Group Members:
Aditya Shah
Deepal Rathod
Krunal Kothari
Mayuri More
Rohan Desai
Tanvi Zunjarrao
Vidhi Shah
Vidhi Rajesh Chitalia

DISCRETE EVENT SIMULATION

A) INTRODUCTION

SIMULATION

Many systems are highly complex, precluding the possibility of analytical solution the analytical solutions are extraordinarily complex, requiring vast computing resources Thus, such systems should be studied by means of simulation.

Simulations are of great importance as they prevent the catastrophic failures in the system due to impact of a change. New changes, procedures, information flows etc. can be examined without interrupting the smooth working of real systems. A simulation model is developed to study the working of a system as it evolves over time. A fully developed and validated model can answer a variety of questions about real systems.

"Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of a system."
-Robert E Shannon 1975

DISCRETE EVENT SIMULATION

Discrete-Event Simulation is the one which models the operation of a system as a discrete sequence of events in time. Each event occurs at a instant in time and marks a change of state in the system. Between consecutive events, no change is assumed to occur; thus, the simulation can directly jump in time from one event to the next.

Advantages of Simulation:

a) it permits controlled experimentation.
   you KNOW what parameters are being changed.
b) it permits time compression.
   e.g., weather forecasting...
c) it permits sensitivity analysis (change input vars)
d) it does not disturb the real system.
   which may not even exist, anyway.
e) it is an effective training tool.
   you are not likely to crash a flight simulator,
   or a big chunk of the Internet.

B) APPLICATION

1 – HEALTH CARE APPLICATIONS

Discrete event simulation (DES) is a method of simulating the behavior and performance of a real-life process, facility or system. DES is being used increasingly in health-care services and the increasing speed and memory of computers has allowed the technique to be applied to problems of increasing size and complexity. DES models the system as a series of 'events' (e.g. a birth, a stay in an intensive care unit (ICU), a transfer or a discharge) that occur over time. DES assumes no change in the system between events. In DES, patients are modeled as independent entities each of which can be given associated attribute information. In the case of neonatal simulation this may include parameters such as gestational age or weight at birth, hospital of birth, singleton/twin and current location. The information may be modified as time runs in the simulation model (e.g. the location will be changed depending on the status of the units in the network, and the level of care being received will be modified as the infant progresses). The simulation also accounts for resources. In the neonatal model the key resources are cots (with the highest level of care for each cot specified) and nurses. In order to care for an infant a unit must have the necessary cot and the necessary

nursing staff (applying appropriate guidelines). The model allows each unit to work to a specified level of overcapacity regarding nursing but will monitor the time each unit is undergoing overcapacity. DES models also allow for complex rules specifying where infants may be accepted; for example, there may be two ICUs, but with different facilities (e.g. surgery) or with different limits on gestational ages. DES thus allows complex decision logic to be incorporated that is not as readily possible in other types of modeling.

2 – CHEMICAL INDUSTRY APPLICATIONS

Application of discrete-event simulation to study logistics activities in a chemical plant. Most chemical production involves continuous flow of materials, such as liquid, gas or solid through the manufacturing and logistics processes. Some simulation issues in this area are conceptualizing production operations for simulation, discretization of continuous processes and building adequate level of detail in the models. DES helps to determine the required capacity of logistics operations to allow continuous operations of a chemical manufacturing plant.

3 – SUPPLY CHAIN MANAGEMENT

Simulation could be an effective technique to help analyze supply chain issues, because it can be applied to problems that are too difficult to model and solve analytically. Many analytical models only take a few variables into consideration. This simplification could result in a model that is not a realistic representation of the system. Simulating a supply chain is complex because a model must include several processes. However, to ignore some processes could lead the simulation model to ignore critical functions and activities that affect the performance being measured. To define the processes involved and develop a model that provides the user with a holistic view of the supply chain, thus preventing sub-optimization, is thus a fundamental step in simulating supply chains.

DES was used for analyzing how the process would respond when wireless identification technology was applied in the retail supply chain. A DES model consisting of a producer, DC, a retail outlet and consumers was developed. The model of the retail supply chain represents those processes behaviors that were considered important. The order process was identified Producers Manufacturers Distribution

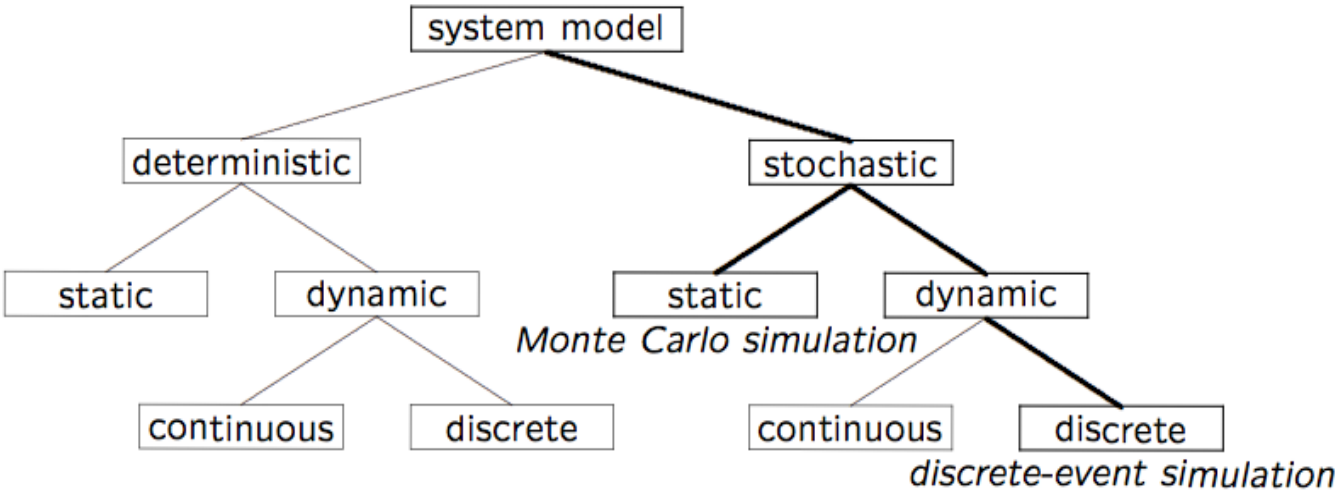
Centers Retail Outlets Consumers Producers Manufacturers Distribution Centers Retail Outlets Consumers 8 as the focus of the study, so this process has a central role in the DES model. The performance variables measured to analyze different scenario results were e.g. variables that control:

- Material handling efficiency
- Consumer service
- Inventory levels
- Lead times

The number of actors involved in the case study reflects directly on the complexity of the assignment to model the retail supply chain, and therefore the size of the problem. DES was identified as a technique for solving this complex problem, since solving it analytically was not viable. DES handled both stochastic and dynamic behaviors, which is the basic problem when the objective is to analyze how a supply chain design behaves and performs over time when different rules and policies are used.

C) THEORY

MODEL TAXONOMY



Form above diagram we get to know the following:

- Discrete-Event Simulation Model

  - Stochastic: some state variables are random
  - Dynamic: time evolution is important
  - Discrete-Event: significant changes occur at discrete time instances
- Monte Carlo Simulation Model
  - Stochastic
  - Static: time evolution is not important

Implementation of Discrete Event Simulation

Operationally, a discrete-event simulation is a chronologically nondecreasing sequence of event occurrences.

event record:

a pairing of an event with its event time

future event list (FEL) (or just *event list*):

a list ordered by nondecreasing simulation time (e.g., in a priority queue)

event (list) driven simulation:

simulation where time is advanced to the time given by the first event record from the event list

Components of discrete-event simulation:

1) System state - The collection of state variables necessary to describe the system at a particular time
2) Simulation clock - A variable giving the current value of simulated time
3) Event list - A list containing the next time when each type of event will occur
4) Statistical counters - Variables used for storing statistical information about system information
5) Initialization routine - A subprogram to initialize the simulation model at time 0
6) Timing routine - A subprogram that determines the next event from the event list and then advances the simulation clock to the time when that event is to occur

---

Approaches to Discrete Event Simulation

Three general approaches:

1. activity scanning
2. event scheduling
3. process interaction

Activity Scanning

- Basic building block is the *activity*
- Model program's code segments consist of sequences of activities (operations) waiting to be executed
- An activity sequence's conditions must be satisfied before it is executed
- Simulation executive moves from event to event executing those activity sequences whose conditions are satisfied

---

Event Scheduling

- Basic building block is the *event*
- Model program's code segments consist of event routines waiting to be executed
- Event routine associated with each event type -- performs required operations for that type
- Simulation executive moves from event to event executing the corresponding event routines

---

Process Interaction

- Basic building block is the *process*
- System consists of a set of interacting processes
- Model program's code for each process includes the operations that it carries out throughout its lifetime
- Event sequence for system consists of merging of event sequences for all processes
- Future event list consists of a sequence of *event nodes* (or *notices*)
- Each event node indicates the event time and process to which it belongs
- Simulation executive carries out the following tasks:
    - placement of processes at particular points of time in the list

- removal of processes from the event list
- activation of the process corresponding to the next event node from the event list
- rescheduling of processes in the event list

- Typically, a process object can be in one of several states:
  - *active* -- process is currently executing
    There is only one such process in a system.
  - *ready* -- process is in event list, waiting for activation at a certain time
  - *idle* (or *blocked*) -- process is not in event list, but eligible to be be reactivated by some other entity (e.g., waiting for a passive resource)
  - *terminated* -- process has completed its sequence of actions, is not in event list, and cannot be reactivated

---

D) ALGORITHM

A/B/c/N/K
A represents the interarrival time distribution
B represents the service-time distribution
c represents number of parallel servers
N represents the system capacity
K represents the size of the calling population

Common types of distr. for A/B:
M: represents exponential or Markov (more about this later)
D: deterministic/constant (not random)
Ek: Erlang of order k
G: general (arbitrary)

M/M/1 QUEUE :

In queueing theory, a discipline within the mathematical theory of probability, an **M/M/1 queue** represents the queue length in a system having a single server, where arrivals are determined by a Poisson process and job service times have an exponential distribution. The model name is written in Kendall's notation. The model is the most elementary of queueing models[1] and an attractive object of study as closed-form expressions can be obtained for many metrics of interest in this model. An extension of this model with more than one server is the M/M/c queue.

An M/M/1 queue is a stochastic process whose state space is the set {0,1,2,3,...} where the value corresponds to the number of customers in the system, including any currently in service.

- Arrivals occur at rate λ according to a Poisson process and move the process from state *i* to *i* + 1.
- Service times have an exponential distribution with rate parameter μ in the M/M/1 queue, where 1/μ is the mean service time.
- A single server serves customers one at a time from the front of the queue, according to a first-come, first-served discipline. When the service is complete the customer leaves the queue and the number of customers in the system reduces by one.
- The buffer is of infinite size, so there is no limit on the number of customers it can contain.

The model can be described as a continuous time Markov chain with transition rate matrix

$$Q = \begin{pmatrix} -\lambda & \lambda & & & \\ \mu & -(\mu+\lambda) & \lambda & & \\ & \mu & -(\mu+\lambda) & \lambda & \\ & & \mu & -(\mu+\lambda) & \lambda \\ & & & & \ddots \end{pmatrix}$$

on the state space {0,1,2,3,...}. This is the same continuous time Markov chain as in a birth–death process. The state space diagram for this chain is as below.

