**Sistem de tip coadă cu două servere legate în serie**

1. **Descrierea problemei**

Schema a fost aplicată pe un model de simulare care reprezintă Administrarea Vaccinurilor COVID-19. Am reprezentat un sistem cu 2 servere operând secvențial. Presupunem că primul server este destinat înregistrării unui pacient și asigurarea respectării măsurilor de siguranță de către acesta în incinta respectivă. Al doilea server este destinat vaccinării efective.

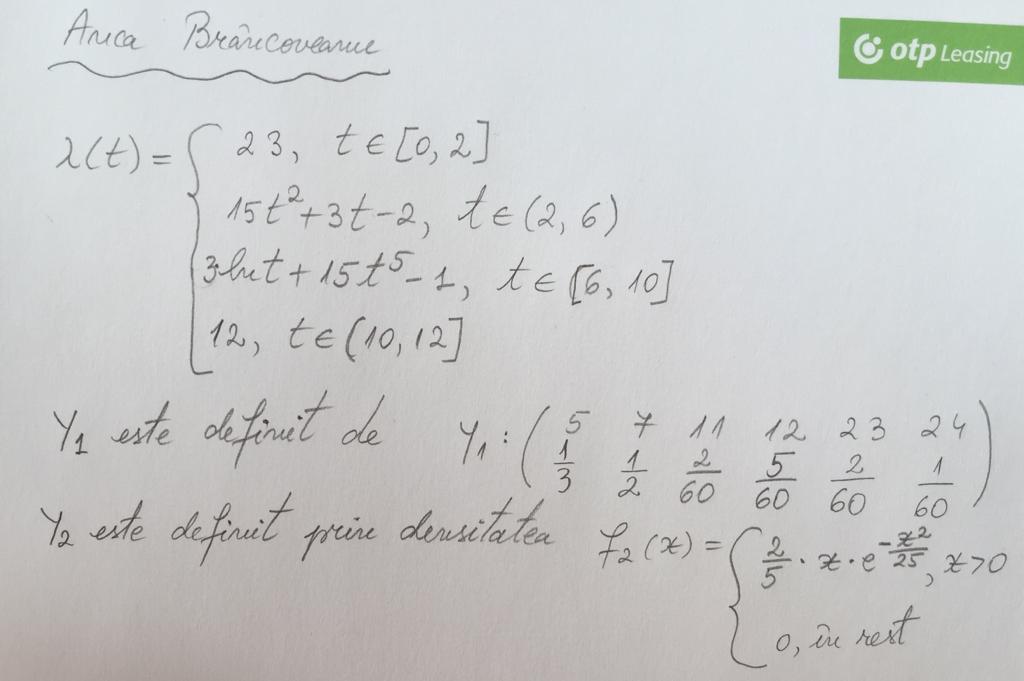
Înregistrarea unui pacient presupune completarea unui formular cu date personale, pentru a fi introduse în baza de date. Motivăm că timpul petrecut de un pacient la acest ghișeu depinde de la persoană la persoană, de aceea recurgem la generarea unei variabile aleatoare Y1 pentru

a simula această diferență. În aceeași idee, procedeul efectiv de vaccinare (activitatea serverului 2) este diferită de la persoană la persoană și poate dura intervale de timp diferite.

1. **Datele problemei**

Pentru realizarea simulării vom folosi următoarele date:

* Variabila de timp: t;
* Variabilele de stare ale sistemului (SS): (n1, n2), unde n1 și n2, reprezintă numărul de pacienți de la serverul 1, respectiv serverul 2 (cei din coadă și cel care este servit);
* Variabile-contor:
  + - NA este numărul de sosiri până la momentul t
    - ND este numărul de plecări până la momentul t
* Variabile output:
  + - A1(n) este momentul sosirii pacientului n (nu la serverul 1, ci în sistem)
    - A2(n) este momentul sosirii pacientului n la serverul 2
    - D(n) este momentul plecării pacientului n din sistem
* Lista de evenimente:
  + - tA este momentul sosirii următorului pacient
    - ti este momentul la care clinetul curent va finaliza interacțiunea cu serverul i, i ϵ {1, 2}
* Funcția de intensitate: Sosirea pacienților se face conform unui proces Poisson neomogen cu funcția de intensitate λ(t). În cazul nostru:



Ipoteze suplimentare formulate de noi:

* + - * + Presupunem că centrul primește pacienți timp de 12 ore pe zi, după care se va închide și nu mai sunt primiți pacienți. Dacă după cele 12 ore mai există persoane în așteptare, vom suplimenta programul pentru a servi pacienții aflați deja în sistem. Vom limita prelungirea programului astfel încât să nu existe situații aberante de tipul muncă peste program sau un interval mai mare de 24h/zi;
        + Considerăm că există o capacitate maximă de persoane în așteptare la ambele servere. În plus, dacă numărul de pacienți aflați în coada de așteptare a serverului 2 este maxim, centrul nu mai primește pacienți. Același comportament este valabil și în cazul în care serverul 2 nu ar servi niciun pacient, dar la serverul 1 s-ar atinge capacitatea maximă de pacienți în așteptare;
        + Pentru acuratețea unui sistem real, am introdus o constantă de răbdare pentru fiecare pacient care așteaptă în coada fiecărui server. Constanta se va atribui cu ajutorul funcției rbinom pentru fiecare pacient și o vom folosi astfel:

De fiecare dată când executăm logica din cazul 1, dacă serverul 1 are deja cel putin un pacient (n1 >=1) atunci, punem în coadă un tuplu de valori (timp\_sosire, constanta de rabdare) care reprezintă noul pacient așezat la coada de așteptare. Același lucru este valabil pentru cazul 2, adică dacă un pacient pleacă de la serverul 1 și serverul 2 are deja cel puțin o persoană de servit, atunci în coada serverului 2 adăugam iarăși un tuplu ca mai sus (timp\_sosire este fie tA, dacă adăugam în q1 la cazul 1, fie t1, dacă adăugam în q2 la cazul 2). La sfârșitul fiecărei iterații (actualizare a stării sistemului prin executarea cazului 1, 2 sau 3), aplicăm o operație de filtrare asupra celor 2 vectori q1 si q2 în care eliminăm toate tuplurile pentru care timp\_sosire + constanta\_rabdare sunt mai mici decât t actual. Cu alte cuvinte, eliminăm toate persoanele care au așteptat constanta\_rabdare timp asociat lor, de la momentul sosirii.

* + - * + Presupunem că fiecare pacient care trece în coada de așteptare pentru serverul 2 nu va părăsi sistemul până când nu va ajunge să fie servit. Dacă un pacient se află în coada de aștptare a serverului 1, acesta are dreptul de a părăsi sistemul dupa o "constantă de răbdare". În acest ultim caz, următoarea persoană va avansa în locul celei care a părăsit centrul;
        + Presupunem că, în fiecare zi avem un număr maxim de doze care pot fi administrate. Numărul rămâne constant pe perioada analizată. Dacă la sfârșitul unei zile avem doze ramase în stoc, acestea vor fi considerate ca și pierdere, deci va scădea profitul pentru acea simulare și implicit profitul mediu pentru acea zi. Astfel, putem menține o evidență a numărului de doze pierdute în fiecare zi.

1. **Rezolvarea problemei**

Schema de simulare constituie în inițializarea parametrilor și testarea cazurilor de mai jos:

Cazul 1: sosirea unui pacient nou

Verificăm dacă serverul 1 este liber, iar dacă serverul 1 are un singur pacient, generăm Y1. Pacientul care tocmai a sosit va fi servit imediat de serverul 1. Valoarea variabilei Y1 se atribuie uniform. Dacă persoanele aflate în așteptare la serverul 2 ocupă un număr maxim de scaune, trebuie să lăsăm scaune libere în caz că serverul 1 servește mai repede, pentru a nu risca ca mai mulți oameni care vin din serverul 1 la serverul 2 să rămână fără scaun.

Cazul 2:

t1<tA înseamnă că serverul 1 se eliberează înainte de sosirea unui pacient nou. Dacă nu este niciun pacient în așteptare la serverul 2, trimitem noul pacient direct la procesarea de către serverul 2. Altfel, înseamnă că există un pacient care este servit în acest moment, deci îl adăugăm în coadă. Regenerăm constanta de răbdare pentru pacientul care acum va intra în serverul 2. Dacă pe serverul 1 nu avem pacienți, t1 tinde la infinit și altfel generăm Y1 și creștem timpul t1 (t1 este momentul în care pacientul curent finalizează interacțiunea cu serverul 1-completarea formularului). Dacă pe serverul 2 avem un singur pacient, simulăm Y2 conform densității de probabilitate și estimat prin metoda respingerii.

Cazul 3:

Serverul 2 se eliberează înainte de a sosi un pacient nou și înainte de finalizarea activității la serverul 2 (vaccinarea). Creștem numărul de plecări până la momentul t și scădem numărul de pacienți (pacientul părăsește sistemul).

1. **Codul comentat**

# n\_0 = 100

# a\_0 = 20220

# lambda = 0.01

install.packages("R.utils")

install.packages("R.oo")

install.packages("methods")

install.packages("base")

install.packages("utils")

install.packages("R.methodsS3")

install.packages("shiny")

library(R.utils)

#FUNCTII PENTRU GENERAREA DE VARIABILE ALEATOARE CU DISTRIBUTIILE DATE

lambda <- 0.01

#Generarea functiei de intensitate a procesului Poisson omogen de parametru t

#Parametrul t va fi exprimat in secunde de la inceperea unei simulari,

#Prin urmare, imparti cu 3600 pentru a obtine ora respectiva

gen\_intensitate <- function(t) {

h <- t / 3600

if(0 <= h && h <= 2){

return(23)

}else if(2 < h && h <= 6){

return(15 \* t ^ 2 + 3 \* t -2)

}

else if(6 < h && h <= 10){

return(3\*ln(t) + 15 \* t^2 + -1)

}

return(12)

}

#Generarea lui Y1, o v.a. discreta cu urmatoarea distributie

#Valorile posibile ale lui Y1, sortate descrescator in functie de probabilitate

Y1\_val <- c(5, 7, 11, 12, 23, 24)

Y1\_prob <- c(20/60, 30/60, 2/60, 5/60, 2/60, 1/60)

sum\_prob <- cumsum(Y1\_prob)

gen\_y1 <- function(){

#Generam uniforma

u <- runif(1,0,1)

return(Y1\_val[which(u < sum\_prob)][1])

