CS-350 - Fundamentals of Computing Systems
Homework Assignment #4

Due on October 21, 2022 at 11:59 pm

Prof. Renato Mancuso

Renato Mancuso

Problem 1

Your career as an independent game developer is going better than expected! You are almost ready to release your first multi-player, called *Goat Simulator: The Heard*[1]. You decide to implement the game server using a single-processor machine that you found in your garage. You tested the first client and noticed that whenever a user plays the game, an average of 10 requests per second are sent to the server. It also appears that the arrival of requests is Poisson-distributed. Furthermore, you have noticed that if any of the requests is dropped/rejected, then the client generating the request will disconnect — which is frustrating for the wannabe goat users. On the other hand, if the average per-request latency grows beyond 0.5 milliseconds, then all the players will experience lags. You can also assume that the service time for each request is exponentially distributed.

a) At the beginning, you strive to have exactly 0% rejection rate by creating a very large request queue in your server. What should be the capacity of the server to prevent the occurrence of lags when 1500 users are playing the game?

b) Assume that you were able to implement the server such that the average per-request service time is 0.06 milliseconds. How many active users can you support before the probability of a generic request of experiencing queuing (i.e. of not being processed right away) grows beyond 81%?

c) You quickly realize that you are not able to meet the ideal specifications demanded by Part a). Apparently, the best you can do is to achieve an average per-request service time of 0.07 msec. You then decide to limit the size of the queue to introduce some rejection in order to limit the lag for everyone. Note that a request occupies space in the queue while it is being processed. Is it possible to have 1500 active users and to set a cap on the request queue somewhere between 10 and 15 to guarantee the constraint on the server latency (less then 0.5 milliseconds) while rejecting less than 11% of all the requests?

Regardless of what you computed above, assume that you set the queue size to 10. Keep considering the given service time of 0.07 msec, and 1500 active users.

d) What is the probability that a request will be processed right away without experiencing any queuing?

e) What is the probability that when a generic request arrives, it finds exactly 5 or 4 other requests in the queue?

[1] A much acclaimed multi-player spin on the award winning Goat Simulator 3

Problem 2

The latest and greatest version of the AliImpress e-commerce infrastructure is structured as a three-tier platform, with three single-CPU servers S1, S2 and S3. S1 handles the arrival of new users that have never seen before—to create a new account. After S1, a request from these new users must visit S3, which is responsible for session creation. After a valid session has been established, a request will move from S3 to S2 for processing. With 0.6 probability, a generic request completes at S2 and leaves the platform. Otherwise, additional processing is required. In this case, a new session needs to be created by visiting S3, and from there the request continues just like any other request visiting S3. Differently from new users' requests, requests from well-known users arrive directly at S2 and from there proceed just like described above — i.e., they might complete and leave rigth away after S2, or go to S3 for an additional session and so on.
At steady-state, the system receives $\lambda_a = 10$ requests per second from new users, and $\lambda_b = 50$ requests per second from well-known users. Moreover, S2 has an average service time of 9 milliseconds; S3 has an average service time of 19 milliseconds. The service time of S1 is unknown, but you were able to measure the average response time at S1, which is 525 milliseconds.

a) Provide a diagram of the system and specify the queuing model used for S1, S2, and S3, the assumptions that need to be made and the name of the theorem used to decompose the system.

b) What is the average service time of S1?

c) What is the utilization of S2?

d) What is the average total number of requests in the system—either waiting for service or being served?

e) What is the capacity of the system, assuming that it always holds that $\lambda_b = 5 \cdot \lambda_a$?

f) What is the response time measured from entry to exit of a generic request, regardless of whether it is from new or existing users?

