Speed Up Scheduling

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS by Research
in
Computer Science and Engineering

by

Yash Gupta
200802043
yash.guptaug08@students.iiit.ac.in

Center for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
May 2013

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled *Speed Up* Scheduling by Yash Gupta, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. Kamalakar Karlapalem

Dedicated to my grandfather Dr. H.L. Gupta, father Yatindra Gupta, mother Shiv Kanya Gupta and sister Shikha Gupta

Acknowledgments

At first, I would like to express my gratitude to my advisor Dr. Kamalakar Karlapalem for his invaluable guidance, motivation and support right from the beginning till the end. His dedication in research, new ideas and constant zeal to improve has inspired me all the while and made me understood the true meaning of research. He guided me through every obstacle that I encountered, trusted me at each step and made me become self-dependent for work. The completion of thesis would not have been possible without his affectionate guidance, motivation and support.

I would also like to thank my friends Tushar, Ajay, Sarvesh, Saket, Harshit who motivated and encouraged me in my difficult times and were always there to support. Thanks for making the 5 years of my college life the most memorable and wonderful years of my life.

Last but not the least I would like to express my gratitude to my parents, Yatindra and Shiv Kanya Gupta, my grandfather Dr. H.L. Gupta and my sister Shikha Gupta for keeping faith in me and encouraging me to work hard despite of obstacle and difficulties. This thesis would not have been possible without their endless love, affection and support. I can never thank my parents enough for all the sacrifices that they have made for me.

Abstract

Queue is required when service provider is not able to handle jobs arriving over the time. In real life, job execution often requires human intervention in it's execution. In a highly flexible and dynamic environment, some jobs might demand for faster execution especially when resources are limited and jobs are competing for acquiring resources. Thus, user may demand for speed up (reduced wait time / faster execution) for some of the jobs present in the queue at run time. In such cases, it is required to accelerate (directly sending the job to the server) jobs requesting for speed up ahead of other jobs present in the queue to facilitate earlier completion of urgent jobs. Since no additional resources are used, therefore such acceleration of jobs might result in slowing down of other jobs present in the queue. In this thesis, we model the problem of Speed Up Scheduling without acquiring any additional resources for the scheduling of on-line speed up requests posed by the user at run-time. Speed Up scheduling needs to speed up all the jobs that requested for it without unnecessarily slowing down the other jobs. Note that it might not be possible to achieve speed up for all the jobs that requested for it as well as it might be the case that while speeding up jobs, even some of the jobs that requested for speed up were slowed down.

In order to address Speed Up problem, we provide implicit Speed Up techniques where the notion of acceleration is incorporated in the priority function. We use concepts of queuing theory and discrete event based simulation model for the design and evaluation of our speed up algorithms. We apply the idea of Speed Up Scheduling to two different applications - Web Scheduling and CPU Scheduling. We demonstrate our results with a simulation based model using trace driven workload and synthetic datasets to show the usefulness of Speed Up in these domains. We further examine how different scheduling algorithms can be compared based on speed up/slow down characteristics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One does not like to wait. In most of the cases, the number of service providers are less compared to the number of jobs requiring service. In such cases, queue is required because the service provider is not able to handle jobs as soon as they arrive. There are many practical scenarios where queuing cannot be avoided. For example, accumulation of web requests in queue arriving at web server, queue of packets at routers, queue of processes waiting for CPU resources etc. In such situations, the choice of scheduling policy determines mean wait time, mean queue length and other performance measures.

In real life, job execution often requires human intervention in its execution. Further, in a highly dynamic environment, a particular job may raise the requirement for urgent attention, especially when the resources are limited. The user may demand for speed up (reduced wait time/faster execution) for some specific job present in the queue at run time. In such a case, we need to accelerate/speed up such job (that requested for speed up based on user demand) ahead of other jobs present in the queue to the server for service. However, such acceleration would result in slowing down of other jobs which are present ahead of it in queue *since no additional resources* are used. Thus, the problem that arises is how to schedule such speed up requests posed by the user so as to facilitate their earlier execution but at the same time ensuring less impact of slow down on the other jobs.

Consider an example of CPU Scheduling of different processes. The processes are usually categorized into different priority classes. All the processes of same priority class are usually scheduled using first come first serve or Round Robin strategy. In such a case, it is possible for a user to request speed up for a specific user process. However, the requested speed up is not urgent enough to move the process to higher priority class but the process needs to be accelerated with respect to those having same priority. Under the assumption of limited resources, some other processes within same priority class have to be delayed in order for one user to achieve requested speed up.

The idea behind speed up is to achieve faster execution of jobs (requested by user at run-time) without acquiring additional resources but with an overhead of some other jobs going to be delayed. Yet we want to facilitate earlier execution of urgent jobs (whose delay might have severe consequences) without unnecessarily slowing down the other jobs. It is important to note that it might not be possible to achieve

speed up for all the jobs that requested for it as well as it might be the case that while speeding up some jobs, even some of the jobs that requested for speed up were slowed down. [9]

1.1 Background and Related Work

The problem of speed up is on-line since we do not have any information about the future jobs that have not yet arrived. Further, users might pose speed up requests dynamically at run time which can further influence the other urgent as well as non-urgent jobs present in the queue. In speed up problem execution needs to be adjusted at run time in order to respond to on-line user requests for speed up. The job priority is not determined according to a predefined scheduling policy assignment criterion like real time scheduling policies, but according to the user on-line requests for speed up posed at run-time. Further in speed up problem, we need to avoid unnecessary penalization to the rest of the jobs (which did not request for speed up). Thus, we even speed up the slowed down non urgent jobs if necessary.

Various scheduling techniques have already been proposed such as First Come First Serve, Shortest Job First, Priority Scheduling, EDF (Earliest Deadline First), Multilevel Queue scheduling, Round Robin etc. These scheduling policies perform effectively in improving the performance of the system in the absence of on-line speed up requests, but when such requests occur we need more fine grained scheduling policies which can speed up all the activities that requested for it at run time. We can compare and contrast different scheduling algorithms based on speed up/slow down characteristics. The results and analysis for such comparisons are presented and explained in later chapters of this thesis. When a specific job completes before its expected execution time with respect to a specific scheduler, then we refer this as speeding up of job. Shortest Job First speeds up the job with shorter duration time, Priority scheduling speeds up the job with highest priority both with respect to FCFS scheduler. However these scheduling policies are unable to handle on-line speed up requests posed by the user at run time in which execution needs to be adjusted at run time thus providing remedies for the jobs that got delayed or requested for faster execution without penalizing more than necessary to the rest of the executing requests. Further existing scheduling policies speed up specific kind of jobs which do not take into consideration the slow down for other jobs. In our problem of speed up, we might have to speed up even the slowed down non-urgent jobs under certain circumstances.

Earliest Deadline First (EDF) scheduling policy focuses on speeding up of jobs with earliest deadlines so that they do not miss their deadlines. EDF scheduling problem seems similar to speed up problem in a sense that it can be executed on-line, the priority depends on the jobs that arrive (specific deadlines) and the priority assignment is done at run-time but there is a difference between preserving deadlines and problem of speed up. *The notion of deadline is absolute whereas concept of speed up is relative*. The deadlines of arriving jobs are absolute and are independent of the state of the queue at their arrival whereas the concept of speed up is defined with respect to expected execution time which is influenced by the queue state. In EDF, the deadlines are met by speeding up of urgent jobs but in our case, there need not be any deadline to achieve speed up. EDF tries to minimize the number of jobs that

miss their deadlines, whereas Speed Up focus is to accelerate/speed up as many jobs as possible which requested for it at run-time. Further, EDF scheduling does not care about the jobs which do not have specific deadlines (non urgent), but in our case we cannot penalize more than necessary to the rest of the executing jobs. In speed up problem, it is more beneficial to achieve the exact requested speed up for urgent job rather than achieving relatively high amount of speed up (than requested) thereby causing higher amount of slow down to the rest of the executing jobs. In current practice, this kind of speed up is handled in an ad-hoc manner resulting in many complaints and inefficient resource utilization.

The problem of speed up has been addressed in [9], [11] and [10]. The authors modeled speed up problem for speeding up of E-commerce activities [10] and improving the response time of business processes [9]. The authors presented three explicit speed up algorithms (MPF, MPF-SD and MinPF) based on *location table* model where they select a specific job based on its queue position and achieve speed up by swapping of jobs in queue. MPF (Maximum Position First) algorithm tries to push forward all the jobs requesting for speed up at the head of the queue and decides their order by trying to swap small positions (near to the head) with the large ones (end of the queue). MPF-SD (MPF-Smaller Duration) algorithm is similar to MPF algorithm but it preserves one property that "*no job that is requesting for speed up is delayed*". The urgent job (requesting for speed up) is pushed ahead in the queue if and only if its duration is smaller than the one with which it swaps. MinPF (Minimum Position First) algorithm is compliment of MPF algorithm. Instead of swapping jobs with maximum distance, it swaps jobs with minimum distance. All the three algorithms are purely positional and computationally expensive with an overhead of maintaining a location table. Further, authors in [9], [11] and [10] used only *achieved ratio* metric (percentage of jobs that achieved their requested speed up) for evaluating the effectiveness of their speed up algorithms. Their work did not consider the impact of slow down on the remaining jobs as a result of speed up which is crucial. Any scheduling algorithm speeding up some urgent jobs requiring urgent attention but causing arbitrarily high slow down to the rest of the jobs would not be practical and fair. Therefore, along with *achieved ratio*, we also consider the impact of slow down to the non urgent jobs as a consequence of speed up and the overall mean wait time. In this thesis, we remodel the speed up problem aiming at speeding up the jobs which requested for it but at the same time providing less scope of slow down for the rest of the executing jobs while keeping the mean wait time reasonable. We provide implicit techniques to address speed up problem where the notion of acceleration is incorporated in the priority function, thus leading to computationally efficient solution.

1.2 Contribution

The problem of speed up has not been adequately studied in literature before (except [9] and [10]) which can respond to on-line requests for speed up. The problem, applications and issues that we address with respect to speed up are as follows:

- **Speed Up Problem Formulation**: We provide a framework for addressing speed up problem and study two major aspects of it - *Speed Up with no constraints* and *Fine Grained Selective Speed Up* based on the restriction on amount of speed up requested.

- **Proposed efficient Speed Up algorithms**: In this thesis, we provide implicit and efficient speed up algorithms to achieve speed up for urgent jobs compared to the existing speed up algorithms which are computationally expensive.

- **Defined metrics for speed up algorithms evaluation**: We define different metrics for analyzing and evaluating the effectiveness of speed up algorithms such as percentage of urgent jobs that achieved speed up, percentage of slowed down non-urgent jobs, mean wait time, 95 percentile, degree of non urgent job impact etc.

- **Applications - Web Scheduling and CPU Scheduling**: The notion of implicit speed up is examined in two different applications - Web Scheduling and CPU Scheduling. We show the usefulness of our proposed Speed Up techniques in these domains using a simulation based model.

- **Speed Up as a scheduling algorithm evaluation metric**: We examine how speed up provides a new mechanism to compare and contrast different scheduling algorithms based on their speed up/slow down characteristics.

1.3 Organization

The rest of the thesis is organized as follows. In Chapter 2, we describe about how jobs could be accelerated in a simple M/M/1 Queuing model and study the impact of such acceleration on performance measures of M/M/1 queuing model. We model the Speed Up scheduling problem in Chapter 3. We present our implicit speed up algorithms to address Speed Up problem in Chapter 4. Chapter 5 presents the experimental analysis and results of our proposed algorithms. Chapter 6 and 7 respectively describes about the application of Speed Up in Web Scheduling and CPU Scheduling including algorithms, experimental analysis and discussion. We conclude our work in Chapter 8 along with possible future work.

Chapter 2

Job Acceleration in M/M/1 Queue

In this chapter, we describe about how jobs could be accelerated in a basic queuing model and study the impact of such acceleration on queuing performance measures like mean sojourn time (total residing time in queue), mean waiting time, waiting time distribution, mean queue length etc.

2.1 M/M/1 Queuing Model

Queuing Models are used to model, analyze and estimate the performance of real systems. The queuing model assumes certain fixed distribution of job's arrival and service time (might be unrealistic) and then estimates the relevant performance measures of the system using concepts of probability and other relevant theory. The queuing model is characterized by:

- ***Arrival process of jobs:*** Usually we assume that inter-arrival times are independent. In many practical situations, jobs arrive according to Poisson stream (exponential inter-arrival times). Jobs might arrive one by one or in batch.
- ***Behavior of jobs:*** Either a job could be patient and can wait for long period of time or could leave the system after a certain threshold amount of waiting.
- ***Service times of job:*** The service time of jobs are usually assumed to be independent of inter-arrival times. For example, service time could be exponentially, geometrically distributed or deterministic in nature.
- ***Scheduling discipline:*** The jobs present in the queue can be served in different order based on the assumed scheduling technique. For example, first come first serve in order of arrival time, random order, priorities etc.
- ***Service Capacity:*** The service capacity is basically the number of servers available for the service of jobs. There might be a single server or multiple servers for the service of arriving jobs.

- ***Waiting room:*** There might be some limitation on the number of jobs in the system waiting for their service.

Kendall provided a shorthand three part notation *a/b/c* in order to characterize the range of queuing models that could be designed based on the above parameters. The first letter denotes the inter-arrival time distribution and the second one is for service time distribution. For example, *M* stands for exponential distribution, *D* for deterministic times etc. The third parameter denotes the number of servers. Thus, *M/M/1* queuing model assumes inter-arrival as well as service times of jobs to be exponentially distributed and consists of a single server. Let λ and μ respectively denotes the arrival rate and service rate for *M/M/1* queuing model. We require that $\rho = \frac{\lambda}{\mu}$ should be less than 1, otherwise the queue length would grow infinitely.