}

#Generarea unei variabile din distributia Exponentiala de parametru lambda

gen\_exp <- function(n, lambda){

U <- runif(n,0,1)

return( -1/lambda \* log(U))

}

#Generarea lui Y2, conform densitatii de probabilitate :

#f(x) = 2 /5 \* x \* exp((-1 \* x ^2) / 25)

#Estimat prin metoda respingerii

gen\_y2 <- function(){

while(TRUE){

Y <- gen\_exp(1,lambda = 1/5)

u <- runif(1,0,1)

if(u <= 1/5 \* Y \* exp((Y/5) \* (1 - Y/5))){

return(Y)

}

}

}

#Generarea Constantelor de rabdare printr-o repartitie Binomiala cu size=100, p=0.2

gen\_binom <- function(size, p){

#Initializam

c <- p / (1-p)

i <- 0

prob <- (1-p) ^ size

f <- prob

while(TRUE){

#Generam u -> Unif(0,1)

u <- runif(1,0,1)

if(u <= f)

return(i)

prob <- c \* (size - i ) / (i +1) \* prob

f <- f + prob

i <- i + 1

}

}

# VARIABILE DE OUTPUT

A\_1 <- c() # momentul sosirii clientului la serverul 1

A\_2 <- c() # momentul sosirii clientului la serverul 2

D <- c() # momentul plecarii clientului din sistem

q1 <- matrix(nrow=0,ncol=2)

q2 <- matrix(nrow=0,ncol=2)

# 1)CONSTANTE DE MEDIU

#A\_1[i] - t > rabdarea -> LEAVE

N\_prog <- 0.5 # numarul de ore in care centrul primeste clienti

N\_extra <- 0.5 # numarul de ore peste program

N\_max2 <- 5 #numarul maxim de scaune in sala de asteptare la serverul 2.

N\_max1 <- 4 #numarul maxim de scaune in sala de asteptare la serverul 1. Dimensiuna maxima a cozii

N\_dif <- 1 #numarul maxim de scaune libere care ar putea fi ocupate de clienti

#serviti de la serverul 1 si cand serverul 2 este plin

N\_doze <- 100 #numarul de doze disponibile in fiecare zi. Profitul va reprezenta

N\_zile <- 31 #numarul de zile asupra caruia facem observarea

# 2) INITIALIZARE

t <- 0 # variabila de timp t

n1 <- 0 # nr de clienti de la serverul 1

n2 <- 0 # nr de clienti de la serverul 2

SS <- c(n1, n2) # starea sistemului

N\_A<- 0 # nr sosiri pana la momentul t

N\_D <- 0 # nr de plecari pana la momentul t

N\_loss <- 0 #nr de clienti care au parasit sistemul fara sa fie serviti

infoServ <- matrix(nrow=2, ncol=6) #vector ce va retine informatiile generate in urma unei simulari

#infoServ[i,1] -> timpul minim de asteptare la serverul i

#infoServ[i,2] -> timpul maxim de asteptare la serverul i

#infoServ[i,3] -> timpul mediu de asteptare la serverul i

#infoServ[i,4] -> nr de clienti pierduti la serverul i

#infoServ[i,5] -> nr total de clienti serviti la serverul i

#infoServ[i,6] -> primul moment de timp la care se pierde un client la serverul i

# Algoritm de generare T\_s

generare\_T\_s <- function(s, lambda){

T\_s <- 0

repeat{

# 1. t = s

t <- s

# 2. generam U\_1 , U\_2

U\_1 <- runif(1)

U\_2 <- runif(1)

# 3.

t <- t - (1 / lambda) \* log(U\_1)

# 4.

if(U\_2 <= gen\_intensitate(t) / lambda){

T\_s <- t

break;

}

}

return(T\_s)

}

#Generare initiala

T\_0 <- generare\_T\_s(0, lambda)

t\_A <<- T\_0

t\_1 <<- Inf

t\_2 <<- Inf

#-------------------------------------------------------------------------------

# 2: CAZUL 1

generare\_gamma <- function(lambda) {

U <- runif(1)

scale <- lambda

return ((-1 / scale) \* sum(log(U)))

}

cazul\_1 <- function(){ # vede variabilele din afara

# caz: t\_A = min(t\_A, t\_1, t\_2)

t <<- t\_A

T\_t <<- generare\_T\_s(t, lambda)

t\_A <<- T\_t

# Daca persoanele aflate in asteptare la serverul 2 ocupa un nr max de scaune

# trebuie sa lasam libere scaune in caz ca serverul 1 serveste mai repede

# ca sa nu riscam ca mai multi oameni care vin din serverul 1 la serverul 2

# sa "ramana fara scaun"

if(n1 >= N\_max1 || (n1 > 1 && n2 >= N\_max2 - N\_dif)){

#Daca totusi se intampla, pierdem

N\_loss <- N\_loss + 1

}else{

if(n1 == 0){

n1 <<- n1 + 1

}else{

#Generam cu rbinom timpul lui de astptare

n\_rabdare <- gen\_binom(100, 0.2)

q1 <<- rbind(q1, c(t, n\_rabdare))

n1 <<- n1 + 1

}

if(n1 == 1){

#Functiile de generare dupa datele primite genereaza valori mici

#In comparatie cu t. Ne vom asuma ca aceste valori sunt, de fapt, minute

Y\_1 <- gen\_y1() \* 60

if( Y\_1 < infoServ[1,1]){

infoServ[1,1] <<- Y\_1

}else if(Y\_1 > infoServ[1,2]){

infoServ[1,2] <<- Y\_1

}

#print(c("Y\_1 generat la caz1:",Y\_1))

infoServ[1,3] <<- infoServ[1,3] + Y\_1

t\_1 <<- t + Y\_1

}

# Output cazul 1)

A\_1\_temp <- A\_1

A\_1 <<- c(A\_1, t)

