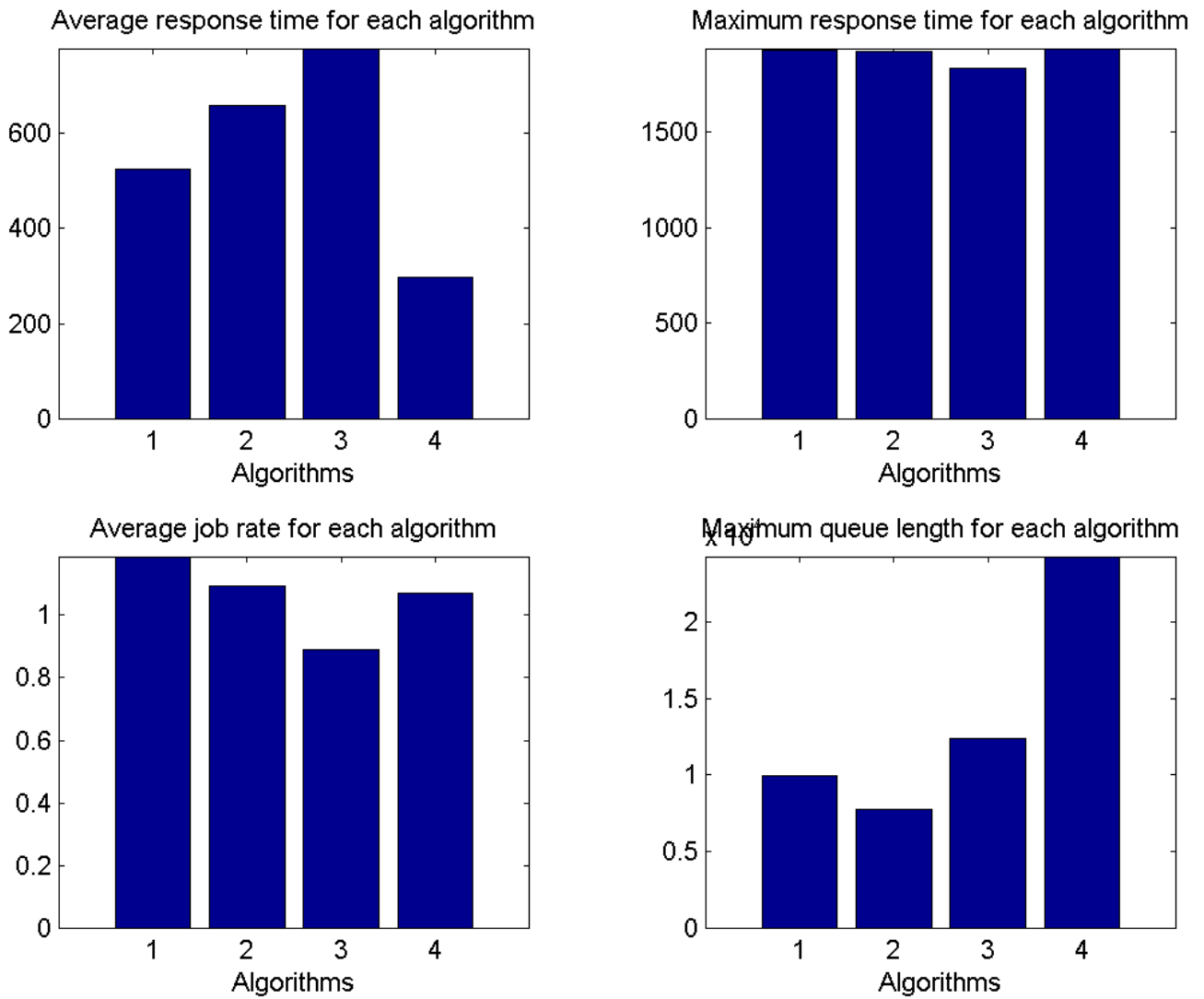


Figure 23 - Results (High arrival rates)

## 3.3.2 Traffic Pattern 2 – Burst in arrival rates

### 3.3.2.1 Base Traffic Pattern 2

The base model for traffic pattern 2 was as follows:

time_loops:	2000
job_range_mean:	8
job_range_sd:	1
mean1 (job CPU time):	500
mean2 (job memory size):	2000
mean3 (job hard drive time):	50
burst_range	500
burst_base1	5
burst_range1	50
burst_base2	10
burst_range2	30

The traffic pattern of this simulation run is shown in Figure 24, together with the performance results contained in Figure 25. Note that the individual job requirements are distributed the same as they are in Traffic Pattern 1. Round robin offers the best average response time, although it gives the worst maximum response time. This seems to be because bursts in arrival rates create short instabilities in server loads that disappear quickly. Round robin deals with this by giving all servers equal numbers of jobs, spreading the burden of the arrival burst. Fuzzy inference achieves the highest average job rate and outperforms least connection and weighted distribution in average response time.

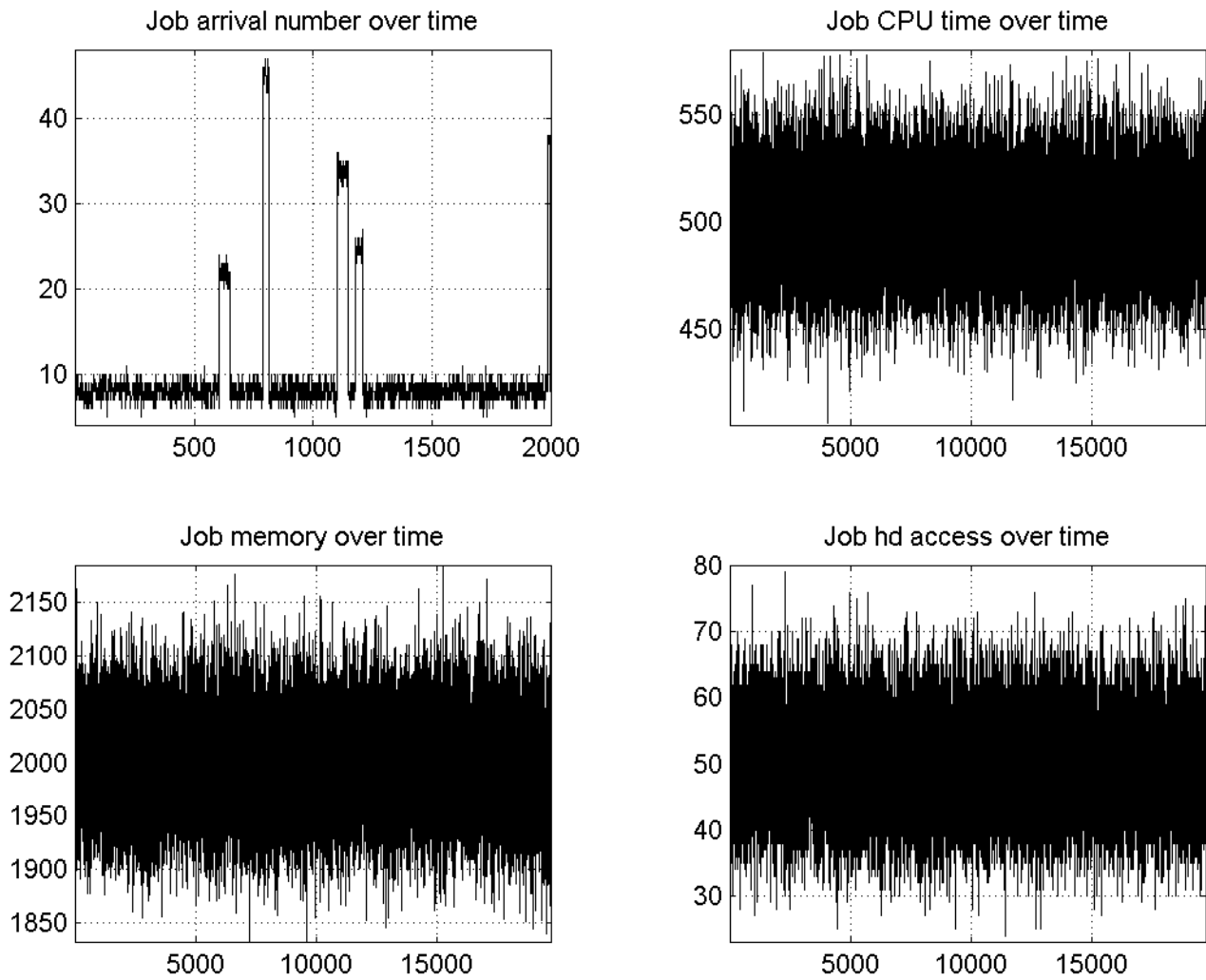


Figure 24 – Base Traffic Pattern 2

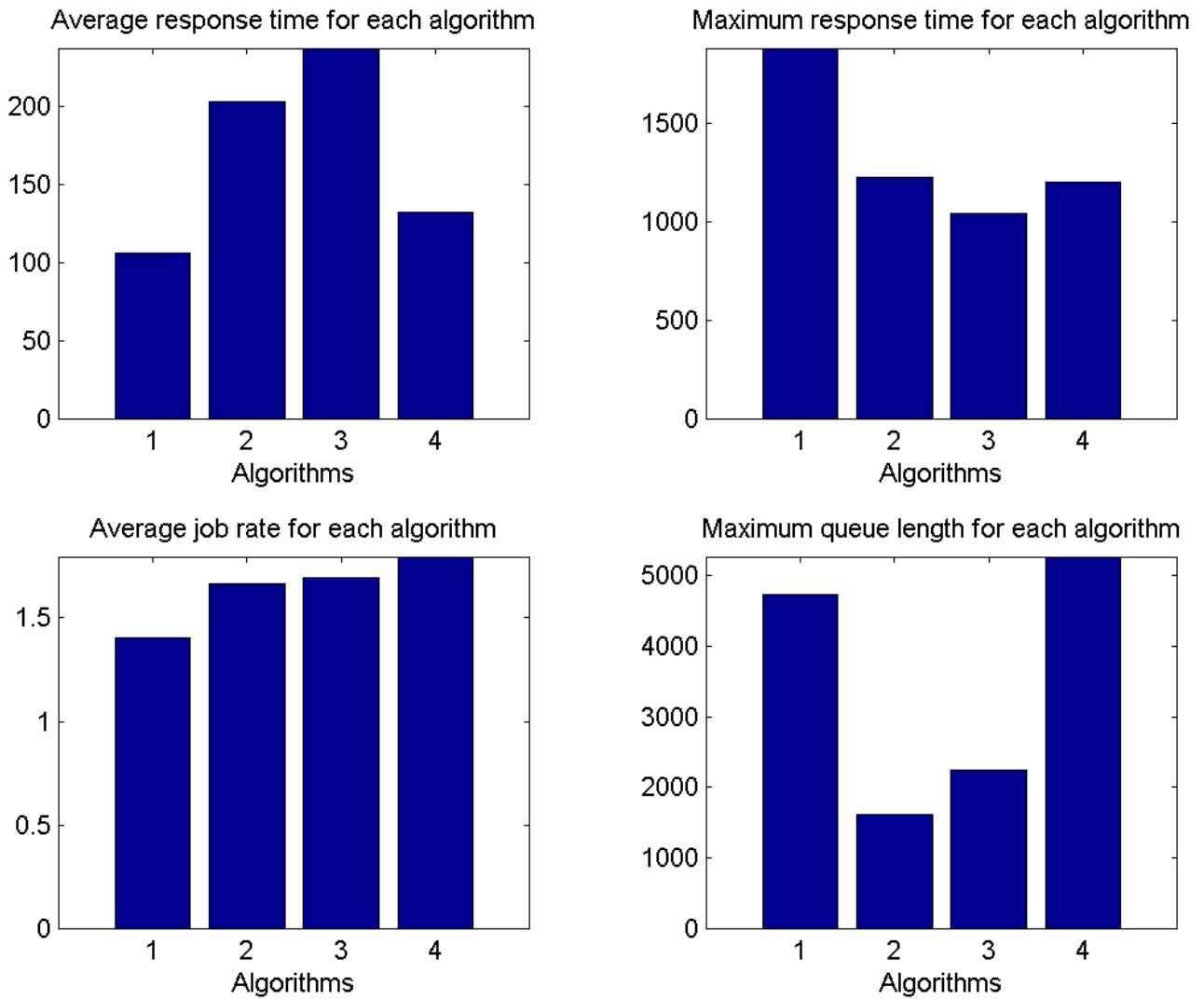


Figure 25 - Results (Base traffic pattern 2)

### 3.3.2.2 Small chance of burst

Changing the chance of a burst happening was achieved by altering the variable `burst_range` from 500 to 1000, which represents a 1 in 1000 chance of a time loop causing a burst in arrival rates. Simulation traffic and results are shown in Figure 26 and Figure 27. The performance is comparable to that of the Traffic Condition 1 (Figure 14) base case, given that the arrival rates seem fairly stable.