2.2 Job acceleration in M/M/1 queue

In *M/M/1* queuing model, first come first serve technique is used as an order of service of jobs present in the queue. If the queue length is large, then an arriving job has to wait until all the jobs ahead of it receive their service. In this section, we study how waiting jobs present in the *M/M/1* queue can be accelerated to the server and how such acceleration affects the mean performance measures of *M/M/1* queue and their distribution over the jobs. We propose following queuing models to facilitate job acceleration in *M/M/1* queue.

2.2.1 M/M/1/Probability Queuing Model

Jobs arrive to the system with exponential inter-arrival times. Single server is present in the system for the arriving jobs having exponential service time distribution. In this queuing model, each job waiting in queue from position 2 to end of the queue is assigned a probability p with which it can be accelerated to the server for service. The job at position 1 in front of server would be served next with a probability $(1 - p)$. No two jobs can simultaneously jump to the server at the same time. Acceleration of a waiting job in $M/M/1/Probability$ queue would take place just after the completion of currently executing job. Thus, after the execution of any job either one of the job waiting in queue from 2^{nd} position to the end would be accelerated (chosen at random) to the server with probability p or the job at 1st position would be served next with probability $(1 - p)$.

Note that for probability parameter p equal to zero, the $M/M/1/Probability$ queuing model would become standard $M/M/1$ queuing model, since the order of service would become FCFS for $p = 0$. If value of p is nearly equal to 1, then the job at position 1^{st} (in front of the server) would be starved (since $(1 - p) \to 0$). Thus, value of p should neither be very high otherwise it would result in starvation for job at 1st position (in front of the server), nor very low otherwise it would not serve the purpose of job acceleration in queue.

2.2.2 M/M/1/Speed Queuing Model

In this queuing model, when i^{th} job enters the system the number of jobs waiting in queue including itself is stored as n_i. After arrival of any job in the system, mean queue length $\bar{n_i}(= \frac{\sum_{k=1}^{i} n_i}{i})$ is updated. If n_i is greater than $\bar{n_i}$, then any job from position mean queue length to the end of the queue is picked at random and is accelerated to the server else, job at position 1 in front of the server gets served. This queuing model could make jobs from position 1 to mean queue length to wait for too long.

2.2.3 M/M/1/FairSpeed Queuing Model

This queuing model is designed in order to overcome the starvation problem raised in $M/M/1/Speed$ queuing model. In this model, acceleration of jobs present in the queue is in accordance with the conditions described in $M/M/1/Speed$ queuing model. However, the only difference is that here the acceleration of jobs (as per $M/M/1/Speed$ model) and first come first serve techniques are used alternately so as to accelerate jobs present in the queue but at the same time ensuring less scope of starvation.

2.3 Mean Performance Measures

Let λ and μ respectively be the arrival rate and service rate of jobs for our proposed queuing models. Let ρ be defined as the ratio of λ and μ. Coefficient of variation for any distribution is defined as the ratio of standard deviation to the mean for that distribution. According to *Pollaczek-Khinchine* [17] mean value formula for poisson arrival of jobs for *any queuing discipline*, the mean number of jobs in the system in steady state, $E(L)$ is given by the following equation:

$$E(L) = \rho + \rho^2 \frac{1 + C^2}{2(1 - \rho)} \tag{2.1}$$

where C is coefficient of variation for service time distribution.

For our proposed queuing models, value of C would be 1 since standard deviation is equal to mean for exponential service time distribution. Thus, on applying *Pollaczek-Khinchine mean value* formula for our proposed queuing models, we get

$$E(L) = \rho + \rho^2 \frac{1 + 1}{2(1 - \rho)}, \tag{2.2}$$

thus,

$$E(L) = \frac{\rho}{1 - \rho} \tag{2.3}$$

Sojourn time for the job is the total time spent by it in the system from its arrival till it finishes. We will apply *Little's Law* for the calculation of *mean sojourn time* denoted by $E(S)$ for our proposed

queuing models. It is important to note that *Little's Law* is not influenced by arrival time distribution, service order, service distribution or practically anything else. By *Little's Law*:

$$E(L) = \lambda E(S) \tag{2.4}$$

Thus, we get

$$E(S) = \frac{E(L)}{\lambda}, \tag{2.5}$$

$$E(S) = \frac{1}{\mu(1-\rho)} \tag{2.6}$$

By definition of Mean Sojourn Time $E(S)$, we have

$$MeanSojournTime = MeanWaitingTime + MeanServiceTime(1/\mu) \tag{2.7}$$

implying Mean Waiting Time $E(W)$ given by

$$E(W) = E(S) - \frac{1}{\mu}, \tag{2.8}$$

$$E(W) = \frac{\rho}{\mu(1-\rho)} \tag{2.9}$$

Also,
Mean Queue Length $E(L^q)$ = Mean number of jobs in system $-$ Mean number of jobs in service.
Thus, mean queue length $E(L^q)$ is given by,

$$E(L^q) = E(L) - \rho, \tag{2.10}$$

$$E(L^q) = \frac{\rho^2}{1-\rho} \tag{2.11}$$

The mean performance measures for our proposed queuing models obtained by 2.3,2.6,2.9 and 2.11 are exactly similar for standard $M/M/1$ queuing model and thus we can conclude that the mean performance measures for $M/M/1$ queuing model is independent of the scheduling discipline. Even though the mean performance measures are same, however there is difference in the waiting time distribution of our proposed queuing models as compared to standard $M/M/1$ model and the results for the same are presented in the next section.

2.4 Results and Conclusions

We coded $M/M/1$, $M/M/1/Probability$, $M/M/1/FairSpeed$ and $M/M/1/Speed$ queuing models in C++ and conducted experiments for different values of $\rho(=\frac{\lambda}{\mu})$. The number of jobs we considered were 1 million. The RAM of the system is 3 GB and processor speed 2.4 GHz. Value of μ used is 5 for all the experiments.

Figure 2.1, 2.2, 2.3 and 2.4 shows the waiting time distribution for $M/M/1/Probability$ queuing

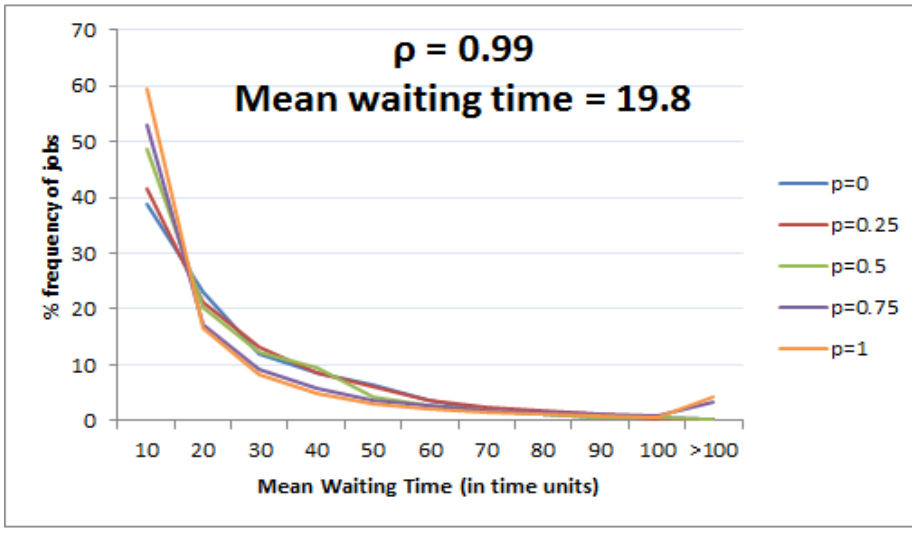

Figure 2.1 $\rho = 0.99$. Waiting time distribution for M/M/1/Probability queuing model for different values of p.

model for different values of ρ. It is important to note that for $p = 0$, the graph line corresponds to standard $M/M/1$ queuing model. Experimental results (Table 2.1) show that the frequency of jobs corresponding to very low waiting time (less than mean waiting time) increases as the value of probability increases and is greater for $M/M/1/Probability$ queuing model for $p > 0$ as compared to $M/M/1$ queuing model. It is because the tendency of job acceleration increases with increase in p and thus, most of the arriving jobs get accelerated to the server resulting in their low waiting time. The frequency of jobs corresponding to low wait time is thus, maximum for $p = 1$. Because of acceleration of some of the jobs, other jobs are slowed down and thus, penalized resulting in high waiting time. It can be seen that the % frequency of jobs corresponding to extremely high waiting time increases with increase in p value. It is because as p increases, the probability of the job at position 1 to get service decreases ($(1 - p)$ decreases as p increases). This results in high waiting time for jobs present near to the server and such penalization can be verified by examining the maximum waiting time for different values of p from Table 2.1. Experimental results verify that as system load ρ increases, the mean wait time also increases. Changing the value of p though doesn't affect the mean performance measures however, it do have an impact on variance of wait time. It is experimentally found from Table 2.1 that as p increases, variance (square of standard deviation) of wait time also increases.

It is experimentally observed that the graph lines corresponding to different values of p tend to meet each other (Figure 2.4) as the value of ρ decreases. It can be explained using the fact that mean queue length from eq. 2.11, is proportional to the square of ρ. As ρ decreases, the mean queue length also decreases and thus, when queue length becomes very small (small value of ρ) there would be negligible effect of changing the probability value on waiting time distribution.

Figure 2.5, 2.6, 2.7 and 2.8 shows the waiting time distribution for $M/M/1$, $M/M/1/FairSpeed$

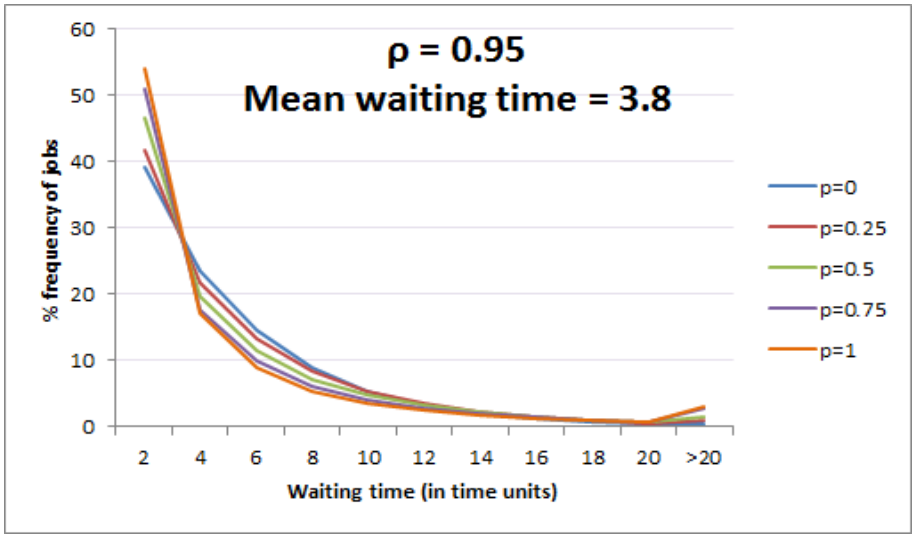

Figure 2.2 $\rho = 0.95$. Waiting time distribution for M/M/1/Probability queuing model for different values of p.

Table 2.1 $\rho = 0.95$ 25 and 95 percentile metric for wait time and Maximum waiting time for M/M/1/Probability queuing model for different values of p. μ_w - mean wait time and σ_w - standard deviation.

Probability	25 percentile	95 percentile	Max wait time	σ_w	μ_w
$p = 0$	0.97	12.10	40	4	3.8
$p = 0.25$	0.82	12.7	51	4.27	3.8
$p = 0.5$	0.61	13.78	66	4.92	3.8
$p = 0.75$	0.44	15.04	71.87	5.66	3.8
$p = 1$	0.36	15.45	1478	8.38	3.8

and $M/M/1/Speed$ queuing models for different values of ρ. The experimental results show that $M/M/1/FairSpeed$ provides less scope of starvation (in terms of maximum wait time from Table 2.2 as well as % frequency of jobs corresponding to extremely high wait time from Fig. 2.5, 2.6, 2.7, 2.8) as compared to $M/M/1/Speed$ queuing model but $M/M/1/Speed$ model is able to accelerate greater number of jobs as compared to $M/M/1/FairSpeed$ queuing model. The waiting time distribution plot lines for $M/M/1$, $M/M/1/FairSpeed$ and $M/M/1/Speed$ queuing models tend to meet each other as value of ρ decreases. From table 2.2, it can be concluded that $M/M/1/FairSpeed$ queuing model is able to accelerate greater number of jobs as compared to $M/M/1$ queuing model and at the same time ensuring less scope of starvation as compared to $M/M/1/Speed$ queuing model in terms of maximum wait time.

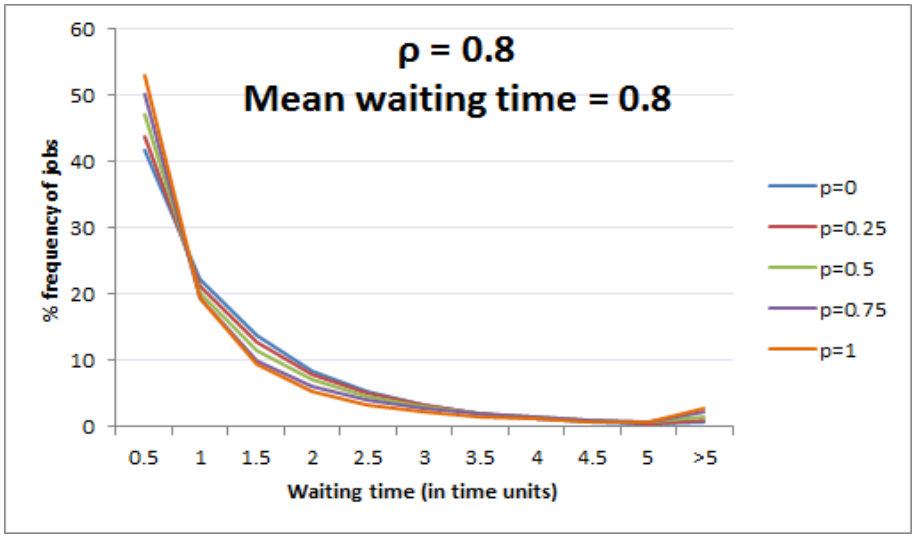

Figure 2.3 $\rho = 0.8$. Waiting time distribution for M/M/1/Probability queuing model for different values of p.

