UNIVERSITY OF EXETER

Jagadeesh Prasad Batchu(720044113), MSc Advanced Computer Science

ECMM424 Computer Modelling and Simulation (Coursework)

Report

M/M/C/C queueing system(multi-server queueing system without queuing space) is a system where the capacity of the system is limited and equal to the number of servers, and there is no waiting positions in the queue. If the servers are full then the job arrival is denied into the servers and it is lost without any delay. It is also known as m-Server Loss Systems.

Code Design for M/M/C/C queueing system.

Import : The code is written in python using Jupyter Notebook. In the first step all the necessary libraries are imported and on the top of the code all libraries which are used are mentioned.

```
import numpy as np
import random
import math
import matplotlib.pyplot as plt
import pandas as pd
```

Numpy is used to create numpy arrays to the servers, status and events. Random is used to generate the random number during the exponential process. Math is used to create infinite values and to find factorials. Matplotlib to plot the graph at the end and Pandas to create a data frame for an effective representation of the report.

Initialising: Class initialisation is done by passing the customers number, arrival rate and service time so that it can be changed regarding the convenience of the problem.

```
class MMCC():
    def __init__(self, total_calls, mean_arrival_time, mean_service_time):
        #Initializing all the parameters that are required
        self.mean_arrival = 1/mean_arrival_time
        self.mean_service = mean_service_time
        self.sim_time = 0.0
        self.C = 16
        self.num_event = self.C + 1
        self.num_calls = 0
        self.num_calls_required = total_calls
        self.next_event_type = 0
        self.server_status =  np.zeros(self.C + 1)
        self.area_server_status = np.zeros(self.C)
        self.time_next_event = np.zeros(self.C + 1)
        self.time_next_event[0] = self.sim_time + self.expon(self.mean_arrival)
        for i in range(1,self.C+1):
            self.time_next_event[i] = math.inf
        self.server_idle = 0
        self.server_utilization = np.zeros(self.C)
        self.total_server_utilization = 0
        self.total_loss = 0
```

All necessary variables are declared here which can be seen in the above picture. C is the number of Servers.

Main Function: After Init function there is a main function inside the class which will run all the functions in the class including the report.

```
#Main method which runs the entire Program
def main(self):
    while (self.num_calls < self.num_calls_required):
        self.timing()
        self.update_time_avg_stats()
        if (self.next_event_type == 0):
            self.arrive()
        else:
            self.j=self.next_event_type
            self.depart()
    self.report()
```

Here, based on the condition(number of calls should be less than total number of calls) the loop runs and at the end it will generate the report.

Timing : It is the first function inside the main function, where the number of events is number of servers + 1 and the simulation time is updated and the time of the last event based on the next event type value which comes from the loop based on the condition where the minimum time of next event is considered.

```
def timing(self):
    self.min_time_next_event = math.inf
    for i in range(0, self.num_event):
        if (self.time_next_event[i] <= self.min_time_next_event):
            self.min_time_next_event=self.time_next_event[i]
            self.next_event_type=i

    self.time_last_event=self.sim_time
    self.sim_time=self.time_next_event[self.next_event_type]
```

Update time : The next function inside main function is Update time average stats, here the time area server status is updated with the past time if the server is busy.

```
def update_time_avg_stats(self):
    self.time_past=self.sim_time-self.time_last_event
    for i in range(1,self.C + 1):
        self.area_server_status[i-1]+=self.time_past*self.server_status[i]
```

Arrive: If the next event type is 0 then this function is implemented. In this function the while loop iterates to find an idle server, if no idle server is found then it becomes loss and the loss value is incremented by one. If it finds the idle server then the time of next event is updated with addition of simulation time and exponential value of mean service time. At the end the number of calls is updated.

```
def arrive(self):
    i = 0
    self.server_idle = 0
    self.time_next_event[0] = self.sim_time + self.expon(self.mean_arrival)

    while (self.server_idle == 0 and i <= self.C):
        if (self.server_status[i] == 0):
            self.server_idle = i
        i += 1

    if (self.server_idle != 0):  #Any of the server is idle
        self.server_status [self.server_idle] = 1
        self.time_next_event[self.server_idle] =self.sim_time + self.expon(self.mean_service)

    else:    #All Servers are busy
        self.total_loss +=1
    self.num_calls+=1
```

