ISTANBUL UNIVERSITY –
JOURNAL OF ELECTRICAL & ELECTRONICS ENGINEERING

YEAR : 2003 (859-877)
VOLUME : 3
NUMBER : 2

DESIGN OF A SCHEDULER: COMPARISON OF DIFFERENT SCHEDULING ALGORITHMS

A. Halim ZAİM

Istanbul University, Engineering Faculty, Computer Engineering Department
34850, Avcilar,Istanbul,Turkey

E-mail: ahzaim@istanbul.edu.tr

ABSTRACT

In this study, we investigate different scheduling algorithms, and compare their performance for systems with multiple priority queues. The scheduler defined in this paper may be thought as the preprocessor in an ATM switch, a network processor in a router or just an ordinary CPU scheduler where multiple processes with different priorities are present in the system. We show that the proposed algorithm outperforms the known scheduling algorithms from the point of balancing the average response times.

Keywords: *Priority scheduling, dynamic scheduling, priority queues..*

I. Introduction

Scheduling theory first appeared in applied mathematics about forty years ago in order to study mathematical questions arising in production planning and scheduling. An increasing interest in this branch or operations research can be attributed to the high level of automation of all branches of human activity. The quality of modern production essentially depends on the planning decisions taken at different stages of the production process. Moreover, as the quality of these decisions is improving, the time and flexibility requirements for decision-making are becoming more important [1].

Scheduling is the process of devising or designing a procedure for a particular objective, specifying the sequence or time for each item in the procedure. Typical scheduling problems are railway time-tabling, project scheduling, production scheduling, mass transit scheduling, hydropower scheduling, scheduling nurse shifts in a hospital, ets... Emerging application examples of scheduling in computer systems are in flexible manufacturing systems, multiprocessor scheduling, multiple queue scheduling, robot activity scheduling, scheduling in large scale networks and hard real-time scheduling. For more information about different scheduling applications refer to ([2]-[5]).

In this study, we investigate the scheduling problem in a computer network router or a switch filtering a number of queues with

Received Date : *5.7.2001*
Accepted Date: 02.05.2003

different priorities. A dynamic priority adjustment proposed for such queues are compared with the well known first-in first-out (FIFO), round robin (RR), head-of-line (HOL) scheduling and a mixed algorithm. More information about these scheduling algorithms can be found in [6].

The paper is organized in the following manner: in Section II, the problem definition and the proposed scheduling algorithm is explained. Section III shows the results, and Section IV concludes the paper.

II. A Scheduling Model for Priority Queues

Consider the following CPU scheduling problem. Programs (jobs, packets, etc.) submitted for execution are partitioned into N classes. The rate of submission for class i is λ_i programs per second. The execution time of a class i program is a random variable with mean EX_i and variance σ_i^2. We may assume that the sequences of program interarrival and execution times from independent and identically distributed (IID) processes. A queuing model for this system is shown in Figure 1.

The queues have infinite capacity, so no program submission is blocked. For simplicity we may assume that an executing program cannot be preempted.

The objective of the scheduling algorithm is the following. We are interested in keeping the average response time for programs of class i, call it ER_i, below a class-dependent, given threshold g_i. The mathematical representation of the problem can then be thought as devising a CPU scheduling algorithm to solve the following problem:

$$ER_i = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} (D_i - A_i) \qquad (1)$$

A dynamic priority adjustment techniques is proposed in [7] to solve the problem defined above. Let T_n denote a program arrival instant such that the program finds the system empty for the n^{th} time as shown in Figure 2. Let D_n denote a departure instant such that a finished program leaves the system empty for the n^{th} time. B_n is called the n^{th} busy cycle; I_n is the n^{th} idle period.

We wish to set up class priorities only at the beginning of each cycle. The priorities will be kept constant throughout the busy cycle. Of course, they may change from busy cycle to idle cycle. From now on, we consider the special case of two program classes only, for simplicity of presentation.

Consider the following quantity:

$$P_i(T_n) = \frac{1}{\lambda_i}\frac{1}{g_i}\frac{1}{T_n}\int (T_n n_i(t))dt, i = 1,2 \quad (2)$$

Here $n_i(t)$ is the queue size including the program currently being executed, if any, of class i at time t. The integral is the time-average queue size; if the system is ergodic and if n is very large, its value should be equal to the average queue size in steady state. Then, from Little's Law [6], the ratio

$$\frac{1}{\lambda_i}\frac{1}{g_i}\frac{1}{T_n}\int (T_n n_i(t))dt \qquad (3)$$

should converge to ER_i.

It is proposed to use $P_1(T_n)$ and $P_2(T_n)$ as the priority indices since (at least for large time values) they represent estimates of ER_i/g_i. The scheduler can be now described as follows:

At time T_n compute $P_i(T_n)$:
If $P_1(T_n)>P_2(T_n)$, give higher priority to class 1.
If $P_1(T_n)<P_2(T_n)$, give higher priority to class 2.
If $P_1(T_n)=P_2(T_n)$, give higher priority to class 2 (arbitrary).

The rational for such an algorithm is the following: By giving it higher priority, such an algorithm always helps the class with the highest current delay. It is dynamic (it needs to measure queue sizes), and thus we expect it to be "adaptive".

1. Defining (*S,Q,P*):

When defining the random processes, we need to decide on the sample space, the set of all events of interest and the probability assignments shown

with (*S,Q,P*) triplet. There are two different approaches for that. One of these approach is true and the other is false. Let us try to pose these approaches and try to make clearer the right choice. The first approach is defining a general set of (*S,Q,P*) and work on this set throughout the paper. The second approach, on the other hand, is defining different (*S,Q,P*) sets for each different random used in the study.

Now, first, let us look at the second one. According to this approach, we define different (*S,Q,P*) sets for each random variable. That means, for example, for the random variable arrival time, we define *S*=[0,1], and *Q*=[0,], and *P* is the probability mass function (*pmf)* of the Poisson Random Variable. For another random variable such as interarrival times which is an exponential random variable, we change the *P* as the probability density function (*pdf)* of the exponential random variable. The problem with this approach, we could not make it clear the definition of the (*S,Q,P*), because if we want to deal with another random variable that we see during the processing time, the (*S,Q,P*) set for this random variable is unknown. Therefore, this is in fact a wrong approach.

The first approach on the other hand says us to define (*S,Q,P*) from scratch for the whole system in a general way. Let us then try to define *S*. *S*, the sample space, contains all possible outcomes of the experiment. That means, for example, for a given seed value, which is in our case the ζ, what would be the values of the random variables such as the arrival random variable, interarrival random variable, etc. It is in fact obvious that, a random variable can take values between 0 and 1, therefore, all possible outcomes range between 0 and 1. As a conclusion we could say that *S*=[0,1].

Q, the set of all events of interest, to which we can assign probabilities is of course uncountable, and we define *Q* as [0, ∞]. *P*, the probability assignment, is a function from *S* to *R* that specifies how events in *Q* are assigned probabilities. There are three ways to assign probabilities to events: measurement, computation, and hypothesis. In this study, we used the computation method, and then defined the probabilities as relative frequencies. That means *P* is the relative frequency of each computation.

For example, to compute the probability of having *k* number of jobs in the system, we run the simulation for a long time, and then calculate the histograms for the outcomes. These histograms from which we could easily calculate the relative frequencies of each outcome, is then used for the probability assignment. This is in fact a known method to approximate the *pdf* of random variables.

2. Random Variables in the System.

Random variables in this system are:
Arrival times of jobs(*A*),
Interarrival times of jobs(*I*),
Number of jobs in the queues(N_{Q1}, N_{Q2}),
Number of jobs in the system(*N*),
Departure times of jobs(*D*),
System Delay(*T*),
Queueing Delays(T_{Q1}, T_{Q2}),
Execution Times of jobs(*X*),
Response Times(R_1,R_2),
Priority Indices(P_1,P_2),

III.Results

In this study we implement different scheduling algorithms for a system with two priority queues with infinite sizes and one scheduler and one server. The priority indices, arrival rates and departure rates of each queue is changed for different scenarios. In the following subsections, we give a number of different tests performed on this system to demonstrate the comparison of different scheduling algorithms by first proving the correctness of the random number generators, and simulation environment.