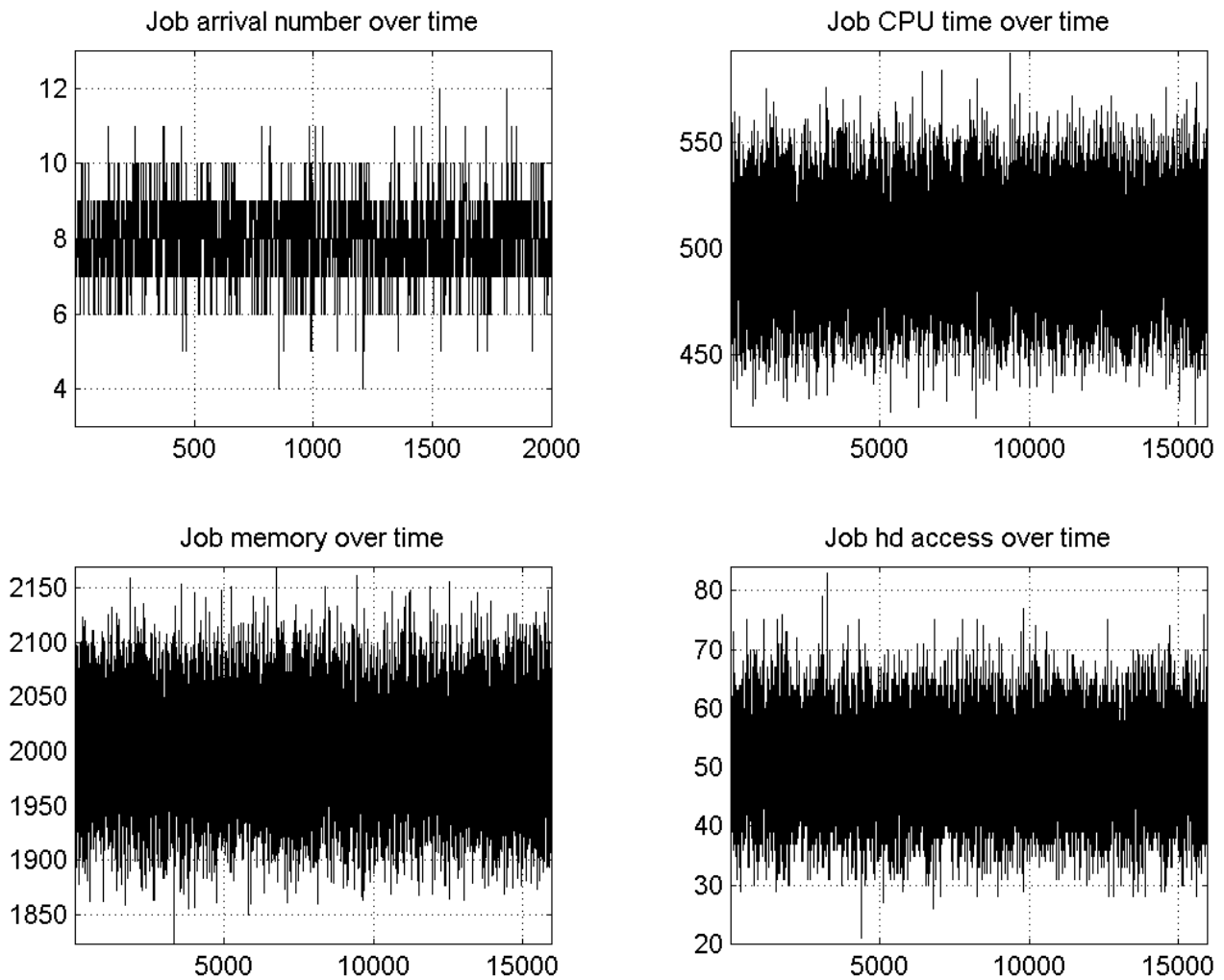


Figure 26 - Traffic Patter 2 (small chance of burst)

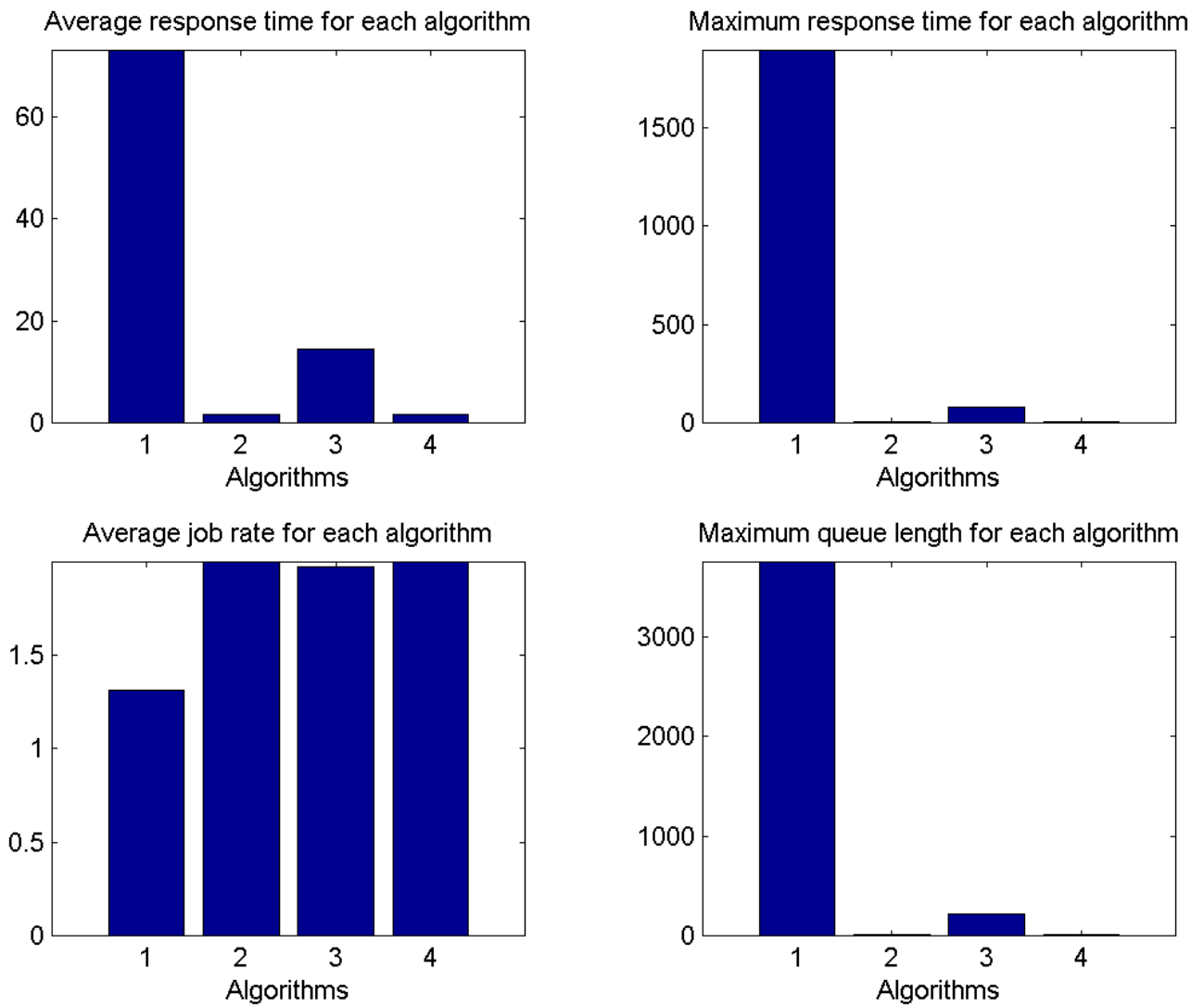


Figure 27 – Results (small chance of burst)

### 3.3.2.3 High chance of burst

The graphs below show the effect of changing the chance of a burst happening from 1 in 500 to 1 in 200 jobs. All other variables are kept the same. Figure 29 shows results that are very similar to the base condition. Average response times have approximately doubled and the job rate has dropped for all algorithms. Once again, the fuzzy inference system has produced the best job rate and the second best average response time.

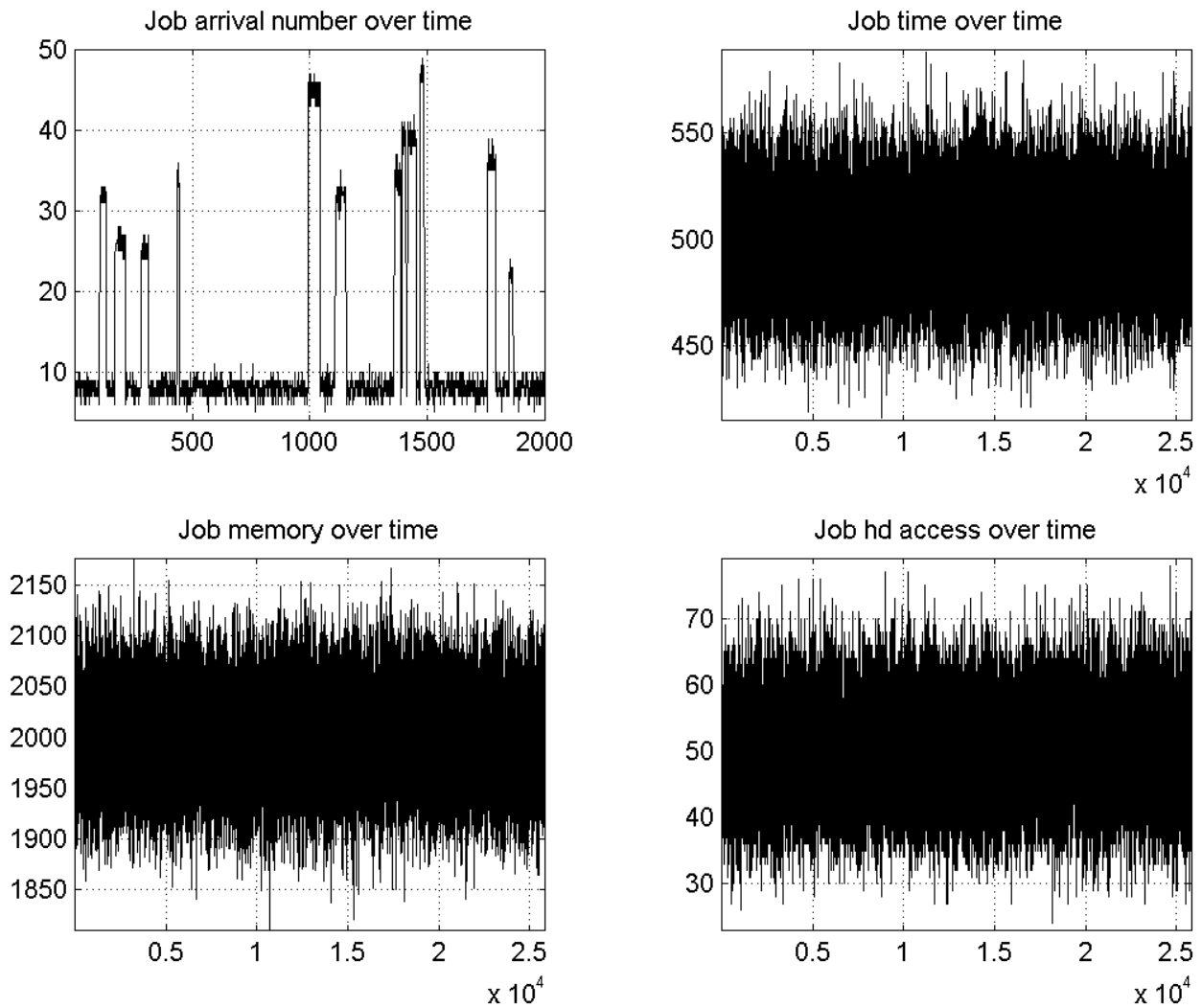


Figure 28 - Traffic Pattern 2 (high chance of burst)

