Department of Computer Science BOSTON UNIVERSITY

CS-350 - Fundamentals of Computing Systems

Midterm Exam #1
Fall 2020

Name: _____

Login: _____ BU ID: _____

Submission instructions: You can either (i) take a print out of the booklet, write your answers down on it, and then scan it as PDF before submitting to Gradescope, or (ii) use the Latex template to to type in answers and submit the PDF through Gradescope, or (iii) simply copy the questions to your favorite text editor, type in your answers below every question, and upload a PDF through Gradescope. **NOTE:** *when uploading your submission, make sure to associate your answers with the correct PDF page.*

Problem #1:	/22
Problem #2:	/20
Problem #3:	/23
Problem #4:	/22
Problem #5:	/23
Total Score:	/110

Note:

- This is a open-book/notes exam.
- Cite any external source you use.
- Calculators are allowed.
- You have 24 hours to complete your exam.
- There are 110 total points.
- Show your work for full marks.
- **Explain all your assumptions clearly.**

I give credit to students who can accurately assess their performance. Thus, after you finish the exam, answer the following for up to five bonus points!

Bonus = max(0, 5 - |GuessedGrade - ActualGrade|)

What do you guess your score (out of 110) will be? _____

Problem 1

You are studying the performance of a program executing on a CPU which features a cache memory to speed up the execution of instructions. Every time the CPU executes an instruction, it can result in a cache "hit" or a cache "miss". Having a cache hit is beneficial to the overall runtime of the program. After studying the cache system, you have measured that an instruction results in a cache hit 30% of the time. This quantity is called *hit rate*. Assume that the hit rate never changes. The program under analysis begins at the entry point; it then executes exactly 10 instructions. After that the execution might follow path A with probability 80%; or path B with probability 20%. When the program executes path A, it will execute 5 more instructions; it will execute 10 more instructions on path B. Regardless of the path taken, the program will terminate at the end of path A or B.

(a) [2 points] Provide the PMF for the total number of instructions that will be executed by the program, regardless of whether they result in cache hits or misses.

Solution:
Call x the total number of instructions that will be executed. There are only two possible outcomes: $P(x = 15) = 0.8$ and $P(x = 20) = 0.2$.

(b) [3 points] How many instructions will be executed on average across many runs of the program?

Solution:
This is simply the expected value of the random variable x. $E[x] = 15 \cdot P(x = 15) + 20 \cdot P(x = 20) = 16$.

(c) [5 points] Assume that we want to observe one run in which the instructions on path are B being executed. How many times, on average, we should execute the program?

Solution:
This is the average of a Geometric distribution with probability of failure $p_A = P(pathA) = 0.8$. Avg. runs to observe path B is $\frac{p_A}{1-p_A} + 1 = \frac{1}{1-p_A} = 5$.

(d) [6 points] Provide the PMF for the number of hits h for the entire program. *HINT:* you can use if/else statements in your formula.

Solution:
Let's distinguish two cases. The case where $h \leq 15$ and the case where $h > 15$. The latter can happen only on path B.

The second case ($h > 15$), can only happen if we take path B, so with $p_B = 1 - p_a = 0.2$. Then, we have a simple Binomial distribution, so $P(h = k) = \binom{20}{k} p_{hit}^k (1 - p_{hit})^{20-k}$ with $15 < k \leq 20$.

The first case holds for values of $0 \leq k \leq 15$. If we take path A with probability $p_A = 0.8$ we will have $P(h = k) = \binom{15}{k} p_{hit}^k (1 - p_{hit})^{15-k}$; while if we go through path B we once again have $P(h = k) = \binom{20}{k} p_{hit}^k (1 - p_{hit})^{20-k}$.

Putting everything together:

$$P(h = k) = \begin{cases} p_A \cdot \binom{15}{k} p_{hit}^k (1 - p_{hit})^{15-k} + p_B \cdot \binom{20}{k} p_{hit}^k (1 - p_{hit})^{20-k} \text{ if } 0 \leq k \leq 15 \\ p_B \cdot \binom{20}{k} p_{hit}^k (1 - p_{hit})^{20-k} \text{ if } 15 < k \leq 20 \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

(e) [6 points] What is the average number of hits you expect to see throughout the execution of the entire program?

Solution:

Here we can consider the two paths independently. If we go through path A, then the mean number of hits will be $15 p_{hit}$; if we go through path B the mean number of hits will be $20 p_{hit}$. The overall average number of hits is just a weighted average of the two: $p_A 15 p_{hit} + p_B 20 p_{hit} = 4.8$.