}

}

# 3: CAZUL 2

cazul\_2 <- function (){

# caz: t\_1 < t\_A && t\_1<= t\_2

t <<- t\_1

if(n1 > 0){

n1 <<- n1 - 1

}

#Adaugam la numarul total de clienti serviti de catre serverul 1

infoServ[1,5] <<- infoServ[1,5] + 1

#Daca nu este niciun client in asteptare la serverul 2, trimitem noul client

#Direct la procesarea de catre serverul 2

if(n2 == 0){

n2 <<- n2 + 1

}else{

#Altfel, insamna ca exista un client care este servit in acest moment

#Deci il adaugam pe acest nou client in coada.

#Regeneram constanta de rabdare pentru clientul care acum va intra in serverul 2

n\_rabdare <- gen\_binom(100,0.2)

#Introducem clientul in serverul2

q2 <<- rbind(q2, c(t, n\_rabdare))

n2 <<- n2 + 1

}

#Daca este singurul client la serverul 2, generam timpul de procesare al acestuia

if(n2 == 1){

#Generarea variabilelor conform repartitiei date de f2(x) genereaza date prea mici

#in comparatie cu t, motiv pentru care ne asumam ca aceste valori ar fi,

#de fapt, minute. Asadar, le vom converti in secunde

Y\_2 <<- gen\_y2() \* 60

if( Y\_2 < infoServ[2,1]){

infoServ[2, 1] <<- Y\_2

}else if(Y\_2 > infoServ[2,2]){

infoServ[2, 2] <<- Y\_2

}

#Actualizam timpul mediu petrecut la serverul 2

infoServ[2,3] <<- infoServ[2,3] + Y\_2

#print(c("Y\_2 generat la caz2:",Y\_2))

t\_2 <<- t + Y\_2

}

if(n1 == 0){

t\_1 <<- Inf

} else{

Y\_1 <<- gen\_y1() \* 60

#print(c("Y\_1 generat la caz2:",Y\_1))

t\_1 <<- t + Y\_1

#Eliminam prima persoana din coada, deoarece este uramtoarea ce va fi servita,

#Daca exista perosoane in coada

if( nrow(q1) > 0){

#print("before Pop:")

#print(q1)

curent <- q1[1, ]

q1 <<- q1[-1,,drop=FALSE]

#print("after Pop:")

# print(dim(q1))

# print(q1)

if( Y\_1 + t - curent[1] < infoServ[1, 1]){

infoServ[1,1] <<- Y\_1 + t - curent[1]

}else if(Y\_1 + t - curent[1]> infoServ[1,2]){

infoServ[1,2] <<- Y\_1 + t - curent[1]

}

#Actualizam la timpul total de servire server 1

infoServ[1,3] <<- infoServ[1,3] + Y\_1 + t - curent[1]

}

}

# Output cazul 2)

N\_temp <- N\_A - n1

A\_2\_temp <- A\_2

if(N\_temp <= length(A\_2))

A\_2[N\_temp] <<- t

else

#print("N\_temp > length (A\_2)")

A\_2 <<- c(A\_2, t)

}

# 4: CAZUL 3

cazul\_3 <- function() {

# t\_2 < t\_A && t\_2 < t\_1

t <<- t\_2

N\_D <<- N\_D + 1

infoServ[2,5] <<- N\_D

if(n2 > 0 ){

n2 <<- n2 - 1

}

if(n2 == 0) {

t\_2 <<- Inf

}else {

Y\_2 <- gen\_y2() \* 60

#print(c("Y\_2 generat la caz3:",Y\_2))

t\_2 <<- t + Y\_2

if(nrow(q2) > 0){

#Adaugam timpul petrecut de clientul din coada la timpii de asteptare

curent <- q2[1, ]

#Eliminam din coada clientul servit.

q2 <<- q2[-1,,drop=FALSE]

#q2 <<- q2[-1,]

#Actualizam timpul minim, maxim petrecut la serverul 2

if( Y\_2 + t - curent[1]< infoServ[2,1]){

infoServ[2,1] <<- Y\_2 + t - curent[1]

}else if(Y\_2 > infoServ[2,2]){

infoServ[2,2] <<- Y\_2 + t - curent[1]

}

#Actualizam timpul mediu petrecut la serverul 2

infoServ[2, 3] <<- infoServ[2,3] + Y\_2 + t - curent[1]

}

}

# Output cazul 3)

D\_temp <- D

D <<- append(D\_temp, t, after=N\_D)