1. Plotting queue sizes as a function of time.

In Figure 3. it is seen in a small time interval the change of the queue size. These values are taken from the experiment made with λ=10, and the time parameter is increased in a discrete way by 0.01. The plot shows that the arrival process is a Poisson Process. The figure shows that queue size increase as a step function which is one of the known properties of the Poisson Process.

In Figure 4. it is seen the changes of the Queue Sizes for both queues in an experiment made with λ_1=0.5, λ_2=0.5. The figure shows that the slope of both curves are equal to 0.5, as it should be. In this figure a further look is used to illustrate the slope of the queues which gives us the parameter of the arrival Poisson process. On the other hand, it is very natural that we could not observe the step function property of the Poisson process from that figure. A closer look to this figure is already given in Figure 3.

2. Plotting histograms of interarrival times, and service times.

Figure 5 shows histograms for interarrival times for an experiment with λ_1=10, λ_2=1. The shapes of both histograms are similar but if we look at these histograms more carefully, we could see the differences in their *x*-axis. In the first histogram the number of jobs arriving with an interarrival time smaller then 0.1 is nearly 5500. The same is true for jobs with interarrival rate smaller than 0.2. As a total the number of jobs with an interarrival rate between 0 and 1 is nearly all of the jobs produced which is nearly equal to 10000. On the other hand, at the second histogram the interarrival rates smaller then 1 is 5500.

To demonstrate the correctness of these histograms, a comparison with real exponential random variables is added. Figure 6 shows the first comparison where the first part is the histogram of interarrival time for queue 1 while the second one is the histogram of values generated randomly with exponential random function with the same parameter λ which is 10 for this queue. Figure 7 is the same experiment repeated with parameters changed according to the λ's of the second queue which is 1.

Figure 6 gives us the histograms for the execution times of jobs in queues 1 and 2. The parameters of the execution times (1/μ's) of queues 1 and 2 are 30, and 5 respectively.

3. Defining system delays (R_1, and R_2), in steady state.

R_1 and R_2 are steady states, meaning the average response time of jobs in each queue as *n* goes to infinity. We can show this as the following formula:

$$ER_i = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} (D_i - A_i) \tag{4}$$

In this formula, we calculate the departure times of each job, and then subtract from the departure time of that job from its arrival time which gives us that job's response time. We add the response time of each job coming to the queues, and calculate their time average. If we increase the number of jobs we are processing, that means if *n* goes to infinity, this will give us the steady state value response time, or in other words the average response time.

4. Expressing $P_i(T_{n+1})$ recursively, in terms of $P_i(T_n)$.

The priority index at time T_n is calculated as the time average of response time until time T_n. We describe this process with the following formula.

$$P_i(T_n) = \frac{1}{N} \sum_{i=1}^{N} (D_i(T_i) - A_i(T_i)) \tag{5}$$

From that formula we can see that sum of the responses is equal to $N*P_i(T_n)$.

$$P_j(T_{n+1}) = \frac{1}{N+1}\left[\sum_{i=1}^{N+1}(D_i(T_i) - A_i(T_i))\right] =$$

$$\frac{1}{N+1}\left[\sum_{i=1}^{N}(D_{ji}(T_i) - A_{ji}(T_i)) + D_j(T_{N+1}) - A_j(T_{N+1})\right] \tag{6}$$

The same is true for the priority index at time T_{n+1}

$$P_j(T_{n+1}) = \frac{1}{N+1}\left[\sum_{i=1}^{N}(D_{ji}(T_i) - A_{ji}(T_i))\right] + \frac{1}{N+1}\left[D_j(T_{N+1}) - A_j(T_{N+1})\right] \tag{7}$$

We know from equation (4) that the summation part of the equation (7) is equal to N*Pi(Tn). If we replace it to equation (7):

From the formula we have obtained, we can see that the priority indices are recursive values, that means they depend both to the previous values and also the response of the job being processed at that time.

Now, let us consider the (P1(Tn), P2(Tn)) plane. The two priority indices determine a point in this plane (for any given n). As n varies, this point moves around the plane. Observe that priority indices are random variables since they depend on the queue size. Assume for the time being that the priority indices converge to some constant values as n approaches infinity.

Figure 9 shows the change of priority indices for queues 1 and 2. In Figure 9, it is obvious that the steady state values for priority indices are very near to 2.00. In fact if we take the average execution times equal, we can calculate the analytical solutions. The analytical solutions for such a system is the same with the M/M/1 queue (see [8]) with $\lambda=\lambda_1+\lambda_2$. The analytical solution to an M/M/1 system with $\lambda=1.5$, and $1/\mu=2$ is given as:

$$T=\frac{1}{\mu(1-\rho)}=\frac{1}{\mu\left(1-\frac{\lambda_1+\lambda_2}{\mu}\right)}=\frac{1}{2\left(1-\frac{1.5}{2}\right)}=\frac{1}{2\left(\frac{1}{4}\right)}=2 \tag{9}$$

Therefore, it is expected to find an average response of 2.00 for FIFO. For the scheduling algorithm proposed, of course the same result is expected. The difference between two is that, FIFO can converge to its steady state value much more easier then the other scheduling method.

Figure 10 gives us the P_1P_2 plane. In Figure 10, we see the result of expected delays for the plane P_1P_2. In this study, we use priority indices interchangeably with the system delays or response times because we get constants $g_i=1$. Figure 10 shows that when we draw a line starting from origin and passing from the steady state point (which is near to 2.00 for this example) we can see oscillation of the priority indices across that line.

That means, at the beginning, priority index 1 is higher than priority index 2. Then after the arrangement of the priority indices in the idle period, the priority index 1 decreases while the priority index 2 increases. Therefore, there is an oscillation around the line defined beforehand. At the end, that means after a reasonable number of arrivals, both indices converges to their steady state values. On the other hand, it is worth mentioning that, the amount of trial necessary for obtaining the steady state depends on both the parameters of the system, and the algorithm used for scheduling. Figure 11 gives us another example for oscillation of P_1P_2 with changing n value.

The Figures 9 and 12 shows the convergence of the priority indices towards their steady state values clearly. For example for the system used in Figure 12, the simulation result for steady state is near to 0.16 for P_1, and 0.28 for P_2. This result may be seen also in that figure while observing the P_1 and P_2 indices individually.

We used the ergodicity notion in obtaining average response times. Ergodicity enables us to calculate statistical parameters of the random process, such as means and correlation, through time averages of a single sample path. In this study, in order to simplify the calculations, we used only one simple path, with the seed = 1. In fact in order to obtain the real means, we should repeat the experiment with different seed values, so that we could obtain different sample paths and calculate the average. However, by the use of ergodicity, we performed the experiment with only one seed, that means for only one sample path and obtained the average by taking the time average.

For example, *P[n customers in the system at time t]* is described with $P_n(t)$, and average number of customers in the system at time t is $EN(t)$. To calculate this value we should repeat the experiment with different seed values, and store the $N(t)$ values for the time we are interested and then take their average. That would be expressed as follows:

$$EN(t)=\sum_{n=0}^{\infty} nP_n(t) \tag{10}$$

We can find the equilibrium distribution as $t\to\infty$, $P_n(t)\to P_n, n=0,1,2,\ldots$

$$EN(t) = \sum_{n=0}^{\infty} nP_n(t) = EN$$

and as $t \to \infty$ (11)

Let N_t be the time average of number of customers in *(0,t)*,i.e. with *N(t)* being the number of customers in the system at time *t*, then

$$N_t = \frac{1}{t} \int_{u=0}^{t} N(u)du \quad (12)$$

That means the system we are interested is ERGODIC which means that $N=N_t=EN(t)$ as $t \to \infty$.