Problem 2

A dear friend of yours called Salak is trying to buy a new graphic card from *woosh.com*. After having filtered through a bunch of different options, Salak has narrowed her options down to three models that have the same exact price (shipping included). To be able to break the tie, she looks at the result of the performance benchmarks available for the three cards. The benchmarks use *Blunder 3D* to generate 3D scenes to be rendered. They run a 3D scene rendering multiple times and collect the time it takes for each rendering (a.k.a. rendering time) to be completed once it has started processing. After performing multiple tests, the benchmark reports the average rendering time and a trustworthy guess about the distribution of the rendering time. The results are as follows:

- **Model A:** The rendering time is uniformly distributed between 5 and 6 microseconds;
- **Model B:** The rendering time is exponentially distributed with a mean of 5 microseconds;
- **Model C:** The rendering time is constant and equal to 6 microseconds;

Salak forwarded to you all this information and happily announced that she is decided to buy the Model B graphic card because according to her "it is the fastest card, duh!". It turned out that she will mostly use the card for 3D modeling with Blunder 3D which generates, on average, 140,000 scene rendering requests every second.

(a) [5 points] Instead of rushing to decide based on the benchmark data alone, what metric(s) you suggest Salak to look into to have a better idea of the performance she should expect when working with Blunder 3D?

Solution:
Assuming that the arrival rate of requests form the 3D rendering software is Poisson, Salak is not considering the queuing time that will arise from the randomness in the performance of the three cards. The metrics Salak should evaluate are q (average queue buildup of rendering requests) and T_q (average response time of rendering requests).

(b) [10 points] Since Salak has not taken CS350, can you show her how you would compare the three models and which one, in your opinion, is to be preferred based on your reasoning?

Solution:
Model A is an M/G/1 with $T_s = 5.5$ microsec. and $\sigma_{T_s} = \sqrt{\frac{(6-5)^2}{12}} = 0.2887$. We can compute $A = \frac{1}{2}(1 + (\frac{\sigma_{T_s}}{T_s})^2) = 0.5014$. Then $T_q = T_s(\frac{\rho A}{1-\rho} + 1) = 14.73$ microsec.

Model B is an M/M/1 system with $T_s = 5$ microsec. We compute $T_q = \frac{T_s}{1-\rho} = 5/(1-0.7) = 16.67$ microsec.

Model C is an M/D/1 system with $T_s = 6$ microsec. We compute $T_q = T_s(\frac{\rho}{2(1-\rho))} + 1) = 21.75$ microsec.

So the best option is Model A because it exhibits the smallest expected value of response time.

(c) [5 points] What speed-up should Salak expect for Blunder 3D traffic (on a per-request basis) when picking Model A instead of Model B?

Solution:
The expected speedup can be computed as speedup $= \frac{T_{q,B}}{T_{q,A}} = 1.13\times$.

Problem 3

Your career as an independent game developer is going better than expected! You are almost ready to release your first multi-player game, called *AutumnGuys*. 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 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) [5 points] 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?

Solution:
The input from each user is $\lambda_u = 10$ req./sec; with 1500 active users we have $\lambda = 1500\lambda_u = 15$ req./msec. We want to make sure that $T_q \leq 0.5$ msec.

We know that $T_q = \frac{q}{\lambda}$ from Little's Law. We can set the constraint $q \leq \lambda 0.5 = 7.5$. Since the queue is very large, assume we are dealing with an M/M/1 system. We have $q = \frac{\rho}{1-\rho}$, from which follows that $\rho = \frac{q}{1+q}$. Then the maximum utilization we can allow to prevent lags is $\rho \leq 0.8823$. Because $\rho = \frac{\lambda}{\mu}$, we have a constraint on the capacity of the server $\mu \geq 17$ req./msec.

(b) [5 points] 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%?

Solution:
With the given $T_s = 0.06$ msec, we are wondering what is that number of active users n for which $n\lambda_u T_s = \rho \leq 0.81$. It follows that $n \leq \frac{0.81}{\lambda_u T_s} = 1350$ users.

(c) [8 points] 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?

Solution:
The limit on the queue size turns the problem to fit an M/M/1/K model, where $K \in \{10, \ldots, 15\}$. We then reason about the probability of rejection. We know that $\rho = 1500\lambda_u T_s = 1.05$. From the M/M/1/K model with $\rho \neq 1$, we know that $P_{rej} = \frac{(1-\rho)\rho^K}{1-\rho^{K+1}}$.

Computing the formula above we find that the first value of K that satisfies the constraint on the rejection rate is K = 11. We have $P_{rej,K=11} = 10.75\%$.

