Sudan University of Science and Technology

Collage of Graduate Studies

College of Engineering

Scheduling Algorithm for Grid Computing using Shortest Job First with Time Quantum

خوارزمية الجدولة للحوسبة الشبكية بإستخدام العمليات الأقصر أولا مع الزمن الكمي

A thesis submitted in partial fulfillment of the requirements for the degree of M.SC.in computer and networks engineering

By:

Raham Hashim Yosuf

Supervised by:

Dr. Rania A. Mokhtar

August 2017

Initiation

الأية

قال الله تعالى:

{أَمَّنْ هُوَ قَانِتٌ آنَاءَ اللَّيْلِ سَاجِدًا وَقَائِمًا يَحْذَرُ الْآخِرَةَ وَيَرْجُو رَحْمَةَ رَبِّهِ قُلْ هَلْ يَسْتَوِي الَّذِينَ يَعْلَمُونَ وَالَّذِينَ لَا يَعْلَمُونَ إِنَّمَا يَتَذَكَّرُ أُولُو الْأَلْبَابِ}

صدق الله العظيم

سورة الزمر الآية (9)

Dedication

Dedicated to

My parents, my sisters

And to my friends

To everyone who tried to guide me to

A better life…..

With my love

Acknowledgement

My thanks for Allah

After that

I would like to thanks my university Sudan University of science and Technology and my college of Graduate Studies Electronics Engineering

Department. And my teachers for inspiring me this project.

I owe my profound gratitude to my thesis supervisor Dr. Rania A.Mokhtar for her valuable guidance, supervision and persistent encouragement. Due to the approach adopted by her in handling my thesis and the way she gave me freedom to think about different things,

I was able to do the constructive thesis. By working under her I have gained priceless knowledge as to how to go about doing an effective research. My greatest thanks to my parents who for their continuous support.

Abstract

One motivation of grid computing is to aggregate the power of widely distributed resources and provide non trivial services to users. To achieve this goal an efficient grid scheduling system is an essential part of the grid. The scheduling of the processes can be implemented in different ways with different scheduling algorithms. In this project we use Shortest Job First [SJF] algorithm where it assigns highest priority to shortest process. If processes are long, it suffers with a long waiting time. When shortest process continues to arrive, the longer process may never get a chance to schedule, this problem is called as Starvation. This research proposes a use of Shortest Job First with time quantum mechanism to overcome this issue. The MATLAB software is used to implement the proposed algorithm, the algorithm achieves better results than the original one in terms of delay time, resources utilization, fair treatment for all the processes and degree of starvation.

ملخص البحث

الهدف الأساسي من الحوسبة الشبكية هو الإستفاده القصوى من المصادر الموزعة وتقديم الخدمات غير البديهية للمستخدمين ولتحقييق هذا الهدف فإن نظام الجدولة هو جزء ضروري في الحوسبة الشبكية لأنه يعمل على جدولة العمليات بطرق مختلفة بناءا على العديد من الخوارزميات وإحدى هذه الخوارزميات هي خوارزمية تنفيذ العمليات القصيرة أولًا والعمليات الأطول تعاني من الإنتظار في الصف إلى أن يتم تنفيذ العمليات القصيرة. وعليه إذا إستمرت العمليات القصيرة في الوصول فإن العمليات الطويلة لا تستطيع الدخول في الجدولة حتى يتم تنفيذها وتسمى هذه المشكلة بالتجويع ويقدم هذا البحث حلاً لهذه المشكلة عبر خوارزمية العملية التي لها زمن تنفيذ أقل مع الزمن الكمي وتعمل هذه الخوارزمية على إختيار أسرع مصدر لتنفيذ العمليات التي لها زمن تنفيذ أقل ثم بعد أن يتم تنفيذها تقوم بتحرير المصدر وذلك إذا كان زمن تنفيذها أقل من أو يساوي الزمن الكمي، أما إذا كان زمن تنفيذها أكبر من الزمن الكمي فإنها تنفذ جزئياً ومن ثم يتم تقسيمها وتوزيعها علي المصادر الحرة الاخرى، أما إذا كانت كل المصادر مشغولة فإنها تحجب وترجع الى اخر الصف وتستمر عملية التنفيذ إلى أن تنفذ كل العمليات الموجودة في صف الإنتظار.

TABLE OF CONTENTS

List of workspace

List of Figures

List of Tables

Abbreviations

ABDRS:	Agent Based Dynamic Resource Scheduling
BAJGS:	Bandwidth Aware Job Grouping based Scheduling
B.T:	Burst Time
CPU:	Central Processing Unit
DJGBS:	Dynamic Job Grouping Based Scheduling
D-MMLQ:	Deadline based Modified Multi Level Queue
DR:	Data Rate
DT:	Delay Transmission
EDF:	Earliest Deadline First
EDSRT:	Earliest Deadline First with Shortest Remaining Time
ERD:	Earliest Release Date
FCFS:	First Come First Service
FF:	First Fit
GBFJS:	Grouping Based Fire grained Job Scheduling
GIS:	Grid Information Service
HJS:	Hierarchical Job Scheduling
HRN:	Highest Response Next
LJF:	Longest Job First
MFQS:	Multilevel Feedback Queue Scheduling
ORC:	Optimal Resource Constraint
RR:	Round Robin
SCTP:	Stream Control Transmission Protocol
SF:	Starvation Free
SFBAJG:	Scheduling Framework for Bandwidth Aware Job Grouping
SJF:	Shortest Job First
SPN:	Shortest Process Next
T.Q:	Time Quantum
VCGRP:	Virtual Computing Grid using Resource Pooling

Chapter one

Introduction

Chapter one

Introduction

1.1 Preface

Grid computing has emerged as a distributed methodology that coordinates the resources that are spread in the heterogeneous distributed environment[1].Grid computing is a type of parallel and distributed system that enables the sharing, selection and aggregation of resources distributed across multiple administrative domains based on their availability, capability, performance, cost and user's quality-of-service requirements[2].Resources can be computers, storage space, instruments, software applications, channels and data, all connected through the Internet and a middleware software layer that provides basic services for security, monitoring and resource management. Resources owned by various administrative organizations are shared under locally defined policies that specify what is shared, who is allowed to access what, and under what conditions.

In most organizations, there are large amounts of underutilized Computing resources. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage. Grid computing looks like a great answer to numerous problems without prodigious management requirements [3].

Effective grid computing is possible, however, only if the resources are scheduled well, the goal of the resource scheduler is to maximize the resource utilization and minimize the processing time of the jobs.

The job scheduling system is responsible to select best suitable machines in a grid for user jobs. Schedulers allocate resources to the jobs to be executed using various algorithms such as First Come First Service [FCFS], Longest Job First [LJF], Earliest Deadline First [EDF] and other algorithms. One type of these algorithms is shortest job first [SJF] also know as Shortest Job Next [SJN] or Shortest Process Next [SPN] the scheduling technique that selects the smallest job.

If process is long, then this long process are suffers with a long waiting time. When shortest jobs continue to arrive, longest jobs may never get a chance to schedule. This problem is called as Starvation. The project proposes a solution to overcome this issue with time quantum mechanism. SJF with time quantum is based on the integration of round robin and SJF scheduling algorithm. In this, priority is calculated on the basis of shortest process and time quantum.

1.2 Problem Statement

In SJF algorithm the resource is allocated to the smallest process. If process is long, then this long process are suffers with a long waiting time. If short Processes are always available to run; the long processes ones never get chance to scheduled. This problem is called starvation.

1.3 Proposed solution:

The proposed solution is algorithm called shortest job first with time quantum which is based on the integration of round robin [RR] and SJF scheduling algorithm. This algorithm can overcome the starvation. In this algorithm the resource is allocated to the shortest process. In this two things can happen. First, the process may be less than or equal to the time quantum. In this case, the process will execute and after completion release the resource by itself. Second, the process may be greater than to the time quantum. In this case, the process will execute for one time quantum, divided and distributed to the free second resource, if resources are busy the process is preempted. Then, the resource scheduler will select the next shortest process to execute. The preempted process will be put at the ready queue. This continues until the execution of all the processes is completed.

1.4 objectives:

- ❖ To optimize the utilization of resources.
- ❖ To reduce end time of job.
- ❖ To enhance performance of job scheduling.
- ❖ To fair treatment for all the processes

1.5 Scope:

The scope of this project is to develop the approach for Shortest Job First scheduling algorithm by using time quantum to reduce the problem of starvation.

1.6 Methodology:

The proposed algorithm is shortest job first [SJF] with time quantum which is based on the integration of round robin and Shortest Job First scheduling algorithms. Firstly select fastest resources to allocate the shortest process, if the process less than or equal to the time quantum. In this case, the process will execute and after completion release the resource by itself. Secondly if process is greater than to the time quantum the Process will execute for one time quantum divided and distributed to the free second resource. If resources are busy the process is pre empted.

Finally the resource scheduler will select the next shortest Process to execute. That means the scheduler select the pre empted process because it is the shortest process at the ready queue. The scheduler continues to select processes until the execution of all the processes is completed. The result is calculated using MATLAB.

1.7 Thesis outlines:

The thesis includes five chapters, chapter one provides introduction of the project, determine the problem statement, proposed solution and objectives. While chapter two covers the literature review which includes theoretical background algorithms that have the relevant related work.

In chapter three the methodology section, Explain the algorithms how it implemented by MATLAB while chapter four presents the result of implementation; furthermore, it also compares the results of the proposed algorithm with other scenario of similar work. And chapter five includes the conclusion and future work.

