ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC KỸ THUẬT MÁY TÍNH

ĐÁNH GIÁ HIỆU NĂNG HỆ THỐNG

BÀI TẬP LỚN

Mô phỏng hệ thống hàng dùng Simpy

Chủ đề 7

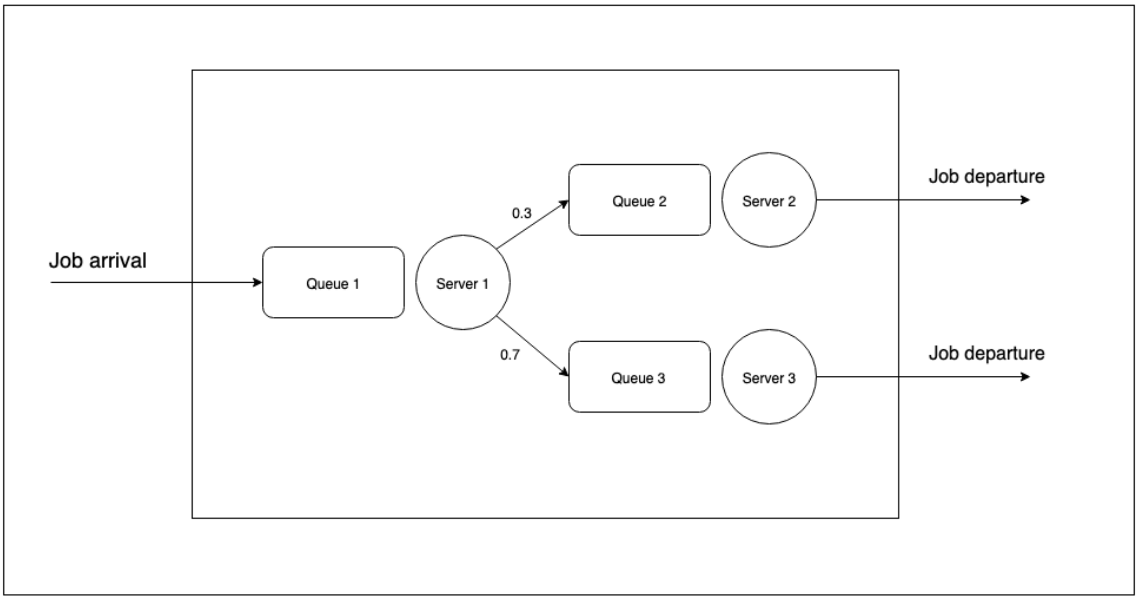
GVHD: **Trần Văn Hoài**
Thành viên : Đỗ Xuân Thơ - 1713365
Nguyễn Phước Sang - 1712938
Tô Ngọc Duy - 1710813

TP. HỒ CHÍ MINH, THÁNG 11/2019

Mục lục

1 Giới thiệu

Chủ đề 7:
Mạng 3 hàng $Q_1 = M/M(\mu_1)/1$, $Q_2 = M/M(\mu_2)/1$, $Q_3 = M/M(\mu_3)/1$, trong đó $Q_1 \to Q_2$ với $p_{12} = 0.7$; $Q_1 \to Q_3$ với $p_{13} = 0.3$; quá trình đến Q_1 với λ, công việc sau khi qua Q_2, Q_3 sẽ rời khỏi hệ thống.



2 Quy trình đánh giá hiệu năng hệ thống

2.1 State goals and Define the System

- **Goals:**
  - Đánh giá hiệu quả khi áp dụng các kỹ thuật Transient Removal hay Terminating Simulation đối với hệ thống
  - Đánh giá kết quả của mô hình mô phỏng so với mô hình phân tích .
- **System:** three Servers + Job arrival

2.2 List Services and Outcomes

- **Services:** Đưa một Job qua System
- **Outcome:**
  - Job qua hệ thống thành công
  - Job còn nằm ở trong hệ thống

2.3 Select Metrics

- Hiệu suất sử dụng hệ thống (Utilization)
- Thời gian phục vụ trung bình của hệ thống (Mean service time)
- Số lượng Job trung bình nằm trong hệ thống (Mean Job in System)
- Phương sai của số lượng Job trung bình nằm trong hệ thống (Variance of Job in System)

2.4 List Parameters

2.4.1 System parameter

- **Service rate of Server 1** : μ_1
- **Service rate of Server 2** : μ_2
- **Service rate of Server 3** : μ_3