In our system to calculate the time average we should repeat the process with different seed values and take the average of the number in the system at that time. To make this for every random variable used in the system and for every time interval is very time consuming therefore, we used the ergodicity notion and calculated the random variables by working on the same sample path with fixed ζ. That means we could use time averaging for obtaining ER_1 and ER_2.

5. Comparing Different Scheduling Algorithms

In this study, we simulated a few scheduling algorithms such as First In First Out (FIFO), Round Robin (RR), The dynamic priority adjustment algorithm proposed in this paper, Head Of Line (HOL), and mixed. We don't explain FIFO, RR and HOL algorithms. To learn more about these algorithms refer to [8]. The mixed algorithm works similar to FIFO. The only change is on the choice of choosing among the queues. That is the queues are FIFO in themselves however, we decide from which queue to choose by a random number. We pick a random number and according to a predefined acceptance parameter we choose queue 1 if that number we generated is below that acceptance parameter. Otherwise we choose the second queue. So we give priority to one of the queues with certain probability for all the time.

Using these scheduling algorithms, we calculate P_1 and P_2, the steady state values of the priority indices. We then plot these values in the P_1P_2 plane for all the scheduling algorithms we have simulated. That way, we could compare different scheduling algorithms.

In order to simulate these algorithms, we define a queue structure formed from three different participants: interarrival time, execution time, and priority. Interarrival times are produced according to an exponential distribution with parameter λ,and the execution times are produced again according to an exponential distribution but this time its parameter is μ. The idea of using exponential distribution for interarrival times comes from the fact that, if the arrival process is Poisson, then the interarrival times of these processes are exponential. Therefore, if we produce exponential interarrival times, the coming process becomes Poisson. In fact we can see this Poisson effect from Figure 3 and 4 which show the arriving times.

In order to compare the behaviors of these algorithms, we plot the steady state responses in P_1P_2 plane. We test the algorithms with four different (λ,μ), combinations. The resulting graphs are shown in Figures 13,14,15 and 16 for ($\lambda_1=\lambda_2=0.5$, $\mu_1=\mu_2=2$); ($\lambda_1=0.5$, $\lambda_2=1$, $\mu_1=\mu_2=2$); ($\lambda_1=\lambda_2=0.5$, $\mu_1=2$, $\mu_2=1$); and ($\lambda_1=10$, $\lambda_2=1$, $\mu_1=30$, $\mu_2=5$) respectively. In these graphs, there are in fact 6 point instead of five, because we plot both HOL(1 2) and HOL(2 1).

As seen in these figures, the results lie in a straight line between the points obtained in trials performed with HOL(1 2) and HOL(2 1). In fact this result is very normal because, in HOL(1 2), the first queue has always priorities to the second queue, therefore, the jobs in the second queue wait longer than the jobs in queue 1. As a consequence, in this case, the average response time of the first queue is smaller than the average response time of the second queue. The inverse is also true for HOL(2 1). At the same time there is not another scheduling algorithm which gives a total priority to one of the queue. The others balance the two queue in a way. Therefore, it is again very natural that, average response time of queue in other algorithms is between these two bounds. For the proposed algorithm, if the arrival rates and execution times are not very different, this algorithm nearly produce the same average responses for both queues, which is also natural. For the mixed scheduling algorithm, the important parameter is the acceptance probability which is 0.3 in our examples. If we change this

parameter, we could easily observe that we move on that straight line between its two bounds. For the RR algorithm, again the important parameter is the processors processing time. If we take this time larger then the average execution time of jobs in one or both of the queue, then we could see a nonpreemptive system. In that case, the system become a nonpreemptive RR, and of course the results approaches to the one obtained with FIFO. Therefore, for that system, it is important to take the processors' execution time smaller than the average execution time of jobs in both queue.

Another important factor in these simulations is the necessary condition of stability, which is the fraction of λ and μ. As it is known ρ which is equal to λ/μ should be smaller than 1. Therefore, λ should be smaller than μ. Otherwise the system become unstable, and we could not reach to a steady state value.

Conclusion

In this paper, we investigate different scheduling algorithms in a comparative way. We define an environment to generate random processes. Using a simulator, we test first the correctness of the environment and then compare the scheduling algorithms from the point of priority handling among two queues with different priorities. We also show how to analyze these priority schemes analytically. As a future work, it is possible to adapt this study to more sophisticated scheduling algorithms.

References

1. Sotskov, Y.N. And Tanaev, V.S., *Scheduling theory and practice: Minsk Group results*, Intelligent Systems Engineering, Spring, 1994.

2. Jing Z., Li L., Sun H., Chen Y., *Performance of priority scheduling to support differentiated services in ATM switches*, Communication Technology Proceedings, 2000. WCC - ICCT 2000. International Conference on , Volume: 1 , 2000, Page(s): 463 -470 vol.1

3. Baruah, S.K.; Deji Chen; Mok, A., *Static-priority scheduling of multiframe tasks Real-Time Systems*, 1999. Proceedings of the 11th Euromicro Conference on , 1999, Page(s): 38 -45

4. Kim H., Lee S. and Lee J., *Alternative priority scheduling in dynamic priority systems*, Engineering of Complex Computer Systems, 1996. Proceedings., Second IEEE International Conference on , 1996, Page(s): 239 -246

5. Katcher, D.I.; Sathaye, S.S.; Strosnider, J.K., *Fixed priority scheduling with limited priority levels*, Computers, IEEE Transactions on , Volume: 44 Issue: 9 , Sept. 1995, Page(s): 1140 - 1144