Table 2.2 $\rho = 0.95$ Maximum waiting time and 25,95 percentile metric for wait time for M/M/1, M/M/1/FairSpeed and M/M/1/Speed queuing model.μ_w - mean wait time and σ_w - standard deviation.

Queuing model	25 percentile	95 percentile	Max wait time	σ_w	μ_w
M/M/1	0.97	12.10	40	4	3.8
M/M/1/FairSpeed	0.55	14.99	66	5.45	3.8
M/M/1/Speed	0.41	12.93	1869	17.02	3.8

2.5 Summary

In this chapter, we examined how waiting jobs present in simple M/M/1 queuing model can be accelerated to the server and studied the impact of such acceleration on performance measures of M/M/1 queuing model. We proposed three queuing models in order to facilitate job acceleration in M/M/1 queue. Using queuing theory, we find that mean performance measures like mean sojourn time, mean wait time, mean queue length etc. in steady state remains the same for M/M/1 queuing model as compared to our proposed three queuing models irrespective of the scheduling discipline. However, acceleration of jobs in M/M/1 queue do affect their waiting time distribution.

M/M/1/Probability queuing model accelerate jobs based on the value of probability parameter p. High value of p ensures faster execution of arriving jobs and could lead to high waiting time for job at position 1. The M/M/1/Speed model uses the value of mean queue length to accelerate jobs present from mean queue length position to the end. In order to reduce the starvation problem for jobs from position 1 to mean queue length, a fairer version M/M/1/FairSpeed model is proposed which facilitates job acceleration but at the same time avoid starvation (high wait time) for jobs present near to the server. The graph lines corresponding to wait time distribution for different queuing models tend to meet each

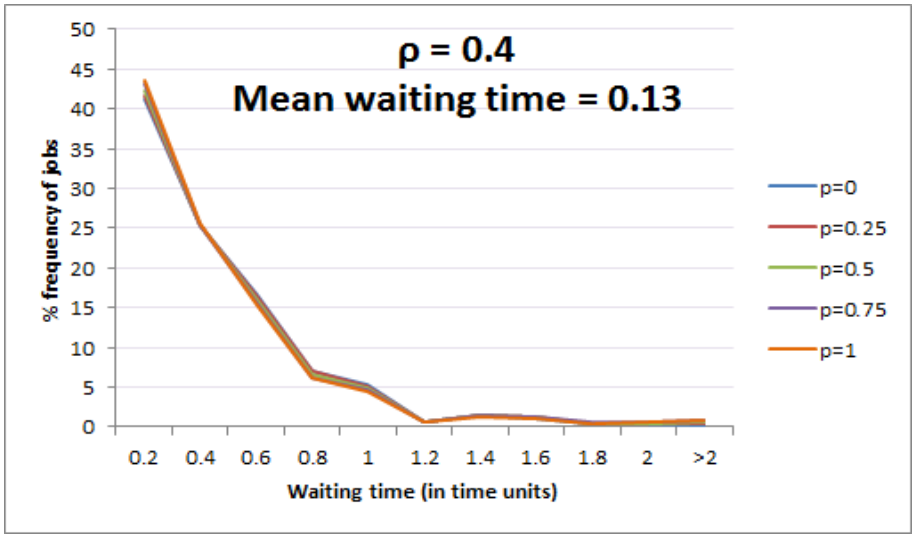

Figure 2.4 $\rho = 0.4$. Waiting time distribution for M/M/1/Probability queuing model for different values of p.

other as ρ approaches very small value showing that scheduling discipline would not have any significant influence on wait time distribution in that case.

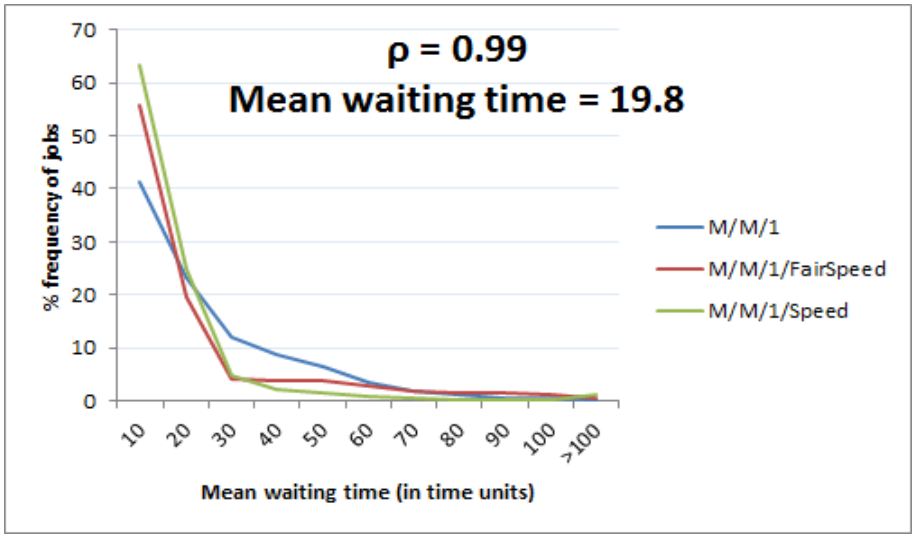

Figure 2.5 $\rho = 0.99$. Waiting time distribution for M/M/1/, M/M/1/Speed and M/M/1/FairSpeed queuing models.

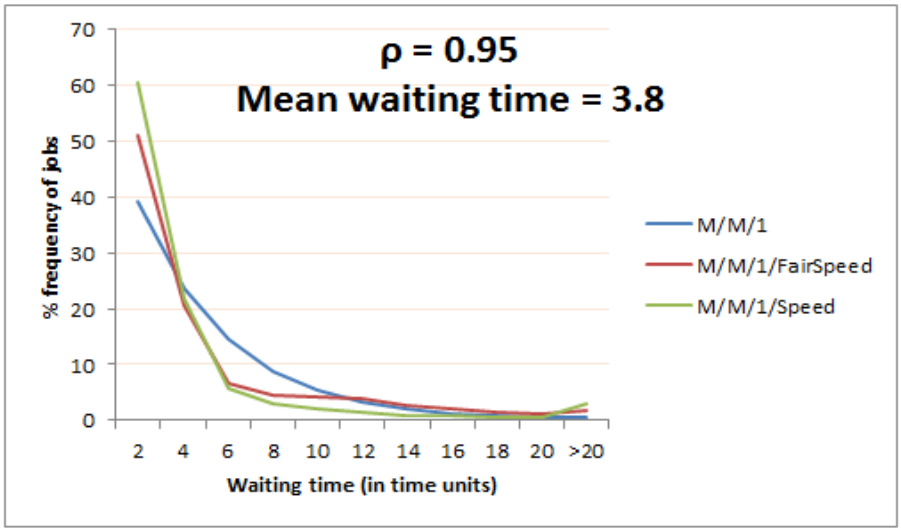

Figure 2.6 $\rho = 0.95$. Waiting time distribution for M/M/1/, M/M/1/Speed and M/M/1/FairSpeed queuing models.

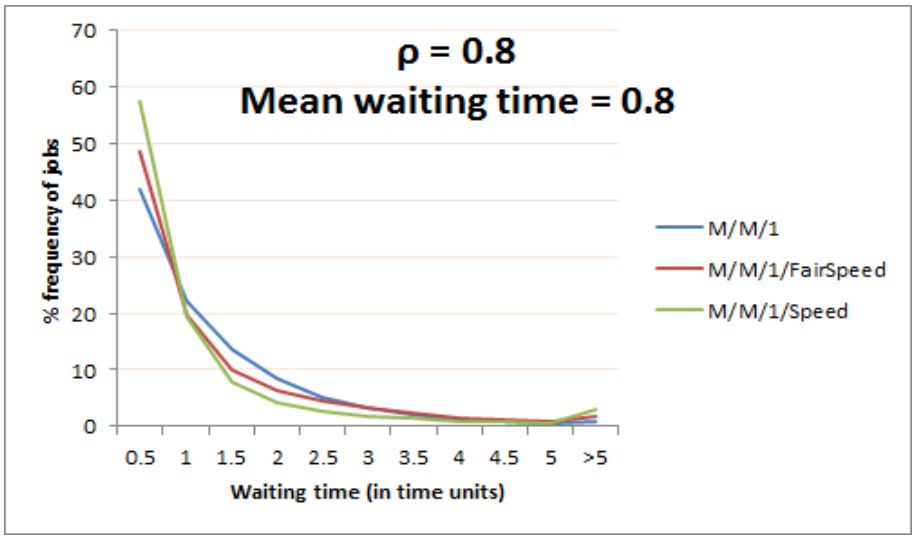

Figure 2.7 $\rho = 0.8$. Waiting time distribution for M/M/1/, M/M/1/Speed and M/M/1/FairSpeed queuing models.

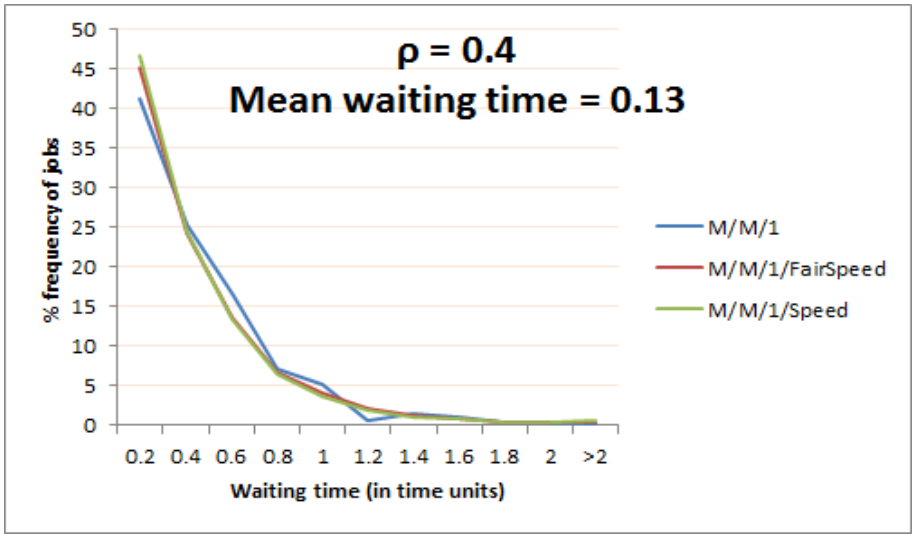

Figure 2.8 $\rho = 0.4$. Waiting time distribution for M/M/1/, M/M/1/Speed and M/M/1/FairSpeed queuing models.

Chapter 3

Speed Up Problem

In this chapter, we model the Speed Up problem *without acquiring additional resources* in order to speed up all the jobs that requested for it. We present the different issues that arise while speeding up jobs and explain how such acceleration would impact other urgent (requesting for speed up) as well as non-urgent jobs present in the queue.

Throughout this work, a single server queuing model is used for addressing the problem of speed up. Even in the presence of multiple queues, we need to support speed up within every single queue associated with a specific server. Load balancing (movement of jobs across different queues to equally distribute workload on multiple servers) is a separate problem which may or may not help in achieving speed up (when the queues are equally loaded). Supporting speed up through load balancers is beyond the scope of this work.

3.1 Problem Formulation

Consider jobs arriving at a system with respect to time. The system comprises of a single server which can serve at most one job at any point of time and a queue where the jobs accumulate and wait for their service. *The jobs are assumed to be non pre-emptable in this work.* The job once finished leaves the system. The state of the system is a snapshot of queue at a particular instant of time.

Definition 1. *State of the system at any point of time t denoted by $S(t)$, is defined as the tuple $< j_1, j_2, j_3, ..., j_l >$ where l is the queue length at time t and j_i is the job at ith position waiting in queue for service, such that $1 \leq i \leq l$ and each job j_i having arrival time strictly less than t. The job getting its service at t is not included in $S(t)$.*

The concept of speed up by itself is relative [9]. Therefore, we define an expected execution time for each job j and then define speed up with respect to it. The expected execution time for job j (expected total time spent in the system from its arrival till it is completely finished) with respect to scheduling policy P is defined as the summation of the processing time of all the jobs that would be executed

before j depending upon P plus the processing time of job j. Note that the processing time of job is the service time of job also referred to as duration of job. If P scheduling policy is assumed to be FCFS, then expected execution time for job j would be defined as summation of processing time of all the jobs present in the queue which arrived before j plus the processing time of j. *Note that, the definition of expected execution time would change if some other scheduling technique is assumed.* Throughout this work, we use the term expected execution time with respect to FCFS scheduler.

Definition 2. *The expected execution time for job j with respect to FCFS scheduler, arriving at time t, denoted by $T_{exp}(j)$ is defined as*

$$T_{exp}(j) = d(j) + \sum_{\forall k \in S(t)} d(k) + d_{rem}(j_s), \tag{3.1}$$

where $d(k)$ denotes the processing time/duration for job k and $d_{rem}(j_s)$ denotes the remaining processing time of job j_s being served at the server. If there is no job at server being served at t, then value of $d_{rem}(j_s)$ is zero.