}

#-------------------------------------------------------------------------------

resetare\_variabile <- function(){

#O noua zi, resetam masura de timp

t <<- 0

#O noua zi, niciun client de procesat, asteptam clientii

n1 <<- 0

n2 <<- 0

#Generam T\_0 prin care setam momentul de timp al sosirii primului client

T\_0 <- generare\_T\_s(0, lambda)

t\_A <<- T\_0

t\_1 <<- Inf

t\_2 <<- Inf

A\_1 <<- c()

A\_2 <- c()

D <<- c()

N\_A <<- 0

N\_D <<- 0

q1 <<- matrix(nrow=0,ncol=2)

q2 <<- matrix(nrow=0,ncol=2)

#Fiecare zi incepe cu 100 de doze

N\_doze <<- 100

#Timp minim petrecut

infoServ[1,1] <<- Inf

infoServ[2,1] <<- Inf

#Timp maxim petrecut

infoServ[1,2] <<- 0

infoServ[2,2] <<- 0

#Timp petrecut

infoServ[1,3] <<- 0

infoServ[2,3] <<- 0

#N clienti pierduti

infoServ[1,4] <<- 0

infoServ[2,4] <<- 0

#N clienti serviti

infoServ[1,5] <<- 0

infoServ[2,5] <<- 0

#Primul mom de timp la care se pierde un client

infoServ[1,6] <<- 0

infoServ[2,6] <<- 0

}

# ------------------------------------------------------------------------------

simulare\_zi <- function(dummy = 1,n\_sim, n\_zi) {

#De fiecare data cand apelam simulare\_zi, ptc folosim variabile globale

#Avem nevoie sa dam reset

resetare\_variabile()

#Initial, consideram ca centrul isi incepe activitatea la timpul 0.

#Consideram ca fiecare variabila de timp reprezinta numarul de secunde

#De la deschiderea centrului.

#Astfel, este nevoie sa ne oprim atunci cand t depaseste N\_prog \* 3600 = 43200

while(TRUE){

# print("-----------------------")

#Centrul se va inchide daca una din cele 2 conditii este indeplinita:

#1. Fie au trecut si numarul de ore extra pe care le-am alocat servirii

#2. Fie programul de primire s-a terminat si am reusit sa servim toti clientii

if(t >= (N\_prog + N\_extra) \* 3600 ||

t >= N\_prog \* 3600 && n1 == 0 && n2 == 0){

# return metrics -> N\_loss, length(A\_2) numar doze,

#print(c("Ramasi in n1", n1))

#print(c("Ramasi in n2", n2))

#print(c("t\_A=",t\_A))

#print(c("t\_1=",t\_1))

#print(c("t\_2=",t\_2))

#print(c("Doze folosite=",N\_D))

#Media timpului de astepare pentru cele 2 servere

infoServ[1,3] <<- infoServ[1,3] / infoServ[1,5]

infoServ[2,3] <<- infoServ[2,3] / infoServ[2,5]

break

}

if(t\_A == min(t\_A, t\_1, t\_2)){

#print("Intra cazul 1")

#Primim clientii doar daca se indeplinesc simultan:

#1. Timpul curent nu a depasit programul de munca

#2. Coada serverului 2 nu este la capacitate maxima

#3. Coada serverului 1 nu este la capacitate maxima

if(t < N\_prog \* 3600){

#print("Cazul 1")

cazul\_1()

}

else{

t\_A <<- Inf

#print("Au trecut orele de primire clienti. Overtime.")

}

}

else if(t\_1 < t\_A && t\_1 <= t\_2){

#print("Intra cazul 2")

cazul\_2()

}

else if(t\_2 < t\_A && t\_2 < t\_1){

# print("Intra cazul 3")

cazul\_3()

}

#Dat fiind t momentul de timp actual, eliminam din cele 2 cozi toti clientii

#care si-au pierdut rabdarea

len\_q1\_inainte <- nrow(q1)

len\_q2\_inainte <- nrow(q2)

if(nrow(q1) > 0){

if(infoServ[1,6] == 0){

q1\_de\_elim <- subset(q1, q1[, 1] + q1[, 2] <= t)

q1 <<- subset(q1, q1[, 1] + q1[, 2] > t)

if(nrow(q1\_de\_elim) > 0){

infoServ[1,6] <<- q1\_de\_elim[1, 1] + q1\_de\_elim[1, 2]

}

}else{

q1 <<- subset(q1, q1[, 1] + q1[, 2] > t)

}

}

if(nrow(q2) > 0){

if(infoServ[2,6] == 0){

q2\_de\_elim <- subset(q2, q2[, 1] + q2[, 2] <= t)

q2 <<- subset(q2, q2[, 1] + q2[, 2] > t)

if(nrow(q2\_de\_elim) > 0){

infoServ[2,6] <<- q2\_de\_elim[1, 1] + q2\_de\_elim[1, 2]

}

}else{

q2 <<- subset(q2, q2[, 1] + q2[, 2] > t)

}

}

n1 <<- n1 + nrow(q1) - len\_q1\_inainte

n2 <<- n2 + nrow(q2) - len\_q2\_inainte

##print(c(length(q1[,1]), len\_q1\_inainte))

#print(c(length(q2[,2]), len\_q2\_inainte))

#print("After:Q1 and Q2:")

#print(q1)

#print(q2)

#print(c("t\_A=",t\_A))