Depart: If the next event type is not equal to 0 then this function is implemented where it will update the server status of the next event type to 0 which is idle and the time of next event to maximum.

```
def depart(self):
    self.server_status [self.j] = 0
    self.time_next_event [self.j] = math.inf
```

Exponential: This function the exponential value of the mean service and the mean arrival is calculated and returned.

```
def expon(self,mean):
    return (-1 * mean * math.log(random.random()))
```

Report: After completing all the calls the loops terminates and enters into report function here the report is generated. Here the report is divided into two types, Simulation and Formulated Report. Where Simulated Report gives Total Server Utilisation and Blocking probability values by calculating the data which came from simulation, and Formulated Report Total Server Utilisation and Blocking probability values by calculating the formula associated with.

```
def report(self):
    #Simulated values of Server Utilization and Bolocking Probability
    for i in range(0,self.C):
        self.server_utilization[i] = self.area_server_status[i] / self.sim_time
        self.total_server_utilization += self.area_server_status[i]
    self.total_server_utilization = self.total_server_utilization/(self.sim_time * self.C)
    Simulated_total_server_utilization.append(self.total_server_utilization)

    Blocking_Probability = self.total_loss/self.num_customers_required
    Simulated_Blocking_probabilty.append(Blocking_Probability)

    #Formulated values of Server Utilization and Bolocking Probability
    μ = 1/self.mean_service
    Service.append(μ)
    λ = 1/self.mean_arrival
    Lamda.append(λ)

    Denomenator = 0
    for j in range (0,self.C+1):
        Denomenator += ((λ/μ)**j)/(math.factorial(j))
    BP = (((λ/μ)**self.C)/(math.factorial(self.C)))/Denomenator #BP = Blocking Pobability
    SU = (λ/(self.C*μ)) * (1 - BP) #SU = Server Utilisation
    Formulated_Blocking_probabilty.append(BP)
    Formulated_total_server_utilization.append(SU)
```

The calculations made are seen in the above code.

Calling: Here outside the class the global and function calling is taking place, where 10,000,000 calls are used and the mean interval time starts with 0.01 up to 0.1 inside the loop.

```
Formulated_total_server_utilization = []
Simulated_total_server_utilization  = []
Formulated_Blocking_probabilty = []
Simulated_Blocking_probabilty = []
Lamda = []
Service = []
mean_interarrival_time = 0.01
x = []
for i in range(10):
    x.append(mean_interarrival_time)
    myObject = MMCC(total_customers = 10_000_000, mean_arrival_time = mean_interarrival_time, mean_service_time = 100)
    myObject.main()
    mean_interarrival_time += 0.01
```

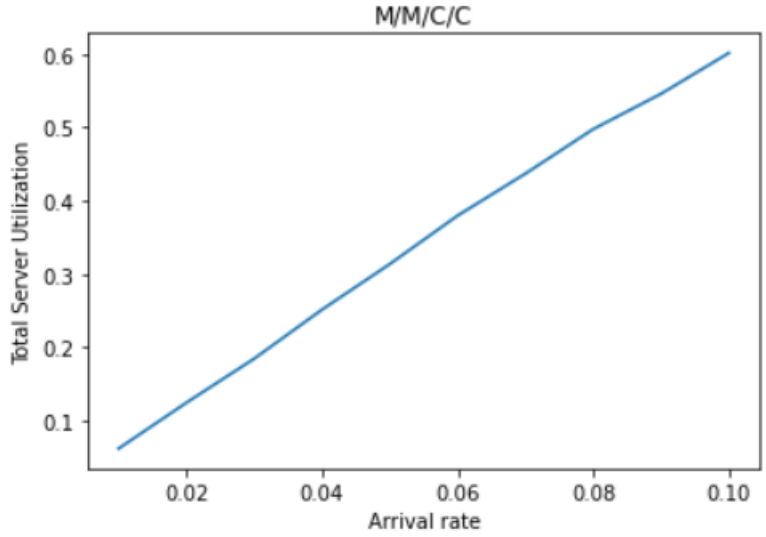
Actual Report: The report values for the arrival rate (0.01- 0.1 call/second) is shown below with respect to values from simulation and formulation.