The accepted traffic rate in this case is $\lambda_{in} = (1 - 0.1075)1500\lambda_u = 13.3882$ req./msec. We can also compute $q = \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}} = 6.0781$. Finally, $T_q = \frac{q}{\lambda_{in}} = 0.454$ msec. So by limiting the queue size to 11 requests, we can achieve the desired latency while dropping less than 11% of our requests.

(d) [5 points] 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. What is the probability of the following events: (1) that a request will be processed right away without experiencing any queuing; (2) that when a generic request arrives, it finds exactly 5 other requests in the system — including those being served.

Solution:
For (1), we just need to compute $P(S_0) = \frac{1-\rho}{1-\rho^{K+1}} = 0.07038$; for (2) we compute $P(S_5) = 0.08984$.

Problem 4

The enterprise database system called *ChupaData* has been deployed over a 2-CPU server. The database is being used to store the prices of various Halloween decorations. The database is implemented so to spawn as many worker threads as the number of CPUs available in the system. This way, when new transactions are submitted to the DB, they are queued in a global queue, and then served by the first available worker thread. To process each transaction, a worker thread takes 0.13 seconds on average. During the month of October, the DB receives an average of 10 requests per second. But during the month of November, the traffic is expected to double.

(a) [2 points] Compute the capacity of the deployed *ChupaData* DB.

Solution:
The capacity is the maximum sustainable throughput, so that value of λ^* such that $\rho = \frac{\lambda^*}{2\mu} < 1$. Hence $\lambda^* = 2/T_s = 15.385$.

(b) [5 points] If we were to observe the system throughout the month of October, how likely it is to find out that both worker threads are busy processing transactions?

Solution:
To solve this part, we need to compute C for an M/M/2 system. First, we know that $\rho = 5 \cdot 0.13 = 0.65$. From this, we compute $K = \frac{1+2\rho}{1+2\rho+(2\rho)^2/2} = 0.7313$. And finally the probability that both servers are busy is $C = \frac{1-K}{1-\rho K} = 0.5121$.

(c) [5 points] Is it fair to advertise our system in October by saying that a generic transaction will start receiving service, on average, after being queued for less than 100 milliseconds?

Solution:
To solve this part, we need to compute the current average waiting time T_w. Let's start with q for an M/M/2 system. We know that $q = 2\rho + \frac{C\rho}{1-\rho} = 2.25$ requests. The average number of requests waiting for service are $w = q - 2\rho = 0.95$. Finally, from Little's Law we have $T_w = w/\lambda = 0.095$ seconds. So it is fair to advertise the system as stated in October.

(d) [5 points] Is it wise to keep the same exact system configuration for the month of November? Motivate your answer.

Solution:
It's not wise because if we double the input traffic, we have $\rho' = 2\rho = 1.3$, which means the server will explode!

(e) [5 points] In order to prepare for the increase in traffic in November, you decide to re-deploy the DB on a three-CPU machine. What per-request response time average you expect in November after this upgrade?

Solution:

Here we simply re-compute T_q for an M/M/3 system with the new $\lambda = 20$ req./sec for November traffic. This leads to $\rho = 0.8667$.

We can then recompute $K = \frac{1+3\rho+(3\rho)^2/2}{1+3\rho+(3\rho)^2/2+(3\rho)^3/6} = 0.7044$.

Next, $C = \frac{1-K}{1-\rho K} = 0.7589$.

Finally, $q = 3\rho + \frac{C\rho}{1-\rho} = 7.5345$ requests.

Problem 5

You are studying the performance of an Audio/Video Encoder which processes encoding requests coming to the system, on average, every 0.5 seconds. The inter-arrival of requests appears to be exponentially distributed. The A/V Encoder has three resources that it internally uses to process requests, namely a CPU for general processing, a GPU for video encoding, and a DSP (Digital Signal Processor) for audio encoding. Each request is comprised of a sequence of chunks that need to be processed sequentially.
The first chunk of a new request always requires processing at the CPU. After a CPU chunk is processed, there are three possibilities: (1) the request might not require any further processing (and be considered done at the encoder) with probability 0.4. Otherwise, (2) with probability 0.2, the next chunk requires processing at the GPU. Alternatively, (3) the next chunk requires DSP processing.
After processing at the GPU, the request might be done and leave the system with probability 0.6. Similarly, after a chunk is completed at the DSP, the request is completed and it leaves the system with probability 0.7. It is always the case that, if a request did not complete after a GPU or DSP chunk, the next chunk will require processing at the CPU. If this happens, the request goes back to the CPU and continues from there as if it was a new request.

System Resource	**CPU**	**GPU**	**DSP**
Avg. Processing Time (ms)	300	1600	950

Table 1: Average processing time of CPU, GPU, and DSP chunks.