Definition 3. *The actual execution time for job j, denoted by $T_{actual}(j)$ is the actual total time spent by the job in the system from its arrival time till it leaves the system (completely finishes).*

$$T_{actual}(j) = T_{finish}(j) - T_{arrival}(j) \tag{3.2}$$

where $T_{finish}(j)$ and $T_{arrival}(j)$ respectively denotes the finish time (time at which job completes its execution) and arrival time for job j.

Note that the *actual execution time* for job can be computed only after the job finishes.

Definition 4. *A job j is said to achieve speed up iff $T_{exp}(j) - T_{actual}(j) > 0$ and is said to be slowed down iff $T_{exp}(j) - T_{actual}(j) < 0$.*

Note that jobs for which their expected execution time *is equal to* their actual execution time, are neither speeded up nor slowed down. If first come first serve scheduling would have been used, then actual execution time will be equal to expected execution time for all of the jobs and thus, jobs will neither be speeded up nor slowed down. Depending on the values of expected and actual execution time we can say whether a specific job is *speeded up* or *slowed down*.

We dynamically label some of the arriving jobs as urgent (*i.e.* they are requesting speed up). The objective of *speed up problem* is to speed up all the jobs that are requesting for speed up without penalizing more than necessary to the rest of the jobs present in the queue, while keeping the mean wait time low for all the jobs. It might not be possible to achieve speed up for all jobs which requested for it. For example, consider a scenario where all the jobs present in the queue are requesting for speed up, then it

Table 3.1 Information of jobs arriving to the system.

Job Id	Arrival Time	Job Duration
j_0	0	5
j_1	1	5
$j\prime_2$	2	3
$j\prime_3$	3	4
j_4	4	3
$j\prime_5$	5	5

is impossible to achieve speed up for all. Also, it might be the case that in the process of speeding up of some of the urgent jobs, even some of the jobs that requested for speed up were slowed down. This case may arise when an urgent job having high processing time is accelerated ahead of another urgent job in the queue.

Consider a queue of jobs as $< j_1, j_2, j_3, ..., j_l >$ present in the system at some point of time t where l is the queue length at t. Suppose job j_i for some $1 < i \leq l$ has requested for speed up. In order to achieve speed up for job j_i, it needs to be accelerated to the server. This acceleration will have no impact on all the jobs j_k where $i + 1 \leq k \leq l$. But all jobs j_p such that $1 \leq p \leq i - 1$ ahead of j_i will suffer a delay equal to $d(j_i)$ (duration time of job j_i). Let there be an urgent job j_k for some $k \in [1, i-1]$ which is temporarily delayed because of the acceleration of j_i, but it is well possible that the job j_k later got accelerated so that eventually it achieved *speed up* despite of facing an initial delay. *Therefore it is not always the case, that a delayed job will be slowed down by the time it completes its execution.* Thus, priority/importance of any job at any point of time t dynamically changes depending on how much it has been delayed till t because of acceleration of other jobs. The scheduling decisions need to be made according to the on-line speed up requests posed by the user at run-time.

Definition 5. *A job j is said to achieve the speed up of x time units iff $T_{exp}(j) - T_{actual}(j) = x$ and $x > 0$ and is said to be slowed down by z time units iff $T_{actual}(j) - T_{exp}(j) = z$ and $z > 0$.*

Example 1. *Consider the jobs arriving as per Table 3.1. Let jobs $j\prime_2$, $j\prime_3$ and $j\prime_5$ are the jobs which are requesting for speed up. Suppose the order of schedule is $j_0, j\prime_5, j\prime_3, j\prime_2, j_4, j_1$. First job j_0 is picked by the server for service at time $t = 0$. At $t = 0$ since there is no job in the queue therefore, $T_{exp}(j_0)$ is 5 equal to its own duration time. Thus, job j_0 is neither speeded up nor slowed down. At $t = 1$, job j_1 arrives and it finds that there is no job in the queue waiting for its service and j_0 is being served by the server, therefore $T_{exp}(j_1)$ is $5+4$ (remaining service time of j_0) that is 9 time units. At $t = 2$, job $j\prime_2$ finds job j_1 waiting in the queue, and thus $T_{exp}(j\prime_2)$ is 3 ($d(j\prime_2)$) $+ 5$ ($d(j_1)$) $+ 3$ (remaining service time of j_0) that is 11 time units. Similarly, expected execution time values for $j\prime_3$, j_4 and $j\prime_5$ are respectively 14, 16 and 20 units. At $t = 5$, when j_0 finishes, urgent job $j\prime_5$ is speeded up to the server for service. Then*

Table 3.2 Speed Up/Slow down information.

Job Id	Arrival Time	Job Duration	T_{exp}	T_{actual}	Speed Up/Slow Down
j_0	0	5	5	5	no speed-up/slow-down
j_1	1	5	9	24	slow down by 15 units
$j\prime_2$	2	3	11	15	slow down by 4 units
$j\prime_3$	3	4	14	11	speed up by 3 units
j_4	4	3	16	16	no speed-up/slow-down
$j\prime_5$	5	5	20	5	speed up by 15 units

all the jobs from j_1 to j_4 will be delayed by 5 time units including the urgent jobs $j\prime_2$ and $j\prime_3$. The job $j\prime_5$ finishes at t = 10. The actual execution time for $j\prime_5$ is thus, 5 units. Since $T_{exp}(j\prime_5) > T_{actual}(j\prime_5)$, $j\prime_5$ is said to achieve speed up of 20-5 = 15 units. Now the delayed job $j\prime_3$ is selected for service after $j\prime_5$ at $t = 10$. Then $j\prime_3$ will finish at $t = 14$. The $T_{actual}(j\prime_3)$ is thus $14 - 3 = 11$ time units. However, the expected execution time for $j\prime_3$ was 14 units. Thus even after facing the initial delay, $j\prime_3$ is successful in achieving speed up of 3 time units. Now job $j\prime_2$ will be scheduled next and will finish at $t = 17$. The actual execution time for $j\prime_2$ is thus 17-2 = 15 units. However, the expected execution time for $j\prime_2$ is 11 units. Thus, job $j\prime_2$ is slowed down by 15-11 = 4 units despite of requesting for speed up. Similarly job j_4 followed by j_1 executes. The information about expected execution time, actual execution time, speed up/slow down for this example as per the supposed schedule is shown in Table 3.2.

3.2 Problem Classification

The problem of speed up can be classified into following two categories depending upon whether the urgent jobs are requesting for an exact amount of speed up or not :

1. *Speed Up with no constraints:* In this problem, some of the jobs present in the queue are requesting *speed up*, however there is no constraint on the amount of speed up that they request. The purpose is to achieve speed up for as many urgent jobs as possible without unnecessarily slowing down the other jobs while keeping the mean wait time low for all the jobs.

2. *Fine Grained Selective speed up:* In this problem, the urgent jobs request *speed up* of some specific time units. The maximum amount of *speed up* that a job j can achieve is clearly equal to the summation of duration times of all the jobs that are in front of it at the time of it's arrival. So, each urgent job can request for some percentage of the maximum *speed up* that it can achieve at that time. Formally, amount of *speed up* requested by an urgent job j denoted by $RSU(j)$

arriving at time t, is given by

$$RSU(j) = (\frac{p_j}{100}) \times \sum_{\forall k \in S(t)} d(k) \tag{3.3}$$

for some $p_j \in (0, 100]$ where $RSU(j)$ stands for ***Requested Speed Up*** by job j, $S(t)$ represents the system state at time t and $d(j)$ is the duration time of job j.

The objective of *Fine Grained Selective Speed Up* is to achieve (almost) exact requested *speed up* for as many urgent jobs as possible without unnecessarily slowing down other jobs while keeping the mean wait time low for all the jobs.

To summarize, *Speed Up* problem aims at meeting the following objectives:

- To maximize the number of urgent jobs which achieve *speed up* (exact requested *speed up* in case of fine grained selective speed up).
- Avoid unnecessary *slow down* for other jobs while accelerating urgent jobs.
- Reducing the overall mean wait time for all the jobs.
- Avoid starvation for any of the job (whether urgent or non-urgent).

3.3 Summary

In this chapter, we modeled Speed Up problem without acquiring additional resources and classify the problem in two major categories - Speed Up with no constraints and Fine Grained Selective Speed Up. The problem of Fine Grained Selective Speed Up has not been studied in literature before and is a specific case of Speed Up problem. Our speed up algorithms to address both the cases of speed up problem are explained and discussed in next chapter.

Chapter 4

Speed Up Algorithms

In this chapter, we present speed up algorithms in order to achieve speed up for the jobs that requested for it. We provide implicit techniques where the notion of acceleration is incorporated in the priority function. We present following three strategies for addressing *speed up* problem depending on the amount of slow down that can be tolerated for the non-urgent jobs:

1. *Urgent Dominant Speed Up (UDSU):* The focus of this technique is to accelerate the urgent jobs as much as possible without even bothering about the *slow down* caused to other non-urgent jobs while keeping the mean wait time low. This strategy gives very high priority to the urgent jobs and might even starve non-urgent jobs in the presence of continuous arrival of urgent jobs.

2. *Non-Urgent Benevolent Speed Up (NUBSU):* This strategy gives high priority to the urgent jobs, but at the same time it opportunistically tries to accelerate *slowed down* non urgent jobs as long as such acceleration is not a hurdle in achieving speed up for rest of the urgent jobs present in the queue at that time.

3. *Fair Speed Up (FSU):* This strategy is fair to all the jobs and does not lead to high *slow down* for any of the job. The *Fair Speed Up* strategy is neither biased towards urgent jobs nor does create starvation for any job while at the same time aiming at reducing the overall mean wait time. With this strategy we can still implicitly achieve speed up for some jobs.

The priority function for job j, denoted by $P(j)$ is defined as follows:

$$P(j) = T_{arrival}(j) * d(j) \tag{4.1}$$

where $T_{arrival}(j)$ denotes the arrival time of job j and $d(j)$ denotes it's duration time.

The job with lower value of priority function is preferred. The idea is to give preference to small size jobs (to improve mean wait time) but this preference takes into consideration the arrival time of job to avoid starvation. The priority function ensures no starvation for jobs while still preferring shorter jobs (to improve mean wait time) because for each job j, since job sizes are finite, there exists certain arrival time (for some other job $j\prime$) $T_{arrival}(j\prime) > T_{arrival}(j)$ such that all the jobs $j\prime$ after j (irrespective of

their sizes) will have higher priority value than j. Thus, each job j would eventually be served.

Though the priority function has some commonality with existing scheduling techniques, but we study it in context of speed up and slow down. Further, we extend our priority function for different applications (refer Chapter 5 and 6).

Definition 6. *Let $U(t)$ be any set comprising of jobs present in the system at time t including the jobs that arrive at t. A job j is said to be Candidate with respect to set $U(t)$ for some function $f(j)$ denoted by $C_{U,f}(t)$, iff $j \in U(t)$ and $f(j)$ is minimum among all the jobs present in $U(t)$. If there are multiple such jobs with minimum function value, then $C_{U,f}(t)$ is the one with shortest duration time among them.*

At any point of time t, let $J_{urgent}(t)$ and $J_{nonurgent}(t)$ respectively denote the set of all the *urgent* (jobs requesting for speed up) and *non-urgent* jobs waiting in the queue for service.

4.1 Speed Up with no constraints

In this problem, some of the jobs present in the queue are requesting for speed up. However, there is no constraint on the amount of speed up requested. The objective is to achieve speed up for urgent jobs (actual execution time should be less than expected execution time), without penalizing more than necessary to the rest of the jobs.

4.1.1 Urgent Dominant Speed Up Algorithm (UDSU)

Algorithm 1 UDSU algorithm.

Input: Input queue of jobs (queue size ≥ 1) at time t.
Output: Next job to be executed.

1: $j \leftarrow$ *null*.
2: **if** $J_{urgent}(t) = \phi$ **then**
3: $j \leftarrow C_{J_{nonurgent},P}(t)$
// $T_{arrival}(j) * d(j)$ is minimum and $j \in J_{nonurgent}(t)$
4: **else**
5: $j \leftarrow C_{J_{urgent},P}(t)$
// $T_{arrival}(j) * d(j)$ is minimum and $j \in J_{urgent}(t)$
6: **end if**
7: **return** j

UDSU Algorithm accelerates the urgent jobs to the server based on the priority value of it. The urgent job with least P (as defined in 4.1) function value is chosen (in case of ties the one with shorter duration time). If there is no urgent job present in the queue, then the non-urgent job with least P priority function value is chosen. The notion of acceleration is incorporated in the priority function.

This algorithm completely ignores the non-urgent jobs in the presence of urgent jobs and focuses on accelerating as many urgent jobs as possible, thus referred to as *Urgent Dominant Speed Up* algorithm. Note that this algorithm will cause the other *non-urgent* jobs to have arbitrarily high *slow down* or can even create starvation for them in the presence of continuous arrival of *urgent* jobs.

To implement this algorithm, we need to maintain two separate *priority queues* one for the *urgent* and the other one for *non-urgent* jobs, based on their priority value. The time complexity of *UDSU* algorithm to pick up the next job at time t is $O(log(\ |J_{urgent}(t)|) + log(\ |J_{nonurgent}(t)|)\ )$ and is quite efficient.

4.1.2 Non-Urgent Benevolent Speed Up Algorithm (NUBSU)

This algorithm gives high priority to the jobs which are requesting speed up, but is not unfair towards other jobs. It optimistically tries to accelerate other jobs as long as such acceleration is not posing any hurdle in achieving speed up for existing urgent jobs.