2.4.2 Workload parameter

- **Arrival rate to Server 1** : λ_1
- **MaxTime of Simulation** : MAXTIME
- **M replication for transient removal and terminating simulation** : M
- **Probability of N or more jobs in the system** : N
- **Knee được xác định bởi Tan= delta(lenght of queue)/delta(time)** : Tan
- **Độ tin cậy 1-alpha** : alpha

2.5 List Factors to Study

- **Arrival rate to Server 1** : λ_1
- **Service rate of Server 1** : μ_1
- **Service rate of Server 2** : μ_2
- **Service rate of Server 3** : μ_3
- **MaxTime of Simulation** : MAXTIME
- **M replication for transient removal and terminating simulation** : M
- **Knee được xác định bởi Tan= delta(lenght of queue)/delta(time)** : Tan
- **Độ tin cậy 1-alpha** : alpha

2.6 Select Evaluation Technique

- Kĩ thuật đánh giá : Mô phỏng

2.7 Select Workload

- Synthesis Workload :
  - JobGenerator with arrival time is exponential distributed
  - M replication
  - MaxTime of Simulation : MAXTIME

2.8 Design Experiments

- Mô phỏng được thiết kế bằng ngôn ngữ python:

2.8.1 Danh sách các package được sử dụng

```
import simpy
#import random
import numpy.random as random
import scipy.stats as ss
import math
import matplotlib.pyplot as plt
```

2.8.2 List parameter

```
   ''' ------------------------ '''
''' Parameters               '''
''' ------------------------ '''
MU1=8
MU2=4
MU3=8
LAMDA=7.5
LOGGED = True
VERBOSE = False
MAXSIMTIME = 200
'''
Probability of N or more jobs in the system
'''
N = 2
''' M relplication for transient removal and terminating simulation'''
M = 10
''' Knee duoc xac dinh boi Tan= delta(lenght of queue)/delta(time) '''
Tan = 0.0001
''' Do tin cay 1-alpha '''
alpha = 0.05
```

2.8.3 Định nghĩa Job: Job

```
class Job:
def __init__(self, name, arrtime, duration,server):
    self.name = name
    self.arrtime = arrtime
    self.duration = duration
    self.server=server
def __str__(self):
    return '%s at %d, length %d,at Server %s' %(self.name, self.arrtime,
        self.duration,self.server.name)
```

2.8.4 Định nghĩa trạng thái của hàng đợi : state

```
class state:
def __init__(self,job_len,time):
    self.job_len=job_len
    self.time= time
```