We can write a probability mass function dependent on *t* to describe the probability that the M/M/1 queue is in a particular state at a given time. We assume that the queue is initially in state *i* and write $p_k(t)$ for the probability of being in state *k* at time *t*. Then[2]

$$p_k(t) = e^{-(\lambda+\mu)t}\left[\rho^{\frac{k-i}{2}} I_{k-i}(at) + \rho^{\frac{k-i-1}{2}} I_{k+i+1}(at) + (1-\rho)\rho^k \sum_{j=k+i+2}^{\infty} \rho^{-j/2} I_j(at)\right]$$

The model is considered stable only if λ < μ. If, on average, arrivals happen faster than service completions the queue will grow indefinitely long and the system will not have a stationary distribution. The stationary distribution is the limiting distribution for large values of *t*.

Various performance measures can be computed explicitly for the M/M/1 queue. We write ρ = λ/μ for the utilization of the buffer and require ρ < 1 for the queue to be stable. ρ represents the average proportion of time which the server is occupied.

Average number of customers in the system

The probability that the stationary process is in state *I* contains *i* customers, including those in service

$$\pi_i = (1-\rho)\rho^i.$$

Busy period of server

The busy period is the time period measured between the instant a customer arrives to an empty system until the instant a customer departs leaving behind an empty system. The busy period has probability density function

$$f(t) = \begin{cases} \frac{1}{t\sqrt{\rho}} e^{-(\lambda+\mu)t} I_1(2t\sqrt{\lambda\mu}) & t > 0 \\ 0 & \text{otherwise} \end{cases}$$

where I_1 is a modified Bessel function of the first kind, obtained by using Laplace transforms and inverting the solution.

The Laplace transform of the M/M/1 busy period is given by

$$\mathbb{E}(e^{-\theta F}) = \frac{1}{2\lambda}(\lambda + \mu + \theta - \sqrt{(\lambda + \mu + \theta)^2 - 4\lambda\mu})$$

which gives the moments of the busy period, in particular the mean is $1/(\mu - \lambda)$ and variance is given by

$$\frac{1 + \frac{\lambda}{\mu}}{\mu^2(1 - \frac{\lambda}{\mu})^3}.$$

Response time

The average response time or sojourn time (total time a customer spends in the system) does not depend on scheduling discipline and can be computed using Little's law as $1/(\mu - \lambda)$. The average time spent waiting is $1/(\mu - \lambda) - 1/\mu = \rho/(\mu - \lambda)$. The distribution of response times experienced does depend on scheduling discipline.

First-come, first-served discipline

For customers who arrive and find the queue as a stationary process, the response time they experience (the sum of both waiting time and service time) has transform $(\mu - \lambda)/(s + \mu - \lambda)$ and therefore probability density function.

$$f(t) = \begin{cases} (\mu - \lambda)e^{-(\mu - \lambda)t} & t > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Processor sharing discipline

In an M/M/1-PS queue there is no waiting line and all jobs receive an equal proportion of the service capacity. Suppose the single server serves at rate 16 and there are 4 jobs in the system, each job will experience service at rate 4. The rate at which jobs receive service changes each time a job arrives at or departs from the system

For customers who arrive to find the queue as a stationary process, the Laplace transform of the distribution of response times experienced by customers was published in 1970,[17] for which an integral representation is known.[18] The waiting time distribution (response time less service time) for a customer requiring *x* amount of service has transform[4]:356

$$W^*(s|x) = \frac{(1-\rho)(1-\rho r^2)e^{-[\lambda(1-r)+s]x}}{(1-\rho r^2)-\rho(1-r)^2 e^{-(\mu/r-\lambda r)x}}$$

where *r* is the smaller root of the equation

$$\lambda r^2 - (\lambda + \mu + s)r + \mu = 0.$$

The mean response time for a job arriving and requiring amount *x* of service can therefore be computed as $x\,\mu/(\mu - \lambda)$. An alternative approach computes the same results using a spectral expansion method.

M/M/C QUEUE:

M/M/C queue system is a classical example of queueing theory and traffic theory. The system consists of a single common queue with FIFO discipline and **C** servers. Entities are arrived as a Poisson process. Queue capacity and timeout (maximum waiting time) are infinite. Service time value is exponentially distributed. The free server selection rule has a random selection.

There is no real system where the number of waiting places is infinite, or client can wait for service indefinitely. Therefore, the M/M/C/K,T queue system with a certain capacity and limited timeout should to be used instead. Such a model can be used to simulate a large number of simple systems like barbershop, easy call center etc.

It should be noted that for the practical purposes the classical M/M/C queue system cannot be used. There is no real system where the number of waiting places is infinite, or client can wait for service indefinitely. Therefore, the M/M/C/K,T queue system with a certain capacity and limited timeout should to be used instead. Such a model can be used to simulate a large number of simple systems like barbershop, easy call center etc.

Model parameters

- **Capacity** is the total number of entities for simulation.
- **NumberOfServers** is the number of servers.
- **InterArrivalTime** is mean value between entities arrivals.
- **ServiceTime** is mean value of service time.
- **ExpTime** is a block function returning exponentially distributed random values.
- **RandomSelection** is the block parameter for selection rule.

E) NUMERICAL

1) A baker is trying to figure out how many dozens of bagels to bake each day. The probabiltity distribution of the number of bagel customers is as follows:

Number of Customer/Day	8	10	12	14
Probability	0.35	0.30	0.25	0.10

Customers order 1,2,3 or 4 dozen bagels according to the following probability distribution.

Number of dozen Ordered/Customer	1	2	3	4
Probability	0.4	0.3	0.2	0.1

Solution:

Profit = Revenue from retail sales - Cost of bagels made + Revenue from grocery store sales - Lost profit.