Chapter two

Literature review

Chapter two

Literature review

2.1 Previous work:

The job scheduling system is responsible to select best suitable machines in a grid for user jobs. Schedulers allocate resources to the jobs to be executed using various algorithms. This chapter describes the algorithms and determines the difference between them.

In [4] one of the main challenges is resource or job scheduling in the grid. They present such an algorithm which helps in scheduling computational resource to the jobs in efficient way.

They use Earliest Deadline First with Shortest Remaining Time. [EDSRTF] is a scheduling algorithm in which scheduling is done according to the deadline of jobs and remaining time. The job with the earliest deadline will get resource first while the job with large value of deadline will have to wait irrespective of their execution time. by Comparison of the proposed algorithm EDSRTF in terms of waiting and turnaround time of jobs with two other algorithms First Come First Serve [FCFS] and Longest Job First [LJF]. Average waiting time and average turnaround time of proposed algorithm is less in comparison to other scheduling algorithms so EDSRTF is best among them.

In [5] the main aim is to develop the new approach for SJF scheduling algorithm which help to reduce the problem of starvation in a heavily loaded computer system. The ASJF algorithm reduces the starvation Problem of the simple SJF architecture. This ASJF algorithm is based on

SJF and multilevel feedback queue scheduling [MFQS] technique**.** They made a comparative study of SJF and ASJF algorithm. It is concluded

That the ASJF algorithm is superior in terms of minimizing degree of starvation, increasing fairness, decrease in response time and Timely resource allocation to individual process**.** ASJF algorithm clearly shows maximum CPU utilization and efficient handling of resources. As

MFQS algorithm is merged with SJF, the technique of dividing the processes to various queues and switching of processes among them will further reduce the problem of starvation.

In [6] the proposed work is about to keep only the most required data items in cache. The requirement depends on the frequency of the reuse of data item. The work is about to identify the frequency of each data items from previous history.

The improvement of algorithm using cache technique allows us to reduce computational cost because the results calculated in previous times not to be calculated in the progressive process.

In [7] they propose a scheduling technique which classifies the subtasks based on the priority assigned by the user. Multi-level queue is used for reducing starvation. The proposed Starvation Free [SF] Scheduling algorithm can be applied widely and it helps in scheduling resources efficiently, resulting in a starvation free grid environment.

Results from simulation experiments demonstrate that the algorithm optimizes the resource nodes and resource utilization rate gets substantial increase. The multi-level queue based scheduling Algorithm for the heterogeneous grid environment has high performance as compared to the other scheduling algorithms for the grid environment.

In [8] the proposed solution for scheduling the jobs using Deadline based- Modified Multi-Level Queue [D-MMLQ] for multiprocessor scheduling. The proposed algorithm fuses two vital concepts for Handling job allocation and execution through multi-level queue. The approach proposes that the starvation problem of low priority jobs or jobs at lower end of queue, hence increasing the overall competence of multiprocessor system. D-MMLQ algorithm is the better utilization of resources than traditional EDF algorithm.

In [9] he is interested in two distinct functionalities: global scheduling and local scheduling. He considers a model composed of global scheduler with its global queue and local scheduler also with its queue. The model focuses in increase the resource utilization at global queue and decrease makespane time at local queue. The objective of this work is to investigate the mechanisms of scheduling problems in grid computing and to find whether scheduling algorithms used at global and local queue are similar.

In [10] an algorithm is designed for an efficient job scheduling algorithm to maximize the resource utilization and minimize processing time of the jobs. They have proposed an efficient three scheduling algorithm [Shortest Job First, First Come First Serve, Round Robin] for jobs and

Send to the queue. They have got some better performance in terms of processing time than job scheduling on the FIFO algorithm. Also they have implemented Shortest Job First algorithm along with all three algorithms for performance analysis.

In [11] they are present an improved algorithm based on priority scheduling which further reduces total time and hence maximizes resource utilization. It is based upon grouping of algorithms on the basis

Of priority. And proposed an efficient four scheduling algorithm [Shortest Job First, First Come First Serve, Round Robin and priority

Scheduling] for jobs and send to the queue to minimize processing time of the jobs.

In [12] they are proposing a backfilling scheduling algorithm is proposed to select many jobs to be backfilled from the waiting job queue. The Main aim of their proposed work is to make develop scheduling jobs based on priority using backfilling for grid computing.

In [13] they are proposing a backfilling scheduling algorithm is proposed to select many jobs to be backfilled from the waiting job queue. The Main aim of their proposed work is to make develop scheduling jobs based on priority using backfilling for grid computing.

Table 2.2 summary of related works

NO	Authors	Paper Title and Year	Algorithms	Overview and Results
4	Dipti Sharma, Mr. Pradeep Mittal	Job Scheduling Algorithm for Computational Grid in Grid Computing Environment,2013	EDSRTF	The scheduling done according to the deadline of jobs and remaining time. Average waiting time and average turnaround time is less in comparison to other scheduling algorithms
8	Manupriya Hasija, Akhil Kaushik	D-MMLQ Algorithm for Multi-level Queue Scheduling, 2014	D-MMLQ	The algorithm Handling job allocation and execution through multi-level queue. it is better utilization of Resources than EDF
9	Mohamed Eisa	Improving Grid Computing Scheduling using Heuristic Algorithms,2013	global scheduling and local scheduling	It focuses in increase the resource utilization at global queue

No	Authors	Paper Title and Year	Algorithm	Overview and Results
10	Pinky Rosemarry, Ravinder Singh, Payal Singhal and Dilip Sisodia	Grouping Based job scheduling algorithm using priority queue and hybrid algorithm in grid Computing, 2012	SJF,FCFS,RR	It used three algorithms to maximize the resource utilization and minimize processing time of the jobs.
13	Sandip Fakira Lokhande1, Sachin D. Chavhan, Prof. S. R. Jadhao,	Grid Computing Scheduling Jobs Based on Priority using Backfilling, 2015	backfilling scheduling algorithm	To select many jobs to be backfilled from the waiting job queue. the aim is to make develop scheduling jobs based on priority

Chapter three

Methodology

Chapter three

Methodology

3.1 Introduction

This section covers techniques, tools which are chosen for calculation delay time, end time of process and resources utilization.

3.2 System algorithm:

The algorithm is Shortest Job First [SJF] with time quantum. In this algorithm the channel is allocated to the shortest process. In this two things can happen. First, if process less than or equal to the time quantum, then the process will be execute and after completion release the channel by itself. Second. If process greater than time quantum, then the process will execute for 1 time quantum, divided and distributed to the free second channel, if channels are busy the process is preempted. Then, the channel scheduler will select the next shortest Process to execute. The preempted process will be put at the ready queue. This continues until the execution of all the processes is completed. This project is executed by MATLAB because it is a high performance language for technical computing. It integrates computation, visualization and programming environment. Furthermore MATLAB is modern programming language environment it has sophisticated data structures, contains built in editing and debugging tools and supports object oriented programming. These factors make MATLAB excellent tools for teaching and research. MATLAB has

Many advantages Compared to the conventional computer languages for solving technical problems.

3.3 Mathematical model

The Mathematical Equations that’s used in simulation in MATLAB are:

3.3.1 Data rate [DR]

Average number of bits or data passing through a communication link in a data transmission system from one channel to another .common data rate units are multiples of bits per second (bit/S) and byte per second(B/S).

3.3.2 Delay transmission [DT]:

Amount of time required to push all of the packets bits onto the channel.

Delay transmission can be mathematically expressed Equation (3.1) calculate the Delay transmission [DT]

$$\mathrm{DT} = \frac{\mathrm{data}}{\mathrm{DR}} \qquad \text{Equation} \quad (3.1)$$

Where*:*

DT = Delay transmission in (sec).

Data = data transmit in (bit).

DR = Data rate in (bit/sec).

3.3.3 End time:

The actual time at which a job finishes its processing unit (second).

3.3.4 Waiting time:

It is the amount of time a process waits in the ready queue unit (second).

3.3.5 Channel utilization:

The maximum use of channel when it is busy.

3.3.5 Time Quantum:

Time gives to job and interrupting the job if it is not completed by then

3.4 Sudo code:

```
3.4.1Select channel.
3.4.2 Select shortest process from the ready queue.
3.4.3Check, if process <= Time quantum (T.Q)
3.4.3. A If yes, then scheduler will allocate the channel to that process, the process will complete its execution and will release the channel by itself.
3.4.3. B If no, then scheduler will allocate the channel to the process; the process will execute for one T.Q, divided and distributed to the free second channel
3.4.4 If channels are busy the process is preempted.
3.4.5 Repeat step 3, if any process is available in ready queue.
3.4.6 Exit
```