2.8.5 Định nghĩa Server1: server-inout

```
class Server_inout:
def __init__(self, env,name,server_out1,mu1,server_out2,mu2):
    self.env = env
    self.name = name
    self.server_out1=server_out1
    self.server_out2=server_out2
    self.servicetime1=float(1/float(mu1))
    self.servicetime2=float(1/float(mu2))
    self.Jobs = list(())
    self.queue = list(())
    self.system=list(())
    self.serversleeping = None
    ''' statistics '''
    self.waitingTime = 0
    self.numberofJobqueue = 0
    self.numberofJobsystem = 0
    self.varofJobSystem=0
    self.probabilityofNJob=0
    self.idleTime = 0
    self.jobsDone = 0
    ''' register a new server process '''
    self.env.process( self.serve() )
def compute_numberofJob(self,k):
    numberofqueuextime = 0
    i=0
    while i<len(self.queue)-1:
        numberofqueuextime+=self.queue[i].job_len*(self.queue[i+1].time-self.queue[i].time)
        i+=1
```

```
        self.numberofqueue=float(numberofqueuextime/MAXSIMTIME)
        numberofsystemxtime = 0
        i=k
        while i<len(self.system)-1:
            numberofsystemxtime+=self.system[i].job_len*(self.system[i+1].time-self.system[i].time)
            i+=1
        self.numberofJobsystem=float(numberofsystemxtime/(MAXSIMTIME-self.system[k].time))
    def compute_varofJobSystem(self,k):
        '''(Jobsystem*time)^2'''
        '''
             a=(self.numberofJobsystem**2)*self.system[0].time/MAXSIMTIME+((self.system[len(self.system)-1].
        job_len-self.numberofJobsystem)**2)*(MAXSIMTIME-self.system[len(self.system)-1].time)/MAXSIMTIME'''
        a=0
        i=k
        while i<len(self.system)-1:
            a+=((self.system[i].job_len-self.numberofJobsystem)**2)*(self.system[i+1].time-
            self.system[i].time)/(MAXSIMTIME-self.system[k].time)
            i+=1
        self.varofJobSystem=a
    def compute_probilityofNJobs(self,n,k):
        a=0
        i=k
        if (n==0) :
            self.probabilityofNJob = 1
        else :
            while i<len(self.system)-1:
                    if (self.system[i].job_len>=n):
                        a+=self.system[i+1].time-self.system[i].time
                    i+=1
            self.probabilityofNJob=a/MAXSIMTIME
    def serve(self):
        while True:
            ''' do nothing, just change server to idle
              and then yield a wait event which takes infinite time
            '''
            if len(self.Jobs)==0:
                self.serversleeping = env.process( self.waiting( self.env ))
                t1 = self.env.now
                yield self.serversleeping
                ''' accumulate the server idle time'''
                self.idleTime += self.env.now - t1
            else:
                j=self.Jobs.pop(0)
                ''' add queue_state to queue list'''
                self.queue.append(state(len(self.Jobs),self.env.now))
                ''' sum up the waiting time'''
                self.waitingTime += self.env.now - j.arrtime
                ''' yield an event for the job finish'''
                yield self.env.timeout( j.duration )
                ''' sum up the jobs done '''
                self.jobsDone += 1
                ''' add system_state to system list'''
                self.system.append(state(len(self.Jobs),self.env.now))
```

```
            '''append Job to server_out1 or server_out2'''
            a=random.randint(1,10)
            if VERBOSE:
                print('%s done : t = %.2f , %s'%(j.name,self.env.now,self.name))
            if( a>=1 and a<=3) :
                duration1=random.exponential(self.servicetime1)
                if VERBOSE:
                    print('%s come : t = %.2f , duration time = %d ,
                        %s'%(j.name,self.env.now,duration1
                    ,str(self.server_out1.name)))
                self.server_out1.Jobs.append(Job(j.name,self.env.now,duration1,self.server_out1))
                if not self.server_out1.serversleeping.triggered:
                    self.server_out1.serversleeping.interrupt()
            else:
                duration2=random.exponential(self.servicetime2)
                if VERBOSE:
                    print('%s come : t = %.2f ,duration time = %d ,
                        %s'%(j.name,self.env.now,duration2,
                    str(self.server_out2.name)))
                self.server_out2.Jobs.append(Job(j.name,self.env.now,duration2,self.server_out2))
                if not self.server_out2.serversleeping.triggered:
                    self.server_out2.serversleeping.interrupt()
    def waiting(self, env):
        try:
            if VERBOSE:
                print( '%s is idle at %.2f' %(self.name, self.env.now) )
            yield self.env.timeout( MAXSIMTIME )
        except simpy.Interrupt as i:
            if VERBOSE:
                print('%s waken up and works at %.2f' %(self.name, self.env.now) )
```

2.8.6 Định nghĩa Server 2, Server 3: server-in

```
    class Server_in:
    def __init__(self, env,name):
        self.env = env
        self.name = name
        self.Jobs = list(())
        self.serversleeping = None
        ''' statistics '''
        self.waitingTime = 0
        self.idleTime = 0
        self.jobsDone = 0
        ''' register a new server process '''
        self.env.process(self.serve())

    def serve(self):
        while True:
            ''' do nothing, just change server to idle
              and then yield a wait event which takes infinite time
            '''
```

```
        if len(self.Jobs)==0 :
            self.serversleeping =self.env.process( self.waiting( self.env ))
            t1 = self.env.now
            yield self.serversleeping
            ''' accumulate the server idle time'''
            self.idleTime += self.env.now - t1
        else:
            j = self.Jobs.pop( 0 )
            ''' sum up the j = self.Jobs.pop( 0 )waiting time'''

            self.waitingTime += self.env.now - j.arrtime
            ''' yield an event for the job finish'''
            yield self.env.timeout( j.duration )
            ''' sum up the jobs done '''

            self.jobsDone += 1
            if VERBOSE:
                print('%s done : t = %.2f ,%s'%(j.name,self.env.now,self.name))

def waiting(self, env):
    try:
        if VERBOSE:
            print( '%s is idle at %.2f' % (self.name,self.env.now) )
        yield self.env.timeout( MAXSIMTIME )
    except simpy.Interrupt as i:
        if VERBOSE:
             print('%s waken up and works at %.2f' % (self.name,self.env.now))
```