FSU = Formulated Server Utilisation SSU = Simulated Server Utilisation

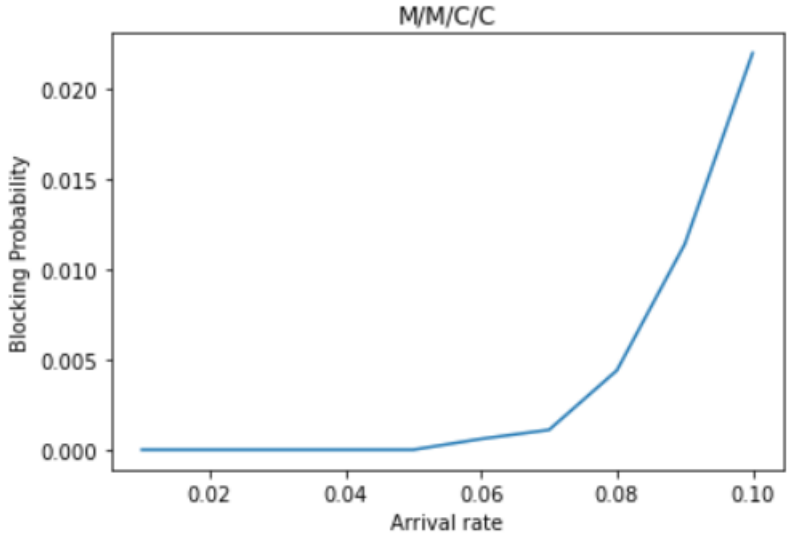
FBP = Formulated Blocking Probability SBP = Simulated Blocking Probability

	μ	λ	FSU	SSU	FBP	SBP
0	0.01	0.01	0.062500	0.062222	1.758271e-14	0.0000
1	0.01	0.02	0.125000	0.124756	4.239078e-10	0.0000
2	0.01	0.03	0.187500	0.184818	1.024323e-07	0.0000
3	0.01	0.04	0.249999	0.251677	3.759783e-06	0.0000
4	0.01	0.05	0.312485	0.313857	4.914017e-05	0.0000
5	0.01	0.06	0.374875	0.379890	3.342793e-04	0.0006
6	0.01	0.07	0.436866	0.437135	1.449786e-03	0.0011
7	0.01	0.08	0.497735	0.497816	4.529832e-03	0.0044
8	0.01	0.09	0.556283	0.545996	1.105250e-02	0.0114
9	0.01	0.10	0.611061	0.601648	2.230187e-02	0.0220

Server Utilisation: The Graphical representation of Server Utilisation based on the Arrival rate is shown below. From the graph with increasing arrival rate the server utilisation is also increasing.



Blocking Probability: The Graphical representation of Blocking Probability based on the Arrival rate is shown below. From the graph with increasing arrival rate the blocking probability is also increasing. And the probability is steady until 0.07 arrival rate and then the graph spiked suddenly.



The maximum value for input rates so that the Blocking Probability < 0.01 is at 0.088 – 0.089 from the above report it is seen. The Blocking probability is more than 0.01 from the arrival rate of 0.89 and more.

Code Design for M1+M2/M/C/C queueing system.

Most of the code is similar to the code of M/M/C/C Queueing system.

Import: Necessary libraries are imported.

```
import math
import numpy as np
import random
```