Chapter four

Result and Discussion

Chapter four

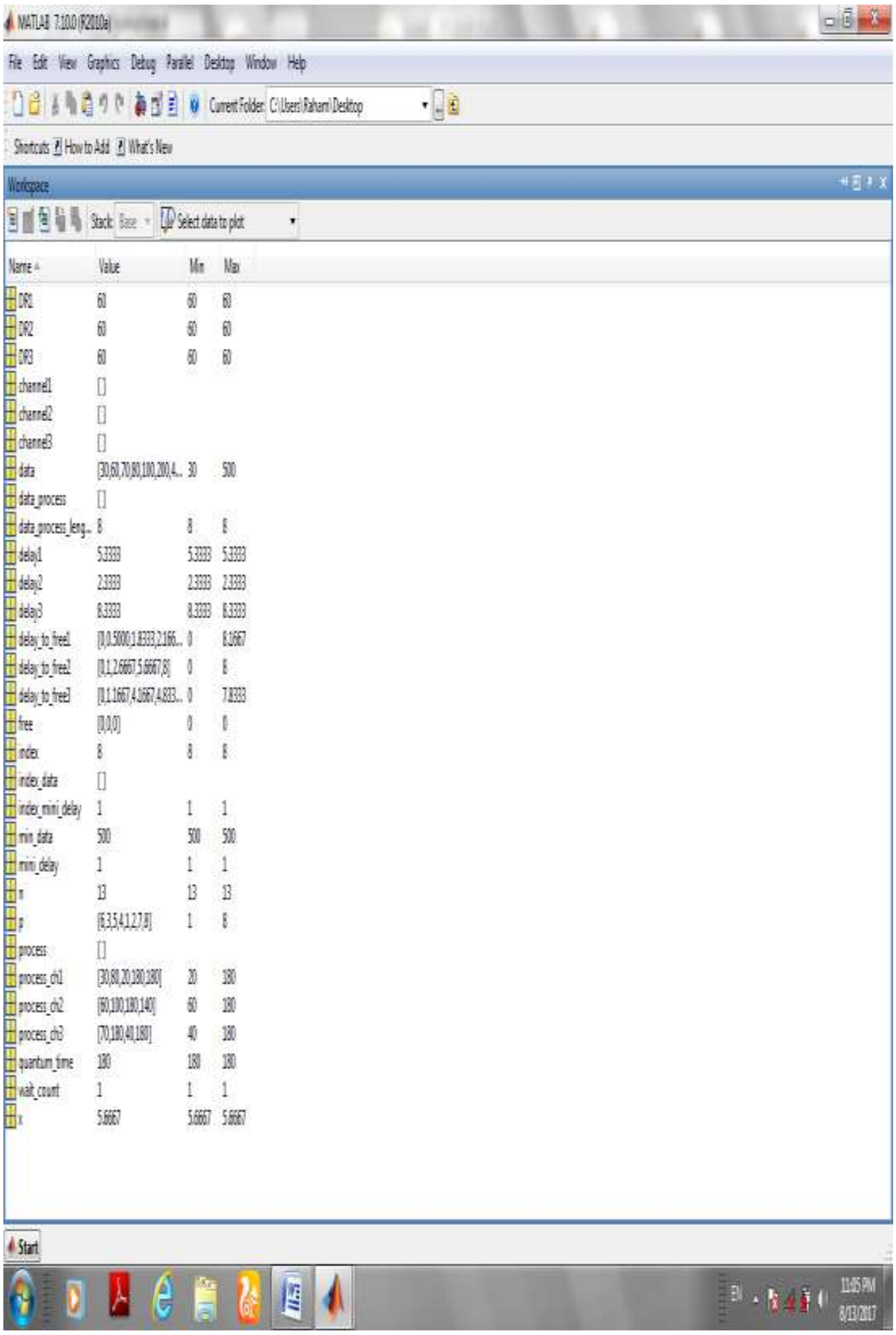
Result and Discussion

4.1 Introduction:

The project used Shortest Job First with time quantum and Shortest Job First algorithms when data rate is equal [60] and data rate is different [2080 100].

Firstly arrange processes from minimum to maximum in the ready queue the processes are [30 60 70 80 100 200 400 500]; it used short process; medium process and long process; secondly calculate delay to free to determine which channel work. Finally allocate the channel that has minimum delay to the minimum process. The algorithms are used three channels to allocated processes.

Workspace 4.1.1 Shortest Job First with time quantum when data rate is equal [60] and time quantum is equal 180


MATLAB 7.10.0 (R2010a)

Current Folder: C:\Users\Rahami\Desktop

Workspace

Name	Value	Min	Max
DR1	60	60	60
DR2	60	60	60
DR3	60	60	60
channel1	[]		
channel2	[]		
channel3	[]		
data	[30,60,70,80,100,200,4...	30	500
data_process	[]		
data_process_leng...	8	8	8
delay1	5.3333	5.3333	5.3333
delay2	2.3333	2.3333	2.3333
delay3	8.3333	8.3333	8.3333
delay_to_free1	[0,0.5000,1.8333,2.166...	0	8.1667
delay_to_free2	[0,1,2.6667,5.6667,8]	0	8
delay_to_free3	[0,1.1667,4.1667,4.833...	0	7.8333
free	[0,0,0]	0	0
index	8	8	8
index_data	[]		
index_mini_delay	1	1	1
min_data	500	500	500
mini_delay	1	1	1
n	13	13	13
p	[6,3,5,4,1,2,7,8]	1	8
process	[]		
process_ch1	[30,80,20,180,180]	20	180
process_ch2	[60,100,180,140]	60	180
process_ch3	[70,180,40,180]	40	180
quantum_time	180	180	180
wait_count	1	1	1
x	5.6667	5.6667	5.6667

This workspace [4.1.1] explains the algorithm [SJF] using time quantum. Data rate value equal 60, time quantum equal 180, the channels are [channel1, channel2, channel3] and the process is not arrange. After arrange data all channels are free that means select minimum processes [30 60 70] to allocated channel1, channel2 and channel3. Channel 1 is became free because it is have minimum data and process 80 is allocated in it and process 100 is allocated into channel 2 after process 80 complete its execution and release channel process 200 is greater than time quantum [180] the process is divided to 180 and 20 and distributed into channel 1 and channel 3, process 400 is also greater than time quantum divided to 180, 180, 40 and distributed into channel 1,channel 2 and channel 3,process 500 is greater than time quantum divided to 180,180,140 distributed into channel 1 ,channel 3 and channel 2 .

End time of all processes that is the time when process is complete its execution and release channel. For example process 30 is start its execution at 0 and finish at 0.5000 second that means end time of execution and delay to free calculate using the difference between the waiting time and end time .

Table 4.1.1 Shortest Job First with time quantum

processes	Waiting time	End time	Delay to free
30	0	0.5000	0.5000
60	0	1	1
70	0	1.167	1.167
80	0.5000	1.833	1.333
100	1	2.667	1.667
200	1.167	4.167	3
400	2.167	5.667	3.5
500	4.833	8.167	3.33