2.8.7 Định nghĩa bộ tạo Job : JobGenerator

```
class JobGenerator:
def __init__(self, env, server,mu,lam):
    self.env= env
    self.server= server
    self.servicetime=float(1/float(mu))
    self.interarrivaltime =float( 1/float(lam))
    env.process( self.generatejobs(env) )

def generatejobs(self, env):
    i = 1
    while True:
        '''yield an event for new job arrival'''
        job_interarrival = random.exponential( self.interarrivaltime )
        yield env.timeout( job_interarrival )
        ''' generate service time and add job to the list'''
        job_duration=random.exponential( self.servicetime )
        self.server.Jobs.append( Job('Job %s' %i, self.env.now, job_duration,self.server) )
        self.server.queue.append(state(len(self.server.Jobs),self.env.now))
        self.server.system.append(state(len(self.server.Jobs),self.env.now))
        if VERBOSE:
            print( 'Job %d come : t = %.2f, duration time = %.2f, arrival time = %.2f ,%s'
```

```
        %( i, self.env.now, job_duration, job_interarrival,self.server.name ) )
i += 1
if not self.server.serversleeping.triggered:
    self.server.serversleeping.interrupt( 'Wake up, please.' )
```

2.9 Analyze and Interpret Data

2.9.1 Start simulation

```
''' start simulation '''
env = simpy.Environment()
MyServer2 = Server_in( env,"Server B")
MyServer3 = Server_in( env,"Server C")
MyServer1 = Server_inout( env,"Server A",MyServer2,MU2,MyServer3,MU3)
MyJobGenerator = JobGenerator( env, MyServer1,MU1,LAMDA )
env.run( until = MAXSIMTIME )
MyServer1.compute_numberofJob(0)
MyServer1.compute_varofJobSystem(0)
MyServer1.compute_probilityofNJobs(N,0)
Jobsys=MyServer1.numberofJobsystem
Varsys=MyServer1.varofJobSystem
ProbNjob=MyServer1.probabilityofNJob
```

2.9.2 Transient removal

```
    ''' be used to creat M replication MM1 '''

listEnv=list()
listJobGeneration=list()
listServer1=list()
listServer2=list()
listServer3=list()

''' Mean across replication '''
meanJnumberofJob=list()
''' Mean of last n-1 observations '''
meanLnumberofJob=list()
''' Relative change '''
meanRelativeChange=list()
i=0
while i<M:
    listEnv.append(simpy.Environment())
    listServer2.append(Server_in(listEnv[i],"Server B"))
    listServer3.append(Server_in(listEnv[i],"Server C"))
    listServer1.append(Server_inout(listEnv[i],"Server A",listServer2[i],MU2,listServer3[i],MU3))
    listJobGeneration.append(JobGenerator(listEnv[i],listServer1[i],MU1,LAMDA))
    env=listEnv[i]
    env.run( until = MAXSIMTIME )
    listServer1[i].compute_numberofJob(0)
    listServer1[i].compute_varofJobSystem(0)
```

```
    listServer1[i].compute_probilityofNJobs(N,0)
    i+=1
''' determind min of length M replication '''
min_=len(listServer1[0].system)
i=1
while i<M:
    min_=min(min_,len(listServer1[i].system))
    i+=1
''' transient removal :
number of job system
Variance of the number of jobs in the system
Probability of 3 or more jobs in the system
'''
''' compute MeanJ'''
i=0
while i< min_:
    j=0
    sumJob=0.0
    sumTime=0.0
    while j<M:
        sumJob+=listServer1[j].system[i].job_len
        sumTime+=listServer1[j].system[i].time
        j+=1
    meanJnumberofJob.append(state(float(sumJob/M),float(sumTime/M)))
    i+=1
'''compute Overall Mean'''
i=0
sumJob=0.0
while i<min_:
    sumJob+=meanJnumberofJob[i].job_len
    i+=1
overallMeanJob=float(sumJob/min_)
''' compute MeanL'''
i=0
while i< min_:
    j=i
    sumJob=0.0
    sumTime=0.0
    while j<min_:
        sumJob+=meanJnumberofJob[j].job_len
        sumTime+=meanJnumberofJob[j].time
        j+=1
    meanLnumberofJob.append(state(float(sumJob/(min_-i)),meanJnumberofJob[i].time))
    i+=1
'''compute relative change '''
i=0
while i< min_:
    meanRelativeChange.append(state(float((meanLnumberofJob[i].job_len-overallMeanJob)/overallMeanJob),meanLnumb
    i+=1
''' determind knee '''
tan=list()
i=0
while i<min_-1:
```

```
    tan.append(state((meanRelativeChange[i+1].job_len-meanRelativeChange[i].job_len),i))
    if (tan[i].job_len<Tan):
        break
    i+=1
k=i
'''
recomputing with knee = k
number of job system
Variance of the number of jobs in the system
Probability of 3 or more jobs in the system
'''
MyServer1.system=meanJnumberofJob
MyServer1.compute_numberofJob(k)
MyServer1.compute_varofJobSystem(k)
MyServer1.compute_probilityofNJobs(N,k)
Jobsys1=MyServer1.numberofJobsystem
Varsys1=MyServer1.varofJobSystem
ProbNjob1=MyServer1.probabilityofNJob
```

