Queuing Systems

Computer Exercise 1

2022

1 Instructions

To be able to complete the computer exercise within the time limit, you have to prepare yourself by completing the home assignments before the exercise. If you have any questions regarding the home assignments, ask your tutor.

In this computer exercise, queuing systems will analyzed theoretically and through simulations. python files for this purpose can be downloaded from the home page of the course.

For every task, there are a number of questions to answer. Some of the questions are marked (**) and these shall be discussed with the tutor before proceeding with the computer exercise. You are supposed to do some thinking yourself before talking to the tutor. If the tutor is occupied elsewhere, you can continue until he becomes available.

1.1 Vocabulary

State transition diagram = tillståndsdiagram
Steady state probabilities = tillståndsannolikhet (p_i)
Loss system = upptagetsystem (det finns inga buffertplatser)
Call congestion = sannolikheten att en kund spärras

2 Home Assignments

In the first part of this computer exercise, a web server will be studied by using a queuing model. First the server is modelled as an M/M/1 system with infinite buffer. The arrival and service rates are λ and μ, respectively.

1. Draw the state transition diagram of our model.

2. Derive the steady state probabilities, p_k, meaning the probability for having k jobs in the system.

3. Derive the expression for the mean number of jobs in the system, $E(N)$, from the steady state probabilities, p_k.

4. Derive the mean time a job spends in the system, $E(T)$, from the mean number of jobs, $E(N)$.

To make our model more realistic, we now assume that the queueing system is limited to L positions.

5. Draw the corresponding state transition diagram.

6. Derive the steady state probabilities, p_k, of the new model.

7. Express the probability that a customer is blocked using the steady state probabilities, p_k.

8. The python script below simuates a queuing system. Study the script and answer the questions after the code.

```
import heapq
import random
import matplotlib.pyplot as plt
import numpy as np
from queue import Queue

# import matplotlib as mpl
# mpl.use('tkagg')
# import matplotlib.pyplot as plt

signalList = []
def send(signalType, evTime, destination, info):
    heapq.heappush(signalList, (evTime, signalType, destination, info))

GENERATE = 1
ARRIVAL = 2
MEASUREMENT = 3
DEPARTURE = 4

simTime = 0.0
stopTime = 10000.0

class larger():
    def __gt__(self, other):
        return False

class generator(larger):
    def __init__(self, sendTo,lmbda):
        self.sendTo = sendTo
        self.lmbda = lmbda
        self.arrivalTimes = []
    def arrivalTime(self):
        return simTime + random.expovariate(self.lmbda)
    def treatSignal(self, x, info):
        if x == GENERATE:
            send(ARRIVAL, simTime, self.sendTo, simTime)
            send(GENERATE, self.arrivalTime(), self, [])
            self.arrivalTimes.append(simTime)
```

```
class queue(larger):
    def __init__(self, mu, L, sendTo):
        self.numberInQueue = 0
        self.sumMeasurements = 0
        self.numberOfMeasurements = 0
        self.measuredValues = []
        self.buffer = Queue(maxsize=0) #maxsize=0 means that there is no upper limit
                                       #for the number of items that can be stored in the
                                       #queue
        self.mu = mu
        self.L = L
        self.sendTo = sendTo
    def serviceTime(self):
        return simTime + random.expovariate(self.mu)
    def treatSignal(self, x, info):
        if x == ARRIVAL:
            if self.numberInQueue == 0:
                send(DEPARTURE,self.serviceTime() , self, []) #Schedule  a departure for t
            self.numberInQueue = self.numberInQueue + 1
            self.buffer.put(info)
        elif x == DEPARTURE:
            self.numberInQueue = self.numberInQueue - 1
            if self.numberInQueue > 0:
                send(DEPARTURE, self.serviceTime(), self, [])  # Schedule  a departure for

            send(ARRIVAL, simTime, self.sendTo, self.buffer.get())
        elif x == MEASUREMENT:
            self.measuredValues.append(self.numberInQueue)
            self.sumMeasurements = self.sumMeasurements + self.numberInQueue
            self.numberOfMeasurements = self.numberOfMeasurements + 1
            send(MEASUREMENT, simTime + random.expovariate(1), self, [])



class sink(larger):
    def __init__(self):
        self.numberArrived = 0
        self.departureTimes = []
        self.totalTime = 0
        self.T = []
    def treatSignal(self, x, info):
        self.numberArrived = self.numberArrived + 1
        self.departureTimes.append(info)
        self.totalTime = self.totalTime + simTime - info
        self.T.append(simTime - info)
```

```
  ##################################################
  #
  # Add code to create a queuing system  here
  #
  ##################################################


while simTime < stopTime:
    actEvent = heapq.heappop(signalList)
    simTime = actEvent[0]
    actEvent[2].treatSignal(actEvent[1], actEvent[3])
```

a. How many servers does the queuing system have?

b. How many places does the buffer have?

c. What is the mean time between arrivals?

d. What is the list *arrivalTimes* used for?

e. What does the list T contain after the simulation run?

f. What does the list *measuredValues* contain after the simulation run?

g. Mark (by e.g. underlining) where the service times are set in the code.

In the second part of the computer exercise, we will study loss systems. These systems can for example be found in wireless cellular networks such as GSM. We assume that a cell in a GSM network has access to m radio channels and that the call length is exponentially distributed with a mean value of $1/\mu$. The number of subscribers in the cell is M (limited amount of subscribers) and the call arrival rate of every single free subscriber is β.

9. Draw the state transition diagram.

10. We assume that the steady state probabilities, p_k, of the system are known. Express the probability the system is full from these probabilities.

11. Express the call congestion from the steady state probabilities, p_k.

12. Which is the most important measure, the probability the system is full or the call congestion? Why?

If the number of subscribers is large in comparison to the number of radio channels ($M/m \gg 10$), our model can be altered.

