EE465:
Probabilistic Methods in Computer Systems Modeling
Fall 2011
Simulation Project
Due: Tuesday, 12/14/2011

Introduction

In this project you will simulate, using C/C++, several different queuing systems. After completing the simulation, you are required to submit a project report containing the answers to the questions posed in the problem statements, along with your source code to be compiled and run by the TAs.

Group A

Group A students have to do Parts 1 and 2 (both simulation and theoretic analysis). Part 3 (which consists of only theoretic analysis) will count as bonus credit for those that want to do it.

Group B

Group B students have to do the theoretic questions of Parts 1 and 2 and, also, do Part 3 (which consists of only theoretic analysis). The simulations for Parts 1 and 2 will count as bonus credit for those that want to try the simulation.

You are required to submit a zip archive containing the following files:

- `Report.pdf` which should include (i) a brief description of what you have done in each part, (ii) answers to the questions in each section, and (iii) the plots asked in the problem statements. No source code should be included in this document. **(This is for both groups)**
- Source files (C or C++). (only for group A and those from group B that want to do the simulations)
- a `Makefile` which should create three executable files `part1`, `part2`, and `part3` after execution of the `make` command in the Unix prompt. See instructions in the Notes section in order to create a simple Makefile. (Only for group A and those from group B that want to do the simulations)
- `README.txt` which includes your instructions for running the program. (only for group A and those from group B that want to do the simulations)

See the Notes section for more information regarding the submission.

Part 1

In this part you will simulate a queuing system for the following 3 cases:

Case I $M/M/1/\infty$ with $\lambda = 0.7, \mu = 1$, FIFO, 5000 arrivals.

Case II $M/M/1/\infty$ with $\lambda = 0.95, \mu = 1$, FIFO, 5000 arrivals.

Case III $M/M/3/6$ with $\lambda = 0.75, \mu = 1$, FIFO, 10000 arrivals.

Case IV $M/M/1/\infty$ with $\lambda = 0.8, \mu = 1$, FIFO, and probability of a served customer to re-enter the system at end of the queue p=0.05. (run for enough arrivals to see convergence with the theoretic results)

Case V $M/M/1/\infty$ with $\lambda = 0.3, \mu = 1$, FIFO, and probability of a served customer to re-enter the system at the end of the queue p=0.4. (run for enough arrivals to see convergence with the theoretic results)

Required Input

Your program should generate a user prompt so that the user can enter the case number.

Required Output

The program should show the following quantities on the screen:

- Utilization
- Average number of packets in the system
- Average delay in the system
- The blocking probability
- Idle period of server
- The distribution of the number of packets in the system

Sample output

Your code should generate an executable file with name `part1`. We should be able to run your program as following and get the output as shown: (The numbers in the output are not the actual numbers that you are supposed to obtain through simulation.)

```
 $./part1 1
Utilization: 0.5
Average number of packets in the system: 6.3
Average delay in the system: 2.3
Blocking ratio: 0.1
Idle period of server: 0.3
Stationary distribution: 0->0.2, 1->0.2, 2->0.4, 3->0.1 ...
```

Questions

(a) Find the theoretical values of each of the output results above.

(b) What are the simulated results? (Just give the output of one run.) Make a table that compares the theoretical and simulated values for each result.

(c) Do the simulation results match the theoretical results for all cases? If not, what might be the cause of the deviation in the case/s that the matching is bad?

(d) Modify your simulator to find three more quantities: (i) average service time, (ii) the average of the second moment of the service time, and (iii) the average waiting time in the queue. Collect the data for Case I. Verify the PK formula using the simulation data. These results should be provided in the report, however they are not part of the required simulator output.

Part 2

In this part you will simulate an $M/G/1$ FIFO queue with the following parameters:

- $\lambda = 0.2$
- The service time S has the following probability density function:

$$f(S) = \frac{\alpha m^{\alpha}}{1 - \left(\frac{m}{M}\right)^{\alpha}} S^{-\alpha-1}, \quad m \leq S \leq M \tag{1}$$

where $m = 1, M = 10^4, \alpha = 1.4$. This distribution is called a Pareto Bounded distribution, $B(m, M, \alpha)$(Note that this is a distribution with a large variance, so expect to run your simulation for more arrivals than before to get converged results.)