#print(c("t\_1=",t\_1))

#print(c("t\_2=",t\_2))

#print(c("n1=",n1))

#print(c("n2=",n2))

#Numarul de clienti pierduti per fiecare sever este diferenta dintre lungimea dinainte

#de eliminare si lungimea curenta

if(len\_q1\_inainte > nrow(q1)){

infoServ[1, 4] <<- infoServ[1, 4] + (len\_q1\_inainte - nrow(q1))

}

if(len\_q2\_inainte > nrow(q2))

infoServ[2, 4] <<- infoServ[2, 4] + (len\_q2\_inainte - nrow(q2))

}

#print("Metrics:")

#Odata obtinute valorile pentru o simulare pentru o zi, acutalizam matricea agregatoare

#print(infoServ)

for (i in 0:1){

#Actualizarea timpului minim petrecut la serverul i din toate simularile

agregator[2 \* n\_zi - i, 1] <<- min(agregator[2 \* n\_zi - i, 1], infoServ[2 - i, 1])

#Actualizarea timpului maxim petrecut la serverul i din toate simularile

agregator[2 \* n\_zi - i,2] <<- max(agregator[2 \* n\_zi - i ,2], infoServ[2 - i,2])

#Actualizarea timpului mediu petrecut la serverul i din toate simularile

agregator[2 \* n\_zi - i,3] <<- (1 / n\_sim) \* infoServ[2 - i , 3] + agregator[2 \* n\_zi - i, 3]

#Actualizarea numarului mediu de clienti pierduti la serverul i

agregator[2 \* n\_zi - i,4] <<- (1 / n\_sim) \* infoServ[2 - i, 4] + agregator[2 \* n\_zi - i, 4]

#Actualizarea numarului mediu de clienti serviti la serverul i

agregator[2 \* n\_zi - i,5] <<- (1 / n\_sim) \* infoServ[2 - i, 5] + agregator[2 \* n\_zi - i, 5]

#Actualizarea numarului mediu de clienti serviti la serverul i

agregator[2 \* n\_zi - i,6] <<- (1 / n\_sim) \* infoServ[2 - i, 6] + agregator[2 \* n\_zi - i, 6]

}

#print(c("Program incheiat.Last t:", t))

#print(agregator)

#Adunam la profitul mediu pentru aceasta zi, profitul obtinut pentru aceasta simulare

profit <<- (1 / n\_sim) \* (N\_doze - N\_D) + profit

#print(c("PROFIT:", profit))

#print("-------------------------")

}

#castig[3,1]

#Rulam simularea pentru 7 zile ->castig normal aka castig[1,1]

#Rulam simularea pentru 7 zile, dar N\_prog ++ ->castig[2,1]

#Rulam simularea pentru 7 zile, dar N\_prog -- ->castig[3,1]

#castig[2,1] - castig[1,1] pentur N\_prog ++

#castig[3,1] - castig[1,1] pentur N\_prog --

#Simulare\_zi va fi metoda apelata pentru a obtine metricele aferente unei

#zile lucratoare

#for i in 1:7{

#In fiecare zi din perioada specificata,e.g. 1:7 o saptamana

#Rulam 10^6 scenarii posibile

# for sim in 1:1e6

#metrics = simulare\_zi()

#Pentru fiecare dintre cele 10^6 valori ale metricelor,

#Calculam o valoare medie orientativa pentru ziua respectiva.

#Adaugam in grafice

#pe Ox numarul zilei si pe Oy una din metrice.

#Cate un grafic per metrica.

#}

#-------------------------------------------------------------------------------

agregator <- matrix(nrow=2 \* N\_zile , ncol=6)

profit <- 0

profituri <- c()

main <- function(){

n <- 5

N\_prog =1

saptamana <- c()

for(i in 1:N\_zile){

#Pentru fiecare zi rulam n simulari

agregator[2 \* i - 1, 2:6] <<- 0

agregator[2 \* i, 2:6] <<- 0

agregator[2 \* i - 1, 1] <<- Inf

agregator[2 \* i, 1] <<- Inf

profit <<- 0

sapply(1:n, simulare\_zi, n\_sim=n, n\_zi=i)

profituri <<- c(profituri, profit)

#print(c("Sfarist de simulare pentru ziua: ",i))

#print(agregator)

}

#Dupa finalizarea simularilor pentru N\_zile, in agregator vom avea

#Pe fiecare rand de indice 2 \* i - 1 informatii de la serverul 1 pentru ziua i

#Pe fiecare rand de indice 2 \* i informatii de la serverul 2 pentru ziua i

#par(mfrow=2, mfcol=2)

ylabs <- c("Timp minim de asteptare",

"Timp maxim de asteptare",

"Timp mediu de astepare",

"Clienti pierduti",

"Clienti serviti",

"Primul Moment La care se pierde un client")

plot\_grafic <- function(i){

ymin <- min(agregator[, i])

ymax <- max(agregator[, i])

plot(seq(1,N\_zile), agregator[seq(1,2\*N\_zile,2), i], col='red', xlab='Zilele Saptamanii',

type='b',

ylab=ylabs[i],

ylim = c(ymin, ymax))

points(seq(1,N\_zile), agregator[seq(2,2\*N\_zile,2), i], col='blue', xlab='Zilele Saptamanii')

lines(seq(1,N\_zile), agregator[seq(2,2\*N\_zile,2), i], col='blue', xlab='Zilele Saptamanii',lty=1)

legend("topright",

legend = c("Serveul 1", "Serverul 2"),

col = c("red","blue"),

bty="n",

pch = "o",

lty =1)