Definition 7. *A job j is said to be opportunistically forwardable at time t if $j \in J_{nonurgent}(t)$ and $\forall k \in J_{urgent}(t)$,*

$$[T_{wait}(k,t) + d(j) + \sum_{P(k\prime)\leq P(k)} d(k\prime) + d_{rem}(j_s)] < T_{exp}(k) \tag{4.2}$$

where $k\prime \in J_{urgent(t)}$ such that $P(k\prime) \leq P(k)$, $T_{wait}(k,t)$ denotes the waiting time of job k till time t and $d_{rem}(j_s)$ is the remaining duration time of job j_s getting serviced at server at t if any else $zero$.

Algorithm 2 NUBSU algorithm

Input: Input queue of jobs (queue size ≥ 1) at time t.
Output: Next job to be executed.

1: $j \leftarrow$ *null*.
2: **if** $J_{nonurgent}(t) = \phi$ **or** $C_{J_{nonurgent},P}(t)$ is **not** *opportunistically forwardable* at t **then**
3: $j \leftarrow C_{J_{urgent},P}(t)$
// $T_{arrival}(j) * d(j)$ is minimum and $j \in J_{urgent}(t)$
4: **else**
5: $j \leftarrow C_{J_{nonurgent},P}(t)$
// $T_{arrival}(j) * d(j)$ is minimum and $j \in J_{nonurgent}(t)$
6: **end if**
7: **return** j

A non urgent job $j_{nonurgent}$ is *opportunistically forwardable* at time t, if it's acceleration to the server for service will not affect the existing urgent jobs (present in queue) in achieving their speed up at t. That is, existing urgent jobs can achieve *speed up* even after the acceleration of non-urgent job

$j_{nonurgent}$ to the server for service. In such case, each urgent job j_{urgent} has to further wait for all the other urgent jobs which have lower $P(j)$ function value (higher preference) than j_{urgent} plus the duration time of $j_{nonurgent}$ since $j_{nonurgent}$ is *opportunistically forwardable* and thus, will be accelerated for service before j_{urgent}. Thus, for j_{urgent} to achieve *speed up*, the time that it has to further wait (including it's own duration time) plus the time that it has already spent waiting in queue till now should be less than it's expected execution time.

Non Urgent Benevolent Speed Up Algorithm gives high priority to the urgent jobs, but is still optimistic towards non urgent jobs in a sense that it accelerates them whenever such acceleration does not pose any hurdle for urgent jobs in achieving their speed up. The important question is does acceleration of *opportunistically forwardable* job always ensure that the other urgent jobs will achieve *speed up*? The answer is no. Our algorithms are on-line and we do not have any information about the jobs arriving in future. So it may happen that at time t, a non-urgent job $j_{nonurgent}$ is *opportunistically forwardable* and thus is forwarded to the server for service, but during the execution of $j_{nonurgent}$, a bunch of urgent jobs arrives in queue say at time $t + \delta t$ and some of which are having their $Priority$ function value lower than some existing urgent job j_{urgent} (might be because of very low duration time). So j_{urgent} for which summation of duration times of all the urgent jobs having lower $Priority$ function value plus duration of $j_{nonurgent}$ was less than $T_{exp}(j_{urgent})$ at time t, can now become greater than $T_{exp}(j_{urgent})$ at time $t + \delta t$ (because of arrival of some urgent jobs which would have been executed before j_{urgent} because of their low P function value). Thus, j_{urgent} cannot achieve speed up. This situation is analogous to non preemptive SJF scheduling where recently arrived short job has to wait for currently executing long job to finish (since the algorithm is on-line and non preemptive). To measure the probability of such events, we define the notion of *successful/unsuccessful* opportunistically forwarded non urgent jobs. A *successful opportunistically forwarded non-urgent job* $j_{nonurgent}$ is the one whose acceleration to the server for service does not impact any other existing urgent job in achieving speed up by the time it complete its execution. Similarly, *unsuccessful opportunistically forwarded job* $j_{nonurgent}$ is the one whose acceleration prevented some of the existing urgent jobs in achieving their speed up.

Definition 8. *Degree of Non Urgent Job Impact (DNUJI) for NUBSU algorithm is defined as the ratio of number of unsuccessful opportunistically forwarded jobs to the total number of opportunistically forwarded jobs.*

$$DNUJI = \frac{NumberOfUnsuccessfulOpportunisticallyForwardedJobs}{TotalNumberOfOpportunisticallyForwardedJobs} \tag{4.3}$$

To implement *NUBSU* algorithm, we maintain a *priority queue* for non urgent jobs based on P function value. For urgent jobs, we maintain a AVL tree based on their P function value. To check whether $C_{J_{nonurgent},P}(t)$ is opportunistically forwardable, we need to traverse the AVL tree in increasing order of P function values by doing an in-order traversal of tree and checking for each urgent job k,

whether

$$[T_{wait}(k,t) + d(C_{J_{nonurgent},P}(t)) + \sum d(k\prime) + d_{rem}(j_s)] < T_{exp}(k) \tag{4.4}$$

where $k\prime \in J_{urgent(t)}$ such that $P(k\prime) \leq P(k)$ and $d_{rem}(j_s)$ denotes the remaining duration time of currently executing job j_s. The time complexity for obtaining $C_{J_{urgent},P}(t)$ and $C_{J_{nonurgent},P}(t)$ is $O(log(\ |J_{urgent}(t)|) + log(\ |J_{nonurgent}(t)|)\ )$. However, the time complexity of determining whether $C_{J_{nonurgent},P}(t)$ is *opportunistically forwardable* is $O(\ |J_{urgent}(t)|)$ since we need to traverse the whole AVL tree containing urgent jobs at time t. Thus, the overall time complexity of *NUBSU Algorithm* for picking up the next job at time t is $O(log(\ |J_{urgent}(t)|) + log(\ |J_{nonurgent}(t)|) + |J_{urgent}(t)|\ )$. This algorithm is computationally more expensive than *UDSU* algorithm.

4.1.3 Fair Speed Up Algorithm (FSU)

This algorithm is not biased to any specific kind of job. It chooses the job which is having least P function value among all the jobs present in the queue (including both urgent and non-urgent jobs) to schedule next. If there are multiple such jobs, then the job with smaller duration time is selected. Note that, both *UDSU* as well as *NUBSU* algorithms can create starvation for non-urgent jobs in the presence of continuous arrival of urgent jobs. However, *FSU* provides less scope of starvation for any of the job. Each job would eventually be served because the next job will be the one having least P function value among all the jobs present in the queue. Therefore, a job waiting for too long will have low value of arrival time as compared to other jobs which arrived after it, and since job sizes are finite, thus a time t will always exist such that its P function value is minimum among all the jobs in the system. The major drawback of this algorithm is that it does not serve the purpose of achieving speed up for as many urgent jobs as possible, but we can still implicitly achieve speed up for some of the jobs. The benefit of this algorithm is that it does not starve any of the job along with improving the overall mean wait time.

To implement this algorithm, we need only one *priority queue* containing all the jobs present in the queue based on their P priority value. Thus, the time complexity to pick up the next job for *FSU* algorithm is $O(log\ l\ )$ where l is the length of the queue.

4.2 Fine Grained Selective speed Up

In this speed up problem, each of the urgent job requests for a specific amount of *speed up* denoted by $RSU(j)$ which from eq. 3.3 is defined as

$$RSU(j) = \frac{p_j}{100} \times (\sum_{\forall k \in S(T_{arrival}(j))} d(k)) \tag{4.5}$$

for some $p_j \in (0, 100]$.

By the definition 2 of $T_{exp}(j)$, $RSU(j)$ can also be written as

$$RSU(j) = \frac{p_j}{100} \times (T_{exp}(j) - d(j) - d_{rem}(j_s)) \tag{4.6}$$

Definition 9. *An urgent job j_{urgent} is said to be successfully speeded up iff*

$$(T_{exp}(j_{urgent}) - T_{actual}(j_{urgent})) \geq RSU(j_{urgent}) \tag{4.7}$$

The objective of *Fine Grained Selective Speed Up* is to maximize the number of *successfully speeded up* jobs without unnecessarily slowing down other jobs while keeping the mean wait time low.

Definition 10. *The current speed up for an urgent job j_{urgent} at time t, denoted by $CSU(j_{urgent}, t)$ is defined as follows:*

$$CSU(j_{urgent}, t) = T_{exp}(j_{urgent}) - (T_{wait}(j_{urgent}, t) + d(j_{urgent}) + d_{rem}(j_s)) \tag{4.8}$$

where

$$T_{wait}(j_{urgent}, t) = t - T_{arrival}(j_{urgent}) \tag{4.9}$$

The *current speed up* at point of time t for job j represents the amount of speed up that it will achieve if it would have been accelerated next to the server for service just after the execution of job currently being served.

The priority of an urgent job j denoted by $P_{urgent}(j, t)$ is defined as:

$$P_{urgent}(j, t) = CSU(j, t) - RSU(j) \tag{4.10}$$

Using eq. 4.6, 4.8 and 4.9, we get

$$P_{urgent}(j, t) = [T_{arrival}(j) + (1 - p_j/100) * (T_{exp}(j) - d(j) - d_{rem}(j_s))] - t \tag{4.11}$$

However, the time t at which we will calculate the priority value and $d_{rem}(j_s)$ remains the same for all urgent jobs, so value of t and $d_{rem}(j_s)$ won't affect in comparing the priorities and thus, we can ignore it. Therefore,

$$P_{urgent}(j) = T_{arrival}(j) + (1 - p_j/100) * [T_{exp}(j) - d(j)] \tag{4.12}$$

The function P_{urgent} is used as priority function among urgent jobs (lower the value of function, higher the importance/preference of it). It can be inferred from eq.4.12 that a job with low arrival time requesting for high percentage of speed up having low expected execution time is preferred. Note that, an urgent job j will be *successfully speeded up* iff $CSU(j, t) \geq RSU(j)$ where t is the time at which job j was accelerated to the server for service. For non-urgent jobs, we use previous priority function P defined in eq. 4.1 for computing their priority.

In order to present the three strategies for speed up for this case, we present a *Generic Probabilistic Speed Up (GPSU)* algorithm which based on the value of probability p can mimic *UDSU*, *NUBSU* and *FSU* algorithm.

Algorithm 3 GPSU Algorithm.

Input: Input queue of jobs (queue size ≥ 1) at time t and probability parameter p
Output: Next job to be executed.

1: $j \leftarrow$ *null.*
2: **if** $J_{urgent}(t) = \phi$ **then**
3: $j \leftarrow C_{J_{nonurgent},P}(t)$
4: **return** j
5: **end if**
6: **if** $J_{nonurgent}(t) = \phi$ **then**
7: $j \leftarrow C_{J_{urgent},P_{urgent}}(t)$
8: **return** j
9: **end if**
10: Generate a random number x between 0 to 1 inclusive.
11: **if** $x \leq p$ **then**
12: $j \leftarrow C_{J_{urgent},P_{urgent}}(t)$
// $P_{urgent}(j)$ is minimum and $j \in J_{urgent}(t)$
13: **else**
14: $j \leftarrow C_{J_{nonurgent},P}(t)$
// $T_{arrival}(j) * d(j)$ is minimum and $j \in J_{nonurgent}(t)$
15: **end if**
16: **return** j

4.2.1 Generic Probabilistic Speed Up Algorithm (GPSU)

GPSU Algorithm takes a probability parameter p which is a measure of bias between urgent and non-urgent jobs. Higher the value of p, higher is the dominance of urgent jobs over non-urgent jobs. GPSU algorithm tries to mimic UDSU, NUBSU and FSU algorithm based on probability parameter p. For $p = 1$, the algorithm 3 mimics the nature of UDSU algorithm. The algorithm will completely ignore the non urgent jobs in the presence of urgent jobs and will lead to their starvation in the presence of frequent arrival of urgent jobs. For higher values of $p < 1$, the algorithm will pick urgent jobs with very high probability, however at the same time, the algorithm will not create starvation for non urgent jobs since the slowed down non urgent jobs will be picked up by their priority function (low arrival time resulting in low priority function value) with probability (1-p) thereby mimicking *NUBSU* algorithm. For p proportional to the fraction of total jobs requesting for speed up (urgent jobs), the algorithm will behave as *FSU* algorithm since we are giving equal chance to both urgent as well as non-urgent jobs.

The algorithm can be implemented by using two separate *priority queues* one for the urgent jobs (based on $P_{urgent}(j)$ function value as per eq. 4.12) and the other one for non-urgent jobs (based on P function value defined in eq. 4.1). The time complexity of this algorithm is thus, $O(log(\ |J_{urgent}(t)|) + log(\ |J_{nonurgent}(t)|)\ )$ to pick up the next job at time t and thus, is computationally efficient. It is important to note that *GPSU algorithm* can also be used for Speed up with no constraints scenario, with only one minor change that instead of using P_{urgent}, use earlier defined P priority function defined

in eq. 4.1 in the *GPSU* algorithm. The *NUBSU* algorithm had high time complexity and thus, *GPSU* algorithm can be used to mimic its nature by setting the probability parameter p.

Value of p to be used in GPSU algorithm depends on the required preference/dominance of urgent over non-urgent jobs. If jobs requesting for speed up are extremely critical and urgent, then p should be 1 or very close to it. However, if jobs requesting for speed up are not much critical and high slow down for any of the job might have severe consequences then in that case value of p can be set somewhere between 0.6 - 0.9. Value of probability parameter is thus, dependent on the requirement of the environment based on the amount of slow down that can be tolerated and the amount of speed up which is intended/desired by the system.