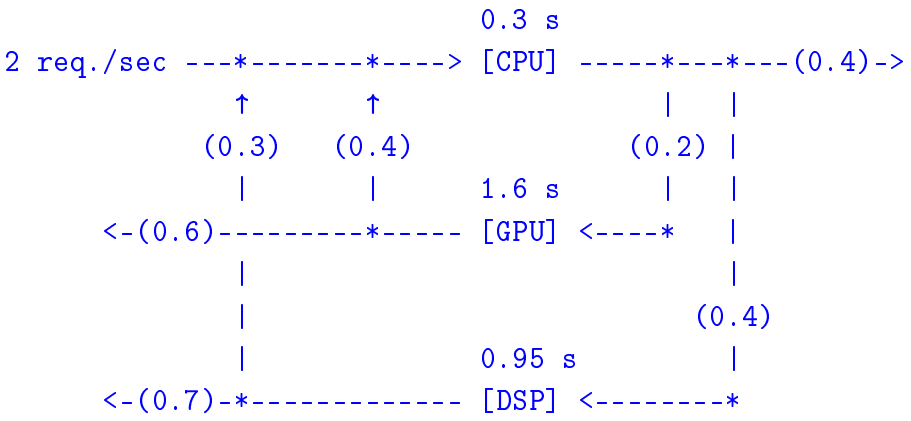
You have measured the average processing time of CPU, GPU, and DSP chunks. They turned out to be exponentially distributed and provided in Table 1.

(a) [3 points] Draw the system diagram, with the routes that the requests can follow as they flow through the system labeled with the corresponding routing probability. State the model you are using to model the resources and the assumptions you are making to approach this problem.

Solution:
Assumptions: (1) CPU, DSP and GPU are M/M/1's — i.e. they have exponentially distributed service time and infinite queues; (2) External arrivals are Poisson; (3) System is at steady-state;

System Diagram (ASCII Art Version)

[points]



(b) [5 points] How many chunks of CPU processing a generic request has on average?

Solution:

Each round of a request through the system and back to the CPU is essentially a single Bernoulli trial. After the first chunk at the CPU, the request comes back to the CPU ("fails") with probability $p = 0.2 \cdot 0.4 + 0.4 \cdot 0.3 = 0.2$. The average number of failures for a Bernoulli with p probability of failure is $p/(1-p) = 0.25$. So on average a request will have 1.25 CPU chunks.

(c) [5 points] Which resource represents the bottleneck in the system? Motivate your answer.

Solution:

Let's first calculate the traffic through all the resources. The external arrivals are $\lambda = 1/0.5 = 2$ req./sec. First, for the CPU we have $\lambda_{cpu} = \frac{\lambda}{1-(0.2 \cdot 0.4 + 0.4 \cdot 0.3)} = \frac{\lambda}{0.8} = 2.5$. Next, $\lambda_{gpu} = 0.2\lambda_{cpu} = 0.5$. Finally, $\lambda_{dsp} = 0.4\lambda_{cpu} = 1$.

Now we can derive the utilization for all the resources. These are $\rho_{cpu} = 2.5T_{cpu} = 0.75$, $\rho_{gpu} = 0.5T_{gpu} = 0.8$, and $\rho_{dsp} = 1T_{dsp} = 0.95$.

So the bottleneck is the DSP because it is the resource with the highest utilization.

(d) [3 points] How many requests on average will be inside the system at steady state? That includes any request that is currently receiving service or being queued at any of the resources of the encoder.

Solution:

For this part, we can analyze the various M/M/1 systems in isolation leveraging the

Jackson's Theorem. We derive $q_{cpu} = \frac{\rho_{cpu}}{1-\rho_{cpu}} = 3$, $q_{gpu} = 4$, and $q_{dsp} = 19$. So the total average number of requests in the system is $q_{tot} = 26$.

(e) [5 points] What is the capacity of the encoder?

Solution:
The capacity is the input rate λ^* that makes the bottleneck explode. Because the bottleneck is the DSP, we set the inequality $\lambda^*_{dsp} T_{dsp} < 1$. It follows that $\lambda^*_{dsp} < 1.0526$. Finally, we know that $\lambda^*_{dsp} = \frac{0.4\lambda^*}{0.8}$. So the maximum traffic (i.e. capacity) that the whole system can sustain is $lambda^* = 2\lambda^*_{dsp} = 2.1052$ req./sec.

(f) [2 points] Can we upgrade the GPU to be twice as fast to improve the capacity of the system?

Solution:
Nope, because the bottleneck is the DSP, so improving the speed of the GPU will not impact the capacity of the system.