Problem 3

The Karatepeli Systems Inc. enterprise has put together an online media conversion system. It allows users to submit multi-party media files comprised of potentially many data payloads. Some of these payloads are audio streams (AS), and others are video streams (VS).
In each file, there is a specific order and sequence in which the payloads need to be processed, for instance, a file might look like the following: AS→VS→VS→AS.... For the same file, the next payload cannot start processing until the previous one has completed processing. The system receives 11 requests per hour that begin with an AS payload. It receives 17 requests per hour that begin with a VS payload.
The media conversion system is comprised of three subsystems. A single-CPU server, namely AUD, is dedicated to pre-process AS payloads, which on average takes 1.5 minutes. Another single-CPU server, namely VID, is dedicated to pre-process VS payloads which takes about 1.1 minutes. Once pre-processing is performed at either the VID or AUD server, the current payload being worked on is passed to the media converter server CONV which is a 3-CPU machine. Here, all the files being worked on are put into a single queue. Whenever any of the 3 CPUs becomes available, the file at the head of the queue can begin processing on the current payload. On average, it takes 2 minutes for a generic payload to be processed at the CONV machine.
Once processing for the current payload is completed at the CONV machine, the next payload can be an AS with probability 0.25 or VS with probability 0.4 and the file is routed to AUD or VID accordingly. The current payload can also be the final one (after which the response can be returned to the user) with probability 0.35.

a) Provide a diagram of the system and state the assumption that must hold for you to analytically solve the system.

b) How many payloads per hour are processed by the CONV machine?

c) Which server is the bottleneck of the system?

d) How many files, on average, are being concurrently handled by the entire system?

e) What is the average response time perceived by the end users when they submit a file for conversion to the system?

f) Consider the case where the rate of arrival of new files at the VS does not change. By how much can the rate of arrival of new files at the AS increase while still allowing the system to reach steady-state?

g) What is the average number of payloads in each media file?

h) What is the probability that a payload arriving at the CONV machine immediately starts service without having to wait in the queue?

Problem 4

Code integration: In this problem you will develop code to simulate an entire system with multiple servers. The following is the workflow of the system. First, class X and class Y requests arrive with rate λ_x and λ_y, respectively. They both enter at server $S0$ which is a single-processor system with average service time $T_{s0,x}$ for X-class requests and $T_{s0,y}$ for Y-class requests. The queue dispatch policy of $S0$ is either FIFO or SJN. From here, the request goes to either $S1$ or $S2$. They go to $S1$ with probability $p^x_{0,1}$ (for X-class requests) and $p^y_{0,1}$ (for Y-class requrests). $S1$ has a single infinite FIFO queue and 1 processor, with average service time T_{s1} (regardless of the request class). $S2$ has a single processor and a FIFO queue with a K_2 maximum size (this includes the request being currently served). The processor has a service time average T_{s2} regardless of the class. Any request that completes processing at $S1$ goes back to $S0$ with probability $p_{1,0}$ regardless of the class, and leaves the system entirely otherwise. Any request that completes processing at $S2$ goes back to $S0$ with probability $p_{2,0}$ regardless of the class, and leaves the system entirely otherwise.
Assume all service times for $S0 - S2$, as well as inter-arrival times of requests from the outside at $S0$ are exponentially distributed.
Write a Java class namely "Simulator" in its own file `Simulator.java` that implements a discrete event simulator. The simulator should have a method with the following prototype: `void simulate(double time)`, that simulates the arrival and execution of requests at the system described above for `time` milliseconds, where `time` is passed as a parameter to the method. You should re-use code you wrote as part of the previous assignments.
Apart from implementing the `simulate(...)` method, the class should also include a `public static void main(String [] args)` function. The `main(...)` function should accept 13 parameters from the calling environment (in the following order):

1. length of simulation time in milliseconds. This should be passed directly as the `time` parameter to the `simulate(...)` function.
2. average arrival rate of requests of class X (λ_x);
3. average arrival rate of requests of class Y (λ_y);
4. average service time at $S0$ for requests of class X ($T_{s0,x}$);
5. average service time at $S0$ for requests of class Y ($T_{s0,y}$);
6. queuing policy which can either be "FIFO" or "SJN";
7. probability $p^x_{0,1}$ of going to $S1$ from $S0$ for X-class requests;
8. probability $p^y_{0,1}$ of going to $S1$ from $S0$ for Y-class requests;
9. average service time at $S1$ for any request (T_{s1});
10. average service time at $S2$ for any request (T_{s2});
11. queue size limit at $S0$ (K_2);
12. probability $p_{1,0}$ of going back to $S0$ from $S1$ for any request ($p_{1,0}$);
13. probability $p_{2,0}$ of going back to $S0$ from $S2$ for any request ($p_{2,0}$).

All times should be intended in milliseconds.
It is responsibility of the `main(...)` function to internally invoke the implemented `simulate(...)` function **only once**. The `simulate(...)` function will need to print in the console the simulated time at which each request arrives at the system (`ARR`), initiates service at Si (`START Si`), and completes service (`DONE Si`).