Let Q = number of dozens baked/day

$S = \sum_i 0_i$, where 0_i = Order quantity in dozens for the ith customer

$Q - S$ = grocery store sales in dozens, $Q > S$

$S - Q$ = dozens of excess demand, $S > Q$

Profit $= \$5.40\min(S,Q) - \$3.80Q + \$2.70(Q-S) - \$1.60(S-Q)$

Number of Customers	Probability	Cumulative Probability	RD Assignment
8	.35	.35	01-35
10	.30	.65	36-65
12	.25	.90	66-90
14	.10	1.00	91-100

Dozens Ordered	Probability	Cumulative Probability	RD Assignment
1	.4	.4	1-4
2	.3	.7	5-7
3	.2	.9	8-9
4	.1	1.0	0

Pre-analysis

$$\begin{aligned} E(\text{Number of Customers}) &= .35(8) + .30(10) + .25(12) + .10(14) \\ &= 10.20 \end{aligned}$$

$$E(\text{Dozens ordered}) = .4(1) + .3(2) + .2(3) + .1(4) = 2$$

$$E(\text{Dozens sold}) = \bar{S} = (10.20)(2) = 20.4$$

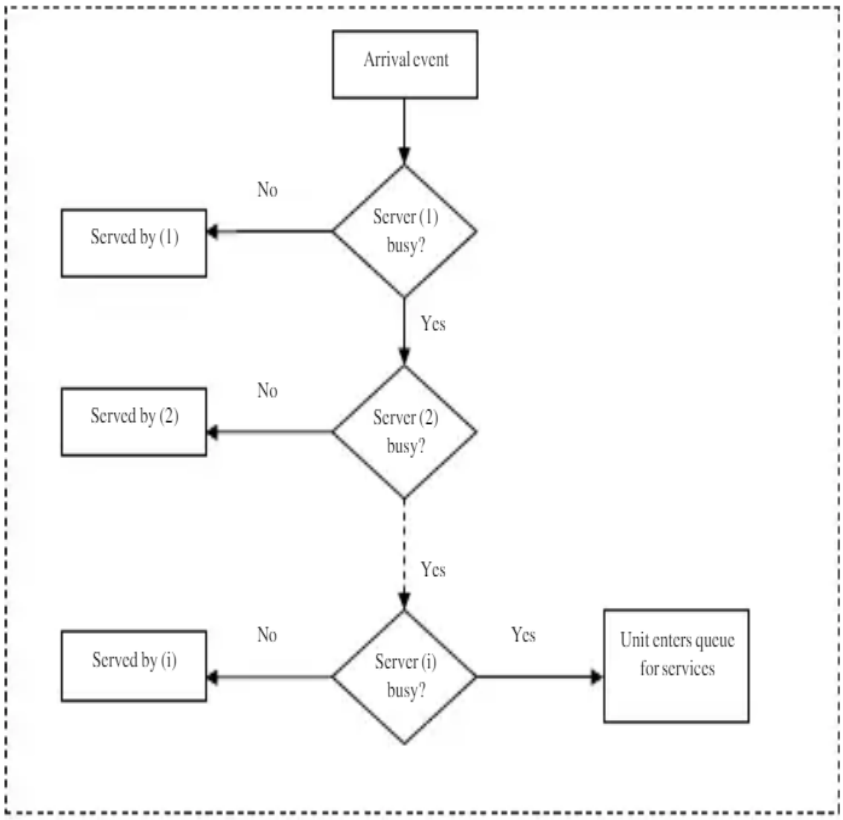
$$\begin{aligned} E(\text{Profit}) &= \$5.40\text{Min}(\bar{S},Q) - \$3.80Q + \$2.70(Q-\bar{S}) - \$1.60(\bar{S}-Q) \\ &= \$5.40\text{Min}(20.4,Q) - \$3.80Q + \$2.70(Q-20.4) \\ &\quad -\$0.67(20.4-Q) \end{aligned}$$

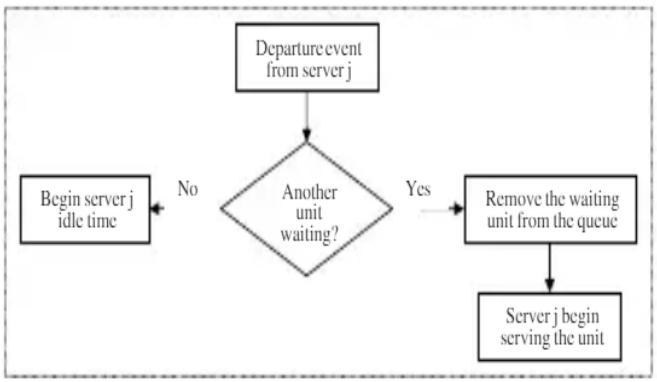
$$
\begin{aligned}
E(\text{Profit}|Q=0) &= 0-0+\$1.60(20.4) = -\$32.64 \\
E(\text{Profit}|Q=10) &= \$5.40(10)-\$3.80(10)+0-\$1.60(20.4-10) \\
&= -\$0.64 \\
E(\text{Profit}|Q=20) &= \$5.40(20)-\$3.80(20)+0-\$1.60(20.4-20) \\
&= \$15.36 \\
E(\text{Profit}|Q=30) &= \$5.40(20.4)-\$3.80(30)+\$2.70(30-20.4)-0 \\
&= \$22.08 \\
E(\text{Profit}|Q=40) &= \$5.40(20.4)-\$3.80(40)+\$2.70(40-20.4)-0 \\
&= \$11.08
\end{aligned}
$$

The pre-analysis, based on expectation only, indicates that simulation of the policies $Q = 20, 30$, and 40 should be sufficient to determine the policy. The simulation should begin with $Q = 30$, then proceed to $Q = 40$, then, most likely to $Q = 20$.

2) Develop and interpret flow diagrams analogous for queueing systems with i channels

For a queueing system with i channels, first rank all the servers by their processing rate. Let (1) denote the fastest server, (2) the second fastest server, and so on.





Initially, conduct a simulation for $Q = 20, 30$ and 40. If the profit is maximized when $Q = 30$, it will become the policy recommendation.

The problem requests that the simulation for each policy should run for 5 days. This is a very short run length to make a policy decision.