Required Input

The user should be able to enter the number of arrivals the simulation needs to run through an input prompt, and the value of the seed. (Recall that the seed is the first number that the pseudo-random generator uses.)

Required Output

The program should show the following quantities on the screen:

- Mean Number of packets in the system
- Average delay per costumer
- Average service time
- Average waiting time
- Idle period of server
- Blocking ratio

Sample Output

Your code should generate an executable file with name `part3`. We should be able to run your program as following and get the output as shown: (Numbers are not valid.)

```
 $./part2 10000 532
Mean Number of packets in the system: 241
Average delay per costumer: 920
Average service time: 3.14
Average waiting time: 900
Idle period of server: 0.145861
Blocking ratio: 0.1
```

Questions

First, you are only required to work with the size distribution. For now, forget about the queuing system.

(a) Find the theoretical values of $\mathbb{E}[S]$ and $\mathbb{E}[S^2]$.

(b) Generate 5 sets of 10^4 samples of such service times using different seeds, and find the average value of each set. Are the means close enough to the actual mean $\mathbb{E}[S]$?

(c) Vary the number of samples that you are using from 10^4 all the way to at least 10^5 and plot the sample mean with respect to the sample size. From the plot, what can you tell about the convergence of the sample mean to the actual mean?

(d) Using the plot, try to identify a number N such that the sample mean of any sample size $n \geq N$ is in the interval $(0.9\mathbb{E}[S], 1.1\mathbb{E}[S])$.

Now, work with the $M/G/1$ queue.

(f) Calculate the theoretical average delay in the system for this queue.

(g) Run the queuing simulation 10 times using different seeds for $L = \max\{N, 10^6\}$ arrivals and find the simulated average delay in the system. Are the results of the simulations with L arrivals satisfactory when you compare them with the theoretical delay? (If you are curious do see how many arrivals one needs to nail down the theoretical value, do the extra credit part.)

(h) Find the variance in the simulated average delay from the values found in part (f). Comment on its value.

Part 3 (Theory only)

In this part you will theoretically analyze a dispatching system. A dispatching system consists of a dispatcher that routes arriving jobs to one of the available servers. Each server has it's own queue and a dispatching policy assigns a queue for each job immediately upon arrival (Fig. 1). There are many dispatching policies that one could follow, eg random (a simple splitting of the arrival process with some probability), round robin (rotation between the queues with some predefined order), size interval task assignment (SITA) (assign the jobs to queues based on non-overlapping job size intervals) etc. We will focus on the SITA policy for this problem.

We have a dispatching system with two servers. The SITA policy for a two server system is as follows:

- Decide on a threshold T for the job sizes.
- If $S \leq T$, where S is the size of the job that arrived, the dispatcher routes the job to the queue of the first server.

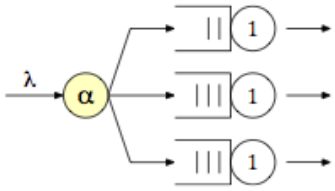

Figure 1: Dispatcher α feeds each server following some dispatching policy.

- Else the dispatcher routes the job to the queue of the second server.

Assume that the job sizes have the same probability density function as in Part 2 (Bounded Pareto $B(m, M, \alpha)$ distribution, with $m = 1, M = 10^4, \alpha = 1.4$) and both servers are of speed 1 (that is the service time for each job is equal to the job size).

Jobs arrive at the dispatcher at Poisson rate λ and are routed to the corresponding queue under the SITA policy.

Questions

(a) Prove that under the SITA policy the arrival process for each queue remains Poisson and compute the rate for these processes as a function of the threshold T. (Tip: Find the probability that an arriving job will go to each server and then use known results for Poisson processes.)