6. Kleinrock, L, "Queueing Systems: Theory", *Wiley-interscience publications*, New York, 1975.

7. Viniotis, Y., "Probability and Random Processes for Electrical Engineers", *McGraw Hill Pub.*, Boston, 1998.

8. Kleinrock, L, "Queueing Systems: Computer Applications", *Wiley-interscience publications*, New York, 1976.

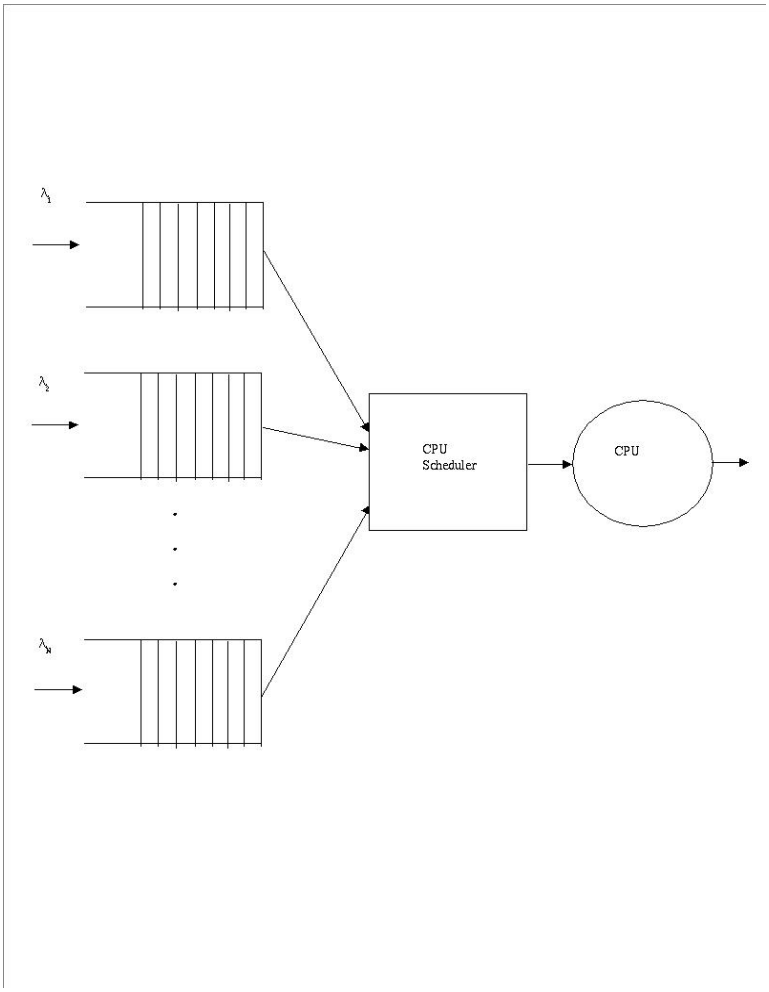

Figure 1: The Queueing Model

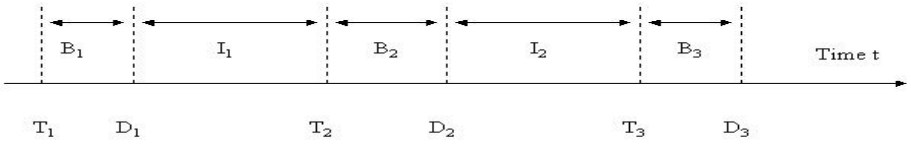

Figure 2: Busy and idle periods

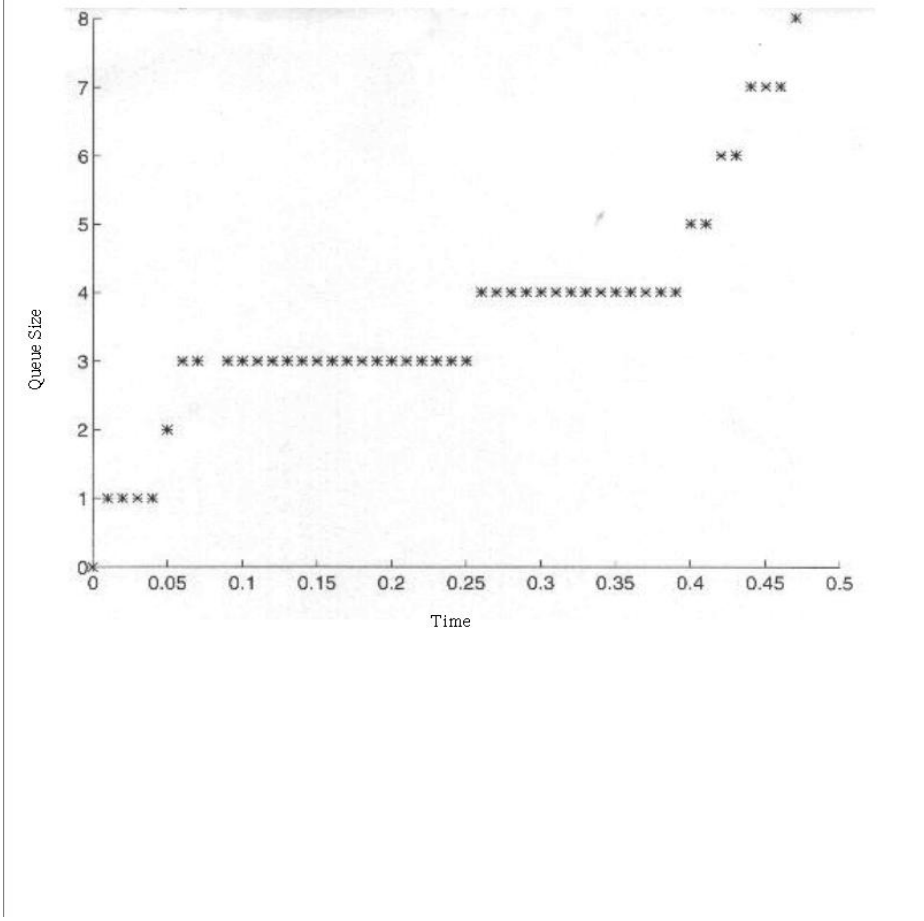

Figure 3: Change in Queue Size with Time

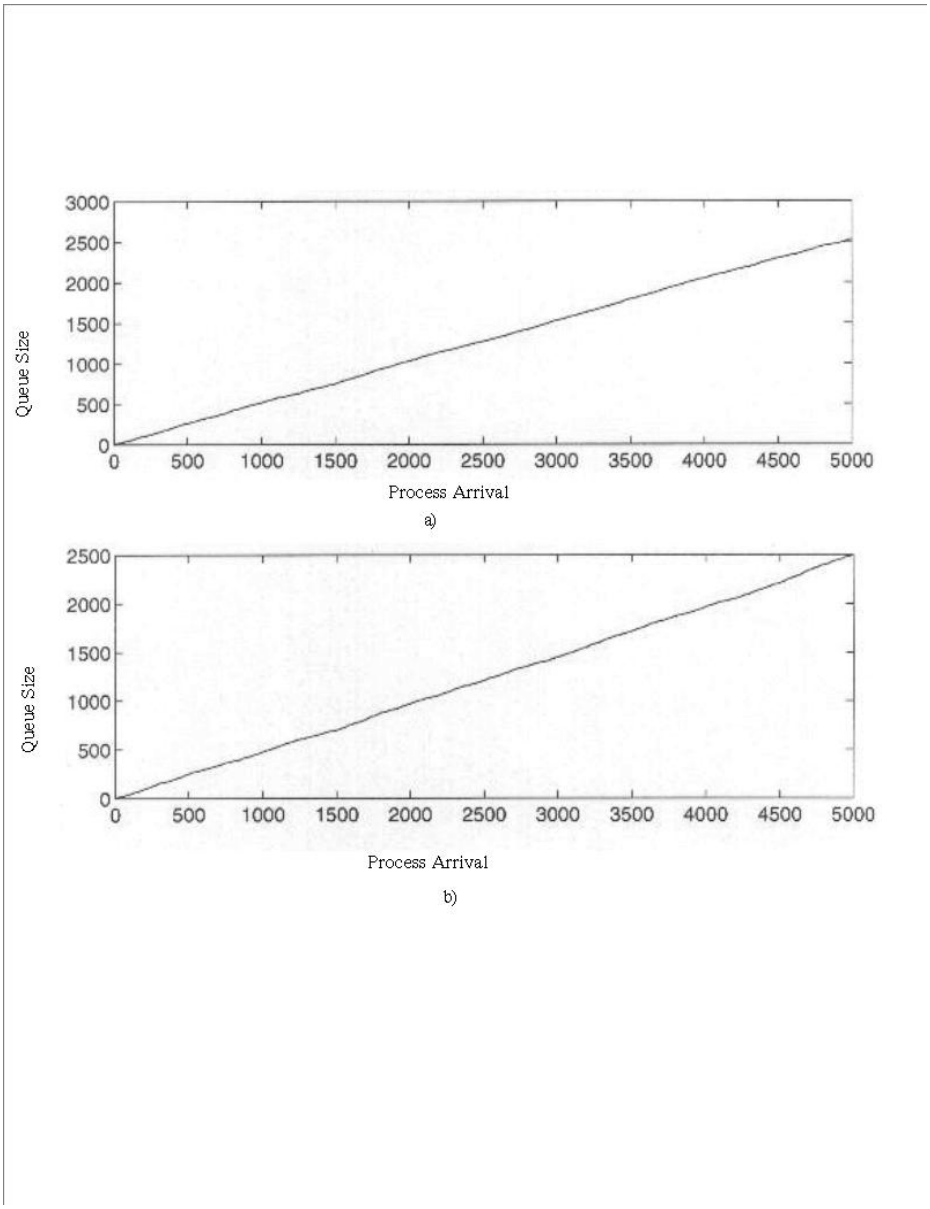

Figure 4: Change in Queue Size with Arrival: a) Queue 1, b) Queue 2.