$Q = 30$

Day	RD for Customer	Number of Customers	RD for Demand	Dozens Ordered	Revenue from Retail $	Lost Profit $
1	44	10	8	3	16.20	0
			2	1	5.40	0
			4	1	5.40	0
			8	3	16.20	0
			1	1	5.40	0
			6	2	10.80	0
			3	1	5.40	0
			0	4	21.60	0
			2	1	5.40	0
			0	4	21.60	0
				21	113.40	0

For Day 1,

Profit = \$113.40 − \$152.00 + \$24.30 − 0 = \$14.30

Days 2, 3, 4 and 5 are now analyzed and the five day total profit is determined.

Problem Statement 1:

1. a) Imagine, you work for a multinational bank. You have the responsibility of setting up the entire call center process. You are expected to tie up with a call centre and tell them the number of servers you require.

You are setting up this call center for a specific feature queries of customers which has an influx of around 20 queries in an hour. Each query takes approximately 15 minutes to be resolved. Find out the number of servers/representatives you need to bring down the average waiting time to less than 30 seconds.

Source Code:

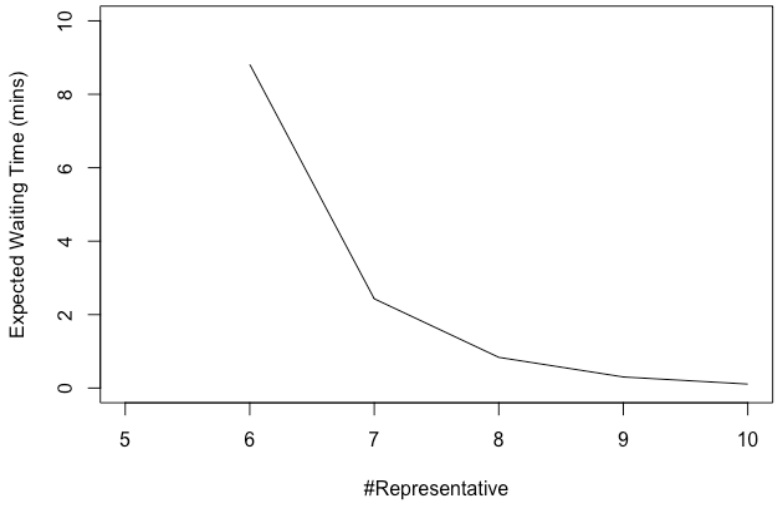
```
Lambda <- 20
Mue <- 4
Rho <- Lambda/Mue
#Create an empty matrix
matrix <- matrix(0,10,2)
matrix[,1] <- 1:10
#Create a function than can find the waiting time
calculatewq <- function(c){
 P0inv <- Rho^c/(factorial(c)*(1-(Rho/c)))
 for (i in 1:c-1) {
  P0inv = P0inv + (Rho^i)/factorial(i)
 }
 P0 = 1/P0inv
 Lq = (Rho^(c+1))*P0/(factorial(c-1)*(c-Rho)^2)
 Wq = 60*Lq/Lambda
 Ls <- Lq + Rho
 Ws <- 60*Ls/Lambda
 #print(paste(Lq," is queue length and ",Wq," is the waiting time"))
 a <- cbind(Lq,Wq,Ls,Ws)
 return(a)
}
#Now populate the matrix with each value of waiting time
for (i in 1:10){
 matrix[i,2] <- calculatewq(i)[2]
```

```
}


A= as.data.frame(matrix)
print(matrix)
row.names(A)<- 1:10
names(A)<- c("X","Y")
A
round(A,1)
plot(x= A$X, y= A$Y, xlab = "#Representative",
    ylab = "Expected Waiting Time (mins)",
    xlim = c(5,10),
    ylim = c(0,10),
    type="l")
```

Output:

```
    X     Y
1   1 -18.8
2   2 -17.9
3   3 -18.4
4   4 -24.1
5   5   NaN
6   6   8.8
7   7   2.4
8   8   0.8
9   9   0.3
10 10   0.1
```



1.b)

Imagine, you are the Operations officer of a Bank branch. Your branch can accommodate a maximum of 50 customers. How many tellers do you need if the number of customer coming in with a rate of 100 customer/hour and a teller resolves a query in 3 minutes ?

You need to make sure that you are able to accommodate more than 99.999% customers. This means only less than 0.001 % customer should go back without entering the branch because the brach already had 50 customers. Also make sure that the wait time is less than 30 seconds.

Source Code:

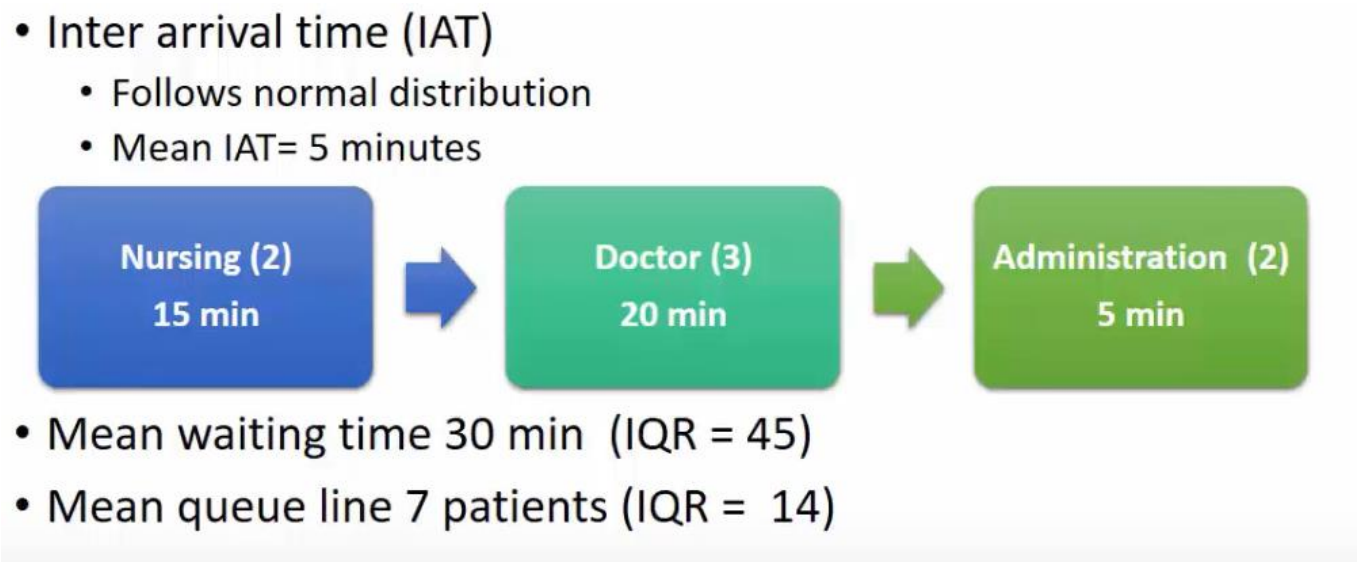
```
Lambda <- 100
Mue <- 20
Rho <- Lambda/Mue
N <- 50
matrix <- matrix(0,100,3)
matrix[,1] <- 1:100
calculatewq(6)
calculatewq <- function(c){
 P0inv <- (Rho^c*(1-((Rho/c)^(N-c+1))))/(factorial(c)*(1-(Rho/c)))
 for (i in 1:c-1) {
  P0inv = P0inv + (Rho^i)/factorial(i)
 }
 P0 = 1/P0inv
 Lq = (Rho^(c+1))*(1-((Rho/c)^(N-c+1))-((N-c+1)*(1-(Rho/c))*((Rho/c)^(N-c))))*P0/(factorial(c-1)*(c-Rho)^2)
 Wq = 60*Lq/Lambda
```

```
  Ls <- Lq + Rho
  Ws <- 60*Ls/Lambda
  PN <- (Rho^N)*P0/(factorial(c)*c^(N-c))
  customer_serviced <- (1 - PN)*100
  #print(paste(Lq," is queue length and ",Wq," is the waiting time"))
  a <- cbind(Lq,Wq,Ls,Ws,customer_serviced)
  return(a)
}
for (i in 1:100){
  matrix[i,2] <- calculatewq(i)[2]
  matrix[i,3] <- calculatewq(i)[5]
}
matrix
matrix=round(matrix,2)
A1= as.data.frame(matrix)
A=A1[c(1:15),c(1,2)]
A
names(A)<- c("X","Y")
A
plot(x= A$X, y= A$Y, xlab = "#Representative",
     ylab = "Avg. Waiting Time (mins)",
     type ="l")
```

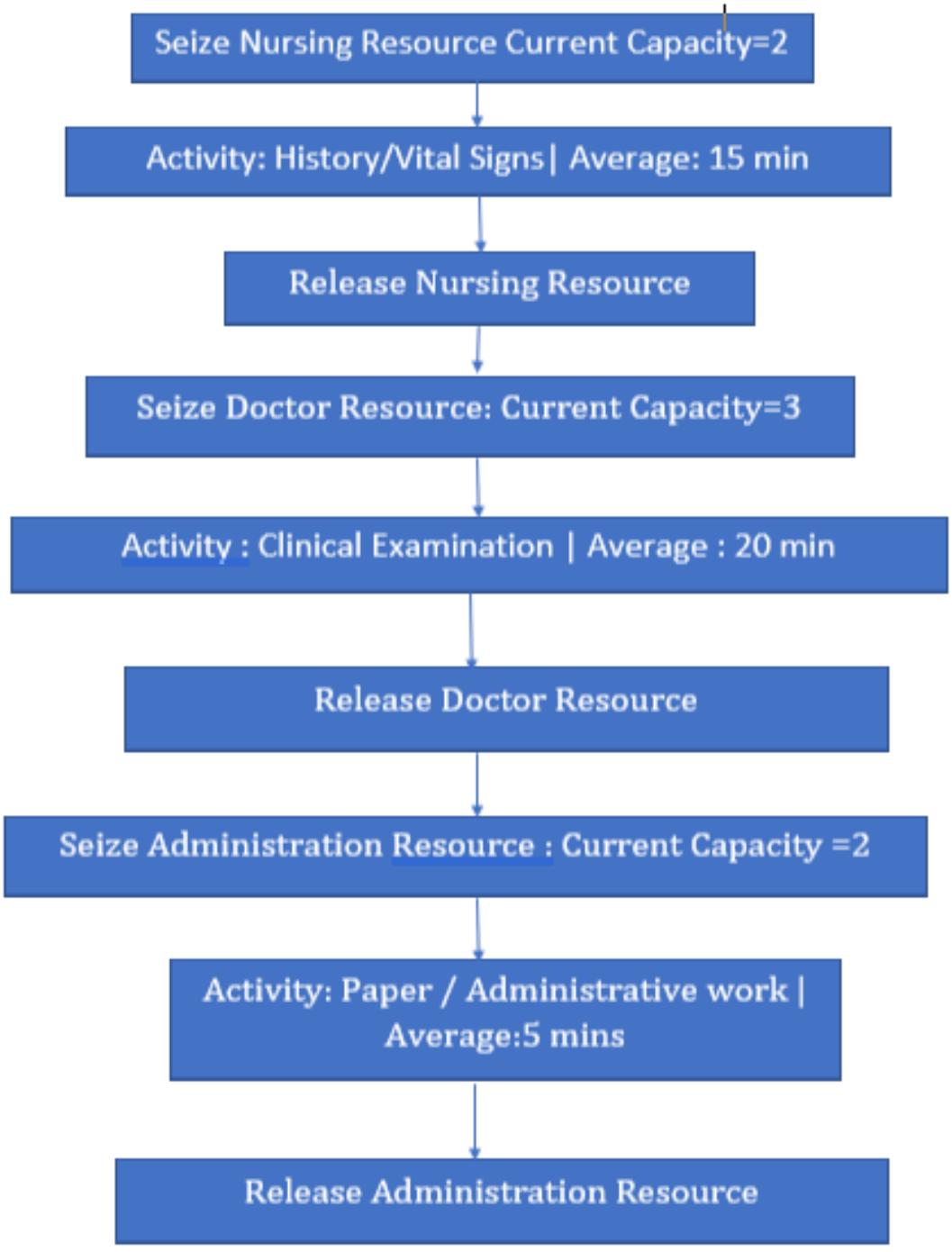
Output:

