CSci 426 FINAL EXAM

VJ Davey

1. PROBLEM 1

   (a) **I.** The estimated steady-state utilization of each service node is as follows (for 100000 jobs):
   $Node1$ 0.64
   $Node2$ 0.72
   $Node3$ 0.80
   $Node4$ 0.73

   (b) **II.** The estimated time-averaged number of jobs in the queue at each service node is as follows (for 100000 jobs):
   $Node1$ 1.35
   $Node2$ 1.63
   $Node3$ 2.65
   $Node4$ 1.63

   (c) **III.** Based upon 128 data points, and with 95% confidence, the expected time spent in the network per job is 1.81 +/- 0.01.

   (d) **IV.** I convinced myself my results were correct by diagramming the system on a separate sheet of paper and calculating the expected interarrival rates to each by hand by making use of the fact that flow in must equal flow out and multiplying respective probabilities to get the interarrival times for each node as follows, knowing that jobs arrive with rate 4, and that for node 4, the entire system before it can be treated as one big node where feedback to 4 occurs with $Pr = .75$ and a job leaves the system otherwise. All right hand side numbers are inversions for the interarrival times:

$$NODE2 : 4 \times .6 = 2.4 \rightarrow .417$$

$$NODE3 : 4 \times .4 = 1.6 \rightarrow .625$$

$$NODE4 : 4 \times .(\frac{.7+.8}{2}) = 3 \rightarrow .333$$

   I verified my calculated interarrival times against what my simulation gave me and found that they were equal. I further convinced myself of the correctness of my simulation by varying both the number of jobs my system simulation would run through, and the number of repetitions of the simulation I would consider for use with the estimate.c program. I found that adding more jobs (with less simulations) and running more simulations (with less jobs) led my simulation to converge to the same numbers on numerous runs, and the interarrival times would match what I found by hand. This is what convinced me my results were correct.

2. PROBLEM 2

   (a) **I.** For 128 runs of the simulation, with 95% confidence, and *for 1000 jobs* the steady-state values of $\bar{l}, \bar{q}$, and $\bar{x}$ of a priority queue are as follows:

$$\bar{l} = 30.07 + / - 1.37$$

$$\bar{q} = 29.07 +/- 1.37$$

$$\bar{x} = 0.99 +/- 0.00$$

Note the size of the intervals!

(b) **II.** For 128 runs of the simulation, With 95% confidence and for *1000 jobs* the steady state values of l, q, and x of a FIFO queue are as follows:

$$\bar{l} = 2.75 +/- 0.10$$

$$\bar{q} = 1.96 +/- 0.10$$

$$\bar{x} = 0.80 +/- 0.01$$

Note the intervals here too!

(c) **III.**I convinced myself my results were correct because I followed the examples defined in chapter 5 of the book concerning alternative queue disciplines. I made use of a linked list with specifically designed enqueue methods to simulate a priority queue where jobs of highest priority (those with the shortest service times) go to the front of the queue in all cases for the priority queue. For the FIFO queue, jobs would always go to the end of the line. Because of this, I was not alarmed when I found that the server utilization was nearly at 1 for my priority queue, while being only at .8 for my FIFO queue. Additionally, for my priority queue, I found that on runs which covered more jobs, that I would have higher values for $\bar{l}$ and $\bar{q}$. I was also convinced my results were okay because upon running my simulation **many** times, I found that the more times my priority queue simulation was ran for any arbitrary number of jobs, the closer the values of $\bar{l}$ and $\bar{q}$ converged to a single number, and the +/- interval became much smaller.

(d) **IV.** Because a priority queue is ran in this problem, this means that jobs with shorter service times are **always** completed before jobs with longer service times. Because the interarrival rate and service rate are relatively close to each other, this means that there is a likelihood that there will be some job generated with a relatively high service time close to the arrival of a job with an low to average service time. That job with a high service time will have a lower priority than most of the other jobs that arrive to the system. Since that job has low priority, it will remain stuck in the queue for a long period of time. As the total number of jobs the system must complete increases, the more likely it is that these type of "permanently stuck in the queue" jobs are encountered and stuck in the queue also. This use of a priority queue where the priority is based on completing shorter jobs is why there is such a noticeable difference from the FIFO queue.