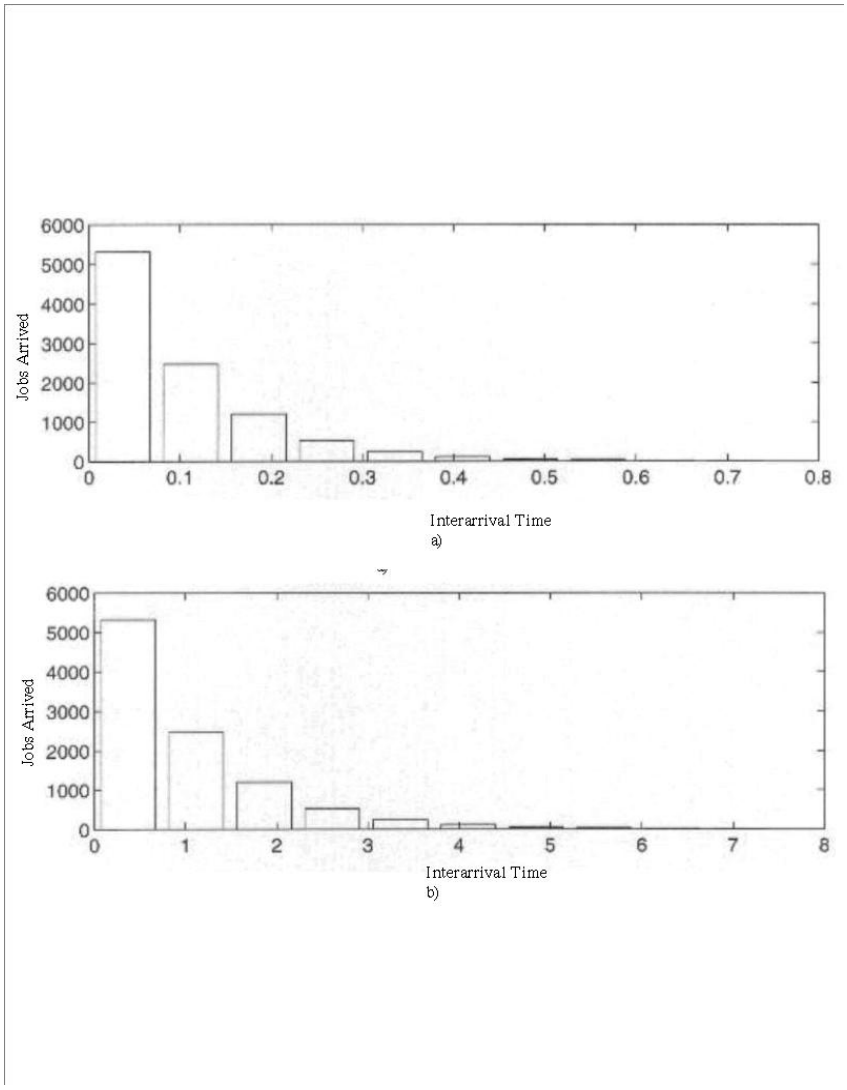

Figure 5: Change in Number of Jobs with Interarrival Time: a)Queue 1, λ=10, b)Queue 2, λ=1.

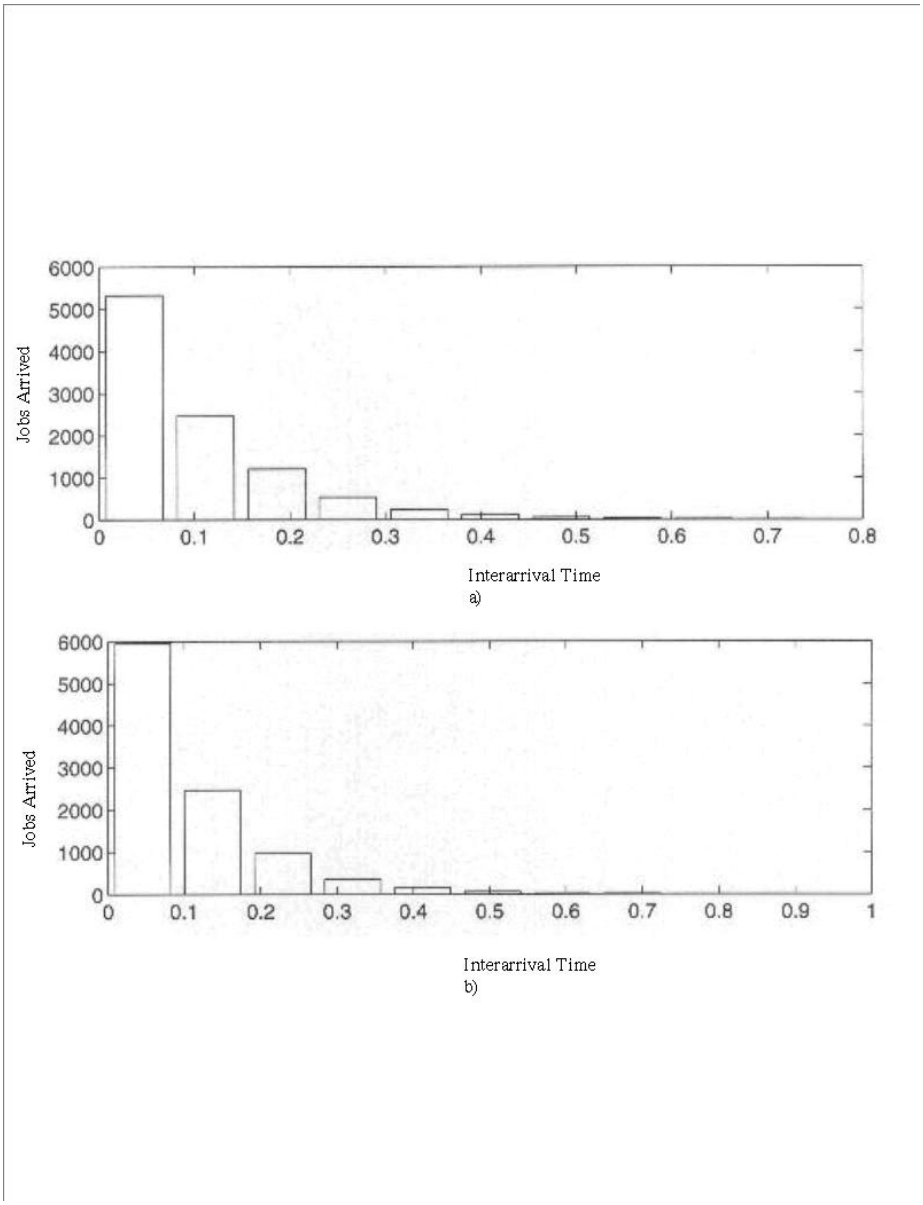

Figure 6: Change in Number of Jobs with Interarrival Time: a)Queue 1, λ=10, b)Exponential with λ=10.