(b) Assume $T = M/2$, where M is the upper bound for the job sizes. Derive the equations for the average number of jobs in the system and the average delay for a job. (Tip: First, argue why the service time in each of the queues follows the same Bounded Pareto distribution but with different bounds).

(c) Find the threshold T that makes the loads ρ_1 and ρ_2 of the servers equal. Compare the average delay in (b) and (c). Comment on the result.

(d) Find the threshold T that minimizes the delay in the system. If you cannot find T in closed form by solving the corresponding equation/s, state the equation/s as a function of T, and then use a numerical solver to find the answer (e.g. www.wolframalpha.com). Compare the average delay in (c) and (d) and comment on the corresponding T values.

Notes

Makefile

In the Unix development environment, you can always put your compile commands into a file called `Makefile` and compile your code by typing `make` in the command prompt. For example if your compilation command is `g++ -o part1 main.cpp`, a simple Makefile can have the following code: (Replace `<TAB>` with the TAB key.)

```
project:
<TAB>g++ -o part1 main.cpp
```

Then, by executing `make` in the command prompt your code will be compiled. You can put the compilation command for all parts of the project in a single Makefile; then your Makefile will look like this:

```
project:
<TAB>g++ -o part1 main1.cpp
<TAB>g++ -o part2 main2.cpp
<TAB>g++ -o part3 main3.cpp
```

Submission Procedure

You also need to upload the zip file mentioned in the introduction (including all files requested) to the Digital Drop Box in the course website. You will use the Send File command to send your source code. The name of your source code MUST be of the form `EE465_Project_LastName_Name.zip`. If you plan to divide your code into multiple files, you need to put all files into this single zipped file, including all source codes, corresponding Makefile, and Readme to tell us how to compile and execute your code.

Development Environment

The code MUST be able to be compiled on USCs SUN machines which run Solaris OS (not Linux or Windows). Your code being able to compile under different compilers but not in UNIX will be treated same as if the code doesnt compile in any machine, and absolutely will not be run on different environments. Therefore make sure your code is successfully compiled and run under UNIX. We will be using aludra.usc.edu to test your code, therefore it makes sense to test it on the same machine before submitting. We will only run `make` and run the three executables as mentioned in the previous section.

Compiler

Use `gcc/g++` version 3.3.2 or newer for compiling your code. Information on how to use `g++` command can be obtained by checking the corresponding man pages by typing `man g++`. You can check for the version by `g++ --version`.

Academic Integrity

This assignment requires individual effort. I will use a sophisticated automated program checker to detect cheating. Be aware that the program checker has caught cheating students before.

Any student caught cheating will be definitely reported to the professor and be assigned an automatic F in the course (not only a zero credit for the assignment). Be honest and earn your grade.

Random Number Generation

For Part 1, change the seed of your random generators every time you run your code, you can use `srand48(unsigned((time(0)` to accomplish this. I am going to test your code at least two times and I dont want to see the same output. For Part 2 you can use `srand48(X0)` to set the seed to `X0`.

Questions?

Please make sure that you read the project description, this section and the questions and answers listed below thoroughly before asking any questions.

Some FAQs

Q. Which C function can be used for generating random number?

A. Use `drand48()` which gives a random number uniformly distributed between 0 and 1. Do not use `rand()` function since it might not give the desired precision needed to conduct a good simulation. You may initialize the random number generator at the beginning of your program (main routine) by using `srand48(unsigned(time(NULL)))`. The header files `stdlib.h` and `time.h` must be included in the program.

Q. In all cases do I have to make sure that the last customer departs from the system? Or at the end of the simulation time (last arrival) I can have some customers left in the queue?

A. Although there may still be some customers in the system, you should stop. In other words, stop your program when the last arrival occurs.

Q. How to generate an arbitrary random variable from a uniform random variable?

A. Refer to the lecture notes.

Q. Is collaboration allowed?

A. No.

Q. Can we use any language other than C/C++?

A. No, unless you take permission from the Professor, but you are strongly encouraged to stick with C/C++ in order to be able to complete Part 2.