	#Representatice	Avg. Waiting Time (mins)	%Customers Attended
1	1	29.25	20
2	2	28.40	40
3	3	27.30	60
4	4	25.20	80
5	5	NaN	NaN
6	6	1.76	100
7	7	0.49	100
8	8	0.17	100
9	9	0.06	100
10	10	0.02	100
11	11	0.01	100
12	12	0.00	100
13	13	0.00	100
14	14	0.00	100
15	15	0.00	100
16	16	0.00	100
17	17	0.00	100
18	18	0.00	100
19	19	0.00	100
20	20	0.00	100
21	21	0.00	100
22	22	0.00	100
23	23	0.00	100
24	24	0.00	100
25	25	0.00	100
26	26	0.00	100
27	27	0.00	100
28	28	0.00	100
29	29	0.00	100
30	30	0.00	100
31	31	0.00	100
32	32	0.00	100
33	33	0.00	100
34	34	0.00	100
35	35	0.00	100
36	36	0.00	100
37	37	0.00	100
38	38	0.00	100
39	39	0.00	100
40	40	0.00	100
41	41	0.00	100
42	42	0.00	100

Problem Statement 2: An outpatient clinic example is shown where the patient is first seen by the nurse for an assessment and history taking, secondly he visits the doctor for his consultation and finally goes to the administrative staff to make a follow-up appointment and complete the paperwork. The activity time for seizing the nurse is normally distributed with mean=15mins, for doctor it is normally distributed with mean 20mins and for the administrative staff its normally distributed with 5mins. The system has 2 nurses, 3 doctors and 2 administrative staff. The interarrival time for patients is normally distributed with mean=5mins. The simulation is running f9hrs(540mins) from 8am-5pm. Find the arrival and service time for patients.



Follow Chart of Outpatient Clinic Example:



Source Code:

```
library(simmer)
library(simmer.plot)
env <- simmer("outpatient clinic")
env
patient <-trajectory("patients' path") %>%
  seize("nurse",1) %>%
  timeout(function() rnorm(1,15)) %>%
  release("nurse",1) %>%
  seize("doctor",1) %>%
  timeout(function() rnorm(1, 20)) %>%
  release("doctor", 1) %>%
  seize("administration", 1) %>%
  timeout(function() rnorm(1,5)) %>%
  release("administration",1)
env %>%
  add_resource("nurse",2) %>%
  add_resource("doctor",3) %>%
  add_resource("administration",2) %>%
  add_generator("patient", patient, function() rnorm(1,5,0.5))
```

Output:

Initial state:

```
simmer environment: outpatient clinic | now: 0 | next: 0
{ Monitor: in memory }
```

{ Resource: nurse | monitored: TRUE | server status: 0(2) | queue status: 0(Inf) }
{ Resource: doctor | monitored: TRUE | server status: 0(3) | queue status: 0(Inf) }
{ Resource: administration | monitored: TRUE | server status: 0(2) | queue status: 0(I
nf) }
{ Source: patient | monitored: 1 | n_generated: 0 }

env %>% run(until=540) #run for 9hrs or 540minutes since 8am-5pm

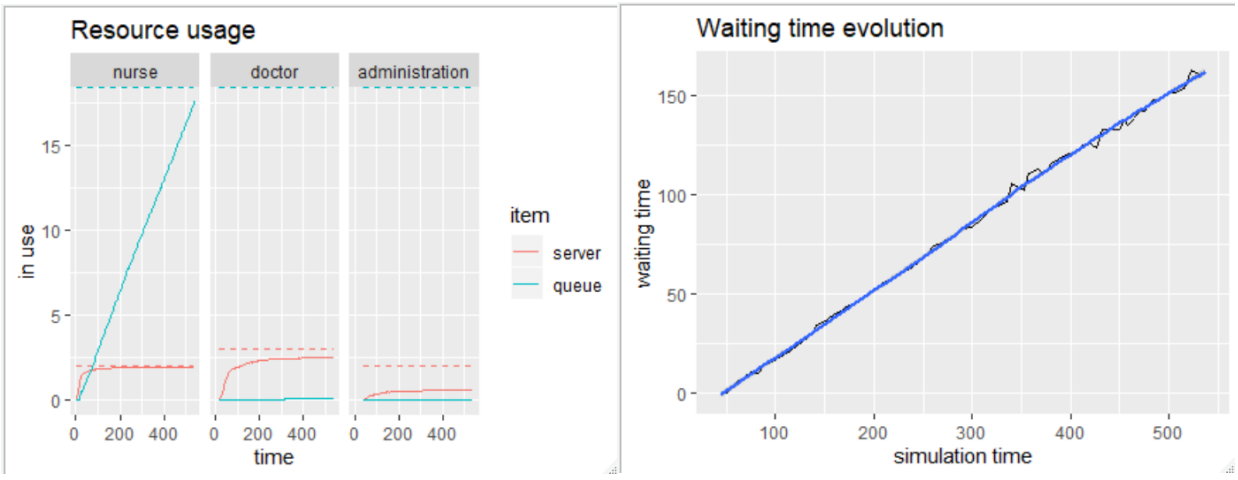
Output:

simmer environment: outpatient clinic | now: 540 | next: 540.269214663727
{ Monitor: in memory }
{ Resource: nurse | monitored: TRUE | server status: 2(2) | queue status: 35(Inf) }
{ Resource: doctor | monitored: TRUE | server status: 3(3) | queue status: 1(Inf) }
{ Resource: administration | monitored: TRUE | server status: 0(2) | queue status: 0(I
nf) }
{ Source: patient | monitored: 1 | n_generated: 109 }

plot(env, what = "resources", metric = "usage", c("nurse","doctor","administration"),
items = c("server","queue"))

plot(env, what = "arrivals", metric = "waiting_time")