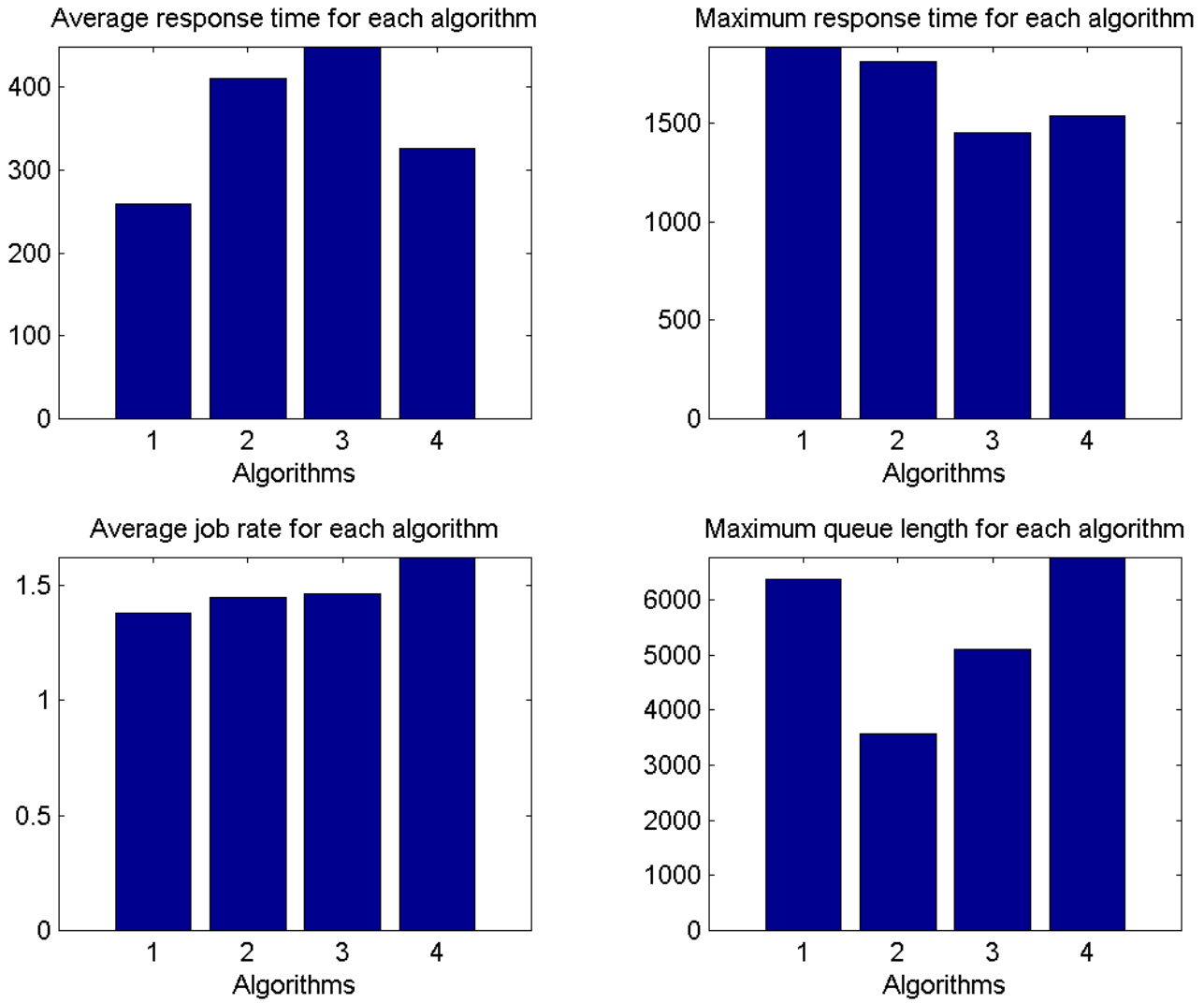


Figure 29 - Results (high chance of burst)

### 3.3.2.4 Short bursts

The next two sections investigate the effect of changing the length of the bursts, not the chance of them happening. The chance of a burst happening is the same as the base Traffic Pattern 2. The following simulation was run using the values of `burst_base1 = 3` and the `burst_range1 = 10`, compared to the base run, which used a `burst_base1 = 5` and `burst_range1` set to 50. Results are show in Figure 30 and Figure 31. The short burst period has created a traffic condition that is very similar to Traffic Condition 1, and the results graph shows that. Short bursts appear to have a minimal effect upon any performance measures.

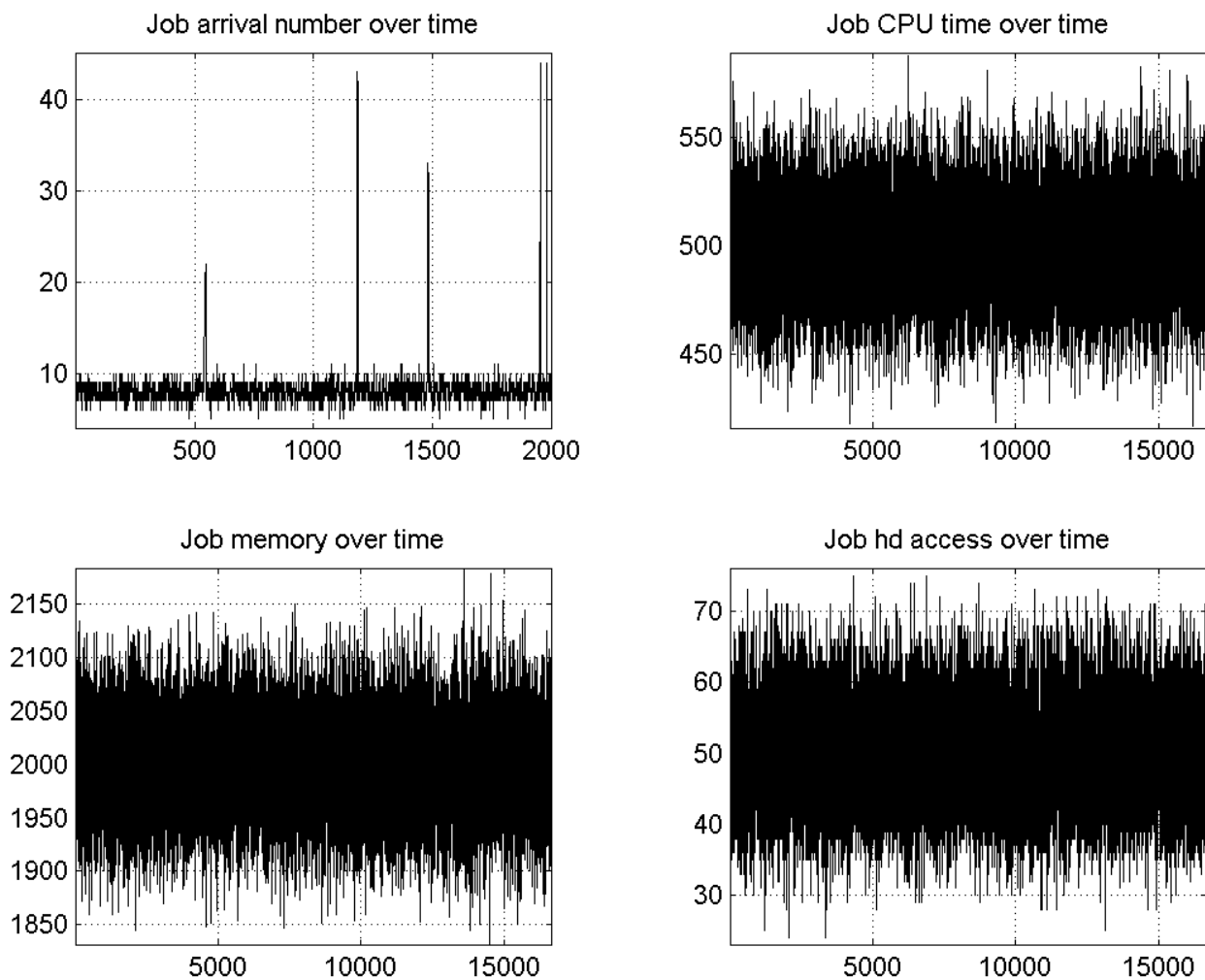


Figure 30 - Traffic Pattern 2 (short bursts)

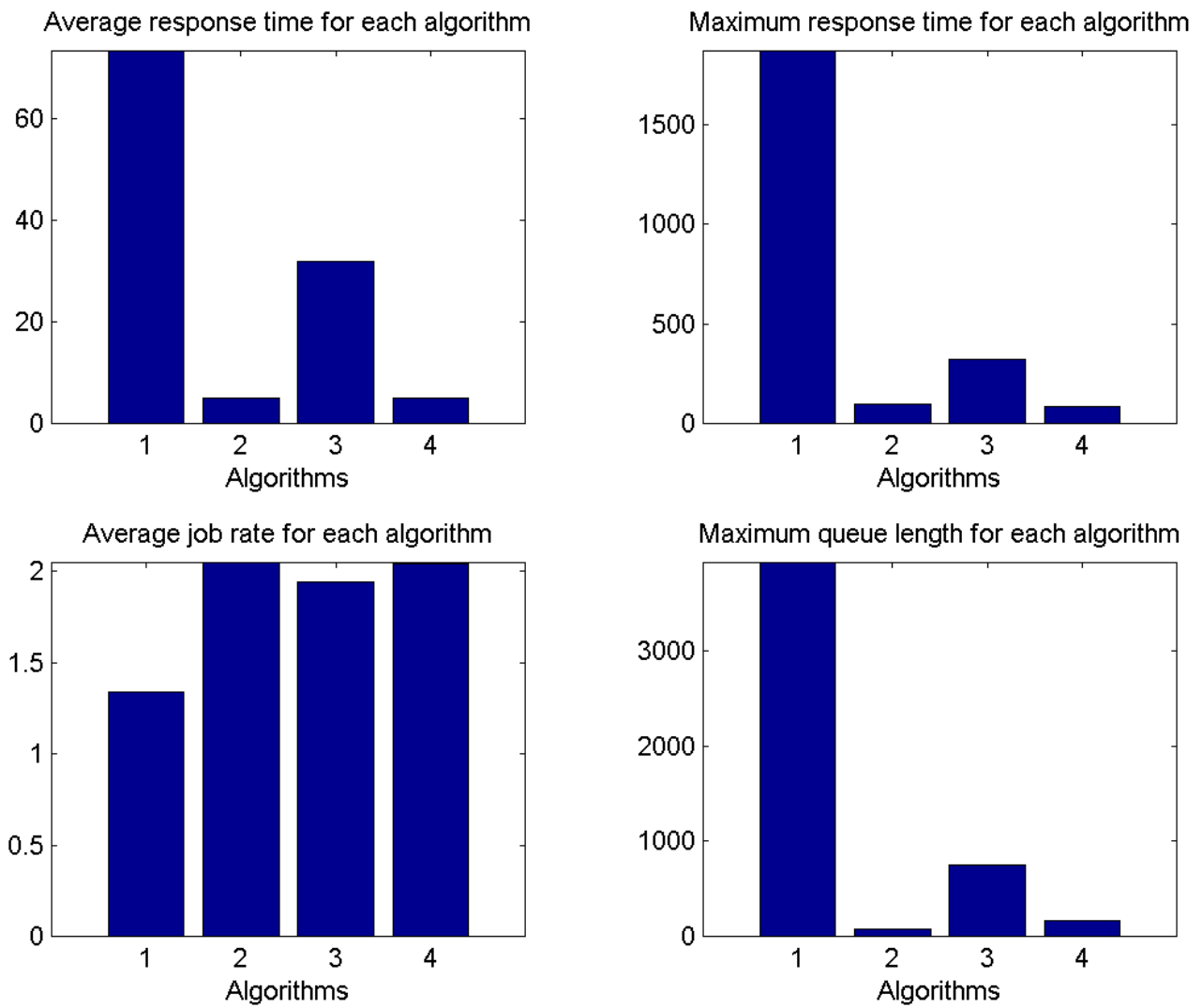


Figure 31 - Results (short bursts)

### 3.3.2.5 Long bursts

This time the length of the bursts was increased to a burst base of 15 and a possible range of 70. Results are shown below in Figure 32 and Figure 33. This time the presence of long bursts in arrival rates has resulted in a similar performance graph to that of the base Traffic Condition 2. A further increase in the length of job arrival bursts will cause the cluster to operate in a similar fashion to that of 3.3.1.4, in which the average arrival rate was raised to 20. Fuzzy inference has offered an average response time that is close to round robin as well as the highest job rate and the second lowest maximum response time. Throughout traffic condition 2, the fuzzy method has shown to be a good compromise between response time and job rate.