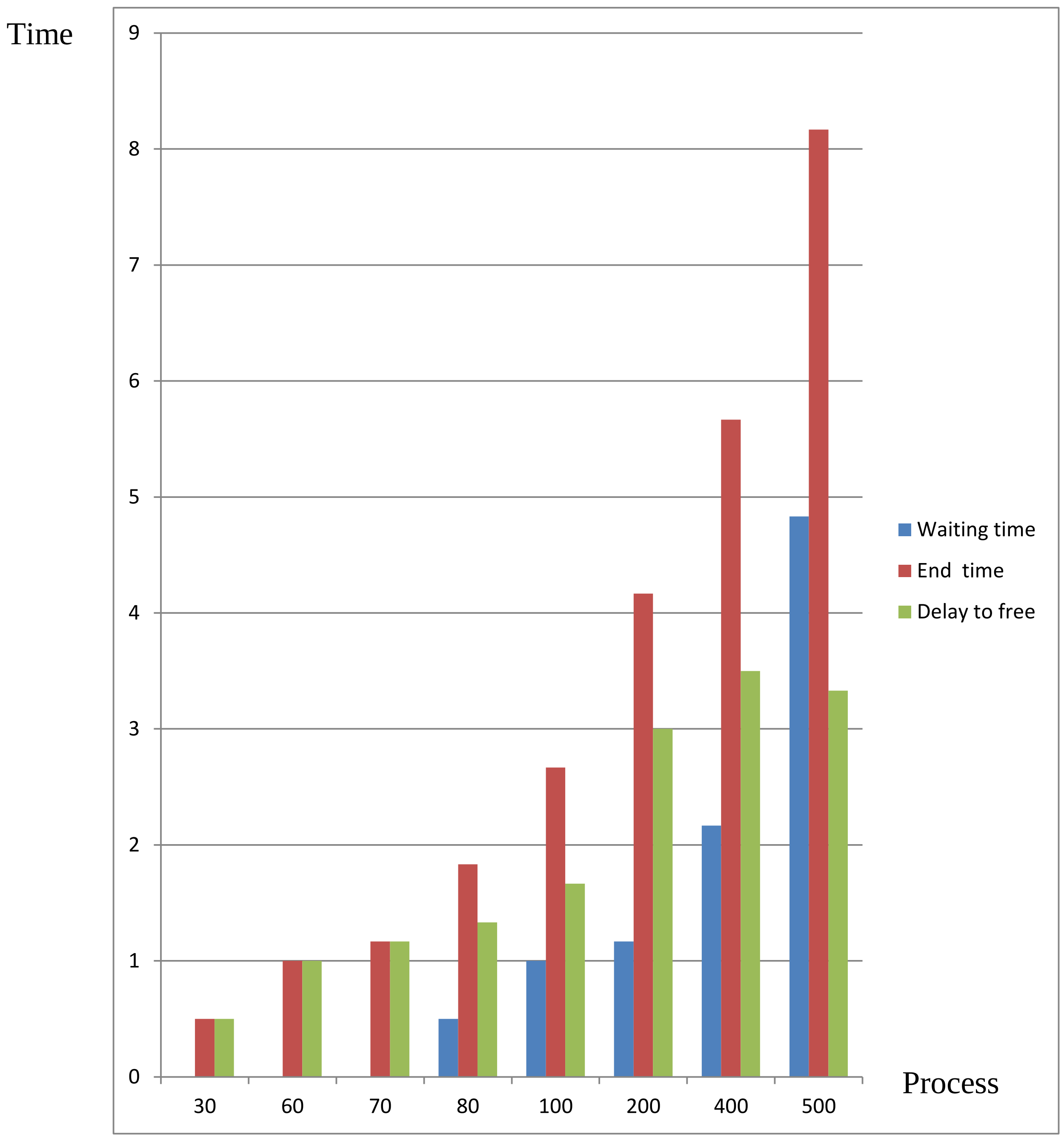


Figure 4.1 Shortest Job First with time quantum

Average waiting time =∑ waiting time of processes ∕ number of processes

0+0+0+0.5000+1+1.167+2.167+4.833/8

= $\frac{9.667}{8}$ = 1.208

Average end time= ∑end time of processes/number of processes

0.5000+1+1.167+1.833+2.667+4.167+5.667+8.167

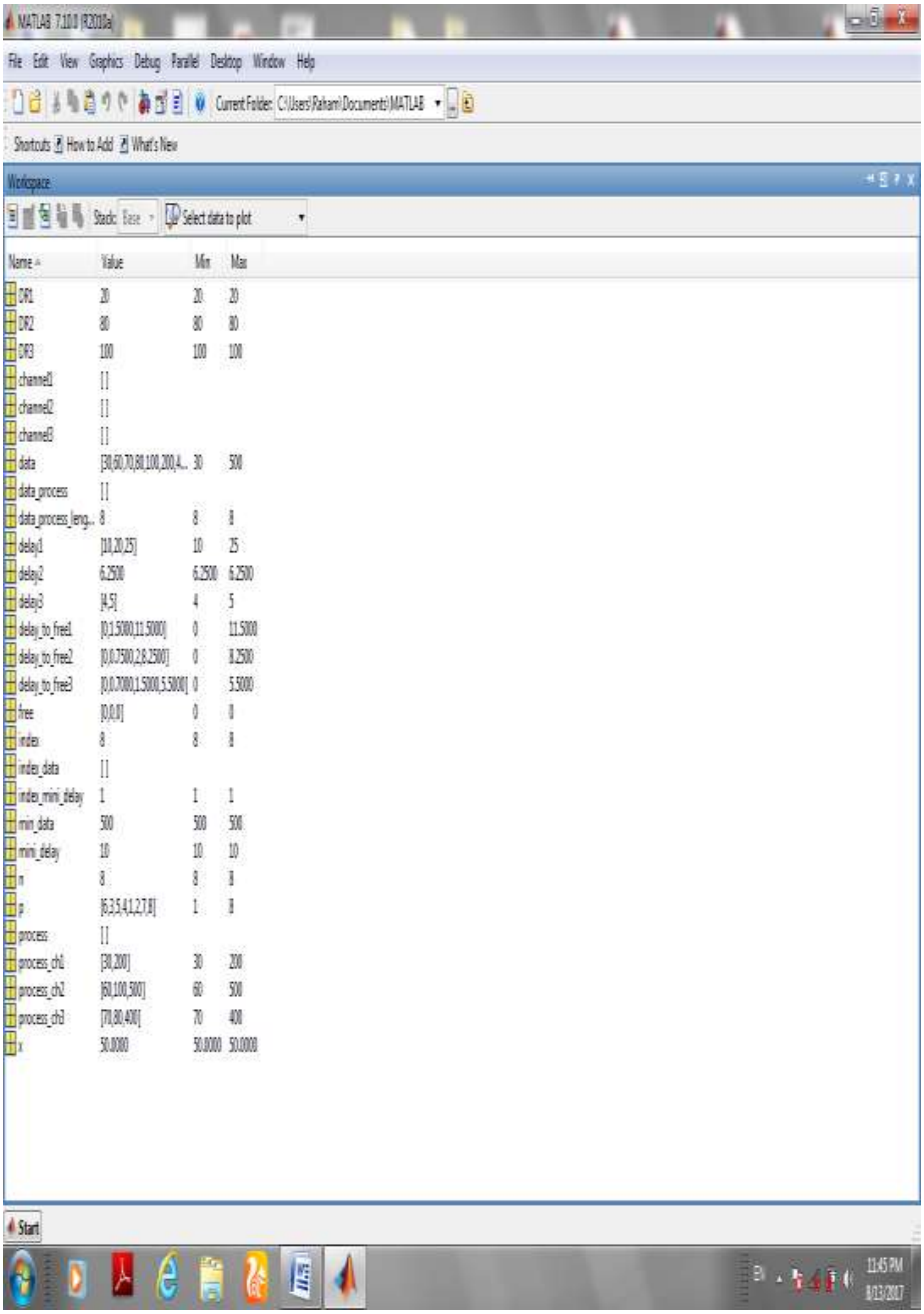
= 25.168/8 = 3.146

Average delay to free = ∑ delay to free of processes/number of processes

=0.5000+1.1.167+1.33+1.667+3+3.5+3.33

= 15.49/8 = 1.94

Workspace 4.1.2 Shortest Job First [SJF] when data rate is equal [60]



Name	Value	Min	Max
DR1	20	20	20
DR2	80	80	80
DR3	100	100	100
channel1	[]		
channel2	[]		
channel3	[]		
data	[30,60,70,80,100,200,4...	30	500
data_process	[]		
data_process_leng...	8	8	8
delay1	[10,20,25]	10	25
delay2	6.2500	6.2500	6.2500
delay3	[4,5]	4	5
delay_to_free1	[0,1.5000,11.5000]	0	11.5000
delay_to_free2	[0,0.7500,2,8.2500]	0	8.2500
delay_to_free3	[0,0.7000,1.5000,5.5000]	0	5.5000
free	[0,0,0]	0	0
index	8	8	8
index_data	[]		
index_mini_delay	1	1	1
min_data	500	500	500
mini_delay	10	10	10
n	8	8	8
p	[6,3,5,4,1,2,7,8]	1	8
process	[]		
process_ch1	[30,200]	30	200
process_ch2	[60,100,500]	60	500
process_ch3	[70,80,400]	70	400
x	50.0000	50.0000	50.0000

This workspace [4.1.2] explains the algorithm [SJF]. Data rate value equal 60, the channels are [channel1, channel2, channel3] and the process is not arrange. After arrange data all channels are free that means select minimum processes [30 60 70] to allocated channel1, channel2 and channel3. Channel 3 is become free because it is have minimum data and process 80 is allocated in it. When process complete execution it release channel by itself. That means process 400 is allocated in channel3, processes 100 and 500 is allocated in channel2 and process 200 is allocated in channel1

Table 4.1.2 Shortest Job First [SJF]

processes	Waiting time	End time	Delay to free
30	0	0.5000	0.5000
60	0	1	1
70	0	1.1667	1.1667
80	0.5000	1.833	1.333
100	1	2.667	1.667
200	1.1667	4.5000	3.333
400	1.833	8.5000	6.667
500	2.667	11	8.333

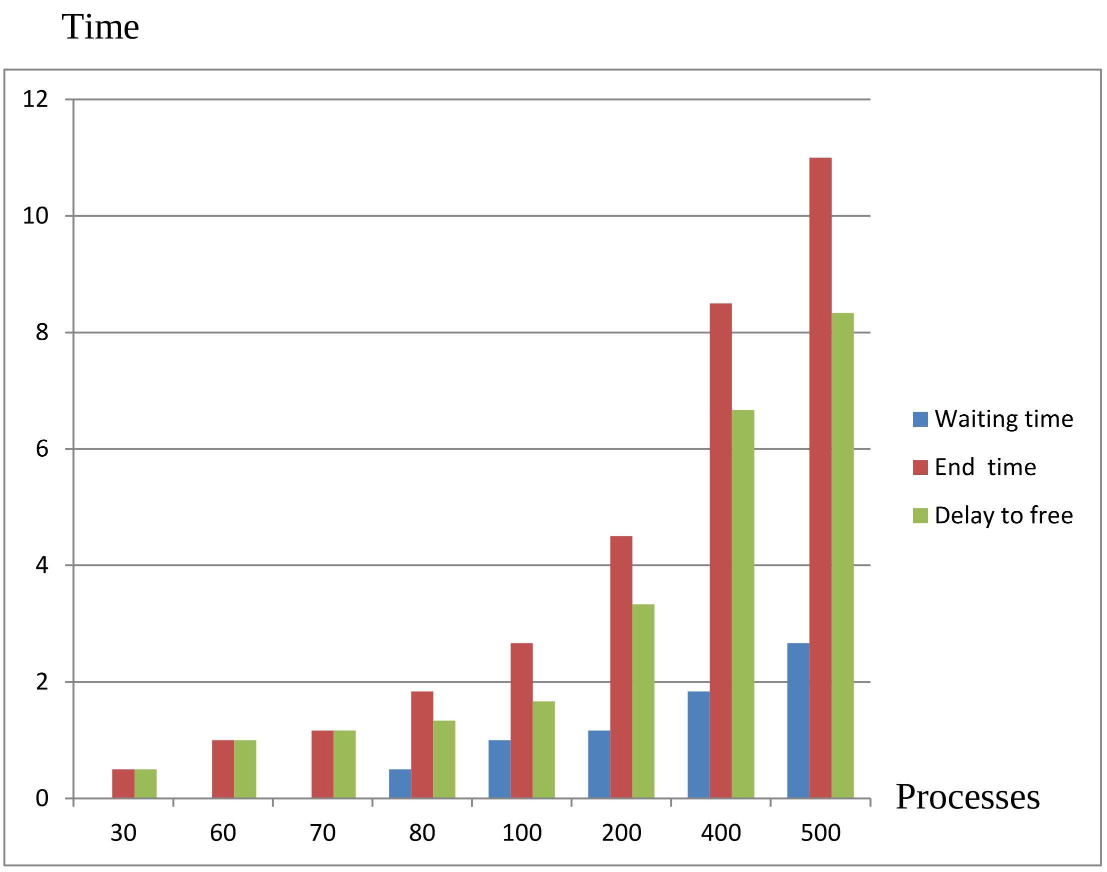


Figure 4.1.2 Shortest Job First [SJF]