2.9.3 Terminating simulations

```
    ''' Terminating simulation'''
''' Mean for each replication from knee k'''
MeanReplication=list()
i=0
j=k
while i<M:
    sumjob=0
    j=k
    while j<len(listServer1[i].system):
        sumjob+=listServer1[i].system[j].job_len
        j+=1
    MeanReplication.append(float(sumjob/(len(listServer1[i].system)-k)))
    i+=1
''' Mean all replication '''
i=0
sumjob=0
while i<M:
    sumjob+=MeanReplication[i]
    i+=1
MeanallReplication=sumjob/M
''' Variance of replicate means '''
i=0
VarofReplication=0
while i<M:
    VarofReplication+=(MeanReplication[i]-MeanallReplication)**2
    i+=1
VarofReplication=VarofReplication/(M-1)
''' Confidence interval for the mean '''
z=-ss.norm.ppf(alpha/2)
x=MeanallReplication-z*math.sqrt(VarofReplication)/math.sqrt(M-1)
y=MeanallReplication+z*math.sqrt(VarofReplication)/math.sqrt(M-1)
```

2.9.4 Show outcome

```
''' print statistics '''
RHO1 = LAMDA/MU1
RHO2 = LAMDA*0.3/MU2
RHO3 = LAMDA*0.7/MU3
RHO4= LAMDA/(MU2+MU3)
print('Initial removal : %.2f'%k)
print('%s'%MyServer1.name)
print( 'JobsDone            : %d' % (MyServer1.jobsDone) )
print( 'Utilization         : %.2f/%.2f'
    % (1.0-MyServer1.idleTime/MAXSIMTIME, RHO1) )
print( 'Mean waiting time   : %.2f/%.2f'
    % (MyServer1.waitingTime/MyServer1.jobsDone, RHO1**2/((1-RHO1)*LAMDA) ) )
print( 'Mean service time   : %.2f/%.2f'
    % ((MAXSIMTIME-MyServer1.idleTime)/MyServer1.jobsDone, 1/MU1) )

print( 'Mean number of Jobs queue: %.2f/%.2f'% (MyServer1.numberofqueue,RHO1**2/(1-RHO1)))
print( 'Mean number of Jobs system: %.2f/%.2f/%.2f'% (Jobsys,Jobsys1,RHO1/(1-RHO1)))
print( 'Variance of the number of jobs in the system : %.2f/%.2f/%.2f'%
     (Varsys,Varsys1,RHO1/((1-RHO1)**2)))
print( 'Probability of %d or more jobs in the system : %.2f/%.2f/%.2f'%
     (N,ProbNjob,ProbNjob1,RHO1**N))
print( 'Variance of %d replication means : %.2f '%(M,VarofReplication))
print( 'Confidence interval for the mean with 1-alpha = %.2f : [%.2f : %.2f]'%(1-alpha,x,y))
print('%s'%MyServer2.name)
print( 'JobsDone            : %d' % (MyServer2.jobsDone) )
print( 'Utilization         : %.2f/%.2f' % (1.0-MyServer2.idleTime/MAXSIMTIME,RHO2) )
print( 'Mean waiting time   : %.2f/%.2f'
    % (MyServer2.waitingTime/MyServer2.jobsDone ,RHO2**2/((1-RHO2)*MU1*0.3)) )
print( 'Mean service time   : %.2f/%.2f'
    % ((MAXSIMTIME-MyServer2.idleTime)/MyServer2.jobsDone, 1/MU2) )
print('%s'%MyServer3.name)
print( 'JobsDone            : %d' % (MyServer3.jobsDone) )
print( 'Utilization         : %.2f/%.2f'
    % (1.0-MyServer3.idleTime/MAXSIMTIME,RHO3) )
print( 'Mean waiting time   : %.2f/%.2f '
    % (MyServer3.waitingTime/MyServer3.jobsDone ,RHO3**2/((1-RHO3)*MU1*0.7)) )
print( 'Mean service time   : %.2f/%.2f'
    % ((MAXSIMTIME-MyServer3.idleTime)/MyServer3.jobsDone, 1/MU3) )
print ('Total system')
print ('Mean utilization =
     %.2f'%((3*MAXSIMTIME-MyServer1.idleTime-MyServer2.idleTime-MyServer3.idleTime)
/(3*MAXSIMTIME)))
print ('Job departure = %d'%(MyServer2.jobsDone+MyServer3.jobsDone))
print ('Mean waiting time of Total System =
     %.2f'%(float((MyServer1.waitingTime+MyServer2.waitingTime
+MyServer3.waitingTime)/(MyServer2.jobsDone+MyServer3.jobsDone))))
print ('Mean service time of Total System =
     %.2f'%(float((3*MAXSIMTIME-MyServer1.idleTime-MyServer2.idleTime
```

```
-MyServer3.idleTime)/(MyServer2.jobsDone+MyServer3.jobsDone))))
i=0
while i<len(meanLnumberofJob):
    if LOGGED:
        qlog.write( '%.6f\t%.6f\n'% (meanJnumberofJob[i].job_len,meanLnumberofJob[i].time ))
    i+=1
if LOGGED:
    qlog.close()
x=list()
y=list()
i=0
while i<len(meanJnumberofJob):
    x.append(meanJnumberofJob[i].time)
    y.append(meanJnumberofJob[i].job_len)
    i+=1
plt.plot(x,y)
plt.xlabel('Time')
plt.ylabel('MeanJ Job in System')
plt.show()
x=list()
y=list()
i=0
while i<len(meanLnumberofJob):
    x.append(meanLnumberofJob[i].time)
    y.append(meanLnumberofJob[i].job_len)
    i+=1
plt.plot(x,y)
plt.xlabel('Time')
plt.ylabel('MeanL Job in System')
plt.show()
x=list()
y=list()
i=0
while i<len(meanRelativeChange):
    x.append(meanRelativeChange[i].time)
    y.append(meanRelativeChange[i].job_len)
    i+=1
plt.plot(x,y)
plt.xlabel('Time')
plt.ylabel('MeanR Job in System')
plt.show()
```