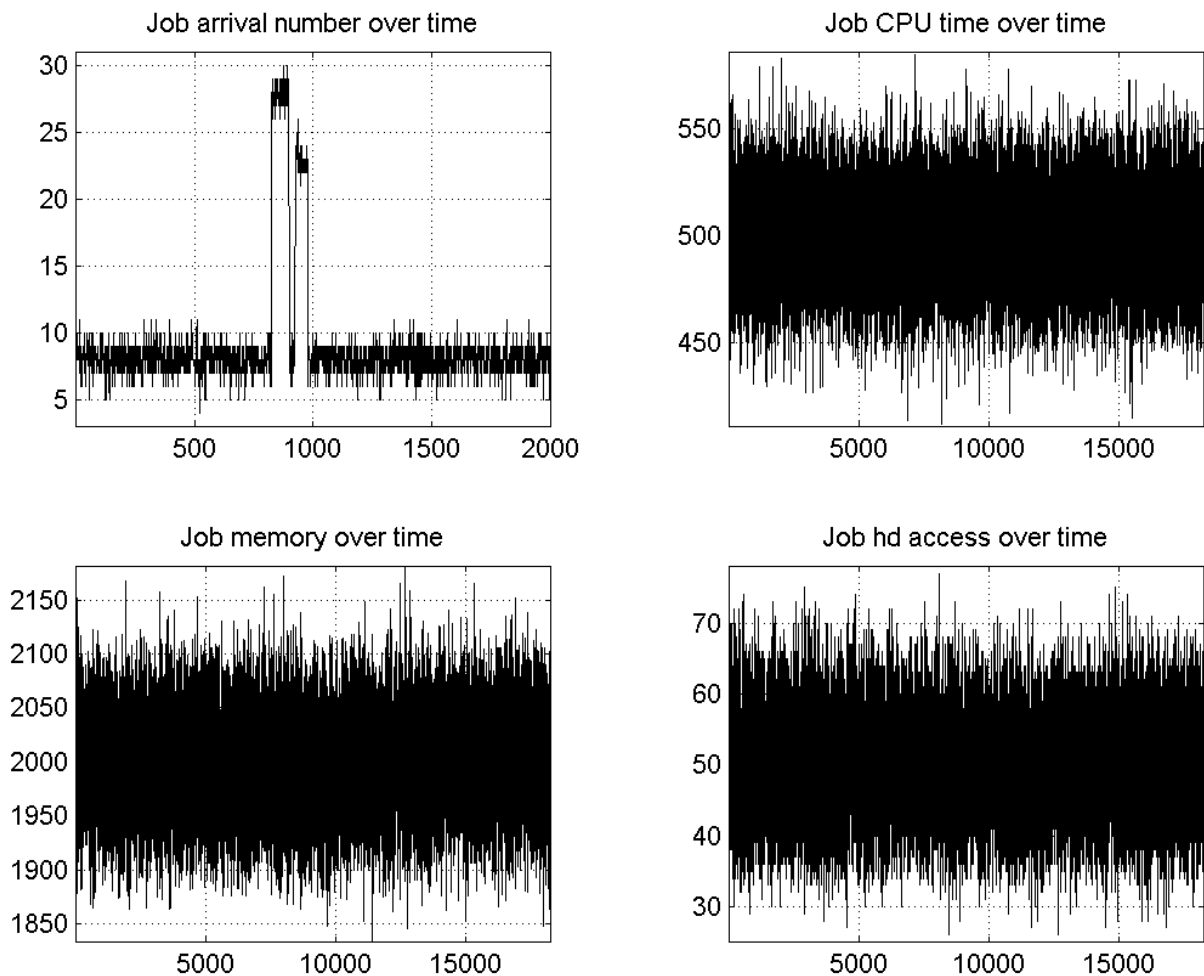


Figure 32 - Traffic Pattern 2 (long bursts)

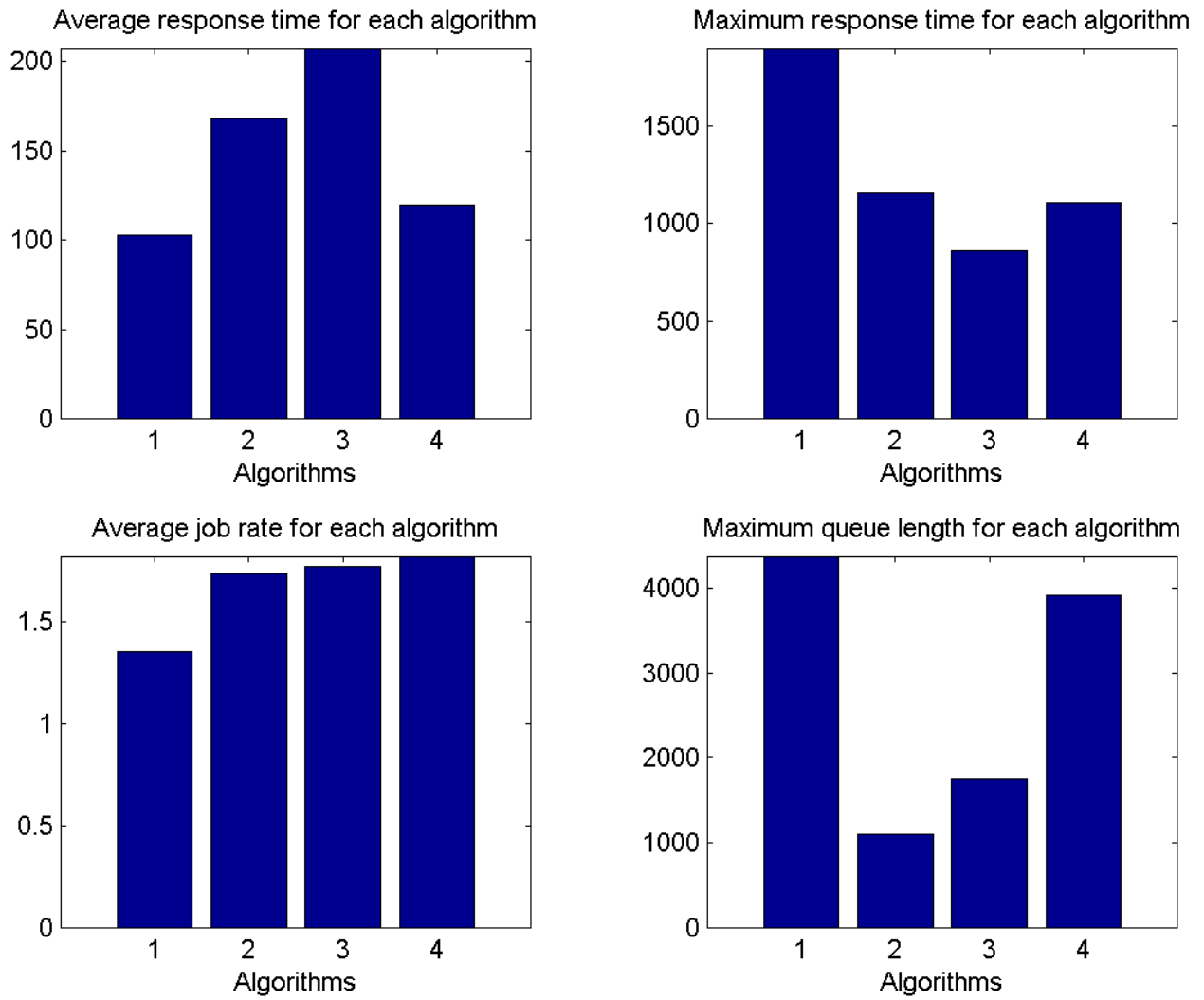


Figure 33 - Results (long bursts)

### 3.3.3 Traffic Pattern 3 – Burst in job requirements

#### 3.3.3.1 Base Traffic Pattern 3

Traffic pattern 3 represents bursts in individual job requirements and keeps the same smooth arrival rate model as traffic pattern 1. The base traffic pattern used for comparison with changes in parameters and other traffic patterns used the following configuration:

```

time_loops:           2000
job_range_mean:       8
job_range_sd:         1
mean1 (job CPU time): 500
mean2 (job memory size): 2000
mean3 (job hard drive time): 50
range1                300
range2                300
range3                300
et                    10
em                    10
eh                    10

```

range1, range2 & range3 refer to the chance of a burst in CPU time, memory and hard drive access occurring. et, em and eh give random multiplication factors for CPU time, memory and hard drive access respectively. This configuration was used in the simulation and the results are shown in

Figure 34 and Figure 35. The performance results are similar to Traffic Condition 1 (3.3.1.1) in that least-connection and fuzzy inference have a very low average response time and the highest job rates. But while the average response times for least-connection and fuzzy inference have only increased by about 0.4, weighted distribution has jumped from 12.5 to 36.4. The maximum queue length for weighted has gone from 198 to 629.

The weakness of weighted distribution lies in its inability to adapt dynamically to changes in network load. If an individual job requires a lot of CPU time or hard drive access, it will cause the server to slow down and become congested. While least connection and the FIS can adapt to this (because the queue at that server will grow), weighted only uses the static CPU power to guide its distribution.