Initialise: Here the arrival rate of New calls and Handover calls is mentioed

```
class MyClass():
    def __init__(self, total_calls, mean_arrival_time_New, mean_arrival_time_Handover, mean_service_time):
        self.mean_arrival_time_Newcalls = 1/mean_arrival_time_New
        self.mean_arrival_time_Handover = 1/mean_arrival_time_Handover
        self.threshold = 2
        self.mean_service = mean_service_time
        self.C = 16
        self.sim_time = 0
        self.num_event = self.C + 2
        self.num_calls = 0
        self.num_calls_NewCalls = 0
        self.num_calls_HandoverCalls = 0
        self.num_calls_required = total_calls
        self.server_status = np.zeros(self.C + 1)
        self.area_server_status = np.zeros(self.C)
        self.time_next_event = np.zeros(self.C + 2)
        self.time_next_event[0] = self.sim_time + self.expon(self.mean_arrival_time_Newcalls)
        self.time_next_event[self.C + 1] = self.sim_time + self.expon(self.mean_arrival_time_Handover)
        if self.time_next_event[0] < self.time_next_event[self.C + 1]:
            self.next_event_type=0
        else:
            self.next_event_type = self.C + 1
        for i in range(1, self.C + 1):
            self.time_next_event[i] = math.inf;
        self.server_idle = 0
        self.server_utilization = np.zeros(self.C)
        self.total_server_utilization = 0
        self.Total_Loss = 0
        self.Total_Loss_NewCalls = 0
        self.Total_Loss_HandoverCalls = 0
```

Main: This Function runs all the functions in the class until the condition satisfies(i.e., running the required number of calls) then it enter into report function and report is generated.

```
    def main(self):
        while ((self.num_calls_NewCalls + self.num_calls_HandoverCalls) < self.num_calls_required):
            self.timing()
            self.update_time_avg_stats()
            if (self.next_event_type == 0):
                self.arrive_NewCalls()
            elif (self.next_event_type == (self.C + 1)):
                self.arrive_HandoverCalls()
            else:
                self.j = self.next_event_type
                self.depart()
        self.report()
```

Timing : This function is same as the above system.

```
def timing(self):
    self.min_time_next_event = math.inf
    for i in range(0, self.num_event):
        if (self.time_next_event[i] <= self.min_time_next_event):
            self.min_time_next_event = self.time_next_event[i]
            self.next_event_type = i

    self.time_last_event=self.sim_time
    self.sim_time = self.time_next_event[self.next_event_type]
```

Update Time:

```
def update_time_avg_stats(self):
    self.time_past = self.sim_time - self.time_last_event
    for i in range(1, self.C + 1):
        self.area_server_status[i-1] += self.time_past * self.server_status[i]
```

New calls : Here the time of next event is updated if the server is idle or else it goes into the loss.

```
def arrive_NewCalls(self):
    i = 0
    self.server_idle = 0
    ##Schedule next arrival
    self.time_next_event[0] = self.sim_time + self.expon(self.mean_arrival_time_Newcalls)
    while (self.server_idle == 0 and i <= self.C):
        if (self.server_status[i] == 0):
            self.server_idle = i
        i += 1
    if (self.server_idle != 0): ## Someone is IDLE
        self.server_status [self.server_idle] = 1
        self.time_next_event[self.server_idle] = self.sim_time + self.expon(self.mean_service)
    else:               ## server is BUSY
        self.Total_Loss_NewCalls += 1
    self.num_customers_NewCalls += 1
```

Handover calls: Here the calls can enter up to the servers excluding the threshold.

```
def arrive_HandoverCalls(self):
    i = 0
    self.server_idle = 0
    ##Schedule next arrival
    self.time_next_event[self.C + 1] = self.sim_time + self.expon(self.mean_arrival_time_Handover)
    while (self.server_idle == 0 and i <= (self.C - self.threshold)):
        if (self.server_status[i] == 0):
            self.server_idle = i
        i += 1
    if (self.server_idle != 0): ## Someone is IDLE
        self.server_status [self.server_idle] = 1
        self.time_next_event[self.server_idle] = self.sim_time + self.expon(self.mean_service)
    else:               ## server is BUSY
        self.Total_Loss_HandoverCalls += 1
    self.num_customers_HandoverCalls += 1
```

Depart:

```
def depart(self):
    self.server_status [self.j] = 0
    self.time_next_event [self.j] = math.inf
```

Exponential:

```
def expon(self, mean):
    return (-1 * mean * math.log(random.random()))
```