2.10 Present Results

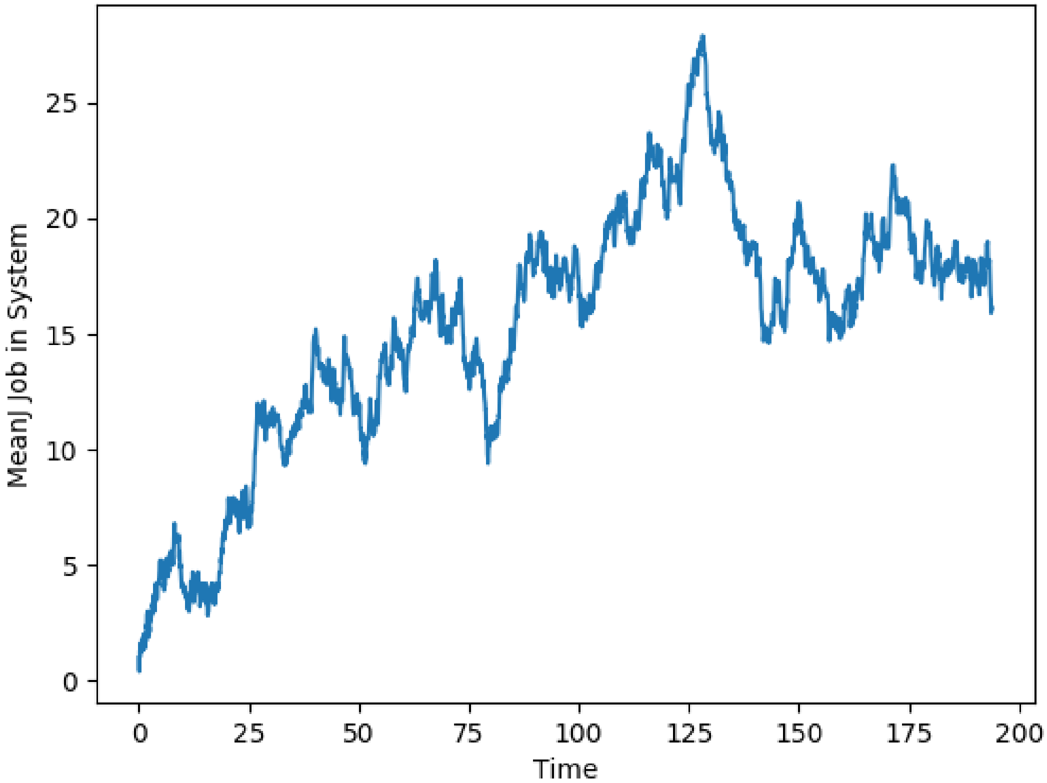
```
   Initial removal : 578.00
Server A
JobsDone             : 1476
Utilization          : 0.92/0.94
Mean waiting time    : 0.75/1.88
Mean service time    : 0.12/0.12
Mean number of Jobs queue: 5.57/14.06
```

```
Mean number of Jobs system: 6.05/16.79/15.00
Variance of the number of jobs in the system : 24.07/11.88/240.00
Probability of 2 or more jobs in the system : 0.80/0.77/0.88
Variance of 10 replication means : 53.73
Confidence interval for the mean with 1-alpha = 0.95 : [12.60 : 22.17]
Server B
JobsDone             : 500
Utilization          : 0.64/0.60
Mean waiting time    : 0.52/0.38
Mean service time    : 0.25/0.25
Server C
JobsDone             : 974
Utilization          : 0.58/0.70
Mean waiting time    : 0.20/0.29
Mean service time    : 0.12/0.12
Total system
Job departure = 1474
Mean utilization = 0.71
Mean waiting time of Total System = 1.05
Mean service time of Total System = 0.29
```