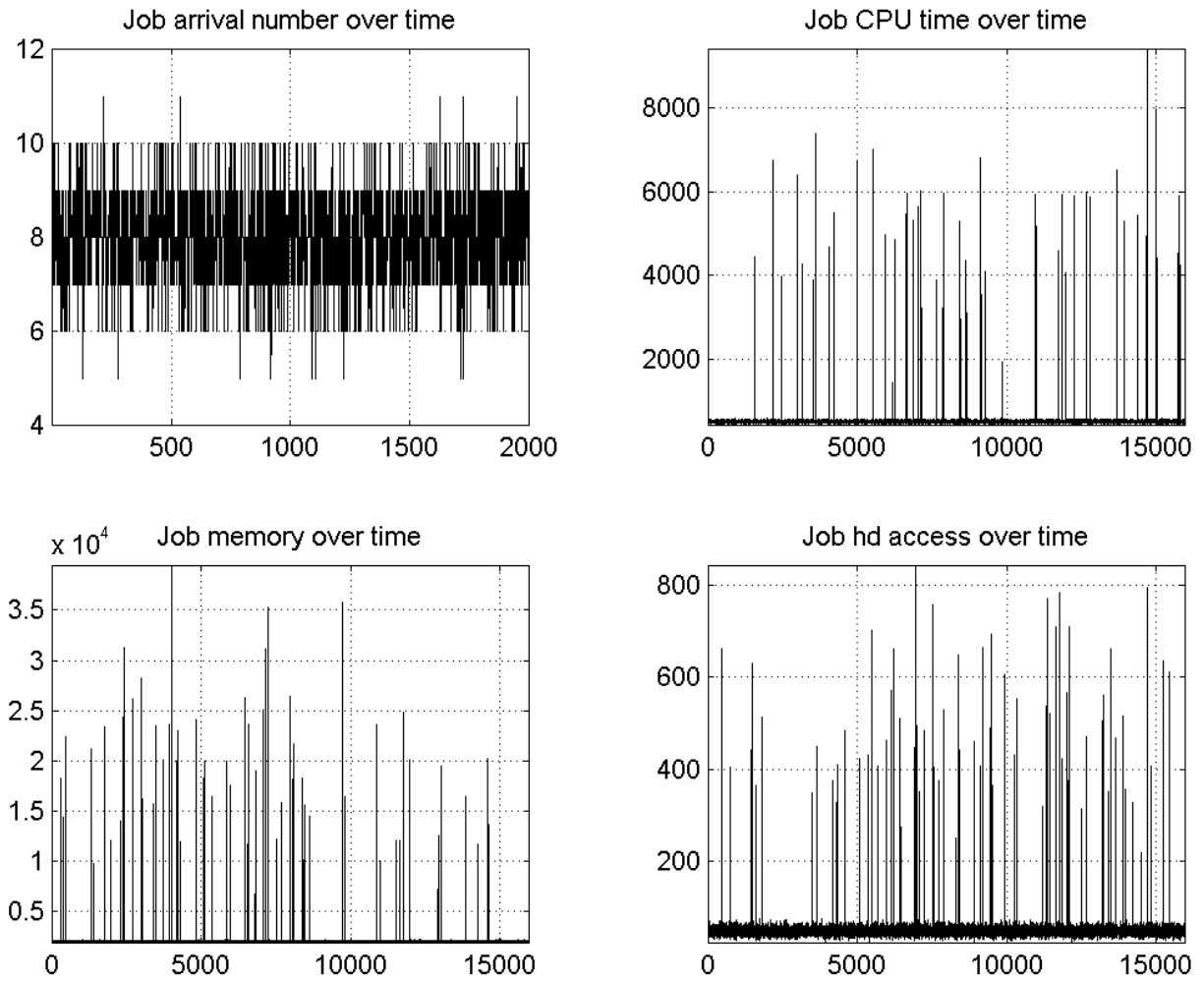


Figure 34 - Base Traffic Pattern 3

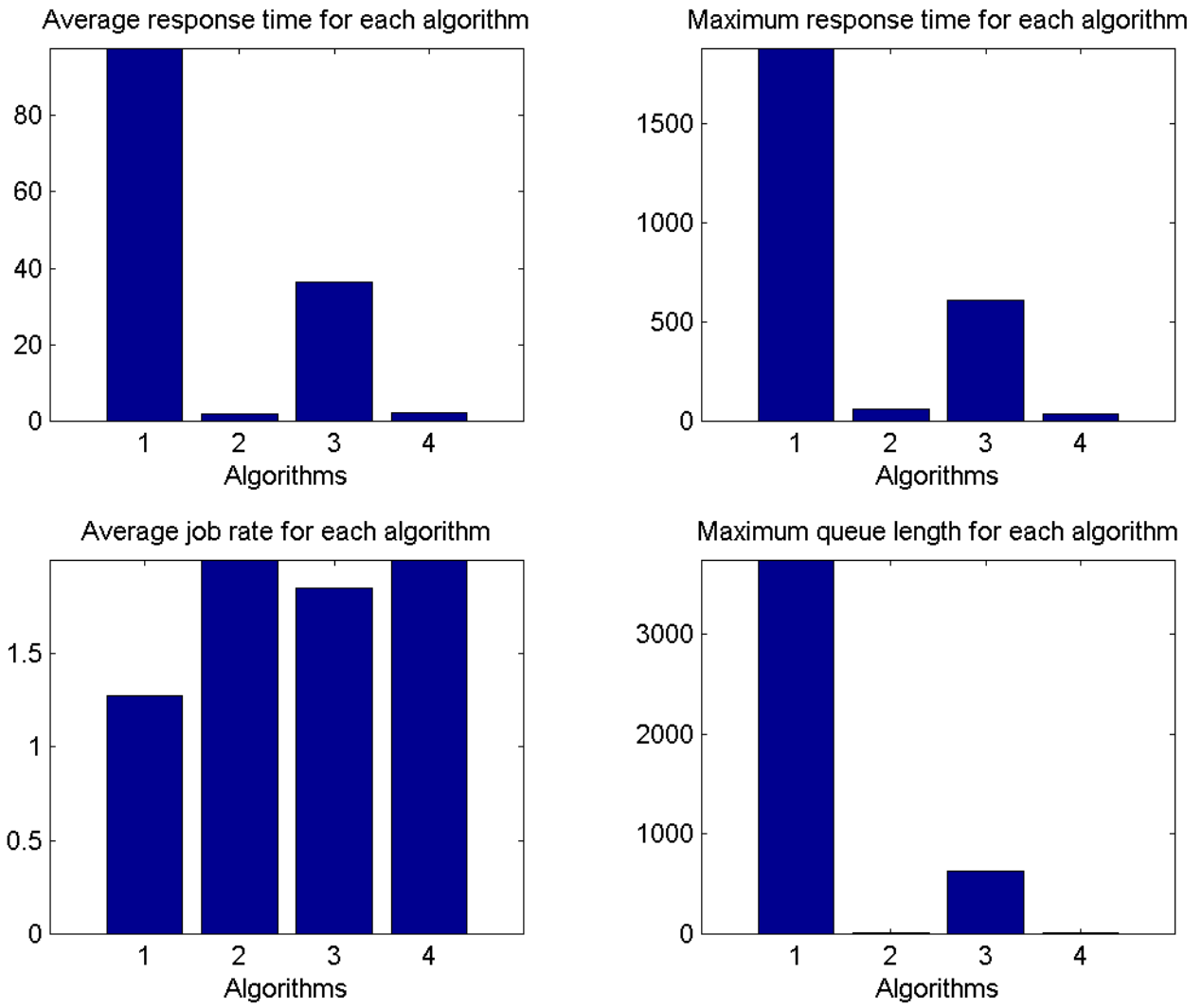


Figure 35 - Results (base Traffic Pattern 3)

### 3.3.3.2 Small chance of burst

This simulation run was implemented by changing the chance of a burst for each attribute to 1 in 2000, compared to 1 in every 300 jobs. The graphs are shown below.

Average response times for least connection and fuzzy inference are roughly 4 each, and weighted is at 20, down 16 from the base traffic. A drop in the chance of a burst happening eases the pressure for weighted distribution, reducing both the average and maximum response time significantly. Round robin performs roughly the same in both tests.

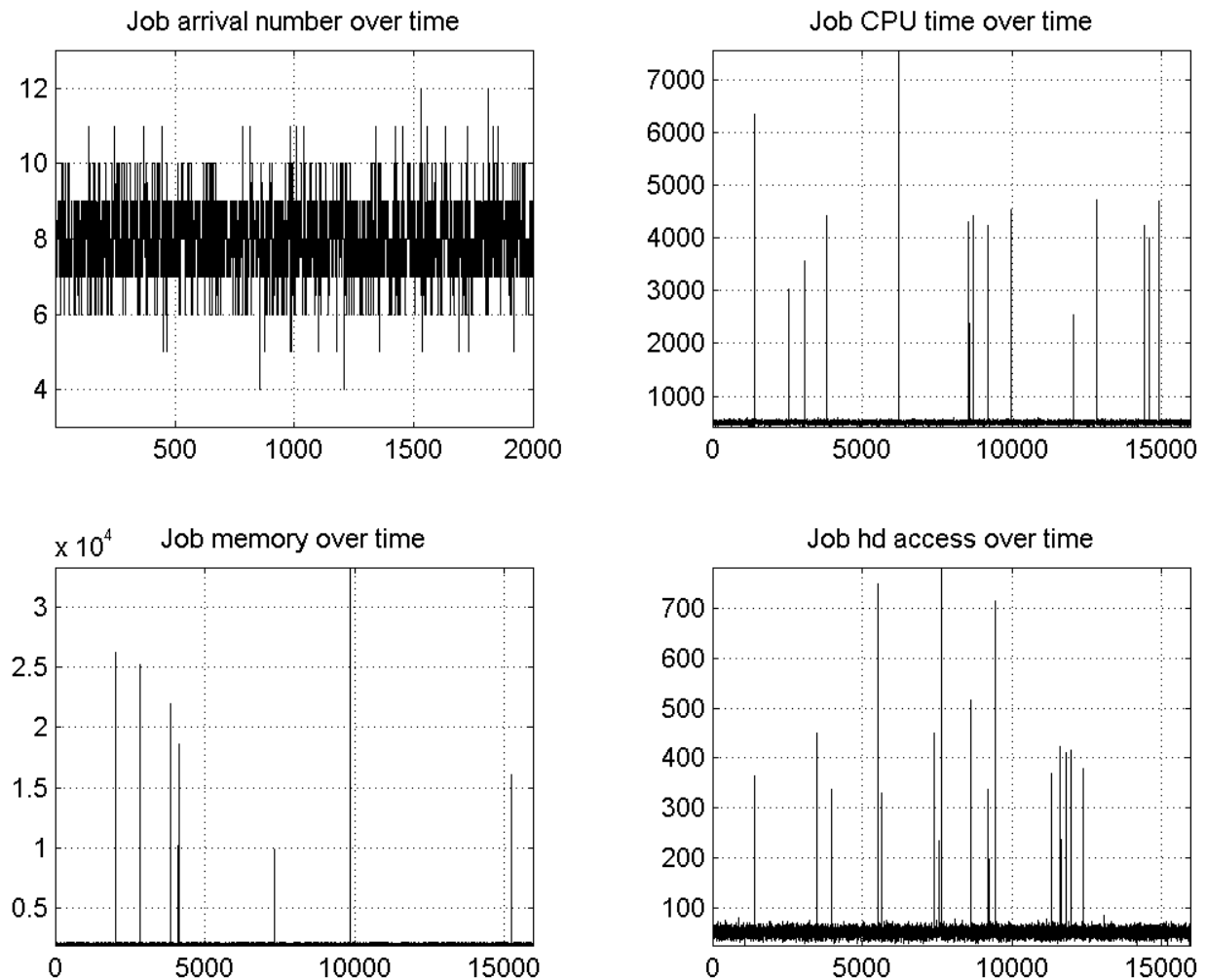


Figure 36 - Traffic Pattern 3 (small chance of burst)