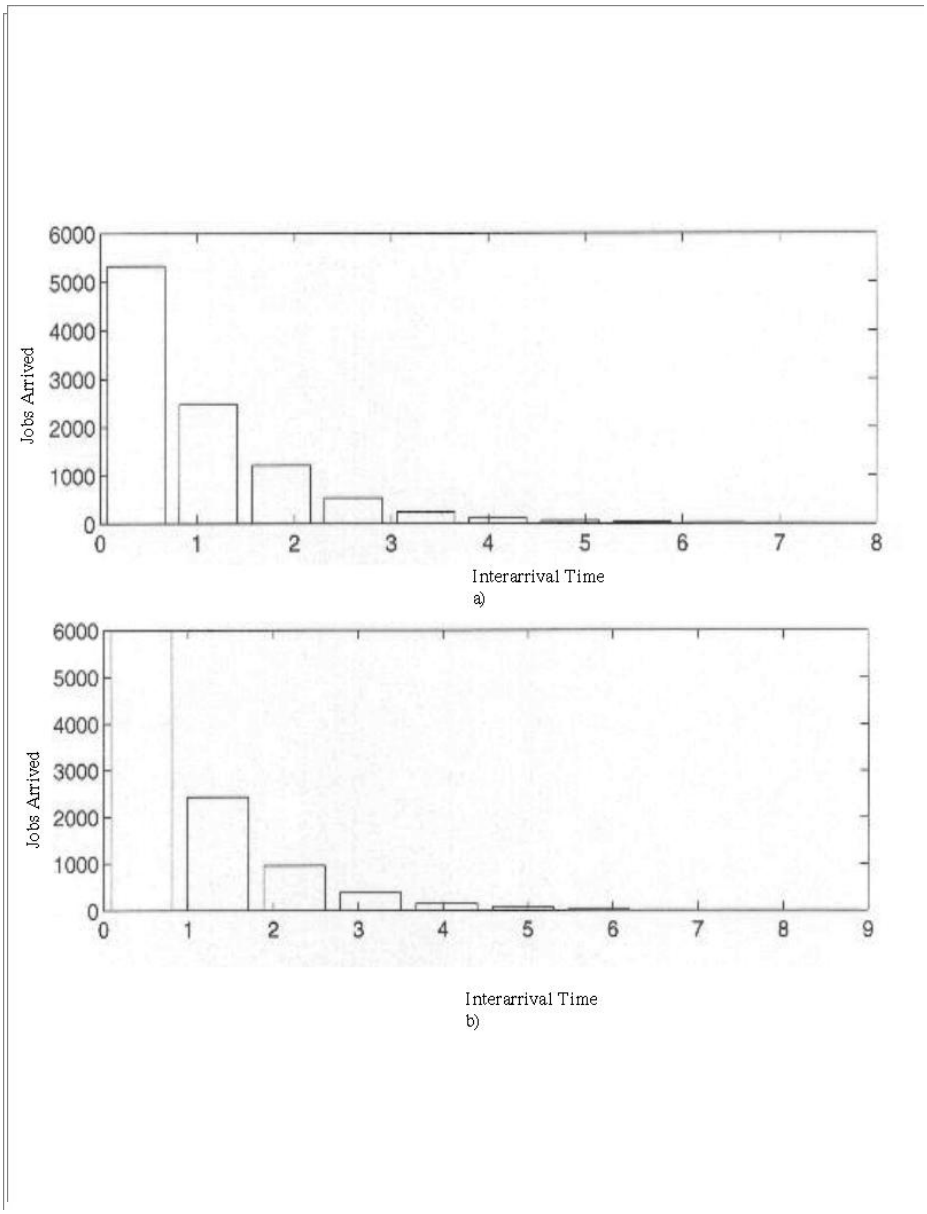

Figure 7: Change in Number of Jobs with Interarrival Time: a)Queue 1, λ=1, b)Exponential with λ=1.

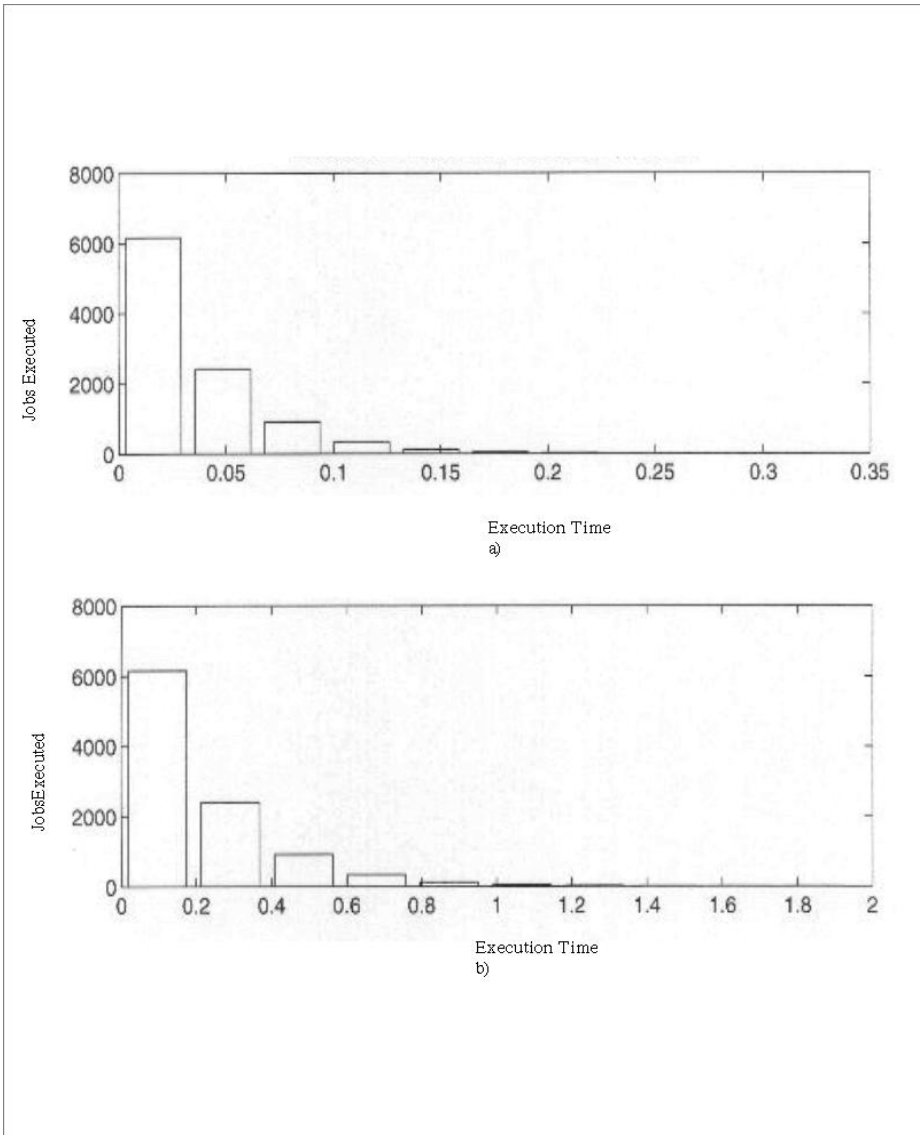

Figure 8: Change in Number of Jobs with Execution Time: a)Queue 1, μ=30, b)Queue 2 with μ=5.