- Mean trajectory by averaging across replications

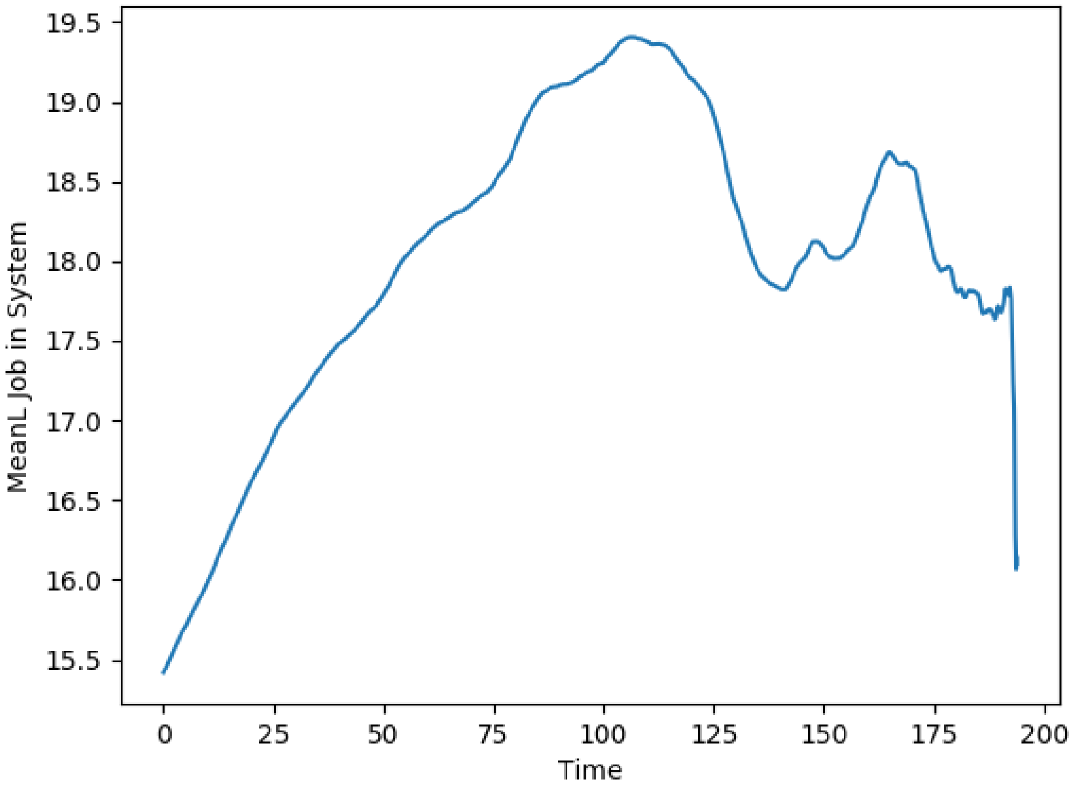
$$\overline{x_j} = \frac{1}{m}\sum_{i=1}^{m} x_{ij}$$

with $j = 1, 2, ..., n$



-Delete the first l observations and get an overall mean from the remaining n l values