Simulation Report: The report is generated based on the values than are generated in the simulation.

```
def report(self):
    for i in range(0, self.C):
        self.server_utilization[i] = self.area_server_status[i]/self.sim_time
        self.total_server_utilization += self.area_server_status[i]
    self.total_server_utilization = self.total_server_utilization/(self.sim_time * self.C)

    print('----------------------------Simulation Report --------------------')
    μ = 1/self.mean_service
    λ1 = 1/self.mean_arrival_time_Newcalls
    λ2 = 1/self.mean_arrival_time_Handover
    SCBP = self.Total_Loss_NewCalls / self.num_customers_NewCalls
    SHFP = self.Total_Loss_HandoverCalls / self.num_customers_HandoverCalls
    SABP = SCBP + 10 * SHFP

    print('                                λ1 = ',λ1)
    print('                                λ2 = ',λ2)
    print('                                 μ = ',μ)
    print('     Loss Probability of New Calls = ', SCBP)
    print('Loss Probability of Handover Calls = ', SHFP)
    print('   Aggregated Blocking Probability = ', SABP)
    print('          Total_server_utilization = ',self.total_server_utilization)
```

Formulated Report: To generate the report based on the formula and inputs.

```
One = 0
Two = 0
C = self.C
N = self.threshold
for i in range(0, C-N+1):
    One +=  (( 1/math.factorial(i)) * (((λ1 + λ2)/μ)**i))

for i in range(C-N+1, C+1):
    Two +=  (( 1/math.factorial(i)) * (((λ1 + λ2)/μ)**(C-N)) * ((λ1/μ)**(i-C+N)))

p0 = 1/(One + Two)
pk_NewCalls = One * p0
Pk_handOver = Two * p0
print(' ------------  -----------Formulated Values -------------------------')
print('                     pk_NewCalls = ',pk_NewCalls)
print('                     Pk_handOver = ',Pk_handOver)
print('                              p0 = ',p0)
```

Function calling: In the Function calling the values can be changed based on the requirements.

```
myObject = MyClass(total_customers = 10_000_000, mean_arrival_time_New = 0.01,
                   mean_arrival_time_Handover = 0.03, mean_service_time = 100)
myObject.main()
```

Maximum value for handover call rates where ABP < 0.02:

By Setting the new calls rate to 0.1 call/second. The maximum value for the handover call rate so that the ABP < 0.02 is not achieved. The lowest ABP value that is achieved is 0.6 at 0.00001 handover call rate. Because of the fixed new calls rates 0.1 which is high and it will affect the handover failure probability which directly effects the Aggregated Blocking Probability.

```
myObject = MyClass(total_calls = 10_000_000, mean_arrival_time_New = 0.1,
                   mean_arrival_time_Handover = 0.00001, mean_service_time = 100)
myObject.main()
```

```
----------------------------Simulation Report --------------------
                              λ1 =  0.1
                              λ2 =  1e-05
                               μ =  0.01
     Loss Probability of New Calls =  0.02234708331003939
Loss Probability of Handover Calls =  0.062436028659160696
   Aggregated Blocking Probability =  0.6467073699016463
          Total_server_utilization =  0.6113588997716187
 ------------  -----------Formulated Values -------------------------
                    pk_NewCalls =  0.9419901781120433
                    Pk_handOver =  0.05800982188795669
                             p0 =  4.661651424213775e-05
```

Maximum value for new call rates where ABP < 0.02:

By Setting the handover rate to 0.03 handover/second. The maximum value for the new call rate so that the ABP < 0.02 is achieved at the new call rate of 0.029.

Below the rate of 0.029 the value of Aggregated Blocking Probability is less than 0.02 and from 0.03 onwards the value increased from 0.02.

```
myObject = MyClass(total_customers = 10_000_000, mean_arrival_time_New = 0.029,
                   mean_arrival_time_Handover = 0.03, mean_service_time = 100)
myObject.main()
```

```
----------------------------Simulation Report --------------------
                              λ1 =  0.029000000000000005
                              λ2 =  0.03
                               μ =  0.01
     Loss Probability of New Calls =  9.326787204869805e-05
Loss Probability of Handover Calls =  0.0019685963784035606
   Aggregated Blocking Probability =  0.019779231656084306
          Total_server_utilization =  0.3684286758742337
 ------------  -----------Formulated Values -------------------------
                    pk_NewCalls =  0.9995552093455028
                    Pk_handOver =  0.0004447906544971919
                             p0 =  0.0027414938907327855
```