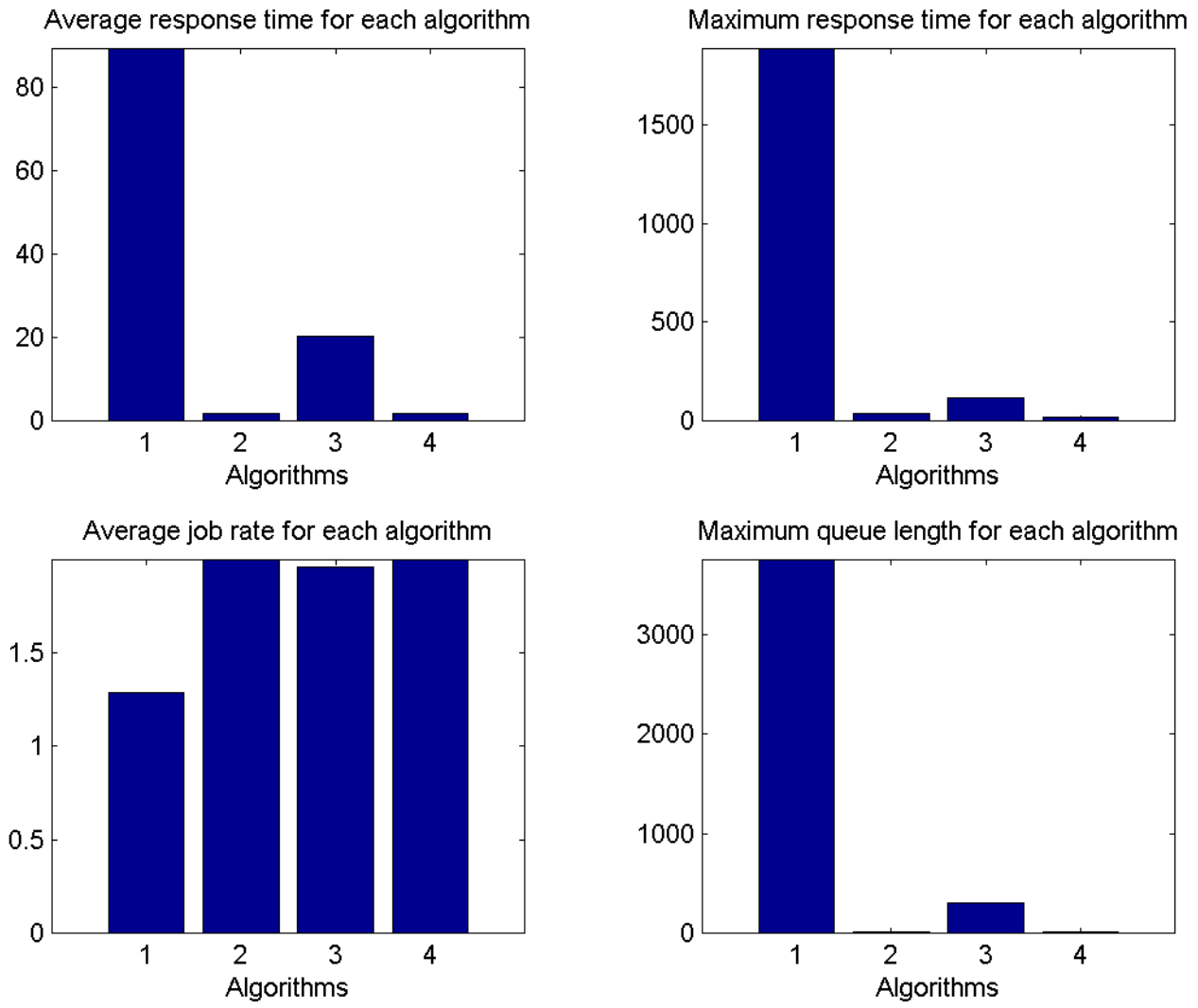


Figure 37 - Results (small chance of burst)

### 3.3.3.3 High chance of burst

This simulation was carried out by altering the chance of a burst for each attribute to 1 in 50 jobs, making it 6 times more likely than the base traffic model. Figure 38 and Figure 39 are shown below. This time weighted distribution provides a higher average response time than any other algorithm, with round robin still giving the worst maximum response time.

When jobs with high requirements frequently congest all servers, their individual powers become irrelevant and it is better to spread the traffic numbers evenly. However, the queue length is still relevant to balancing the load and it can be seen that least connection provides the best average response time and the highest average job rate. The average response time for fuzzy inference is almost triple that of least connection (although it easily outperforms the other two methods). This is because although the FIS gives the greatest consideration to queue length, it still uses CPU power to guide its load balancing decisions, and is therefore susceptible to some of the weakness of weighted distribution.

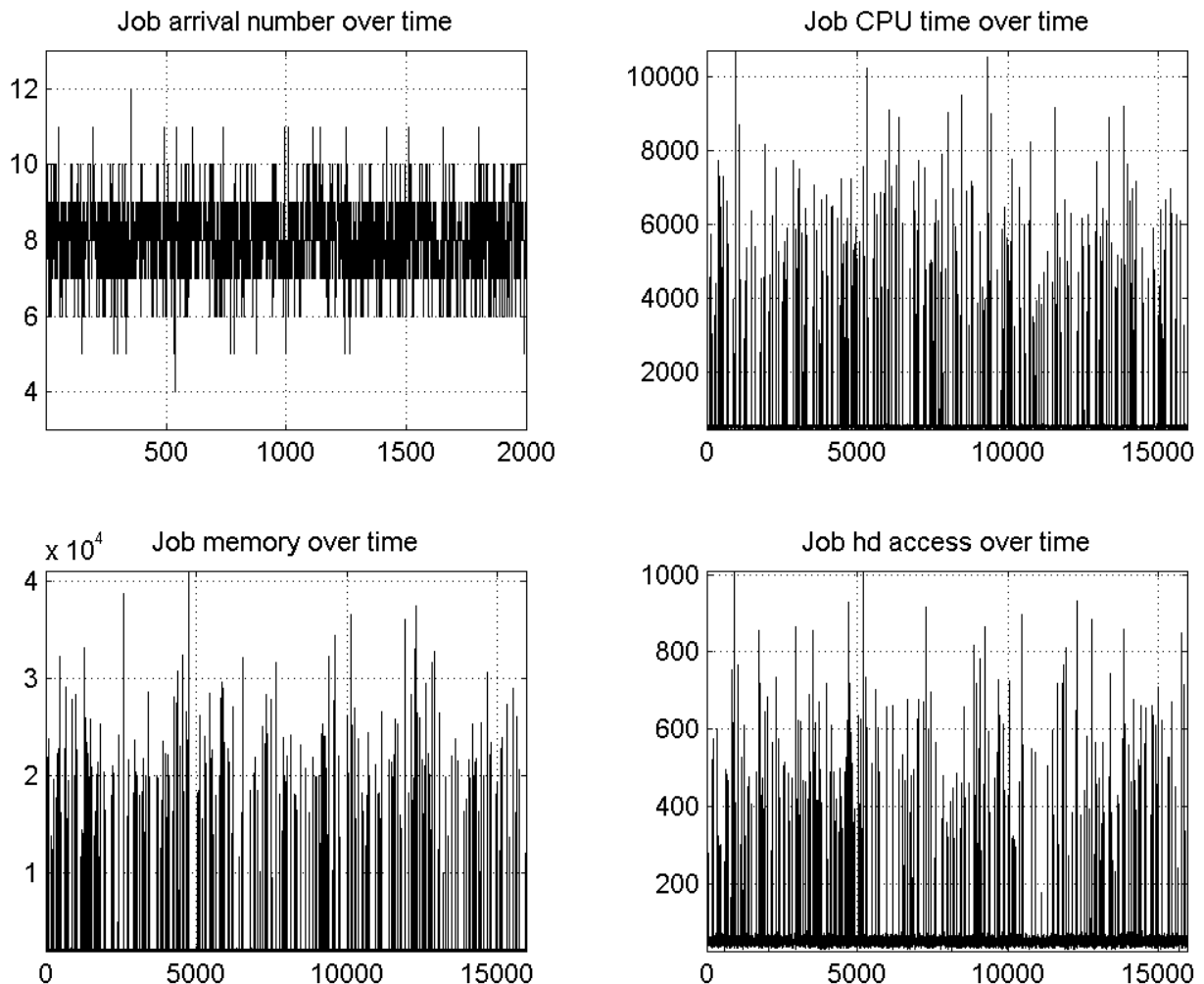


Figure 38 – Traffic Pattern 3 (high chance of burst)

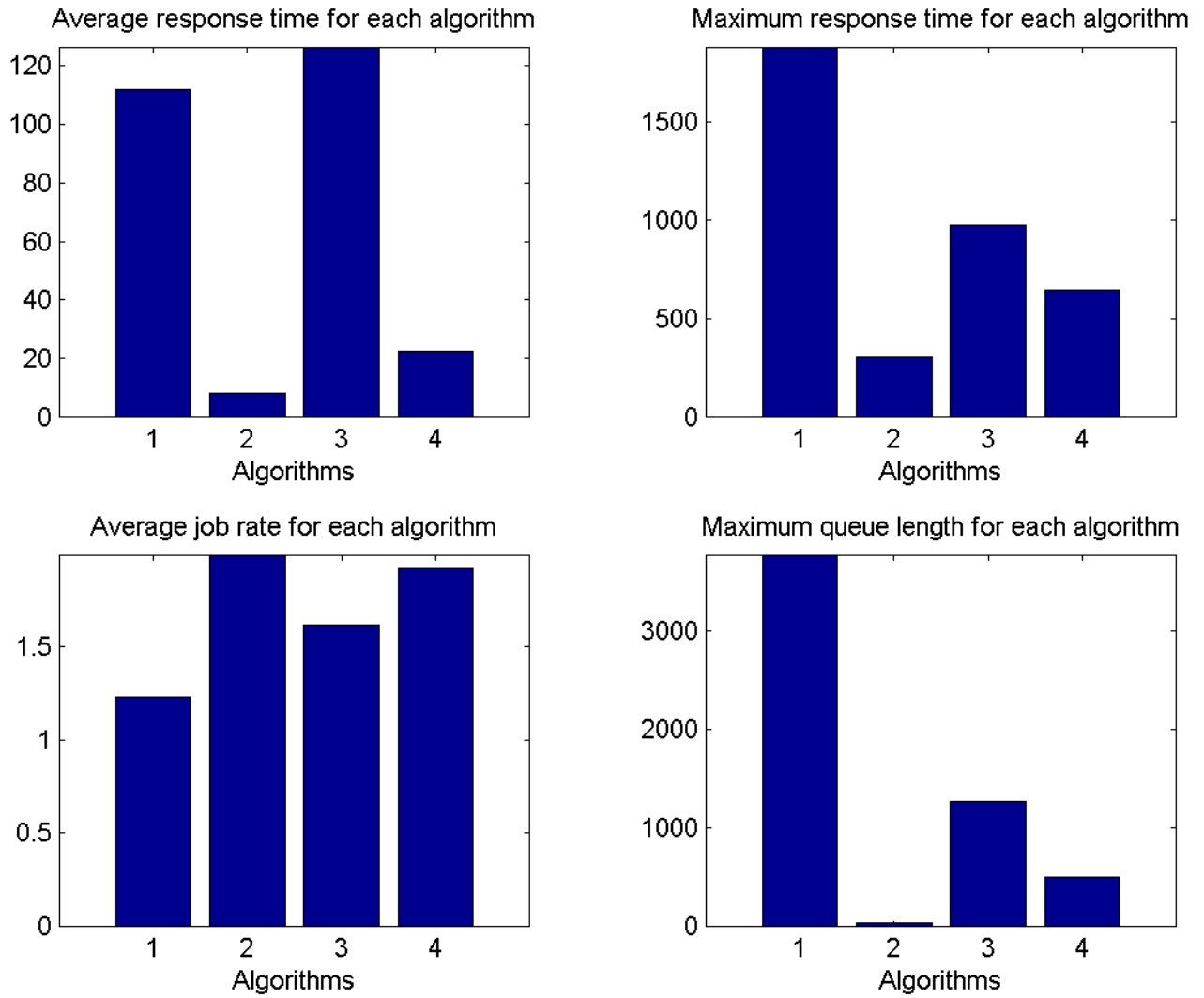


Figure 39 - Results (high chance of burst)

### 3.3.3.4 Small bursts

The following two simulation runs analyse the affect of changing the size of the bursts that occur, and keeping the chance of job bursts fixed at 1 in 300. Figure 40 and Figure 41 show the result of changing the mean size of the burst factor to 3 for each job attribute, as opposed to the default size of 10. The relative performance results are again quite similar in comparison to the base for traffic pattern 3. However, because the bursts are smaller in requirement, the average response time for weighted distribution has dropped from 36 to 25. Least connection and fuzzy inference have only dropped by about 0.5. Decreasing the size of the bursts has lessened the instability in queue length caused by large jobs and has therefore made CPU speed a more accurate predication of load. The average job rate of weighted is also very close to that of least connection and fuzzy inference.