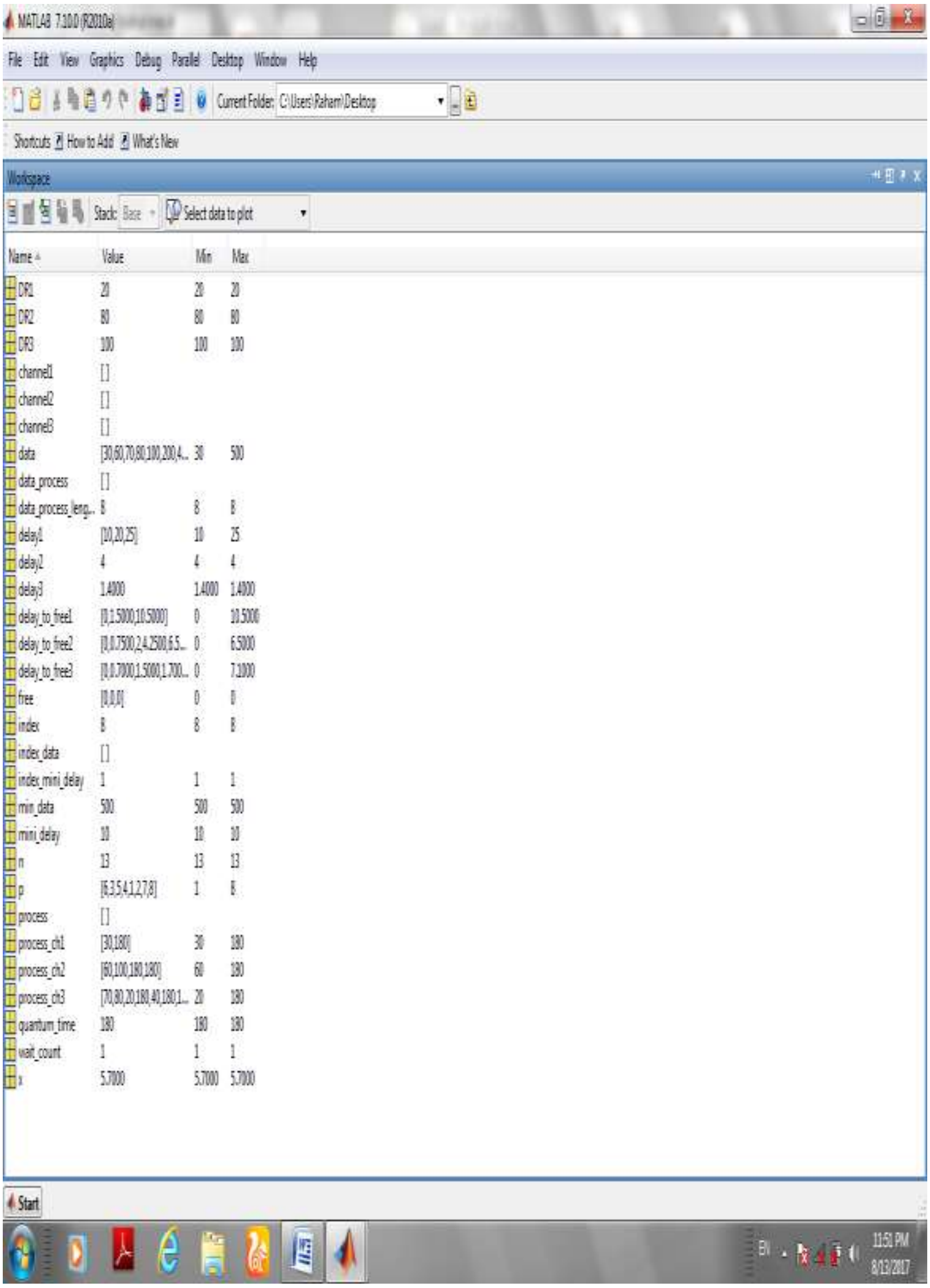
Average waiting time =∑ waiting time of processes / number of processes

$= \frac{7.1667}{8} = 0.8959$

End processes = ∑end time of processes/ number of processes = 31.1667= 3.896

Delay to free = ∑ Delay to free of processes/ number of processes 23.999 = 2.999

Workspace 4.1.3 Shortest Job First with time quantum when data rate is different [20 80, 100]


MATLAB 7.10.0 (R2010a)
File Edit View Graphics Debug Parallel Desktop Window Help
Current Folder: C:\Users\Raham\Desktop
Shortcuts How to Add What's New
Workspace
Stack: Base
Select data to plot
Start
EN
11:51 PM
8/13/2017

Name	Value	Min	Max
DR1	20	20	20
DR2	80	80	80
DR3	100	100	100
channel1	[]		
channel2	[]		
channel3	[]		
data	[30,60,70,80,100,200,4...	30	500
data_process	[]		
data_process_leng...	8	8	8
delay1	[10,20,25]	10	25
delay2	4	4	4
delay3	1.4000	1.4000	1.4000
delay_to_free1	[0,1.5000,10.5000]	0	10.5000
delay_to_free2	[0,0.7500,2,4.2500,6.5...	0	6.5000
delay_to_free3	[0,0.7000,1.5000,1.700...	0	7.1000
free	[0,0,0]	0	0
index	8	8	8
index_data	[]		
index_mini_delay	1	1	1
min_data	500	500	500
mini_delay	10	10	10
n	13	13	13
p	[6,3,5,4,1,2,7,8]	1	8
process	[]		
process_ch1	[30,180]	30	180
process_ch2	[60,100,180,180]	60	180
process_ch3	[70,80,20,180,40,180,1...	20	180
quantum_time	180	180	180
wait_count	1	1	1
x	5.7000	5.7000	5.7000

This workspace [4.1.3] explains the algorithm [SJF] using time quantum when data rate is different [20 80 100]. The channels are [channel1, channel2, channel3] after arrange data all channels are free that means select minimum processes [30 60 70] to allocated channel1, channel2 and channel3. When process complete execution it release channel by itself.

Table 4.1.3 Shortest Job First with time quantum when data rate is different

processes	Waiting time	End time	Delay to free
30	0	1.5000	1.5000
60	0	0.7500	0.7500
70	0	0.7000	0.7000
80	0.7000	1.5000	0.8
100	0.7500	2	1.25
200	1.5000	10.5000	9
400	1.7000	4.2500	2.55
500	3.9000	7.1000	3.2