3. PROBLEM 3

For this problem, I found that the answer as to which setup most improves performance depends largely on what the arrival rate of the jobs, λ, is and what the service rate for the jobs, μ, is. With hand drawn calculations I considered three systems which look like the following:

From my calculations, which are provided with this test on scratch paper, I found that if μ was some constant rate, then we need only consider variations in the arrival rate. For the original server, it can handle arrival rates which are lower than μ with utilization $\bar{x}$ of the server approaching 1 as the arrival rate grows close to the service rate; rates above μ may cause the server to become unstable in this situation.

In the event a second server is added, the system will be able to handle jobs which arrive at rates above μ, and can in fact handle arrival rates up until the point they approach 2μ, at which point the server approaches overflow and instability. However, another metric worth considering is the speed the system can perform the tasks, which directly corresponds to the wait each job may experience in the system. Since the only difference here was that a second server was added, there is no impact on how fast the jobs are performed in the event that the arrival rate is less than the service rate. The second server would only need to activate in situations where the first server is busy, and if the average utilization of the first server is below 1, then the second server might as well not even be there, as the first server can handle all the work, much like the original system.

Finally, in the event the single server were replaced with another server that was 1.7 times as fast, the new faster server could also handle jobs that arrive at rates above μ up until the point the arrival rate approaches 1.7μ; an arrival rate higher than this may lead to instability. A noticeable improvement here is that with this new server, service occurs much more rapidly. In the event arrival rates are below 1.7μ, average waiting times for this node are much shorter than they are for both the original system or the system involving two servers due to the increased speed of the server.

In conclusion, the call on which system change may improve performance the most depends largely on what metric is important to the user regarding performance. In the event the user knows that they have a fixed or limited arrival rate which is well lower than the service rate of the 1.7μ server, the 1.7μ server would most likely be the best choice for improving performance as it could complete services much faster than the 2 server set up so long as its utilization isn't hitting 1. Likewise, in the event the user wants a system where the emphasis is not on speed, but on the workload their system can handle, then the two server setup may be the best choice. This system can handle slightly larger arrival rates than the 1.7μ server, and wont be as close to its instability condition as the arrival rate approaches 1.7μ.

4. PROBLEM 4

   (a) **I.** I am having difficulty constructing a simulation for a nonstationary poisson process. I am following the book's instructions, and I continue to have difficulty getting the simulation to properly produce a nonstationary arrival. My knot counter j will not update accordingly so I hope my following answers based on knowledge from class notes and reading the textbook and the attached code will allow me some form of partial credit. I understand that this single problem is worth a significant amount of points on an assignment worth 40% of the final grade, so I pray that you have mercy on my soul.

   (b) **II.** Had my simulation properly worked, I would assume that the interval estimates of the number in the node would relate to the nonstationary nature of the arrival process because the number in the node would be dependent on both the arrival rate and the service rate. We already know that the service time distribution is fixed as an $Erlang(4, 0.25)$. An arrival rate which varies as the program runs according to the spline given would lead to a variation of average in the number in

the node as the program runs for each of the different time knots.

For slopes of 0, I would imagine that for many runs of the program that the estimate interval widths would decrease as many runs occurred, and for slopes other than zero, I would imagine that the interval widths might be large as many runs occurred. The reason for this difference would be because for two pairs of knots with the same arrival rate, there is time passing where the data can converge to a single average number in the node during the simulation, much like a stationary poisson process. Such data should have small estimate intervals since they would be around the same values.

For two pairs of knots with differing arrival rates (slopes other than 0), the data cannot converge on one single average number in the node because the arrival rate is changing, which means the utilization of the server, and therefore the number in the node must be changing as well. This will lead to larger interval widths for the average number in the node statistic for many runs of the simulation when compared to knot pairs with slope 0.

(c) **III.** If I were to approximate the nonstationary Poisson arrival process with an equivalent stationary arrival process with a constant rate $\bar{\lambda}$, I would initially consider the total area under the spline which totals to 1615 (found geometrically by finding the areas of the rectangles and trapezoids under the spline). From there I would try to find a constant arrival rate (horizontal line) that would give us the same area under its curve as the spline when running to terminal time $\tau = 2000$. I found this by dividing 1615 by 2000 to get a rate $\bar{\lambda} = 0.8075$.