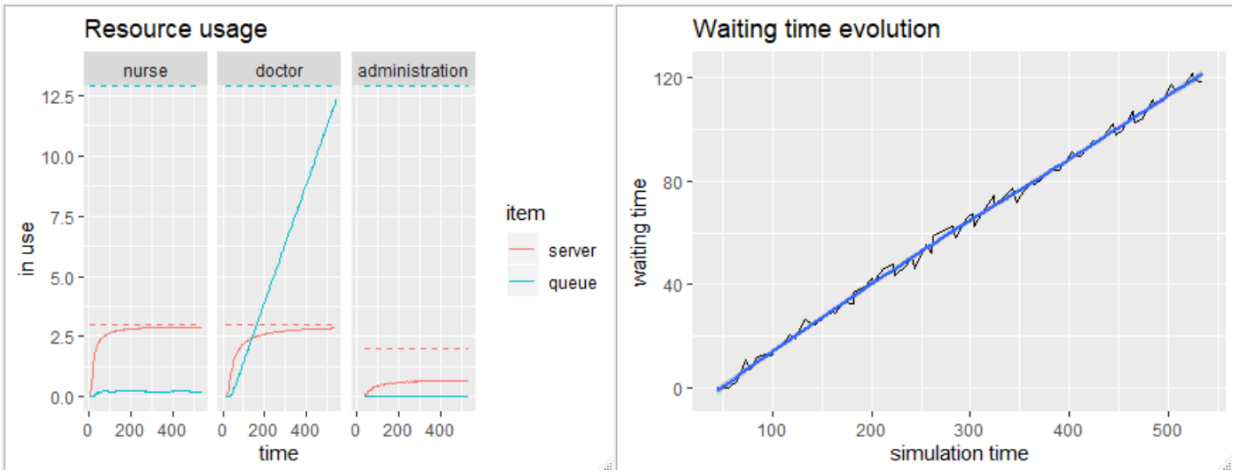
Output:



When you add 1 nurse more i.e nurse=3

Output:

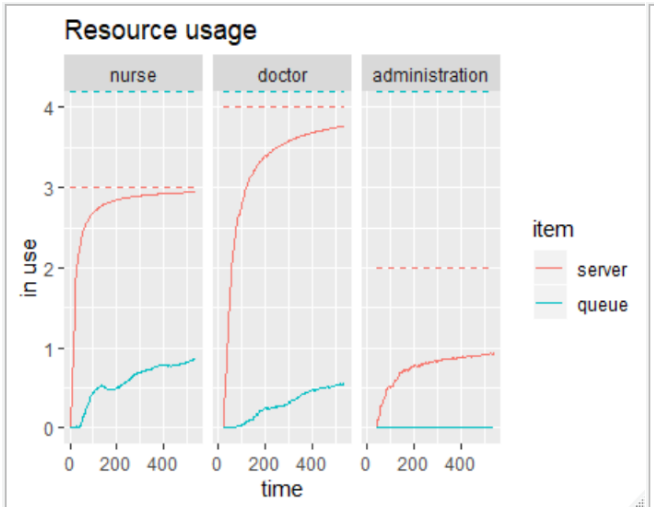
simmer environment: outpatient clinic | now: 540 | next: 540.409942860751
{ Monitor: in memory }
{ Resource: nurse | monitored: TRUE | server status: 2(3) | queue status: 0(Inf) }
{ Resource: doctor | monitored: TRUE | server status: 3(3) | queue status: 27(Inf) }
{ Resource: administration | monitored: TRUE | server status: 0(2) | queue status: 0(I
nf) }
{ Source: patient | monitored: 1 | n_generated: 107 }

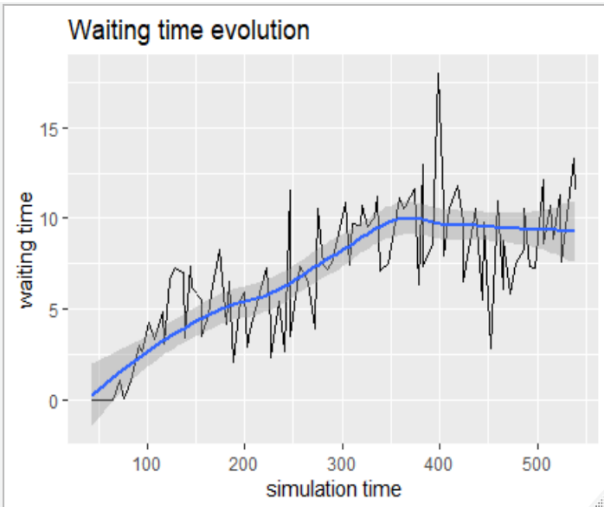


When you add 1 more doctor in the above scenario i.e doctor=4

Output:

simmer environment: outpatient clinic | now: 540 | next: 540.222676648817
{ Monitor: in memory }
{ Resource: nurse | monitored: TRUE | server status: 3(3) | queue status: 2(Inf) }
{ Resource: doctor | monitored: TRUE | server status: 3(4) | queue status: 0(Inf) }
{ Resource: administration | monitored: TRUE | server status: 2(2) | queue status: 0(I
nf) }
{ Source: patient | monitored: 1 | n_generated: 109 }





When you add 1 more administrator:

Output:

simmer environment: outpatient clinic | now: 540 | next: 540.594062906763
{ Monitor: in memory }
{ Resource: nurse | monitored: TRUE | server status: 3(3) | queue status: 2(Inf) }
{ Resource: doctor | monitored: TRUE | server status: 4(4) | queue status: 2(Inf) }
{ Resource: administration | monitored: TRUE | server status: 0(3) | queue status: 0(I
nf) }
{ Source: patient | monitored: 1 | n_generated: 110 }