Whenever a request goes from server Si to Sj (e.g., from $S0$ to $S2$), print a statement of the form: `FROM Si TO Sj`. If the request goes out after Si, print `FROM Si TO OUT`. When a new request arrives at $S2$ while the current total number of requests in the system is already K_2 ($S2$ is full), then the newly arrived request is dropped and discarded (see `DROP S2`). The output must look like this (for simplicity, only X-class requests are considered and $K_2 = 3$):

```
X0 ARR: <timestamp>
X0 START S0: <timestamp>
X1 ARR: <timestamp>
X2 ARR: <timestamp>
X0 DONE S0: <timestamp>
X0 FROM S0 TO S1: <timestamp>
X0 START S1: <timestamp>
X1 START S0: <timestamp>
X1 DONE S0: <timestamp>
X1 FROM S0 TO S2: <timestamp>
X1 START S2: <timestamp>
X2 START S0: <timestamp>
X2 DONE S0: <timestamp>
X2 FROM S0 TO S2: <timestamp>
X3 ARR: <timestamp>
X3 START S0: <timestamp>
X3 DONE S0: <timestamp>
X3 FROM S0 TO S2: <timestamp>
X4 ARR: <timestamp>
X4 START S0: <timestamp>
X4 DONE S0: <timestamp>
X4 FROM S0 TO S2: <timestamp>
X4 DROP S2: <timestamp>
X5 ARR: <timestamp>
X5 START S0: <timestamp>
X6 ARR: <timestamp>
X7 ARR: <timestamp>
X5 DONE S0: <timestamp>
X5 FROM S0 TO S1: <timestamp>
X6 START S0: <timestamp>
X0 DONE S1: <timestamp>
X5 START S1: <timestamp>
X0 FROM S1 TO S0: <timestamp>
X1 DONE S2: <timestamp>
X2 START S2: <timestamp>
X1 FROM S2 TO OUT: <timestamp>

S0 UTIL: <utilization of S0>
S0 QAVG: <avg. number of requests in S0>
S0 WAVG: <avg. number of requests waiting in S0>

S1 UTIL: <utilization of S1>
S1 QAVG: <avg. number of requests in S1>
S1 WAVG: <avg. number of requests waiting in S1>
```

```
S2 UTIL: <utilization of S2>
S2 QAVG: <avg. number of requests in S2>
S2 WAVG: <avg. number of requests waiting in S2>
S2 DROPPED: <total number of dropped requests>

S3 UTIL: <utilization of S3>
S3 QAVG: <avg. number of requests in S3>
S3 WAVG: <avg. number of requests waiting in S3>

QAVG X: <avg. number of X-class requests in the entire system>
QAVG Y: <avg. number of Y-class requests in the entire system>
TRESP X: <avg. response time of X-class requests>
TRESP Y: <avg. response time of Y-class requests>
TWAIT X: <avg. waiting time of X-class requests>
TWAIT Y: <avg. waiting time of Y-class requests>
```

where `<timestamp>` is the simulated time in milliseconds at which the event occurred printed in decimal format. This is also called the **trace** of the simulation. Also, `<utilization>` is a decimal number between 0 and 1, `<avg. number of requests>` is a decimal number, `<avg. response/waiting time of requests>` is a decimal number expressed in milliseconds, and `<total number of dropped requests>` is an integer.

Submission Instructions: in order to submit this homework, please follow the instructions below for exercises and code.

The solutions for Problem 1-3 should be provided in PDF format, placed inside a single PDF file named `hw4.pdf` and submitted via Gradescope. Follow the instructions on the class syllabus if you have not received an invitation to join the Gradescope page for this class. You can perform a partial submission before the deadline, and a second late submission before the late submission deadline.

The solution for Problem 4 should be provided in the form of Java source code. To submit your code, place all the `.java` files inside a compressed folder named `hw4.zip`. Make sure they compile and run correctly according to the provided instructions. The first round of grading will be done by running your code. Use CodeBuddy to submit the entire `hw4.zip` archive at `https://cs-people.bu.edu/rmancuso/courses/cs350-fa22/codebuddy.php?hw=hw4`. You can submit your homework multiple times until the deadline. Only your most recently updated version will be graded. You will be given instructions on Piazza on how to interpret the feedback on the correctness of your code before the deadline.