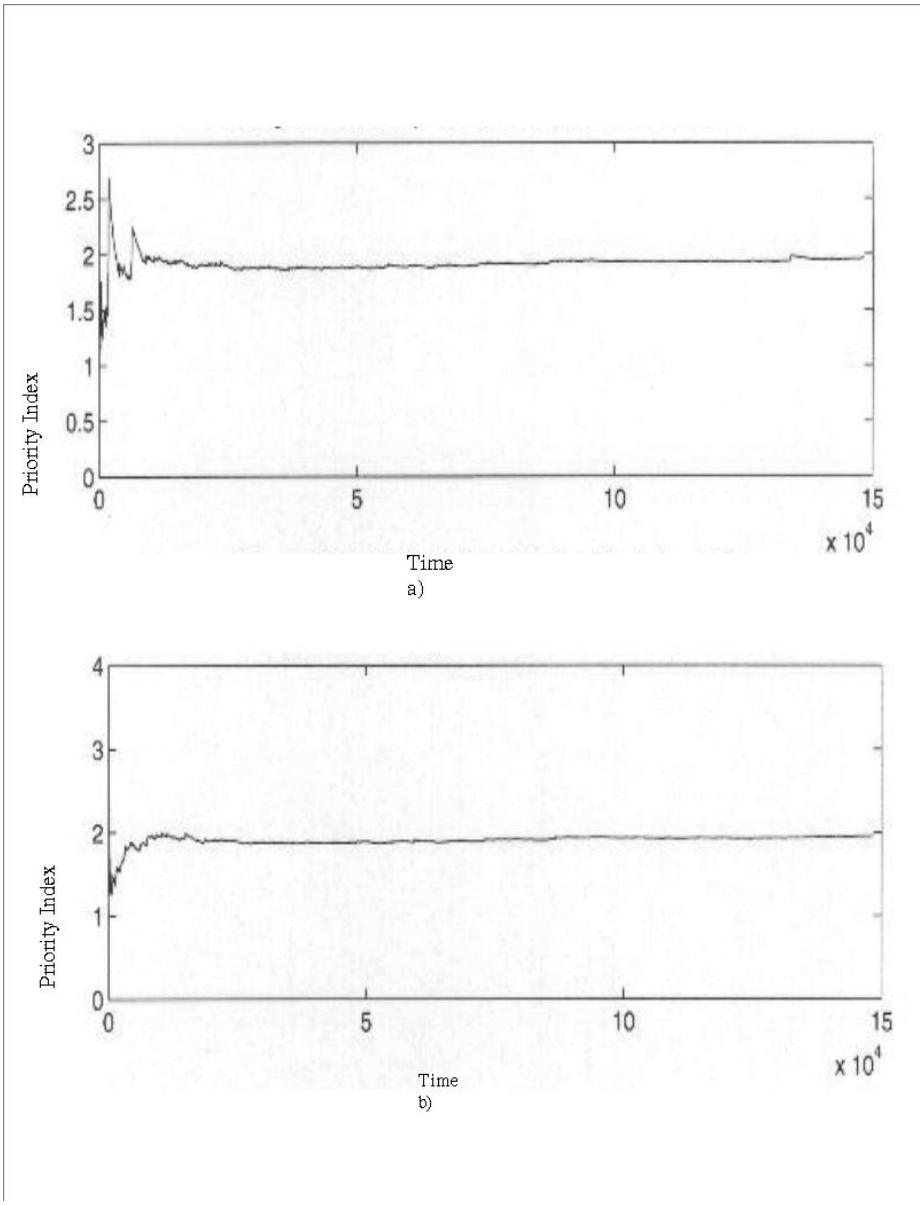

Figure 9: Change in Priority Indices: a)Queue 1, b)Queue 2.

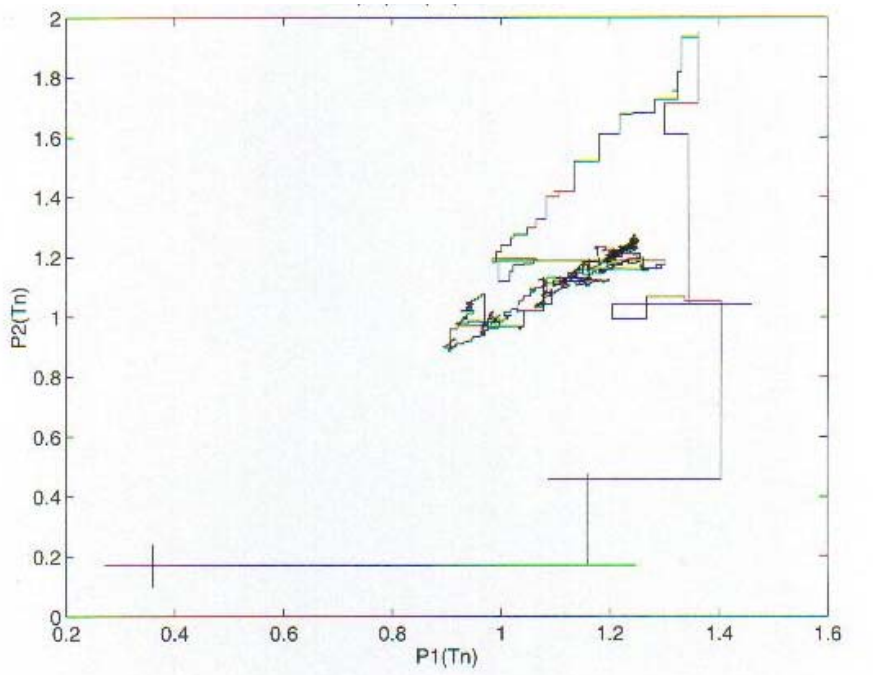

Figure 10: Change in P_1 and P_2 as n Varies(λ_1= 0.5, EX_1.= 0.5, λ_2= 1, EX_2.= 0.5).

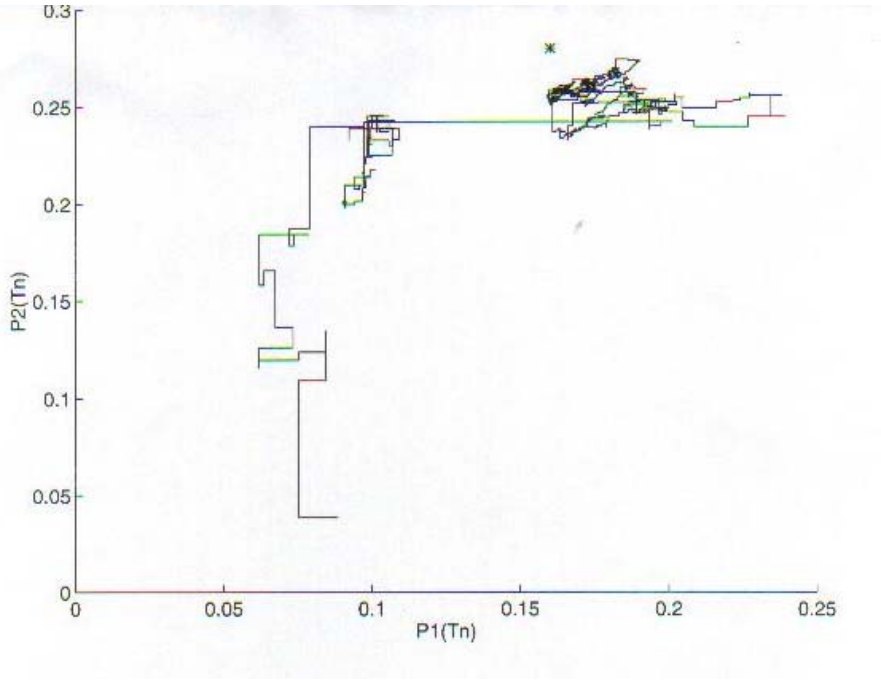

Figure 11: Change in P_1 and P_2 as n Varies(λ_1= 0.1, EX_1.= 0.03, λ_2= 1, EX_2.= 0.2)