}

sapply(1:6, plot\_grafic)

#Plotare profituri

plot(seq(1:N\_zile), profituri,type='b',col="green",xlab="Zi", ylab="Profit")

}

main()

#simulare\_zi(n\_sim=1, n\_zi=1)

# ------------------------------------------------------------------------------

library(shiny)

# Define UI for application that draws a histogram

ui <- fluidPage(

# Application title

titlePanel("Old Faithful Geyser Data"),

# Sidebar with a slider input for number of bins

sidebarLayout(

sidebarPanel(

sliderInput("bins",

"Number of bins:",

min = 1,

max = 50,

value = 30)

),

# Show a plot of the generated distribution

mainPanel(

plotOutput("distPlot")

)

)

)

# Define server logic required to draw a histogram

server <- function(input, output) {

output$distPlot <- renderPlot({

# generate bins based on input$bins from ui.R

x <- faithful[, 2]

bins <- seq(min(x), max(x), length.out = input$bins + 1)

# draw the histogram with the specified number of bins

hist(x, breaks = bins, col = 'darkgray', border = 'white')

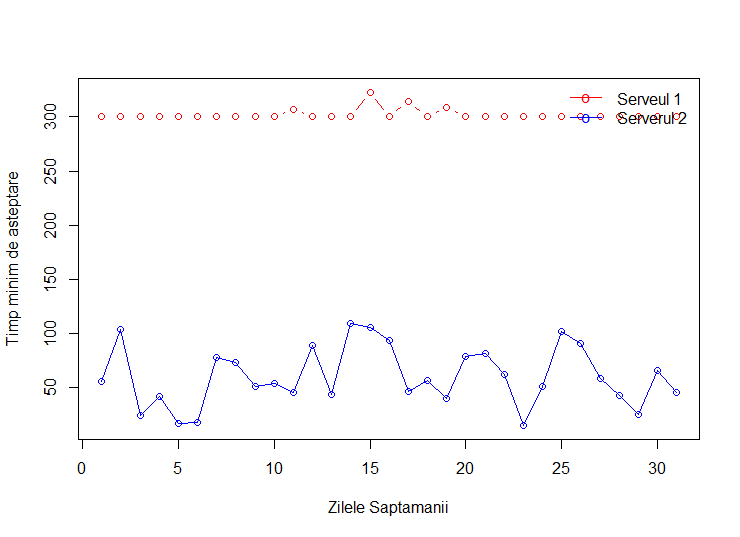
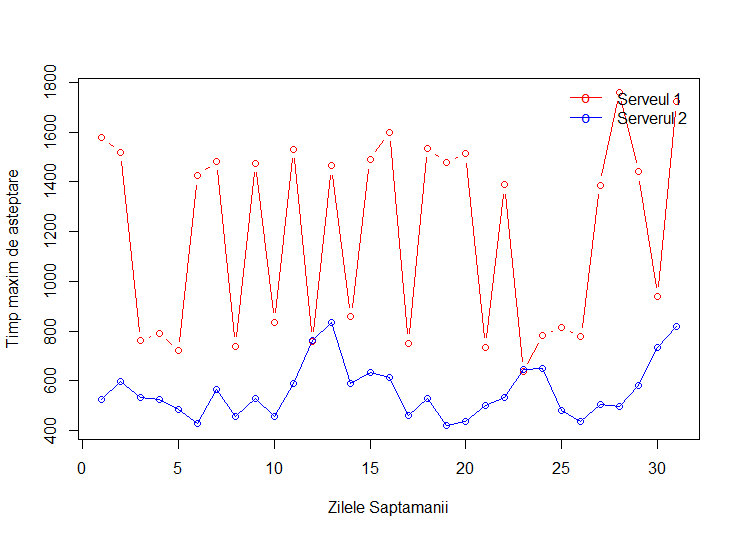
})