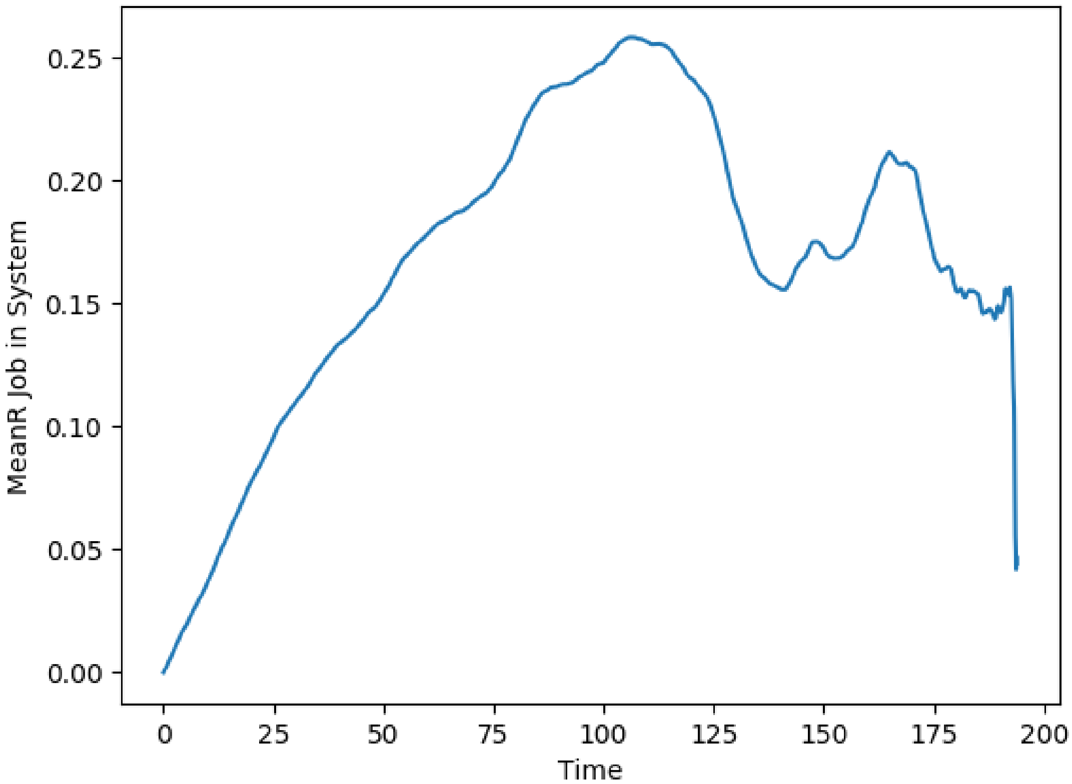
$$\overline{\overline{x_l}} = \frac{1}{n-l} \sum_{i=l+1}^{n} x_j$$



- Compute the relative change

$$\overline{\overline{x_r}} = \frac{\overline{\overline{x_l}} - \overline{\overline{x}}}{\overline{\overline{x}}}$$

$$\text{with } \overline{\overline{x}} = \frac{1}{n}\sum_{i=1}^{n} x_j$$



Validation:

		Before Transient removal	After Transient removal	Analyst model
	Initial removal	578		
Server A	Jobsdone	1476		
	Utilization	0.92		0.94
	Mean waiting time	0.75		1.88
	Mean service time	0.12		0.12
	Mean number of Job queue	5.57		14.06
	Mean number of Job system	6.05	16.79	15
	Variance of the number of Jobs in System	16.79	11.88	240
	Probability of 2 or more Jobs in the System	0.8	0.77	0.88
	Confidence interval for mean with 1-alpha=0.95		[12.6 : 22.17]	
Server B	Jobsdone	500		
	Utilization	0.64		0.6
	Mean waiting time	0.52		0.38
	Mean service time	0.25		0.25
Server C	Jobsdone	974		
	Utilization	0.58		0.7
	Mean waiting time	0.2		0.29
	Mean service time	0.12		0.12
Total System	Jobs departure	1474		
	Mean utilization	0.71		
	Mean waiting time of Total System	1.05		
	Mean service time of Total System	0.29		

Nhận xét: Dựa vào bảng trên ta thấy: Trước khi thực hiện kĩ thuật Transient removal thì kết quả của Simulation model khá gần với Analyst model,nhưng sau khi thực hiện kĩ thuật trên thì với việc loại bỏ được 578 trạng thái đầu tiền chưa ổn định của meanJobofSystem thì kết quả lại rất gần với Analyst model. Đặc biệt , phương sai của Simulation model sau khi thực hiện kĩ thuật Transient Removal (11.88) nhỏ hơn nhiều so với Simulation model trước khi Transient Removal (16.79) và Analyst model (240).