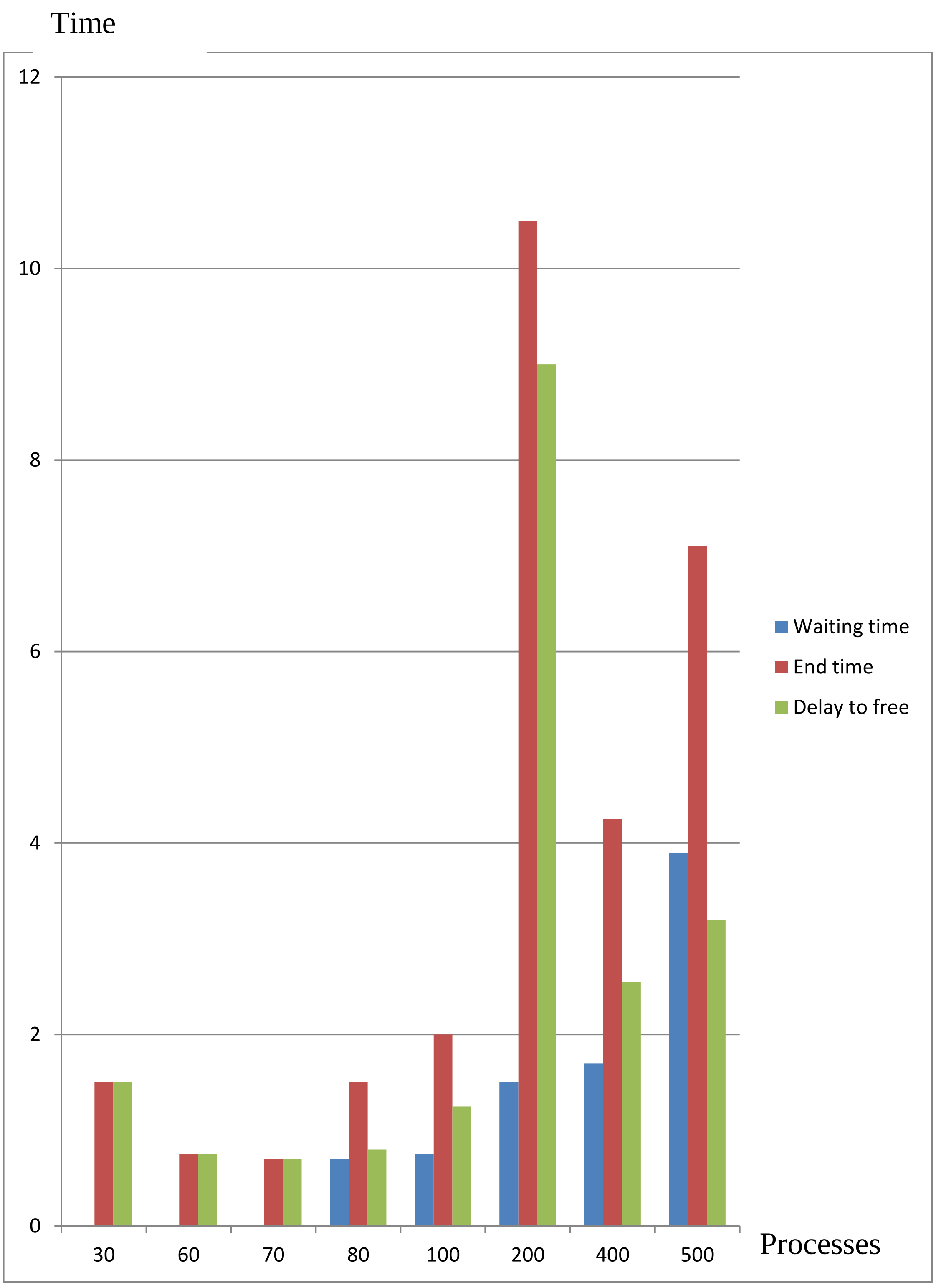


Figure 4.1.3 Shortest Job First with time quantum when data rate is different

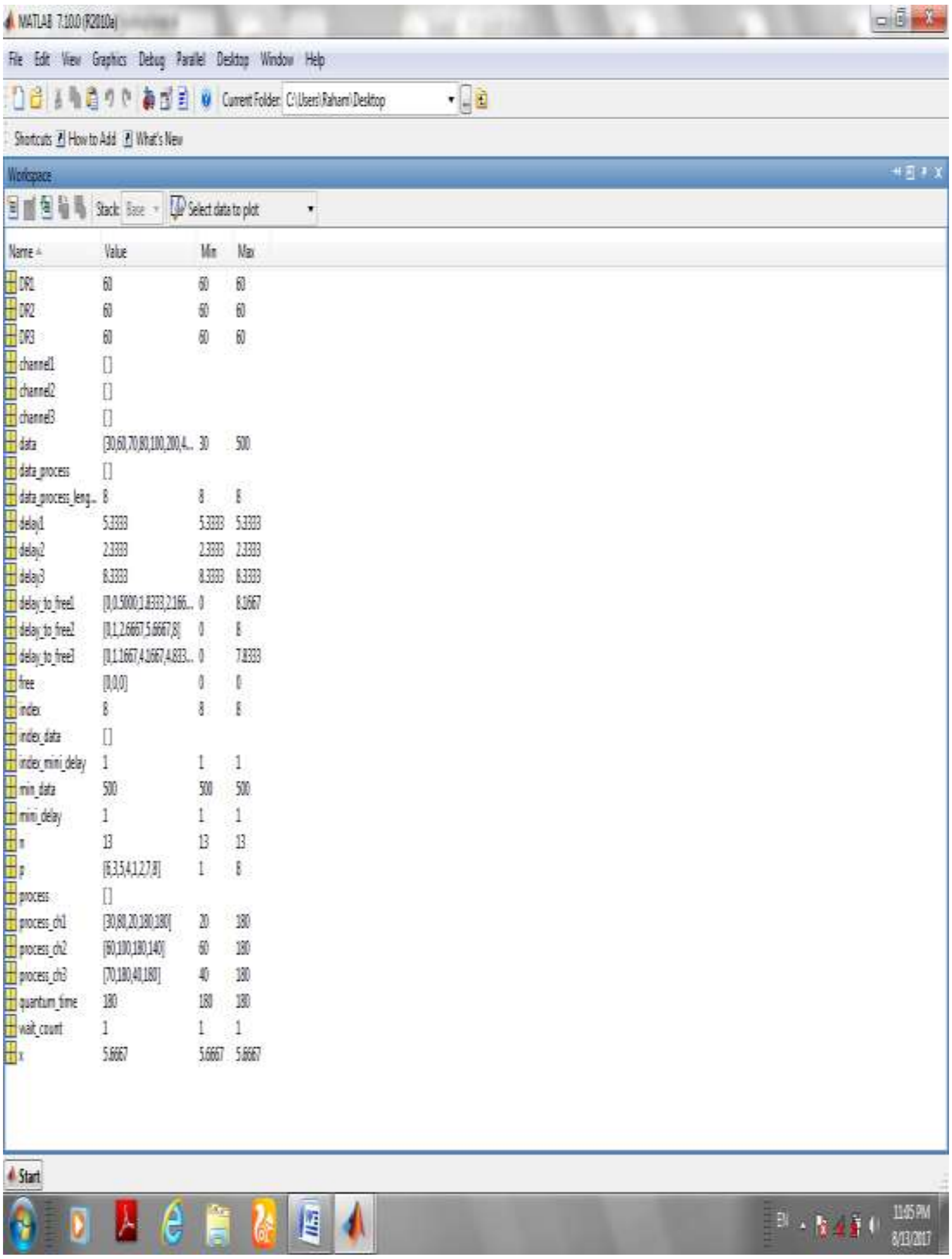
Average waiting time =∑ waiting time of processes ∕ number of processes

$= \frac{8.55}{8} = 1.07$

End time =∑end time of processes/ number of processes = 28.3= 3.54

Delay to free = ∑ Delay to free of processes/ number of processes 19.75 = 2.47

Workspace 4.1.4 Shortest Job First when data rate is different [20, 80, 100]


MATLAB 7.10.0 (R2010a)
File Edit View Graphics Debug Parallel Desktop Window Help
Current Folder: C:\Users\Rahami\Desktop
Shortcuts How to Add What's New
Workspace
Stack: Base
Select data to plot
Start
11:05 PM
8/13/2017

Name	Value	Min	Max
DR1	60	60	60
DR2	60	60	60
DR3	60	60	60
channel1	[]		
channel2	[]		
channel3	[]		
data	[30,60,70,80,100,200,4...	30	500
data_process	[]		
data_process_leng...	8	8	8
delay1	5.3333	5.3333	5.3333
delay2	2.3333	2.3333	2.3333
delay3	8.3333	8.3333	8.3333
delay_to_free1	[0,0.5000,1.8333,2.166...	0	8.1667
delay_to_free2	[0,1,2.6667,5.6667,8]	0	8
delay_to_free3	[0,1.1667,4.1667,4.833...	0	7.8333
free	[0,0,0]	0	0
index	8	8	8
index_data	[]		
index_mini_delay	1	1	1
min_data	500	500	500
mini_delay	1	1	1
n	13	13	13
p	[6,3,5,4,1,2,7,8]	1	8
process	[]		
process_ch1	[30,80,20,180,180]	20	180
process_ch2	[60,100,180,140]	60	180
process_ch3	[70,180,40,180]	40	180
quantum_time	180	180	180
wait_count	1	1	1
x	5.6667	5.6667	5.6667

This workspace [4.1.4] explains the algorithm [SJF] when data rate is different [20 80 100]. The channels are [channel1, channel2, channel3] after arrange data all channels are free that means select minimum processes [30 60 70] to allocated channel1, channel2 and channel3.all processes are greater than time quantum must be divided and distributed into free channels ,when process is complete execution it release channel by itself.

Table 4.1.4 Shortest Job First when data rate is different

processes	Waiting time	End time	Delay to Transmission
30	0	1.5000	1.5000
60	0	0.7500	0.7500
70	0	0.7000	0.7000
80	0.7000	1.5000	0.8
100	0.7500	2	1.25
200	1.5000	11.5000	10
400	1.5000	5.5000	4
500	2	8.2500	6.25

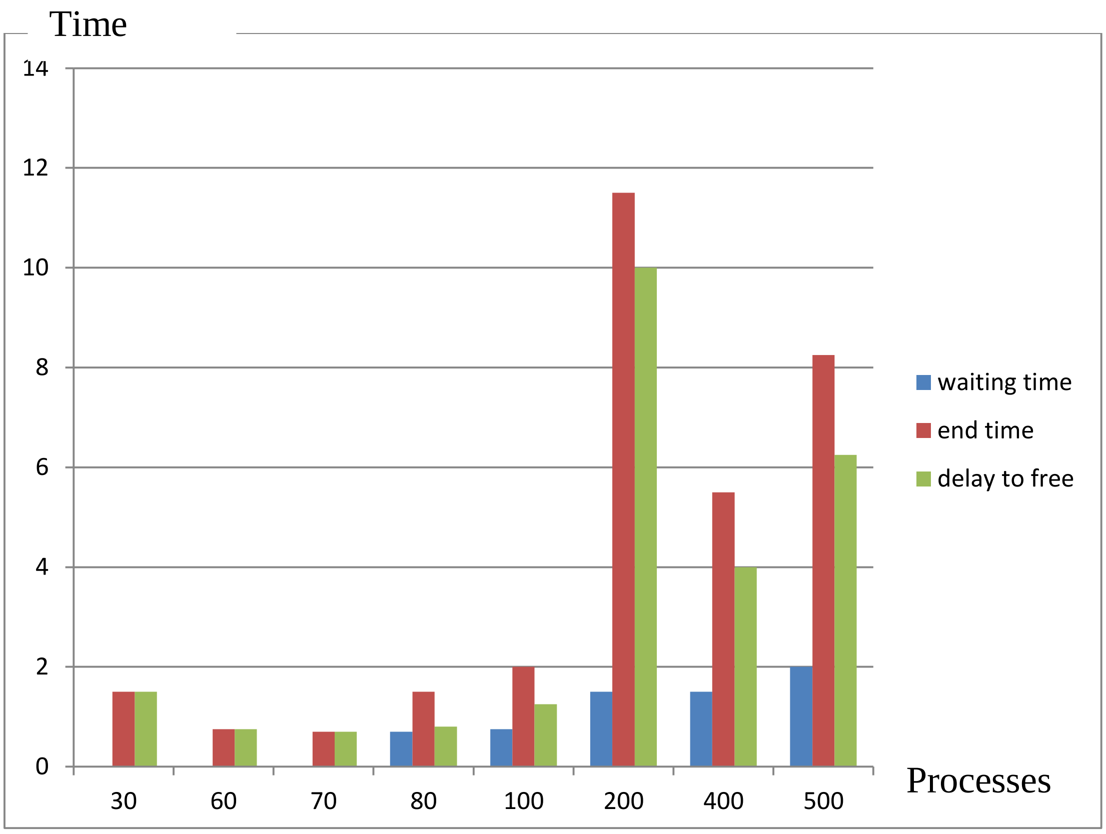


Figure 4.1.4 Shortest Job First when data rate is different

Average waiting time =∑ waiting time of processes / number of processes

$= \frac{6.45}{8} = 0.806$

End time = ∑end time processes/ number of processes = 31.7= 3.96

Delay to free = ∑ Delay to free of processes/ number of processes 25.25 = 3.16

4.2 Summary

From previous results when we compare between two algorithms we find Shortest Job First with time quantum is better than Shortest Job First in terms of end time process, resources utilization, delay to transmission and Fair treatment for all the processes. But in term of waiting time Shortest Job First is better, SJF doesn’t always minimize waiting time.