Chapter 5

Experiments and Results

We used a discrete event based simulation tool for the purpose of comparison analysis and evaluation of our speed up algorithms. We assumed that jobs are arriving in Poisson distribution which is a very good approximation for arrival jobs in real systems. The total number of jobs we considered for the experiments were 10 thousand. The service time of jobs were assumed to be in exponential distribution. We conducted experiments for three different values of system load ρ (λ/μ) equal to 1.1, 1.3 and 1.5. We labeled some of the arriving jobs among total jobs as $urgent$ and conducted experiments by varying proportion of urgent jobs (jobs requesting for speed up) from 0% to 100%. Value of μ used is 0.04 and λ varies depending on the value of ρ.

5.1 Speed Up with no constraints

5.1.1 Percentage of urgent jobs that achieved speed up

The Table 5.1, 5.2 and 5.3 gives the percentage of urgent jobs (jobs requesting for speed up) that achieved speed up for different speed up algorithms based on the proportion of urgent jobs for system load 1.1, 1.3 and 1.5 respectively. UDSU algorithm is successful in achieving near about similar (sometimes even higher) amount of speed up as compared to existing speed up algorithms MPF and MinPF. The UDSU algorithm outperforms MPF-SD in terms of achieved speed up for majority of urgent job proportion. It is interesting to note that the performance of NUBSU algorithm is lower than UDSU algorithm and other existing speed up algorithms in terms of achieved speed up validating our idea that the acceleration of opportunistically forwardable job does not always ensure that every other existing urgent job present in the queue will achieve speed up and thus, such acceleration may even slow down the urgent jobs. But still, NUBSU algorithm speeds up large number of urgent jobs. The percentage of speeded up urgent jobs is lowest for FSU algorithm since it is fair to all and is not biased towards urgent jobs. When all the jobs present in the queue are requesting for speed up (100% case), then all the three speed up (UDSU, NUBSU and FSU) algorithm behaves as same. As the percentage of urgent jobs requesting for speed up increases, achieved speed up decreases.

Table 5.1 $\rho = 1.1$: Percentage of urgent jobs that achieved speed up.

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	-	-	-
10%	**97%**	94%	**97%**	**97%**	94%	84%
20%	**95%**	91.5%	**95%**	**95.5%**	91.5%	84%
30%	**94.67%**	93.3%	**94.33%**	**95%**	90%	84.33%
40%	93.5%	93.5%	**94.25%**	**94.5%**	89.75%	85.25%
50%	94.2%	94.6%	**95.2%**	**95%**	90.6%	86.6%
60%	93.67%	92.83%	**93.83%**	**94.5%**	90.5%	86.33%
70%	92.71%	92%	**93.86%**	**94%**	89.75%	87%
80%	91.62%	90.62%	**93%**	**93%**	90.88%	87.25%
90%	89.78%	89.67%	**90.89%**	**90.56%**	90.56%	87.56%
100%	-	-	-	87.3%	87.3%	87.3%

Table 5.2 $\rho = 1.3$: Percentage of urgent jobs that achieved speed up.

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	-	-	-
10%	**98%**	96%	**98%**	**98%**	94%	89%
20%	**98.5%**	97.5%	**98.5%**	**98.5%**	94%	87%
30%	**98.67%**	98%	**98.67%**	**98.33%**	92.33%	87.67%
40%	**98.75%**	96.75%	**98.75%**	**98.5%**	91.75%	87.75%
50%	97.2%	96.4%	**97.6%**	**97.6%**	93%	87.8%
60%	97.67%	96.83%	**97.83%**	**97.87%**	92.67%	88.33%
70%	97%	96.29%	**97.57%**	**97.14%**	93.14%	88.29%
80%	**97.5%**	95.75%	**97.5%**	**96%**	92.88%	88%
90%	**96.56%**	95.89%	**96.78%**	92.56%	91.22%	88.11%
100%	-	-	-	87.6%	87.6%	87.6%

Table 5.3 $\rho = 1.5$: Percentage of urgent jobs that achieved speed up.

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	-	-	-
10%	**99%**	97%	**99%**	**98%**	94%	81%
20%	**97.5%**	97%	**97.5%**	**98.5%**	94%	83%
30%	**97.67%**	97.33%	**97.67%**	**98.33%**	92.67%	84.67%
40%	**98%**	97.75%	**99%**	**98.5%**	94%	85.75%
50%	**99.2%**	98.2%	**99.2%**	**98.8%**	94.8%	86.8%
60%	**99%**	98%	**98.67%**	**98.87%**	95%	86.5%
70%	**99.14%**	98.43%	**98.86%**	**98.14%**	96%	86.86%
80%	**97%**	97%	**97.25%**	96%	95.12%	87.12%
90%	**96.44%**	96.56%	**96.44%**	92.11%	91.33%	86.89%
100%	-	-	-	86.4%	86.4%	86.4%

Table 5.4 $\rho = 1.1$: Percentage of slowed down non-urgent jobs.

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**8.8%**	**8.8%**	**8.8%**
10%	65.78%	**8.56%**	89.11%	**10.89%**	**9.56%**	**8.44%**
20%	67%	15.25%	94.62%	**13.38%**	**10.25%**	**8.12%**
30%	71.86%	24.57%	95.67%	**16.29%**	**13.14%**	**7.71%**
40%	68.67%	31%	95.83%	**22%**	**16.5%**	**7.83%**
50%	75%	39.8%	96.2%	**27%**	**22.8%**	**8.4%**
60%	80%	51.5%	96.75%	**35.25%**	**35%**	**8%**
70%	82%	49%	96%	**46%**	**41.67%**	**8%**
80%	84.5%	60%	96.5%	**60%**	**59%**	**9%**
90%	88%	82%	97.4%	86%	90%	**12%**
100%	-	-	-	-	-	-

Table 5.5 $\rho = 1.3$: Percentage of slowed down non-urgent jobs.

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**10.4%**	**10.4%**	**10.4%**
10%	41.11%	**9%**	90.75%	**11.22%**	**11.25%**	**10.56%**
20%	65.88%	17.25%	94.62%	**14%**	**12.2%**	**10%**
30%	71.14%	23.71%	96.71%	**17.29%**	**13.57%**	**10%**
40%	80.17%	30.33%	97.33%	**22.83%**	**17.33%**	**10.17%**
50%	89.8%	35.6%	96.8%	**32.6%**	**22.8%**	**9.8%**
60%	76%	45.25%	96.75%	**44%**	**31.75%**	**10.75%**
70%	83%	58%	97.33%	**55%**	**54%**	**11%**
80%	88%	65%	98.5%	81%	**65%**	**11.5%**
90%	92.5%	79%	99%	90%	**80%**	**15%**
100%	-	-	-	-	-	-

Table 5.6 $\rho = 1.5$: Percentage of slowed down non-urgent jobs.

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**13.1%**	**13.1%**	**13.1%**
10%	68.78%	**8.67%**	94.67%	**15.44%**	**13.56%**	**12.44%**
20%	66.12%	17.12%	98.38%	**16.25%**	**14.62%**	**12.12%**
30%	63.43%	24.43%	98.71%	**22.43%**	**16.43%**	**12.29%**
40%	84.5%	33.83%	99%	**31.33%**	**21.83%**	**12.67%**
50%	76.8%	40.6%	98.8%	**39%**	**25.4%**	**13.4%**
60%	81%	54.25%	99%	**52.25%**	**39.25%**	**13.25%**
70%	87.33%	60.33%	98.67%	66.67%	63%	**14%**
80%	91%	73.5%	99%	85.5%	82%	**16%**
90%	95%	86%	99%	92%	90%	**17%**
100%	-	-	-	-	-	-

Table 5.7 $\rho = 1.1$: Mean Wait Time (in time units).

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**473**	**473**	**473**
10%	1742	1575	1713	**495**	**535**	**473**
20%	1738	1459	1711	**536**	**602**	**473**
30%	1715	1372	1694	**558**	**668**	**473**
40%	1709	1288	1692	**584**	**759**	**473**
50%	1747	1164	1675	**633**	**779**	**473**
60%	1705	1137	1669	**738**	**848**	**473**
70%	1672	1069	1625	**823**	**876**	**473**
80%	1670	1073	1591	**925**	**897**	**473**
90%	1562	1259	1548	**1111**	**1111**	**473**
100%	-	-	-	**473**	**473**	**473**

5.1.2 Percentage of slowed down non urgent jobs

Table 5.4, 5.5 and 5.6 shows that our implicit speed up algorithms (UDSU, NUBSU and FSU) performs better than MPF, MPF-SD and MinPF algorithms in terms of slow down caused to the other non-urgent jobs present in the queue. The reason is that MPF, MPF-SD and MinPF algorithms do not consider about the slow down of other non-urgent jobs and greedily aim at achieving high speed up for urgent jobs using location table and swapping of jobs. However, in our speed up algorithms, non-urgent jobs are served using the priority function which takes into consideration the arrival time (to avoid high wait time) as well as duration time (to improve the mean wait time) which avoids high slow down for non urgent jobs. Note that NUBSU algorithm performs better than UDSU algorithm in terms of slow down since NUBSU algorithm whenever it finds some non-urgent job to be opportunistically forwardable, it accelerates it to the server resulting in less percentage of slow downs for non-urgent jobs as compared to UDSU algorithm. It is also observed that as the proportion of urgent jobs increases, the slow down caused to non-urgent jobs also increases. FSU algorithm provides least amount of slow down to non urgent jobs since FSU is fair to all and is not biased towards any specific job.

5.1.3 Mean Wait Time

Table 5.7, 5.8 and 5.9 shows the overall mean wait time for all the jobs present in the queue for different values of system load. The mean wait time for FSU algorithm is lowest owing to its unbiased nature and it remains unaffected with respect to the proportion of urgent jobs since the algorithm is independent of it. Our speed up algorithms outperforms existing position based speed up algorithms (MPF, MinPF and MPF-SD) in terms of overall mean wait time owing to the design of the priority function which prefers job with shorter duration time (resulting in low mean wait time).

Table 5.8 $\rho = 1.3$: Mean Wait Time (in time units).

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**724**	**724**	**724**
10%	2332	2167	2364	**741**	**899**	**724**
20%	2445	2017	2398	**793**	**914**	**724**
30%	2414	1843	2454	**875**	**979**	**724**
40%	2565	1798	2504	**979**	**1060**	**724**
50%	2577	1726	2469	**1147**	**1190**	**724**
60%	2504	1657	2408	**1247**	**1280**	**724**
70%	2542	1644	2440	**1365**	**1404**	**724**
80%	2583	1830	2541	**1816**	**1326**	**724**
90%	2260	1936	2345	**1349**	**1237**	**724**
100%	-	-	-	**724**	**724**	**724**

Table 5.9 $\rho = 1.5$: Mean Wait Time (in time units).

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**944**	**944**	**944**
10%	2944	2675	2947	**1005**	**1039**	**944**
20%	2939	2480	2939	**1071**	**1140**	**944**
30%	2882	2305	2902	**1123**	**1228**	**944**
40%	2959	2229	2894	**1228**	**1311**	**944**
50%	2914	2007	2903	**1337**	**1400**	**944**
60%	2931	2059	2938	**1583**	**1491**	**944**
70%	2889	2052	2856	**1820**	**1665**	**944**
80%	2852	2206	2848	**2142**	**1850**	**944**
90%	2810	2471	2782	**1542**	**1515**	**944**
100%	-	-	-	**944**	**944**	**944**

Table 5.10 $\rho = 1.1$: 95 percentile metric/maximum for wait time (in time units).

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**3269 / 14002**	**3269 / 14002**	**3269 / 14002**
10%	4018 / 7817	3857 / 10160	3911 / 4125	**3421 / 14218**	**3449 / 14008**	**3269 / 14002**
20%	5085 / 11657	3985 / 13053	4136 / 4263	**3661 / 14533**	**3307 / 14394**	**3269 / 14002**
30%	6119 / 15324	4505 / 12132	4491 / 4644	**3875 / 14825**	**3254 / 14498**	**3269 / 14002**
40%	6770 / 15854	4929 / 14302	5423 / 6011	**3658 / 16840**	**3503 / 14829**	**3269 / 14002**
50%	8257 / 17293	5303 / 17525	6049 / 6959	**3822 / 17860**	**3660 / 16049**	**3269 / 14002**
60%	8970 / 14967	5495 / 19537	7046 / 7695	**4873 / 16533**	**4002 / 16452**	**3269 / 14002**
70%	9751 / 21812	6020 / 19069	8321 / 9905	**5707 / 17281**	**4106 / 16798**	**3269 / 14002**
80%	11355 / 22886	6526 / 21901	9638 / 12983	**6485 / 17795**	**4750 / 17795**	**3269 / 14002**
90%	9615 / 21751	7308 / 19959	9875 / 20488	10041 / 20491	9872 / 20491	**3269 / 14002**
100%	-	-	-	*3269 / 14002*	**3269 / 14002**	**3269 / 14002**

Table 5.11 $\rho = 1.3$: 95 percentile metric/maximum for wait time (in time units).

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**5103 / 14623**	**5103 / 14623**	**5103 / 14623**
10%	5587 / 16837	5450 / 12007	5304 / 5712	**5202 / 14623**	**5187 / 14984**	**5103 / 14623**
20%	6963 / 15066	5589 / 13880	5481 / 5712	**5460 / 15950**	**5264 / 14984**	**5103 / 14623**
30%	8602 / 14592	6460 / 16932	5990 / 6351	6593 / 16796	**5406 / 15449**	**5103 / 14623**
40%	9467 / 19757	7374 / 18521	6871 / 7246	7633 / 17487	**5833 / 17142**	**5103 / 14623**
50%	10811 / 19072	7835 / 18594	7837 / 8571	**7547 / 18131**	**6058 / 18131**	**5103 / 14623**
60%	12590 / 22650	9145 / 19750	9383 / 10774	**8086 / 18470**	**6072 / 18470**	**5103 / 14623**
70%	14317 / 20823	9935 / 20649	11758 / 13640	**9912 / 19389**	**9075 / 19389**	**5103 / 14623**
80%	16713 / 21987	10606 / 21999	15602 / 18375	13754 / 20693	**9752 / 20693**	**5103 / 14623**
90%	11681 / 22804	7991 / 22560	11625 / 20883	11040 / 21771	9825 / 21771	**5103 / 14623**
100%	-	-	-	*5103 / 14623*	**5103 / 14623**	**5103 / 14623**

Table 5.12 $\rho = 1.5$: 95 percentile metric/maximum for wait time (in time units).