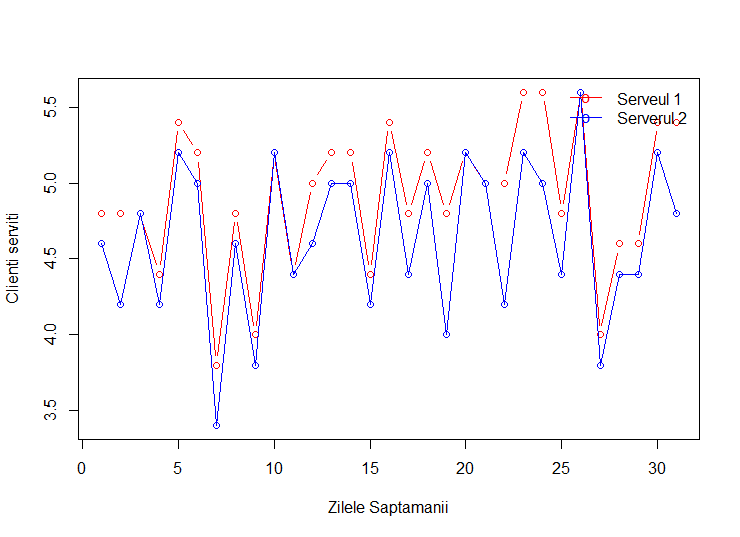
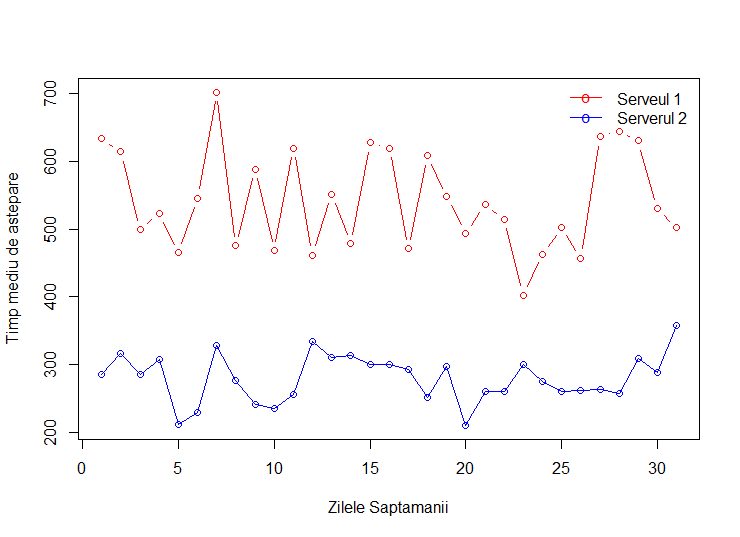
}

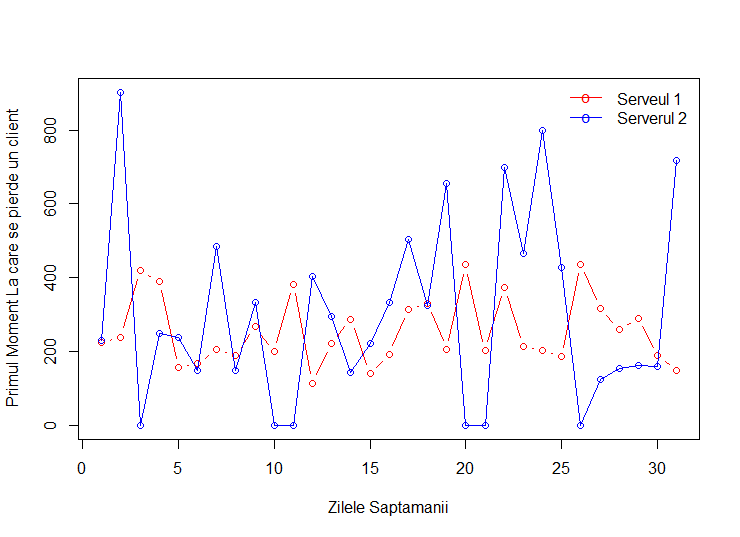
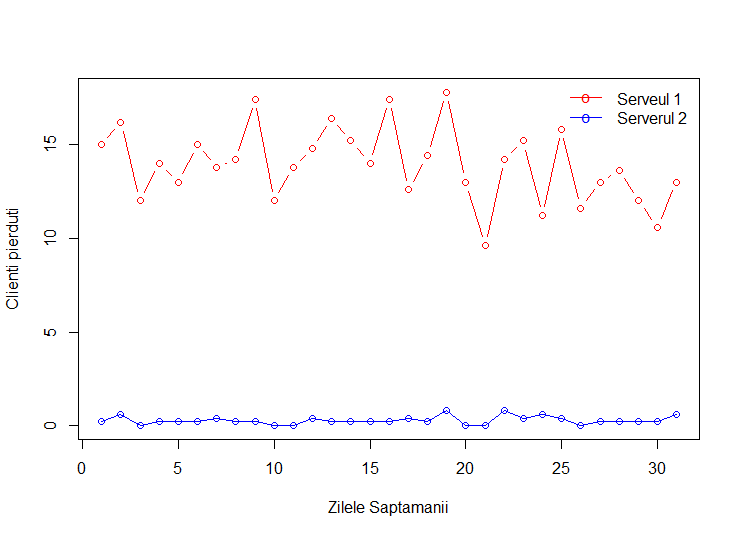
# Run the application

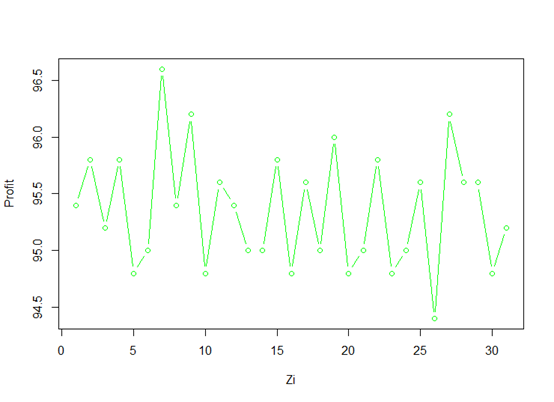
#shinyApp(ui = ui, server = server)

1. **Rezultatele obținute**

 ****

****

**** ****



1. **Comentarii și concluzii**