Chapter five

Conclusion and recommendations

Chapter five

Conclusion and recommendations

5.1 Conclusion

Grid computing can solve more complex tasks in less time and utilizes the resources efficiently. To make grid work properly, best job scheduling strategies have to be employed. Scheduling helps the jobs to get resources properly. This project used proposed a scheduling algorithm Shortest Job First with time quantum and also done its comparison in terms of waiting time, end time and delay to transmission of jobs with algorithm Shortest Job First [SJF]. Average waiting time, average end time and average delay to transmission of jobs are also calculated. The proposed algorithm has following benefits in comparison to Shortest Job First algorithm [SJF]:

5.1.1 Low end time

5.1.2 Low delay to transmission

5.1.3 More resources utilization

5.1.4 Fair treatment for all the processes.

5.1.5 Minimizing degree of starvation.

So Shortest Job First with time quantum is the best.

5.2 Recommendations:

In future Shortest Job First with dynamic time quantum and will also be using dynamic time quantum for ready queue, where time quantum will be calculated every time a process enters or exits the queue. One may be Able to increase the Performance, Throughput and decrease the end Time by above solution.

References

[1] Mrs. Radha, Dr.V.Sumathy “A Detailed Study of Resource Scheduling and Fault Tolerance in Grid”, 2011.

[2] N.A. Azeez1; A.P. idoye; A.O. Adesina; K.K. Agbele; Iyamu Tiko, and I.M. Venter, “Peer to Peer Computing and Grid Computing: Towards a Better Understanding”, 2011.

[3] Manisha Bhardwaj, Sandeep Kumar, A Two Way Scheduling Approach for Effective Resource Scheduling in Grid, International Journal of Computer Networks and Wireless Communications (IJCNWC), Vol.3, No3,PP.243-246, June 2013

[4] Dipti Sharma, Mr. Pradeep Mittal, Job Scheduling Algorithm for Computational Grid in Grid Computing Environment, International Journal of Advanced Research in Computer Science and Software Engineering, PP. 735-743, 5, May 2013.

[5] Himani Aggarwal, Er. Shakti Nagpal, Augmented SJF algorithm with reduced starvation, International Journal of Advanced Research in Computer Science and Software Engineering, pp. 718-723, 6, June 2014.

[6] Er. Himanshu Jain, Kavita Khatkar, Process Scheduling Approach for Starvation Improvement with Time Delay Analysis in Grid Resources Allocation, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4 PP. 197-201, , 8, August 2015.

[7] Kumaresh.V.S, Prasidh.S, Arjunan.B, Subbhaash.S and Sandhya.M.K, Multilevel Queue-Based Scheduling for Heterogeneous

Grid Environment, IJCSI International Journal of Computer Science Issues,pp. 245-248, November 2012.

[8] Manupriya Hasija, Akhil Kaushik, Satvika Kaushik and Manoj Barnela,

D-MMLQ Algorithm for Multi-level Queue Scheduling, IJCSNS International Journal of Computer Science and Network Security, VOL.14, pp. 90-94, July 2014.

[9] Mohamed Eisa, Improving Grid Computing Scheduling using Heuristic Algorithms, International Journal of Computer Applications, Volume 78,pp. 14-17, September 2013.

[10] Pinky Rosemarry, Payal Singhal, Ravinder Singh, A Study of Various Job & Resource Scheduling Algorithms in Grid Computing, International Journal of Computer Science and Information Technologies, pp. 5504-5507, 2012.

[11] Pinky Rosemarry, Ravinder Singh, Payal Singhal and Dilip Sisodia, Grouping Based job scheduling algorithm using priority queue and hybrid algorithm in grid Computing, International Journal of Grid Computing & Applications (IJGCA), Vol.3,pp. 55-65, December 2012.

[12] Qudsia Mateen, Ujala Niazi, Marwah, Grouping based job scheduling Algorithm Using Priority queue, Shortest Job First, Round Robin and First Come First Serve, International Journal of Computer and Communication System Engineering (IJCCSE), Vol. 2 (1), pp. 139-142, 2015.

[13] Sandip Fakira Lokhande1, Sachin D. Chavhan, Prof. S. R. Jadhao,Grid Computing Scheduling Jobs Based on Priority Using Backfilling, International Journal of Electrical Electronics & Computer Science Engineering, pp. 68-72,(April, 2015).