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**6155 / 15183**	**6155 / 15183**	**6155 / 15183**
10%	6538 / 13068	6406 / 14129	6253 / 6680	**6213 / 15480**	**6227 / 15480**	**6155 / 15183**
20%	8185 / 14456	6604 / 16961	6497 / 6707	6860 / 16201	**6497 / 16201**	**6155 / 15183**
30%	9538 / 18402	7419 / 15682	6601 / 6731	7148 / 16633	**6523 / 16633**	**6155 / 15183**
40%	10704 / 18246	8334 / 18321	7519 / 7912	**7495 / 17204**	**6528 / 16936**	**6155 / 15183**
50%	11295 / 20126	8971 / 19189	8953 / 9993	**8854 / 18121**	**6536 / 18121**	**6155 / 15183**
60%	13664 / 22158	10153 / 21665	10385 / 11603	**9904 / 17268**	**7709 / 17268**	**6155 / 15183**
70%	15472 / 24620	11352 / 22056	12713 / 14194	**11226 / 18715**	**9339 / 17858**	**6155 / 15183**
80%	16009 / 23261	12819 / 23468	15181 / 19209	14459 / 20239	**12263 / 20239**	**6155 / 15183**
90%	13598 / 23948	9766 / 22179	13285 / 21617	11568 / 22142	11509 / 22142	**6155 / 15183**
100%	-	-	-	*6155 / 15183*	**6155 / 15183**	**6155 / 15183**

5.1.4 95 percentile metric for wait time

Table 5.10, 5.11 and 5.12 indicates the 95 percentile metric of wait time for different speed up algorithms. Experimental results show that our implicit speed up algorithms (UDSU, NUBSU and FSU) outperforms the existing speed up algorithms for majority of the urgent job proportion.

5.1.5 DNUJI for NUBSU algorithm

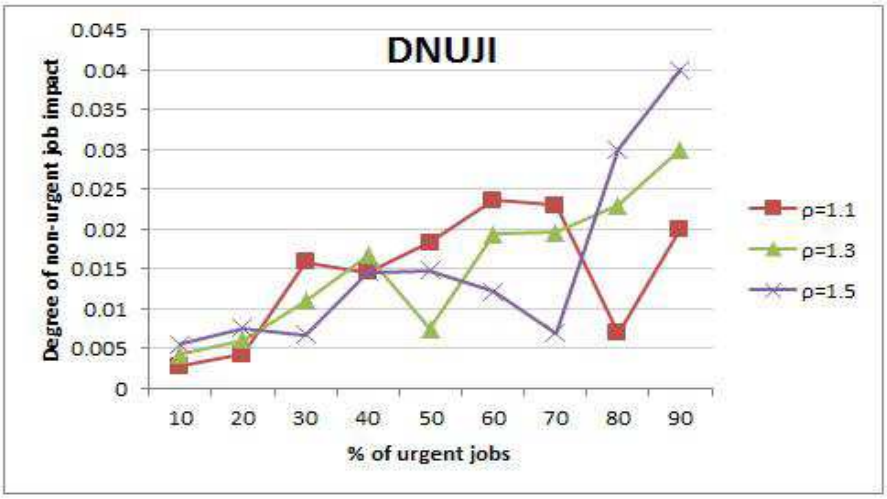

Figure 5.1 Degree of non-urgent job impact (DNUJI) for NUBSU algorithm for different values of system load.

Figure 5.1 shows the plot of DNUJI for NUBSU algorithm based on proportion of urgent jobs for different values of system load ρ. The non-zero value of DNUJI validates our idea that acceleration of opportunistically forwardable job do not always ensure that every other urgent job would achieve speed up. But, experimentally it still happens even less than 5 times out of 100. It is experimentally observed that when very large number of jobs are requesting for speed up ($>= 80\%$), then higher the system load, higher is the value of DNUJI.

5.2 Fine Grained Selective Speed Up

In order to consider Fine Grained Selective Speed Up scenario, among all the urgent jobs (requesting for speed up) we assigned each of them random percentage value $pj \in (0, 100]$ corresponding to which their RSU will be calculated. Note that existing MPF, MPF-SD and MinPF algorithms are not applicable for cases like *Fine Grained Selective Speed Up* where each job j requests for some specific $p_j\%$ of max speed up that it can achieve. Therefore, we present here the results for our GPSU algorithm for different values of probability parameter p.

Table 5.13 $\rho = 1.1$: Percentage of successfully speeded up urgent jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	-	-	-	-
10%	**94%**	**94%**	94%	92%
20%	**92%**	**91.5%**	91%	89%
30%	**92%**	**91.33%**	91%	90%
40%	**91.75%**	**90.5%**	89.25%	88.25%
50%	**92.6%**	**91.8%**	90.8%	82.8%
60%	**90.17%**	**87.83%**	85.17%	76%
70%	**87.57%**	**85.57%**	82.43%	60.43%
80%	**81.5%**	**79.12%**	36.75%	2.2%
90%	**57.11%**	**5%**	3.3%	2.11%
100%	**2.2%**	**2.2%**	2.2%	2.2%

Table 5.14 $\rho = 1.3$: Percentage of successfully speeded up urgent jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	-	-	-	-
10%	**95%**	**95%**	94%	93%
20%	**96%**	**94.5%**	94.2%	92.7%
30%	**95.33%**	**94.67%**	94.33%	91%
40%	**94.5%**	**94.25%**	93%	89.25%
50%	**93.4%**	**93.2%**	89.8%	86.2%
60%	**92.17%**	**91.5%**	87.17%	85.67%
70%	**91.71%**	**90.14%**	79.14%	51.57%
80%	**91%**	**89%**	10.5%	2.75%
90%	**39.22%**	**9.78%**	1.89%	1.44%
100%	**1%**	**1%**	1%	1%

Table 5.15 $\rho = 1.5$: Percentage of successfully speeded up urgent jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	-	-	-	-
10%	**96%**	**95%**	94.5%	93%
20%	**93%**	**92.8%**	92.5%	92.14%
30%	**93.67%**	**93%**	92.67%	91%
40%	**94.5%**	**94.5%**	93.5%	94.5%
50%	**94.8%**	**93.6%**	93.6%	85.8%
60%	**91.33%**	**91.33%**	90.33%	72.5%
70%	**90.71%**	**85.57%**	66.43%	2.12%
80%	**71.38%**	**64.62%**	6.62%	0.71%
90%	**12.56%**	**5.67%**	2.16%	0.42%
100%	**0.1%**	**0.1%**	0.1%	0.1%

Table 5.16 $\rho = 1.1$: Percentage of slowed down non-urgent jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	8.8%	8.8%	**8.8%**	**8.8%**
10%	10.89%	10.89%	**10.78%**	**10.78%**
20%	13.38%	13.38%	**13.25%**	**13%**
30%	16.29%	15.86%	**15.43%**	**15.43%**
40%	22%	21.33%	**20.17%**	**19.67%**
50%	27%	25.8%	**24.4%**	**23.4%**
60%	35.25%	35%	**32.25%**	**28%**
70%	46%	41.33%	**37.33%**	**28.67%**
80%	64%	55%	**32%**	**5.5%**
90%	90%	46%	**9%**	**2%**
100%	-	-	-	-

Table 5.17 $\rho = 1.3$: Percentage of slowed down non-urgent jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	10.4%	10.4%	**10.4%**	**10.4%**
10%	11.22%	11.22%	**11.1%**	**11.1%**
20%	14%	15.62%	**13.75%**	**12.38%**
30%	17.29%	17.29%	**17.29%**	**16.14%**
40%	22.83%	23%	**22.33%**	**25.33%**
50%	32.6%	32.2%	**30%**	**29.4%**
60%	44%	40.5%	**38.5%**	**37.25%**
70%	55%	49.33%	**48.2%**	**39.33%**
80%	81%	60.5%	**27.5%**	**5%**
90%	90%	55%	**14%**	**12%**
100%	-	-	-	-

5.2.1 Percentage of successfully speeded up urgent jobs

Table 5.13, 5.14 and 5.15 shows the percentage of successfully speeded up urgent jobs (jobs for which achieved speed up $>=$ Requested speed up) for GPSU algorithm for different values of probability parameter p. For $p = 1$, maximum successful speed ups were obtained. The % of successful speed ups decreases as probability parameter p decreases, since the tendency to favor urgent jobs decreases with decrease in p. It is also experimentally observed, that as the proportion of urgent jobs requesting for speed up increases, % of successful speed ups decreases.

5.2.2 Percentage of slowed down non-urgent jobs

Table 5.16, 5.17 and 5.18 shows the percentage of slowed down non-urgent jobs for GPSU Algorithm for different values of probability parameter p. Higher the probability parameter, higher is the % of slowed down non-urgent jobs since as p increases, tendency to serve non-urgent jobs decreases which

Table 5.18 $\rho = 1.5$: Percentage of slowed down non-urgent jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	13.1%	13.1%	**13.1%**	**13.1%**
10%	15.44%	15.33%	**15.33%**	**15.22%**
20%	18.25%	18%	**17.25%**	**17%**
30%	22.43%	22.14%	**21.71%**	**21.71%**
40%	31.33%	30.33%	**29.83%**	**29.67%**
50%	39%	38.4%	**38.2%**	**36.4%**
60%	52.25%	50.5%	**47.25%**	**40.75%**
70%	66.67%	62%	**53.67%**	**24.67%**
80%	85.5%	71.5%	**33.5%**	**7.5%**
90%	92%	31%	**5%**	**2%**
100%	-	-	-	-

Table 5.19 $\rho = 1.1$: Overall mean wait time for all the jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	473	473	473	473
10%	495	495	495	495
20%	537	536	530	537
30%	560	559	560	559
40%	589	588	591	587
50%	645	642	644	644
60%	760	749	752	760
70%	868	850	847	947
80%	1039	984	1129	1711
90%	1532	1441	1728	1751
100%	1717	1717	1717	1717

leads to their slow down. Thus, the percentage of slowed down jobs which are not requesting for speed up is least for $p = 0.7$.

5.2.3 Mean Wait Time and 95 percentile metric for wait time

Table 5.19, 5.20 and 5.21 shows the overall mean wait time (in time units) for all the jobs for GPSU Algorithm. 95 percentile metric for wait time for GPSU Algorithm for different values of system load is shown in *Table* 5.22, 5.23 and 5.24.

5.3 Experimentation on process logs dataset

Apart from conducting experiments on M/M/1 queueing system, we also conducted experiments on real process logs obtained using *Process Accounting* utility in linux. For the purpose of gathering logs,

Table 5.20 $\rho = 1.3$: Overall mean wait time for all the jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	724	724	724	724
10%	742	742	742	742
20%	793	805	805	805
30%	878	877	876	893
40%	985	984	981	1012
50%	1163	1157	1193	1179
60%	1286	1264	1292	1310
70%	1472	1413	1537	1616
80%	2158	1780	1937	2400
90%	2223	2059	2449	2444
100%	2440	2440	2440	2440

Table 5.21 $\rho = 1.5$: Overall mean wait time for all the jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	944	944	944	944
10%	1005	1005	1004	1005
20%	1072	1073	1072	1072
30%	1126	1125	1123	1123
40%	1233	1229	1230	1225
50%	1352	1346	1342	1364
60%	1622	1591	1570	1591
70%	1919	1852	1895	2207
80%	2573	2191	2325	2930
90%	2661	2787	2981	3019
100%	2955	2955	2955	2955

Table 5.22 $\rho = 1.1$: 95 percentile metric/maximum for wait time for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	3269 / 14002	3269 / 14002	3269 / 14002	3269 / 14002
10%	3421 / 14218	3421 / 14218	3421 / 14218	3421 / 14218
20%	3661 / 14533	3661 / 14533	3661 / 14533	3661 / 14533
30%	3875 / 14825	3875 / 14825	3875 / 14825	3875 / 14825
40%	3658 / 16840	3658 / 16840	3658 / 16840	3658 / 16835
50%	3822 / 17860	3822 / 17860	3753 / 17860	3822 / 17860
60%	4873 / 16533	4779 / 16533	4660 / 16533	4675 / 16533
70%	5707 / 17281	5707 / 17281	5406 / 17281	3805 / 16798
80%	6485 / 17795	6472 / 17795	2852 / 17795	4723 / 5451
90%	5269 / 20491	3567 / 16954	4522 / 5380	4522 / 5380
100%	4368 / 5318	4368 / 5318	4368 / 5318	4368 / 5318

Table 5.23 $\rho = 1.3$: 95 percentile metric/maximum for wait time for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	5103 / 14623	5103 / 14623	5103 / 14623	5103 / 14623
10%	5202 / 14623	5202 / 14623	5202 / 14623	5202 / 14623
20%	5860 / 15950	5860 / 15950	5860 / 15950	5860 / 15950
30%	6593 / 16796	6593 / 16796	6593 / 16796	6451 / 16796
40%	7633 / 17487	7633 / 17487	7633 / 17487	7588 / 17487
50%	7547 / 18131	7547 / 18131	7547 / 18131	7507 / 18131
60%	8086 / 18470	8086 / 18470	8351 / 18470	7494 / 18470
70%	9952 / 19389	9861 / 19389	9685 / 19233	6069 / 17935
80%	13754 / 20693	10651 / 20693	6133 / 16352	6394 / 7476
90%	10764 / 21771	5752 / 19814	6233 / 7138	6233 / 7138
100%	6077 / 6832	6077 / 6832	6077 / 6832	6077 / 6832

Table 5.24 $\rho = 1.5$: 95 percentile metric/maximum for wait time for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	6155 / 15183	6155 / 15183	6155 / 15183	6155 / 15183
10%	6413 / 15480	6413 / 15480	6413 / 15480	6413 / 15480
20%	6860 / 16201	6860 / 16201	6860 / 16201	6860 / 16201
30%	7148 / 16633	7148 / 16633	7148 / 16633	7148 / 16633
40%	7695 / 17204	7695 / 17204	7695 / 17204	7695 / 17204
50%	8854 / 18121	8854 / 18121	8854 / 18121	8854 / 18121
60%	9904 / 17268	9904 / 17268	9904 / 17268	9387 / 17268
70%	11226 / 18715	11226 / 18715	11028 / 17858	6965 / 16179
80%	14459 / 20239	11145 / 20239	6728 / 16158	7277 / 8304
90%	11521 / 22142	6579 / 16823	7100 / 8072	7100 / 8072
100%	7043 / 7969	7043 / 7969	7043 / 7969	7043 / 7969

Table 5.25 Sample process log records obtained using pacct utility in linux.

command	version	utime	systime	etime	uid	gid	mem	char	pid	ppid	finish time
cc1plus	v3	17.00	1.00	19.00	1000	1000	31632.00	0.00	27379	27378	Sun Dec 15 16:47:50 2013
a.out	v3	16.00	0.00	16.00	1000	1000	7988.00	0.00	27389	27324	Sun Dec 15 16:47:51 2013
ld	v3	5.00	0.00	6.00	1000	1000	10512.00	0.00	27554	27553	Sun Dec 15 16:51:02 2013

Table 5.26 Process logs dataset: Percentage of urgent jobs that achieved speed up for Speed up with no constrains scenario.