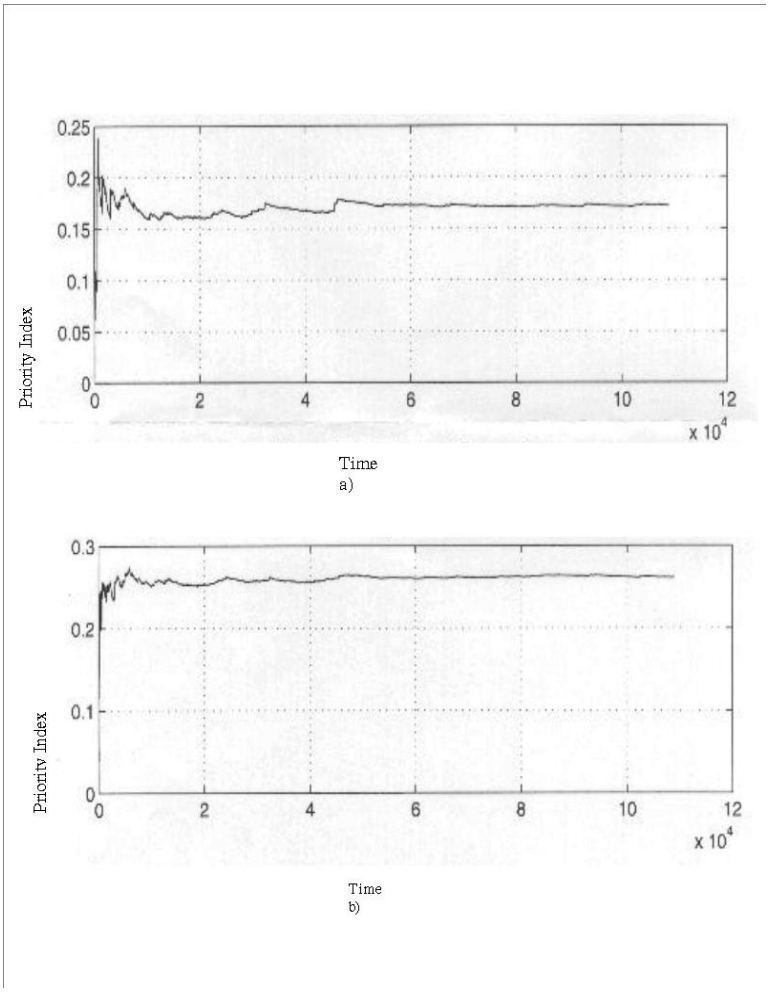

Figure 12: Time vs Priority Indices: a)λ=10,μ=30, b) λ=1,μ=5

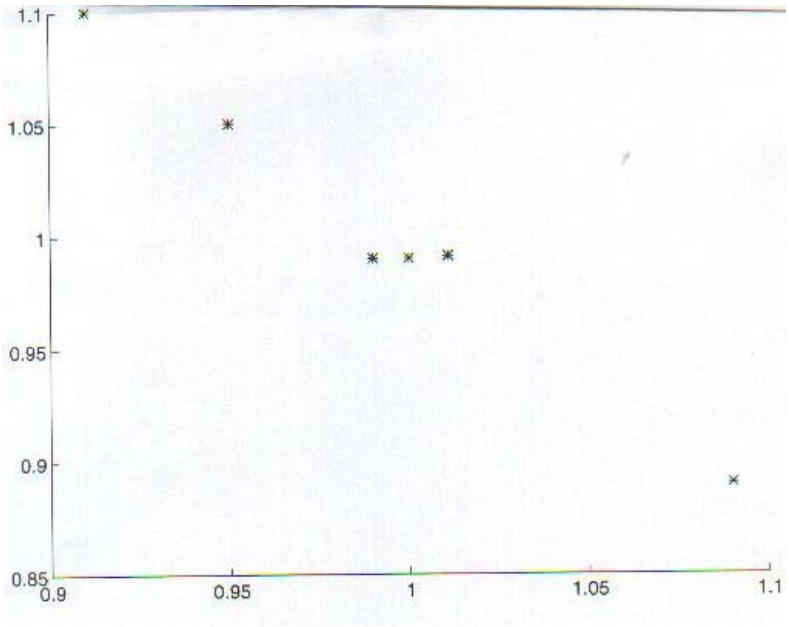

Figure 13: P_1 vs P_2 for ($\lambda_1=\lambda_2=0.5$, $\mu_1=\mu_2=2$)

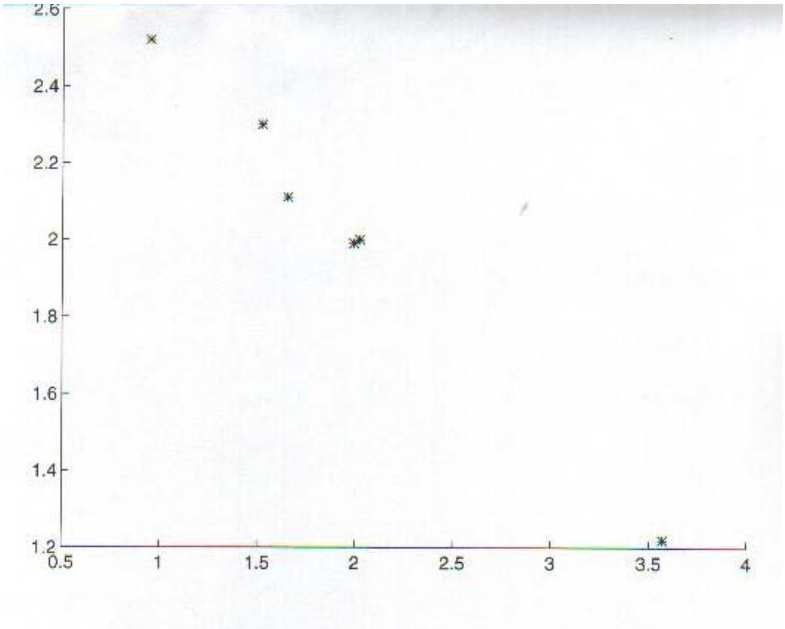

Figure 14: P_1 vs P_2 for ($\lambda_1=0.5$, $\lambda_2=1$, $\mu_1=\mu_2=2$)

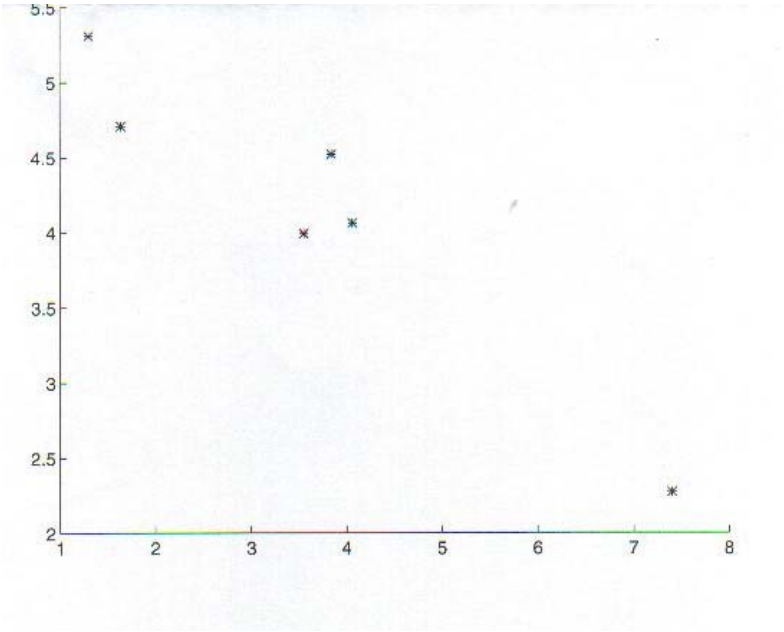

Figure 15: P_1 vs P_2 for ($\lambda_1=\lambda_2=0.5$, $\mu_1=2,\mu_2=1$)

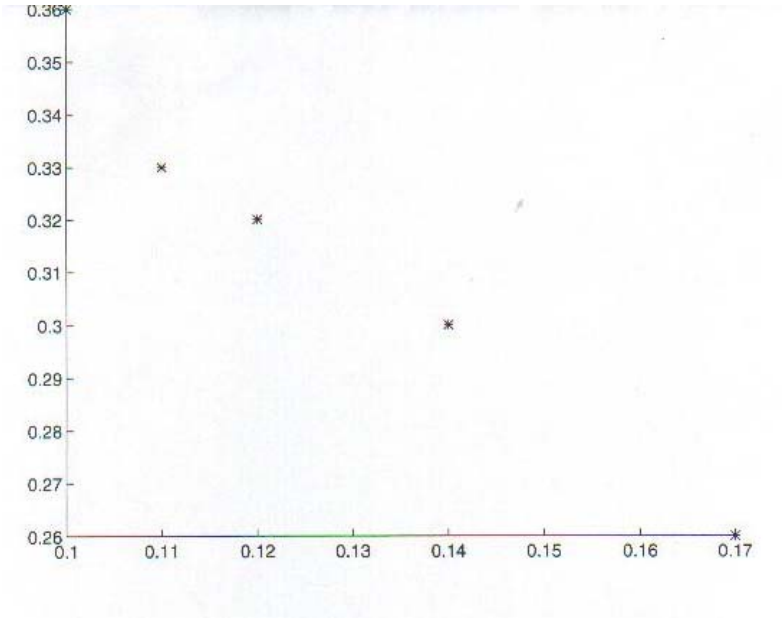

Figure 16: P_1 vs P_2 for ($\lambda_1=10,\lambda_2=1$, $\mu_1=30,\mu_2=5$)