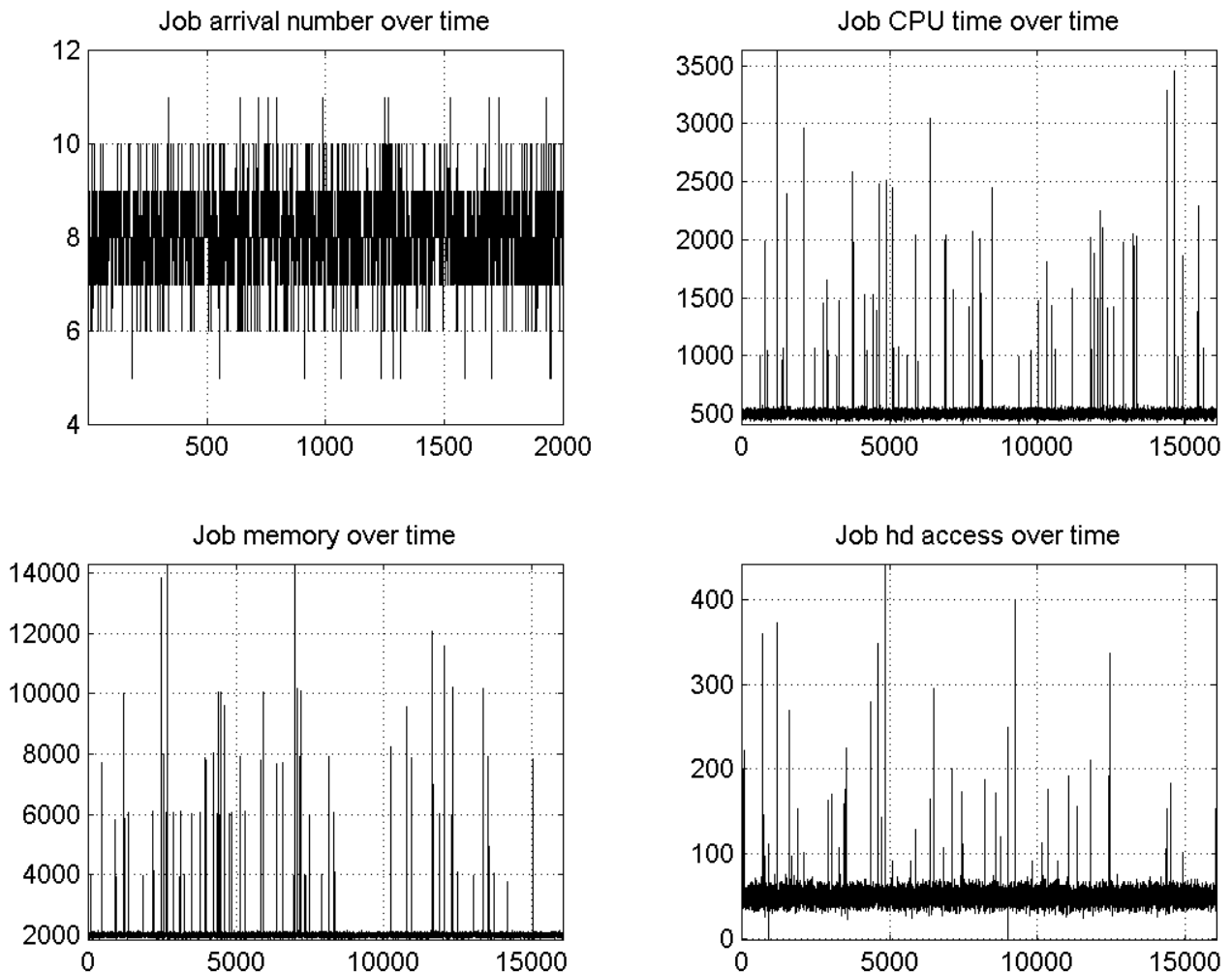


Figure 40 - Traffic Pattern 3 (small burst size)

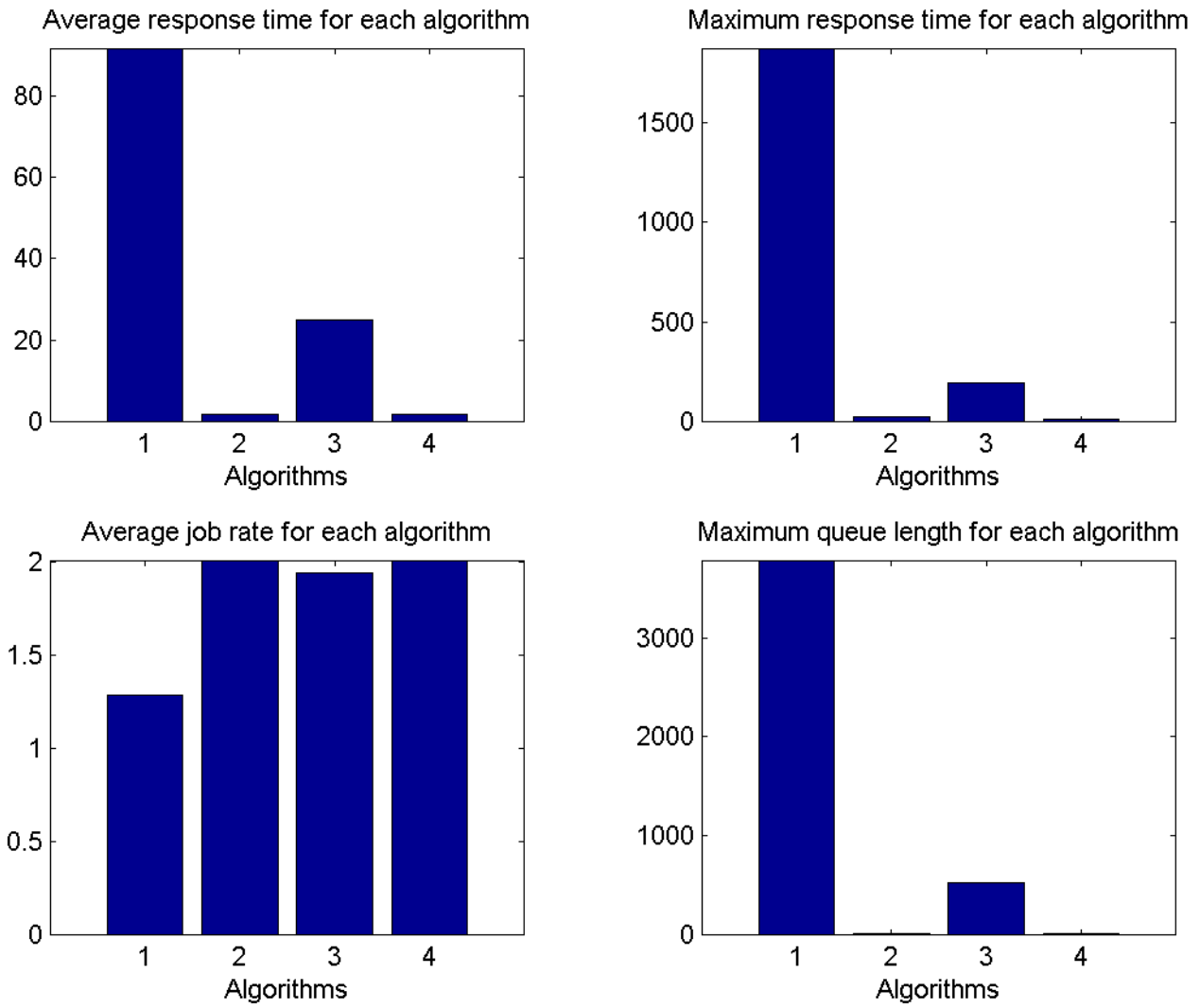


Figure 41 - Results (small burst size)

### 3.3.3.5 Large bursts

The burst factor has been changed from the default size of 10 to 30 for all job characteristics. Results are shown in Figure 42 and Figure 43. Once again, we can see the weakness of weighted distribution, with an increase in average response time of over 40 from the traffic condition 3 base case. The least-connection and fuzzy inference methods rose by 3.8 and 4 respectively. The average job rate of weighted distribution has also dropped slightly. Note the rise in maximum queue length for weighted distribution of 493, compared with 23 and 32 for least-connection and fuzzy inference.

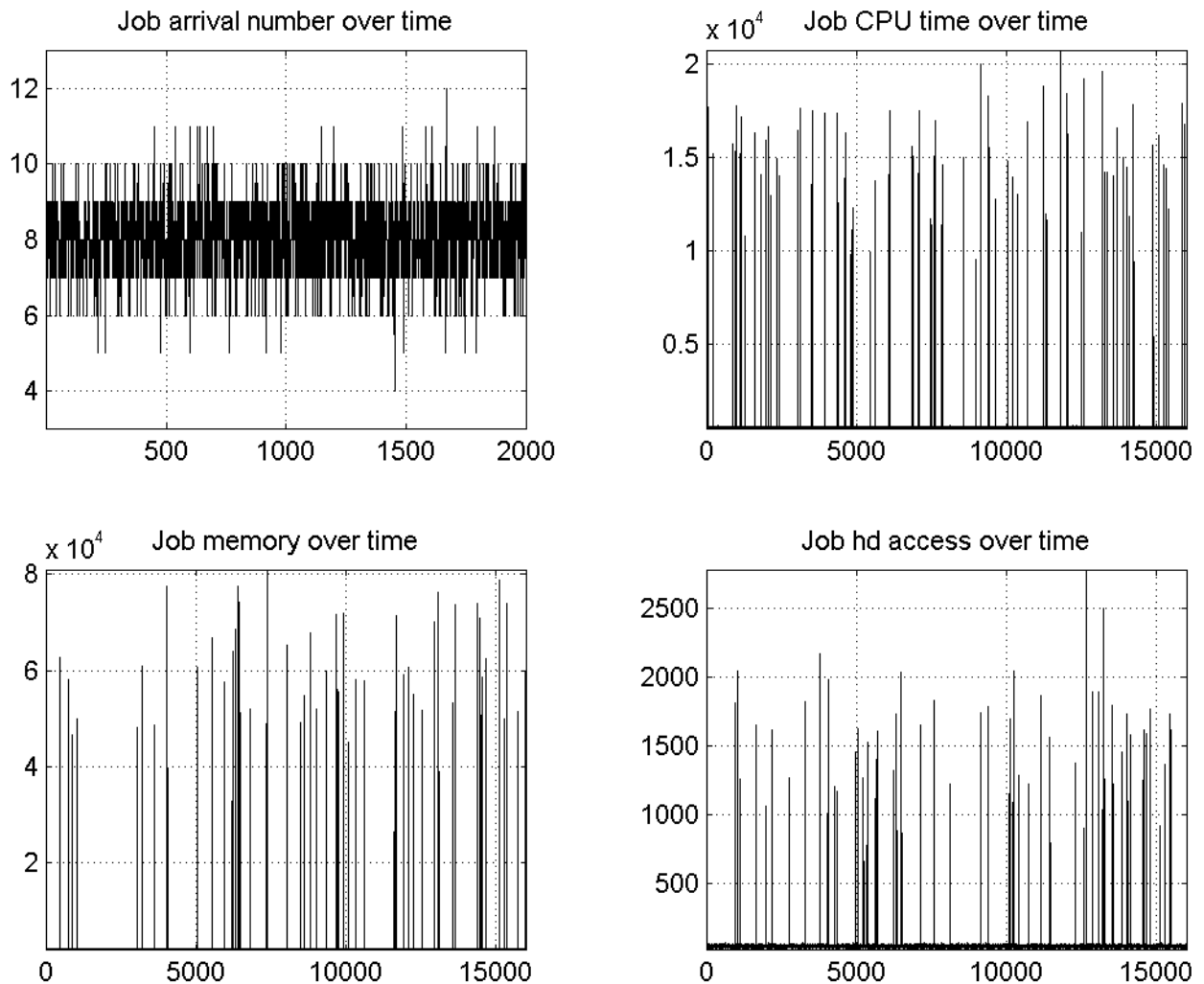


Figure 42 - Traffic Pattern 3 (large burst size)