15. Draw the state transition diagram of the new model.

3 Lab assignment: M/M/1 with infinite queue

First, we assume that the web server can be modeled as an M/M/1 system with infinite queue. Obviously, these assumptions are not entirely correct since the queue must be limited in size due to memory restrictions. It is also doubtful if the service times are exponentially distributed. A python-program simulating an M/M/1 system with $\lambda = 7\ \mathrm{s}^{-1}$ and $\mu = 10\ \mathrm{s}^{-1}$ during 1000 seconds can be found in the python-file MM1.py.

1. Simulate the system and plot the number of jobs in the system at the instant of a measurement event. Hint: Use the N-vector and the python command plot(N). Sketch the plot in the graph below.

2. Calculate the average number of jobs in the system, $E[N]$, from the simulation results. Hint: Use the function np.mean(queue.measuredValues).

3. Calculate theoretically $E(N)$ (see home assignment 3) and compare the obtained value with the simulated value.

4. Make a histogram over the obtained measurements for N. Be inspired by the example of how to draw a histogram in the file plotting.py. What does the histogram show?

5. Normalize the histogram and plot the result in the graph below.

A program in the file pkMM1.py, calculates the steady state probabilities for an M/M/1 system with infinite queue. Plot the steady state distribution in the same diagram as the histogram. Python hint: first plot the histogram as in the previous task, the hold on feature is switched on by default in matplotlib.pyplot. So each time you evoke plt.plot() before plt.show() a drawing is added to the plot.

6. What does the histogram and the curve show and how have they been obtained?

7. Alter the simulation length to 50 s and then 3000 s. Plot the histogram for these two simulation lenght just as in the previous problem. What impact does the simulation length have on the accuracy of the numerical results? (**)

The number of jobs in a system is interesting for the operator when dimensioning buffer sizes, but for the customers, the total response time is a more interesting measure. Now we will continue to study the behavior of an M/M/1 system using the simulation length of 1000 seconds.

8. Calculate the mean response time, T, directly from output vector (T). Hint: Use the numpy-function mean.

We will now investigate what will happen to the mean response time when λ is altered.

9. Plot the number of jobs at the instant of a measurement event for $\rho = 1.1$ and compare it with the corresponding plot for $\rho = 0.7$. Explain the differences. (**)

10. Fill in the mean response time, T, for $\rho = 0.25, 0.5, 0.7, 0.9, 1.0, 1.1$ in the table below.

ρ	0.25	0.5	0.7	0.9	1.0	1.1
T						

11. Plot the mean response time, T, for the investigated ρ-values in the graph below. How does the response time change with the system load?

12. For $\rho = 0.25$ and 0.9, plot elements 1-100 of the T-vector as a function of elements 2-101 by writing plt.plot(T[2:101],T[1:100],'*') in the command prompt.

$\rho = 0.25$

$\rho = 0.9$

13. How is the dependency between consecutive response times influenced by the system load?

The accuracy of the measurements can be improved if the measurements are not started until the system has settled. Then, the first measurements, which perhaps are not really representative for the system (very few customers), are discarded. This is especially important for systems where the initial states are very unusual.

14. Which is the initial state and most usual state for an M/M/1-system with $\rho < 1$? Hint: Use the steady state probabilities when determining the most usual state.

15. How long do we have to wait until the system has settled for an M/M/1 system with $\rho < 1$ and $\rho \geq 1$, respectively? (**)

4 Laboratory assignment: M/G/1- infinite queue

The time between consecutive arrivals and the service time can sometimes not be approximated as exponentially distributed. Now, we will compare the behavior of a system with one server, infinite queue and exponential interarrival times, but with different service time distributions. Deterministic, uniform, exponential and hyperexponential distributions will be investigated.

Simulate the system for the service time distributions using $\lambda = 7$ s^{-1}, a mean service time of 0.1 seconds and a simulation length of 1000 seconds. Hence, the original parameter settings shall be used. Hint: change the service time distribution in queue.serviceTime().

1. Find the mean number of jobs and the mean response times for the different service time distributions in the table below. For an example of the hyperexponential distribution see page 118 in the book. Use the parameters $\alpha = 0.25, \mu_1 = 100$ and $\mu_2 = 1/0.13$

Service time distribution	Kendall's Notation	$E[N]$	T
Deterministic	M/D/1		
Uniform	M/U/1		
Exponential	M/M/1		
Hyper exponential	M/H2/1		

2. How does the variance for the service time seem to influence these mean values ($E[N]$ and T). Hint: The distributions in the table are displayed in ascending order according to the service time variance.

3. For which λ-value do the systems in the previous task become unstable? What is the corresponding λ-value for a G/G/m system? This exercise shall be solved through theoretical reasoning. (**)

In most systems, it is not only the mean response time that is considered to be important but also how many jobs in percent that have a response time less/more than Z seconds. A common criteria used for dimensioning is to not allow more than X percent of the jobs to have a response time longer than Z seconds.

5 Laboratory assignment: M/M/1*limited queue

Set λ=$7s^{-1}$ and limit the buffer to 6 positions by altering the python-file MM1.py. Hint: Insert an admission control feature (with an if) in the arrival event that investigates if the queue is full.

1. How many jobs can now coexist in the system?

2. Determine the probability that a customer is blocked from the simulation results.

3. Determine the probability that the system is full from the simulation result. Hint: Use the histogram function to obtain an estimation of the steady state probabilities.

4. Calculate the theoretical values for the call congestion and the probability the system is full by using the program in pkMM1b.py Increase ρ to 1.1 by increasing λ.

5. How does the load influence the system stability? Compare with the results obtained for a system with infinite queue. (**)