Appendix I

Shortest Job First with quantum time:

```
clc , close all ,clear all

data = [100 200 60 80 70 30 400 500]; %Kbyte

index_data=[];

p=[];

process=[];

for n=1:length(data);

[min_data index]=min(data)

process=[process min_data]

p =[p index]

data(index)=1000;

end

data=process;

process=[]

channel1=[];

channel2=[];

channel3=[];

process_ch1=[];

process_ch2=[];
```

```
process_ch3=[];

delay_to_free1=[0]

delay_to_free2=[0]

delay_to_free3=[0]

DR1=[20]; %Mbps

DR2=[80];

DR3=[100];

delay1=data/DR1; %ms

delay2=data/DR2;

delay3=data/DR3;

free=[1 1 1];

data_process=data;

data_process_length=length(data);

n=0

quantum_time=sum(data)/length(data);

quantum_time=quantum_time

for x=0:.00001:50

%for n=1:data_process_length

    data_process_length=length(data);

   %if n<data_process_length
```

```
 if (free(1)==1 && free(2)==1 && free(3)==1)
     n=n+1
     delay1=data_process/DR1; %ms
     delay2=data_process/DR2;
     delay3=data_process/DR3;
     [mini_delay
index_mini_delay]=max([delay1(1),delay2(1),delay3(1)])
     if index_mini_delay==1
     if data_process(1)<=quantum_time
     process_ch1=[process_ch1 data_process(1)]
     data_process(1)=[]
 delay_to_free1=[delay_to_free1 mini_delay+max(delay_to_free1)]
     wait_count=1;
     else
     process_ch1=[process_ch1 quantum_time]
     data_process(1)=data_process(1)-quantum_time
     delay_to_free1=[delay_to_free1
quantum_time/DR1+max(delay_to_free1)]
     end
     free(1)=0
     elseif index_mini_delay==2
```

```
    if data_process(1)<=quantum_time
    process_ch2=[process_ch2 data_process(1)]
    data_process(1)=[]
    delay_to_free2=[delay_to_free2 mini_delay+max(delay_to_free2)]
    else
    process_ch2=[process_ch2 quantum_time]
    data_process(1)=data_process(1)-quantum_time
    delay_to_free2=[delay_to_free2
(quantum_time/DR2)+max(delay_to_free2)]
    end
    free(2)=0
    elseif index_mini_delay==3
    if data_process(1)<=quantum_time
    process_ch3=[process_ch3 data_process(1)]
    data_process(1)=[]
    delay_to_free3=[delay_to_free3 mini_delay+max(delay_to_free3)]
    else
    process_ch3=[process_ch3 quantum_time]
    data_process(1)=data_process(1)-quantum_time
    delay_to_free3=[delay_to_free3
(quantum_time/DR3)+max(delay_to_free3)]
```

```
     end
     free(3)=0
     end
elseif (free(1)==1 & free(2)==1 & free(3)==0)
    n=n+1
 delay1=data_process/DR1; %ms
    delay2=data_process/DR2;
    [mini_delay index_mini_delay]=max([delay1(1),delay2(1)])
    if index_mini_delay==1
    if data_process(1)<=quantum_time
    process_ch1=[process_ch1 data_process(1)]
    data_process(1)=[]
    delay_to_free1=[delay_to_free1 mini_delay+max(delay_to_free1)]
    else
    process_ch1=[process_ch1 quantum_time]
    data_process(1)=data_process(1)-quantum_time
    delay_to_free1=[delay_to_free1
(quantum_time/DR1)+max(delay_to_free1)]
    end
    free(1)=0
    elseif index_mini_delay==2
    if data_process(1)<=quantum_time
    process_ch2=[process_ch2 data_process(1)]
    data_process(1)=[]
    delay_to_free2=[delay_to_free2 mini_delay+max(delay_to_free2)]
    else
    process_ch2=[process_ch2 quantum_time]
    data_process(1)=data_process(1)-quantum_time
    delay_to_free2=[delay_to_free2
(quantum_time/DR2)+max(delay_to_free2)]
    end
    free(2)=0
    end
elseif (free(1)==1 & free(2)==0 & free(3)==1)
    n=n+1
    delay1=data_process/DR1; %ms
   delay3=data_process/DR3;
    [mini_delay index_mini_delay]=max([delay1(1),delay3(1)])
    if index_mini_delay==1
    if data_process(1)<=quantum_time
    process_ch1=[process_ch1 data_process(1)]
    data_process(1)=[]
    delay_to_free1=[delay_to_free1 mini_delay+max(delay_to_free1)]

    else

    process_ch1=[process_ch1 quantum_time]

    data_process(1)=data_process(1)-quantum_time

    delay_to_free1=[delay_to_free1
(quantum_time/DR1)+max(delay_to_free1)]

    end

    free(1)=0

    elseif index_mini_delay==2

    if data_process(1)<=quantum_time

    process_ch3=[process_ch3 data_process(1)]

    data_process(1)=[]

    delay_to_free3=[delay_to_free3 mini_delay+max(delay_to_free3)]

    else

    process_ch3=[process_ch3 quantum_time]

    data_process(1)=data_process(1)-quantum_time

        delay_to_free3=[delay_to_free3
(quantum_time/DR3)++max(delay_to_free3)]

    end

    free(3)=0

    end
```

```
  elseif (free(1)==0 & free(2)==1 & free(3)==1)
   n=n+1
   delay2=data_process/DR2
   delay3=data_process/DR3
   [mini_delay index_mini_delay]=max([delay2(1),delay3(1)])
   if index_mini_delay==1
   if data_process(1)<=quantum_time
   process_ch2=[process_ch2 data_process(1)]
   data_process(1)=[]
       delay_to_free2=[delay_to_free2
mini_delay+max(delay_to_free2)]
   else
   process_ch2=[process_ch2 quantum_time]
   data_process(1)=data_process(1)-quantum_time
       delay_to_free2=[delay_to_free2
(quantum_time/DR2)+max(delay_to_free2)]
   end
   free(2)=0
   elseif index_mini_delay==2
   if data_process(1)<=quantum_time
   process_ch3=[process_ch3 data_process(1)]
```

```
    data_process(1)=[]
        delay_to_free3=[delay_to_free3
mini_delay+max(delay_to_free3)]
    else
    process_ch3=[process_ch3 quantum_time]
 data_process(1)=data_process(1)-quantum_time
        delay_to_free3=[delay_to_free3
(quantum_time/DR3)+max(delay_to_free3)]
    end
    free(3)=0
    end
elseif (free(1)==1 & free(2)==0 & free(3)==0)
      n=n+1
    delay1=data_process/DR1; %ms
    if data_process(1)<=quantum_time
    process_ch1=[process_ch1 data_process(1)]
    data_process(1)=[]
        delay_to_free1=[delay_to_free1 delay1(1)+max(delay_to_free1)]
    else
    process_ch1=[process_ch1 quantum_time]
    data_process(1)=data_process(1)-quantum_time
```

```
         delay_to_free1=[delay_to_free1
(quantum_time/DR1)+max(delay_to_free1)]

    end

    free(1)=0

elseif (free(1)==0 & free(2)==1 & free(3)==0)

      n=n+1

    delay2=data_process/DR2; %ms

    if data_process(1)<=quantum_time

    process_ch2=[process_ch2 data_process(1)]

    data_process(1)=[]

        delay_to_free2=[delay_to_free2 delay2(1)+max(delay_to_free2)]

    else

    process_ch2=[process_ch2 quantum_time]

    data_process(1)=data_process(1)-quantum_time

        delay_to_free2=[delay_to_free2
(quantum_time/DR2)+max(delay_to_free2)]

    end

   free(2)=0

  elseif (free(1)==0 & free(2)==0 & free(3)==1)

      n=n+1

    delay3=data_process/DR3; %ms
    if data_process(1)<=quantum_time

    process_ch3=[process_ch3 data_process(1)]

    data_process(1)=[]

        delay_to_free3=[delay_to_free3 delay3(1)+max(delay_to_free3)]

    else

    process_ch3=[process_ch3 quantum_time]

    data_process(1)=data_process(1)-quantum_time

        delay_to_free3=[delay_to_free3
(quantum_time/DR3)+max(delay_to_free3)]

    end

   free(3)=0

  end

 % x

    if max(delay_to_free1)<=x

      free(1)=1



  end

  if max(delay_to_free2)<=x

      free(2)=1

  end

  if max(delay_to_free3)<=x
```

```
       free(3)=1
    end
   %end
   if  length(data_process)==0
      break
  end
 end
```

Appendinx II

Shortest Job Firs SJF]:

```
data = [100 200 60 80 70 30 400 500]; %Kbyte

index_data=[];

p=[];

process=[];

for n=1:length(data);

[min_data index]=min(data)

process=[process min_data]

p =[p index]

data(index)=1000;

end

data=process;

process=[]

channel1=[];

channel2=[];

channel3=[];

process_ch1=[];

process_ch2=[];

process_ch3=[];
```

```
delay_to_free1=[0]

delay_to_free2=[0]

delay_to_free3=[0]

DR1=[20]; %Mbps

DR2=[80];

DR3=[100];

delay1=data/DR1; %ms

delay2=data/DR2;

delay3=data/DR3;

free=[1 1 1];

data_process=data;

data_process_length=length(data);

n=0

for x=0:.00001:50

%for n=1:data_process_length

    if n<data_process_length

   if (free(1)==1 && free(2)==1 && free(3)==1)

       n=n+1

         delay1=data_process/DR1; %ms

         delay2=data_process/DR2;
         delay3=data_process/DR3;
         [mini_delay
index_mini_delay]=max([delay1(1),delay2(1),delay3(1)])
         if index_mini_delay==1
process_ch1=[process_ch1 data_process(1)]
            free(1)=0
       delay_to_free1=[delay_to_free1 mini_delay+max(delay_to_free1)]
         elseif index_mini_delay==2
process_ch2=[process_ch2 data_process(1)]
             free(2)=0
                         delay_to_free2=[delay_to_free2
mini_delay+max(delay_to_free2)]
         elseif index_mini_delay==3
process_ch3=[process_ch3 data_process(1)]
             free(3)=0
                         delay_to_free3=[delay_to_free3
mini_delay+max(delay_to_free3)]
         end
    data_process(1)=[]
   elseif (free(1)==1 & free(2)==1 & free(3)==0)
       n=n+1
```

```
        delay1=data_process/DR1; %ms

        delay2=data_process/DR2;

        [mini_delay index_mini_delay]=min([delay1(1),delay2(1)])

        if index_mini_delay==1

process_ch1=[process_ch1 data_process(1)]

          free(1)=0

                    delay_to_free1=[delay_to_free1
mini_delay+max(delay_to_free1)]

        elseif index_mini_delay==2

process_ch2=[process_ch2 data_process(1)]

           free(2)=0

                     delay_to_free2=[delay_to_free2
mini_delay+max(delay_to_free2)]

        end

   data_process(1)=[]

  elseif (free(1)==1 & free(2)==0 & free(3)==1)

      n=n+1

        delay1=data_process/DR1; %ms

        delay3=data_process/DR3;

        [mini_delay index_mini_delay]=max([delay1(1),delay3(1)])

        if index_mini_delay==1
```

```
process_ch1=[process_ch1 data_process(1)]
            free(1)=0
                        delay_to_free1=[delay_to_free1
mini_delay+max(delay_to_free1)]
         elseif index_mini_delay==2
process_ch3=[process_ch3 data_process(1)]
             free(3)=0
                         delay_to_free3=[delay_to_free3
mini_delay+max(delay_to_free3)]
         end
    data_process(1)=[]
   elseif (free(1)==0 & free(2)==1 & free(3)==1)
       n=n+1
         delay2=data_process/DR2
         delay3=data_process/DR3
         [mini_delay index_mini_delay]=max([delay2(1),delay3(1)])
         if index_mini_delay==1
process_ch2=[process_ch2 data_process(1)]


             free(2)=0
```

```
                    delay_to_free2=[delay_to_free2
mini_delay+max(delay_to_free2)]

       elseif index_mini_delay==2

process_ch3=[process_ch3 data_process(1)]

          free(3)=0

                    delay_to_free3=[delay_to_free3
mini_delay+max(delay_to_free3)]

       end

   data_process(1)=[]

  elseif (free(1)==1 & free(2)==0 & free(3)==0)

     n=n+1

    delay1=data_process/DR1; %ms

process_ch1=[process_ch1 data_process(1)]

   free(1)=0

             delay_to_free1=[delay_to_free1
delay1(1)+max(delay_to_free1)]

   data_process(1)=[]

  elseif (free(1)==0 & free(2)==1 & free(3)==0)



     n=n+1

    delay2=data_process/DR2; %ms
```

```
process_ch2=[process_ch2 data_process(1)]

    free(2)=0

                delay_to_free2=[delay_to_free2
delay2(1)+max(delay_to_free2)]

    data_process(1)=[]

   elseif (free(1)==0 & free(2)==0 & free(3)==1)

       n=n+1

     delay3=data_process/DR3; %ms

    process_ch3=[process_ch3 data_process(1)]

    free(3)=0

                delay_to_free3=[delay_to_free3
delay3(1)+max(delay_to_free3)]

    data_process(1)=[]

   end

  % x



  if max(delay_to_free1)<=x

       free(1)=1

   end

   if max(delay_to_free2)<=x

       free(2)=1
   end
   if max(delay_to_free3)<=x
       free(3)=1
   end
   end
   if  length(data_process)==0
    % break
end
end
```