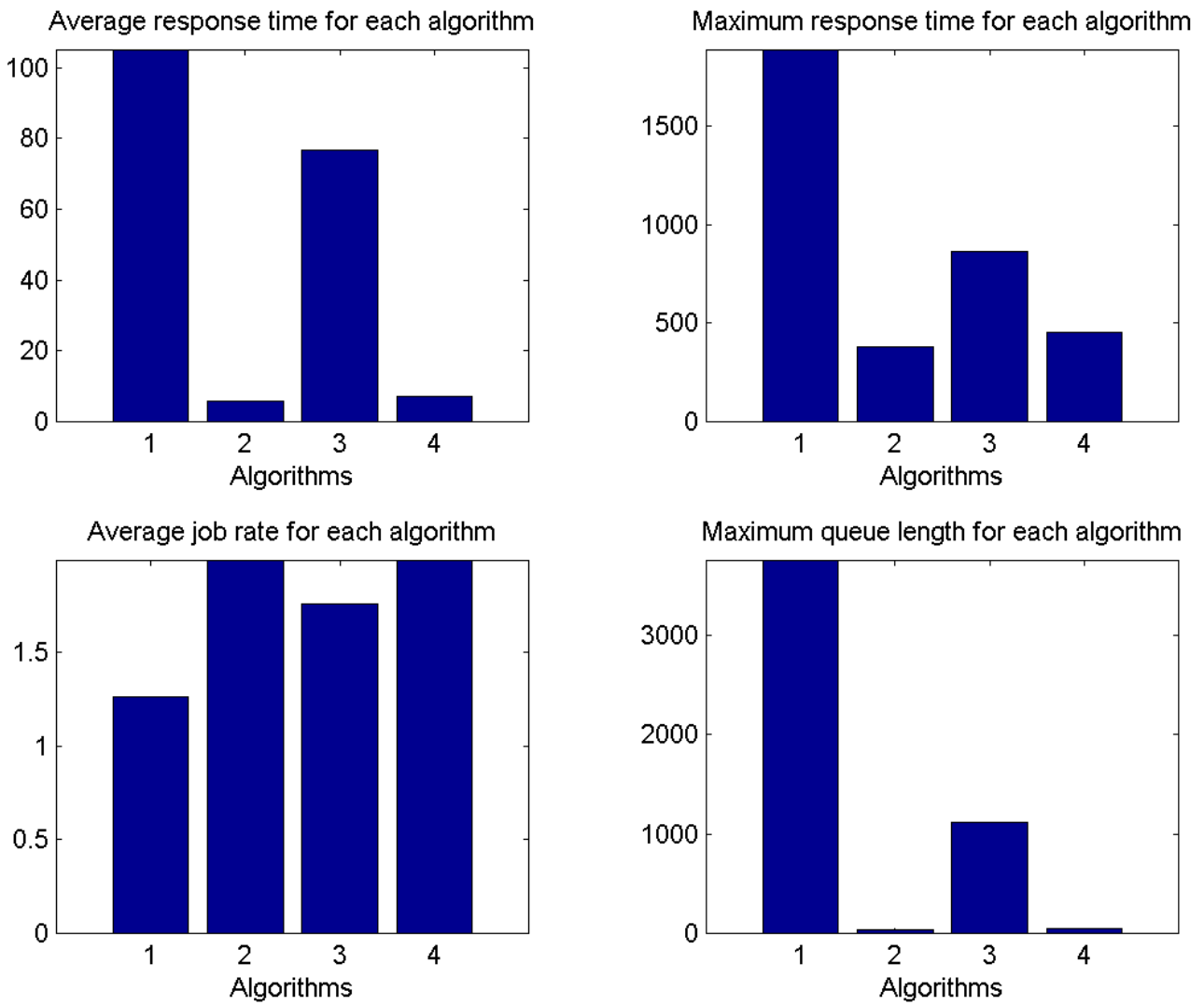


Figure 43 - Results (large burst size)

### 3.4 Performance Results

Both least connection and the FIS work well in traffic condition 1. They offer low average and maximum response times and high job rates. They are not as effective in traffic condition 2; where bursts in arrival rates tend to overload the servers and decrease the effectiveness of a dynamic load balancing technique. They are particularly useful in traffic condition 3 where they can adapt to the changing load caused by bursts in job requirements. Compare this to traffic condition 2, where overloads are caused by arrival rates that affect all servers equally (rather than just the size of one server's queue). Of the existing techniques, least connection is by far the most effective.

Weighted distribution is most effective in situations where servers are equal in power, or their CPU speed is a strong indicator of overall power. It is not effective when bursts in arrival rates or job requirements occur, because it is a static method incapable of adapting to changing network loads.

The more servers there are in a cluster, the less it matters what method is used. If the servers are mostly idle, there is no great need to balance load and a simple method such as round robin or weighted distribution can be used.

When a cluster is subject to a high arrival rate (see 3.2.1, 3.3.1.4, 3.3.2.3 and 3.3.2.5) the most effective method becomes round-robin. This is because the number of arrivals is evenly distributed amongst all servers. Balancing using the queue length becomes less effective in this situation because all server's queues are growing without bound. The FIS method provides a compromise between queue length and CPU power that offers a good alternative in these overload situations.

## 4 CONCLUSIONS

The research in this thesis was motivated by the need to improve upon existing load balancing methods by the application of a relatively new technique called fuzzy inference. It has focussed on the development and implementation of a simulation, and the findings were based on comparisons between existing algorithms and the fuzzy inference system that was developed.

The fuzzy inference method does not always achieve the lowest average response times and job rates and is usually beaten by least connection for the lowest maximum response time. However, it provides stability across a range of traffic conditions, especially in the situation of arrival overloads. It is an intuitive approach to load balancing that can easily incorporate more cluster characteristics.

The system developed and implemented in this research has provided a strong load balancing method. It uses the strength and stability of least connection by incorporating queue length, but also factors in the CPU power of a server to direct jobs to those with processing power. While the fuzzy method uses high level rules to provide an intuitive system that deals with the uncertainty in load balancing, it is far from perfect. If the number of inputs is increased, then the number of possible rules increases exponentially, making it harder to develop simple rules that drive the system.

The uncertainty of a fuzzy system makes it susceptible to changes in traffic conditions. The system needs to be designed to suit the current traffic conditions, and any change could alter its performance. That being said, the use of fuzzy logic is a powerful tool for a developer with a good working knowledge of the problem at hand. The uncertainty of fuzzy logic is its greatest strength, but also its greatest weakness.

### 4.1 Future Work

As mentioned in section 2.2.1, there are a number of limitations to this research. The simulation assumes a simplified version of web traffic and server operation. Any enhancements to these aspects of the simulation would provide more realistic results.

A basic fuzzy system has been developed with two simple inputs. Other possible inputs and enhancements to the simulation include past performance (response times, job rates), network line speed, cache size/speed, types of client requests (http, ftp, cgi, etc).

Ideally, a fuzzy system that has been tried and tested under a simulation could eventually be compiled and implemented on an actual web server cluster with real client traffic. This would give solid feedback as to the effectiveness and run time speed of the fuzzy inference system.

## 5 BIBLIOGRAPHY

1. Schroeder T. Goddard S. Ramamurthy B. Scalable web server clustering technologies. In IEEE Network, vol.14, no.3, May-June 2000, pp. 38– 45, USA, 2000. IEEE.
2. Kopperapu, Chandra. Load balancing servers, firewalls, and caches. Wiley 2002, pp 5 – 32.
3. Shirazi, Behrooz A. Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE 1999 pp. 310 – 311.
4. Eager, D. L., E. D Lazowska, and J. Zahorjan, “A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing”, Performance Evaluation, Vol. 6, 1986 pp 53-68.
5. Wang, Yung-Terng and Morris, J.T. “Load Sharing in Distributed Systems”. IEEE Trans on Computers, vol. C-34 no.3, Mar 1985 pp 204-217.
6. Shivaratri, Niranjana G. et al. “Load Distributing for Locally Distributed Systems”, Computer vol. 25 no. 12, Dec. 1992, pp 33-44.
7. Aweya J. Ouellette M. Montuno DY. Doroy B. Felske K. An adaptive load balancing scheme for web servers. International Journal of Network Management, Jan-Feb 2002, v.12 pp. 3–39, 2002.
8. Eager, Derek L. et al “Adaptive Load Sharing in Homogeneous Distributed Systems”. IEEE Trans on Software Eng. Vol. SE-12, no.5, May 1986, pp. 662-675.
9. Mazingo S. Internet server load balancing. Digital Systems Report, Summer 1999, v.21 pp. 27–29, 1999.
10. Hyeong-Ah Choi Li-Chuan Chen. Approximation algorithms for data distribution with load balancing of web servers. In Proceedings 2001 IEEE International Conference on Cluster Computing, pp. 274–81, Los Alamitos, CA, USA, 2001. IEEE Comput. Soc. 2001.
11. Kaario K. Hamalainen T. Jian Zhang. Tuning of QOS aware load balancing algorithm (qos-lb) for highly loaded server clusters. In Networking - ICN 2001. First International Conference on Networking. Proceedings, Part II (Lecture Notes in Computer Science Vol.2094). Springer-Verlag.
12. Kunz T., “The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme” IEEE Trans. Software Eng., Vol. 17, No. 7, July 1991, pp. 725-730.
13. Livny M. and Melman M., “Load Balancing in Homogeneous Broadcast Distributed Systems” Proc. ACM Computer Network Performance Symp., 1982, pp. 47-55. Proceedings printed as a special issue of ACM Performance Evaluation Rev., Vol. 11, No. 1, 1982, pp. 47-55
14. Sysmaster Server Cluster Load Balancing. Available from: [http://www.sysmaster.com/support\\_tech\\_guide1.htm](http://www.sysmaster.com/support_tech_guide1.htm) [Accessed on 9 July 2003]

15. LVS (Linux Virtual Server Project)  
Available from: <http://www.linuxvirtualserver.org> [Accessed on 5 July 2003]
16. Zadeh, L.A., "Fuzzy Sets", *Information and Control*, 8 (1965), pp. 338-353.
17. Zalinda, Othman, Khairanum Subari and Norhashimah Morad, Application of Fuzzy Inference Systems and Genetic Algorithms in Integrated Process Planning and Scheduling. *International Journal of The Computer, The Internet and Management*, Vol. 10, No2, 2002, pp. 81 – 96.
18. Shaout A. and McAuliffe P. Job scheduling using fuzzy load balancing in distributed system. *Electronics Letters*, 1 Oct 1998, 34:1983–1985, 1998.
19. S. Abu Ayyash, S. Hariri and K. Jabbour, An Expert System Approach to Load Balancing in a Distributed Environment, 36th Midwest Symposium on Circuits and Systems, August, 1993.
20. D. Gupta and P. Bepari. Load sharing in distributed systems. Proc. National Workshop on Distributed Computing, Jadavpur University, Calcutta. January 1999.
21. Chin Lu, C.S. Lui, W.K.Lie, Peter Lie, Micheal Tang, Doris Lau, and Joy Li, Distributed Scheduling Framework-A Load Distribution Facility on Mach, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96), Sunnyvale, Aug. 7-9, 1996, pp.757 – 768.
22. A. Osman, H. Ammar. Dynamic Load Balancing Strategies for Parallel Computers.  
Available from : <http://citeseer.nj.nec.com/osman02dynamic.html> [Accessed on 28 July 2003]
23. Sacha Dierkes. Extension of Bayesian decision theory and fuzzy logic to decision machines for load balancing.  
Available from <http://citeseer.nj.nec.com/dierkes96extension.html> [Accessed on 28 July 2003]
24. S. Dierkes. Load Balancing with a Fuzzy Decision Algorithm. Proceedings of the International Panel Conference on Soft and Intelligent Computing, Budapest, Hungary, 7- 10. October 1996.
25. C. Park, J.G. Kuhl, A fuzzy based distributed load balancing algorithm for Large Distributed Systems. Proceeding of the Second International Symposium on Autonomous Decentralized Systems (ISADS'95) April 25 - 27, 1995 Phoenix, Arizona, USA. IEEE 1995.
26. Mod Backhand: (a load balancing patch to the Apache Web Server).  
Available from: [http://www.backhand.org/mod\\_backhand/](http://www.backhand.org/mod_backhand/)
27. E.H Mamdami. Advances in the linguistic synthesis of fuzzy controllers. *International Journal of Man-Machine-Studies* 7, pages 1-13, 1976.