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	-	-	-
10%	**98.99%**	97.98%	**99.8%**	**98.99%**	97.98%	82.83%
20%	**98.99%**	98.48%	**99.49%**	**98.99%**	95.96%	79.29%
30%	98.32%	**98.99%**	**99.33%**	**99.2%**	94.28%	79.12%
40%	99.14%	**99.75%**	**99.49%**	**99.24%**	95.71%	85.75%
50%	98.79%	**99.39%**	98.79%	**99.39%**	96.16%	80.61%
60%	98.65%	**99.16%**	98.65%	**99.1%**	97.47%	80.64%
70%	98.56%	**98.99%**	98.56%	**98.67%**	96.97%	80.95%
80%	**98.48%**	**99.12%**	**98.48%**	96.09%	95.96%	81.44%
90%	**98.32%**	**98.32%**	**98.2%**	91.69%	90.69%	81.14%
100%	-	-	-	81.33%	81.33%	81.33%

we executed different types of user programs (dynamic programming, recursion, modular exponentiation, binary search , adhoc etc.) randomnly in poisson fashion using shell script on randomnly generated datasets at run time. The RAM of the system is 3GB and processor used is 2.4 GHz. Using pacct utility in linux, we collected process logs for about 10K executed processes which involves both user as well as kernel processes. The minimum and maximum cpu burst time obtained in the logs were 1 and 185 units respectively. Some of the sample log records are shown in table 5.25, where command - command name, version - version of acct file, utime - user time, systime - system time, etime - elapsed time, uid - user id, gid - group id, mem - memory usage, char - number of characters transferred on input/output, pid - process id, ppid - parent pid, finish time - finish time of process. The nature of results obtained are similar as compared to M/M/1 system and are presented here.

5.4 Discussion

The authors in [9] and [10] used only achieved ratio metric (percentage of urgent jobs that achieve speed up) for evaluating the effectiveness of their speed up algorithms. In this research, we along with achieved ratio metric also consider the amount of slow down for rest of the jobs as a result of speed up and overall mean wait time. The experimental results show that our proposed speed up algorithm UDSU is able to achieve near about similar amount of speed up as compared to the existing speed up algorithms MPF, MPF-SD and MinPF. All the three speed up algorithms UDSU, NUBSU and FSU outperforms the

Table 5.27 Process logs dataset: Percentage of slowed down non-urgent jobs for speed up with no constraints scenario.

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**18.16%**	**18.16%**	**18.16%**
10%	84.75%	**9.08%**	97.98%	20.63%	20.29%	**18.27%**
20%	81.2%	**17.78%**	98.87%	22.45%	20.3%	**17.65%**
30%	79.25%	25.94%	99.33%	**25.5%**	**22.48%**	**17.15%**
40%	68.57%	34.12%	99.49%	**33.61%**	**23.87%**	**12.67%**
50%	71.77%	45.16%	99.6%	**40.52%**	**29.64%**	**17.14%**
60%	70.78%	54.66%	98.79%	**52.53%**	**51.21%**	**17.38%**
70%	93.96%	79.8%	99.7%	**78.52%**	**72.48%**	**17.45%**
80%	80.4%	86.88%	99.73%	86.93	**83.92%**	**18.59%**
90%	93%	87%	99.8%	93%	93%	**17%**
100%	-	-	-	-	-	-

Table 5.28 Process logs dataset: Mean Wait Time for Speed Up with no constraints scenario (in time units).

Urgent jobs	MPF	MPF-SD	MinPF	UDSU	NUBSU	FSU
0%	-	-	-	**1122**	**1122**	**1122**
10%	3507	3154	3489	**1185**	**1170**	**1122**
20%	3337	2765	3528	**1334**	**1291**	**1122**
30%	3573	2752	3606	**1516**	**1425**	**1122**
40%	3484	2556	3492	**1599**	**1529**	**1122**
50%	3420	2518	3485	**1735**	**1631**	**1122**
60%	3484	2672	3573	**2484**	**2235**	**1122**
70%	3438	2716	3466	**2416**	**2315**	**1122**
80%	3215	2870	3261	**2186**	**2010**	**1122**
90%	3319	3100	3335	**1763**	**1763**	**1122**
100%	-	-	-	**1122**	**1122**	**1122**

Table 5.29 Process logs dataset: Percentage of successfully speeded up urgent jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	-	-	-	-
10%	**97.98%**	**97.98%**	96.97%	96.97%
20%	**97.98%**	**97.91%**	96.97%	96.46%
30%	**97.93%**	**97.64%**	97.63%	96.64%
40%	**97.22%**	**97.22%**	96.97%	95.96%
50%	**95.96%**	**93.74%**	93.15%	92.73%
60%	**93.43%**	**93.77%**	93.1%	92.42%
70%	**92.06%**	**91.92%**	90.62%	1.01%
80%	**56.68%**	**16.54%**	0.51%	0.38%
90%	**1.97%**	**1.01%**	0.45%	0.34%
100%	**0.3%**	**0.3%**	0.3%	0.3%

Table 5.30 Process logs dataset: Percentage of slowed down non-urgent jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	18.16%	18.16%	**18.16%**	**18.16%**
10%	20.63%	20.63%	**20.21%**	**20.21%**
20%	22.45%	22.45%	**22.32%**	**22.32%**
30%	25.5%	25.4%	**25.36%**	**25.21%**
40%	33.61%	33.6%	**33.51%**	**33.45%**
50%	40.52%	41.13%	**40.32%**	**40.12%**
60%	70.53%	64.48%	**48.61%**	**43.83%**
70%	78.52%	72.82%	**48.66%**	**28.52%**
80%	86.93%	51.79%	**17.09%**	**1.01%**
90%	93%	43%	**2%**	**0.2%**
100%	-	-	-	-

Table 5.31 Process logs dataset: Overall mean wait time for all the jobs for GPSU Algorithm.

Urgent jobs	p=1	p=0.9	p=0.8	p=0.7
0%	1122	1122	1122	1122
10%	1186	1186	1185	1186
20%	1335	1335	1334	1334
30%	1518	1517	1517	1518
40%	1603	1596	1595	1587
50%	1750	1730	1720	1735
60%	2571	2340	2138	1980
70%	2806	2596	2377	2453
80%	3003	2596	2847	3237
90%	3196	2924	3254	3293
100%	3297	3297	3297	3297

existing speed up algorithms in terms of slow down caused to the other non-urgent jobs. It is because existing speed up algorithms do not provide any remedies for delayed jobs and greedily aim at achieving high speed up for urgent jobs. However, our implicit speed up priority function takes into consideration the arrival time (while speeding up urgent jobs) which avoids high slow down for non-urgent jobs. The overall mean wait time is improved dramatically by our speed up algorithms owing to the design of priority function as compared to the existing speed up algorithms which are purely positional. Further, our algorithms are implicit in nature where the notion of acceleration is incorporated in the priority function which leads to a time efficient solution unlike existing algorithms where there is an overhead of swapping of jobs and *location table*. We further presented the GPSU algorithm for *Fine Grained Selective Speed Up* scenario which has not been yet addressed and presented the results for the same.

We used our experimental methodology to be simulation. The experiments performed involved changing various parameters (system load, jobs requesting for speed up, RSU etc.) and the results shown are the statistical averages of multiple runs. We conducted our experiments assuming a M/M/1 queuing system which is widely used queueing system for the analysis of real life situations and is a good approximation for large number of queueing systems [7]. The poisson arrival distribution of jobs assumed in experiments is a very good approximation for job arrival in real systems. Regarding exponential distribution there is an argument from information theory which states that exponential distribution provides the least information or highest entropy, and is therefore a reasonable assumption for service time when no other data is available [7]. Thus, our assumption of M/M/1 system for experimental analysis is valid to a greater extent and our results are significant. We also achieved similar nature of results using real process logs dataset. Further, we also have considered real trace data for experimental analysis presented in later chapters of this thesis.

Chapter 6

Applications: Web Scheduling

In this chapter, we apply the idea of implicit *speed up* for the purpose of scheduling of web requests arriving at web server. We show how our proposed priority function (defined in eq.4.1) for implicit speed up with little modification, can be applied to web scheduling. We show the effectiveness of our algorithms using simulation model on a trace driven workload.

6.1 Introduction

Current demand on busy web servers requires them to serve up to thousands of clients simultaneously. In such a case, the response time suffered by the client is one of the most important factor that determines the web server performance, where the response time is defined as the time duration between the time client makes the request until the time the client receives the last byte of the file requested. The servers cannot afford large delay in response time for users because it might result in rejection of request either because of server timeout or due to user abort. The slow response times and difficult navigation are the most common complaints of Internet Users [12]. The scheduling strategies used by a web server play a very crucial role in determining the response time of users. Now, the question that arises is how to improve the response time of the user but at the same time ensuring no starvation by simply changing the order of servicing of requests arriving at web server.

The traditional scheduling strategy used in web server is Processor-Sharing (PS) scheduling in which each of the n incoming requests gets same amount $1/n$ of the CPU time. PS scheduling is unbiased as it gives equal proportion of time to all.

It is well known from queuing theory that Shortest Remaining Processing Time (SRPT) scheduling is an optimal scheduling algorithm for minimizing the mean response time [14]. SRPT based scheduling policies uses size of requested file to implement SRPT. Various SRPT based scheduling policies [5], [8], [15], [3] and [1] have been proposed for the scheduling of HTTP requests arriving at web server. Still these algorithms are not used in practice. There are several reasons for this. One reason is that SRPT causes starvation for large size requests (because of its tendency to favor small size requests) in the presence of continuous arrival of small sized requests. Also, SRPT relies completely on size of

requested file to determine the mean response time which is not enough because it does not take into consideration the user server interaction parameters present in Internet like throughput of the user connection. Further, SRPT based policies cannot be applied for dynamic HTTP requests where file size is not known in advance.

6.2 Related Work

L. Cherkasova introduced the concept of Alpha-Scheduling to improve the response time for web applications [4]. This strategy lies between FIFO (first-in-first-out) and SRPT. The measure of balance between FIFO and SRPT was decided by the parameter alpha.The major drawback of this strategy is that they have not taken into consideration the user-server interaction parameters over Internet like network bandwidth, which may play a crucial role in influencing the response time of the user.

It is well known from queuing theory that SRPT is an optimal algorithm for minimizing the mean response time when the job size is known in advance [14]. For static requests, the size of the request (time required to service the request) can be well-approximated by the size of the file requested by HTTP request, which is known to the server. The SRPT based scheduling strategies uses this approximation for the scheduling of HTTP requests. The work done to implement SRPT scheduling for web servers has been done on both the application level and the kernel level. M. Crovella and R. Frangioso [5] implemented SRPT connection based scheduling (priority to requests with smaller size) at the application level and got improvement in the response time as compared to traditional PS scheduling but at the cost of drop of throughput by some constant factor. Later, M. Harchol Balter [8] implemented SRPT based scheduling strategy at the kernel level and got better improvements in response time than in [5] and the throughput problem was eliminated.

C. Murta [13] introduced an extension to SRPT known as FCF (Fastest Connection First) scheduling which takes into consideration the network conditions instead of relying only on file size of request. The strategy gives priority to the requests with shorter size issued through faster connections.The information sharing between a Web server and the TCP connections running in the server was taken into consideration. They got improvements in the response time as compared to SRPT, although SRPT provided the shortest server delay. They concluded that scheduling strategies for web servers should take into consideration the network conditions (WAN) and use them to set up priorities for the requests. Ahmad AlSa'deh [1] suggested SRRT (Shortest Remaining Response Time) scheduling for web servers which is also an extension to SRPT. In addition to the file size, the scheduling algorithm also takes into consideration current RTT (Round-Trip Time) and TCP congestion window size for the servicing of HTTP requests. They got an average improvement of about 7.5% over SRPT algorithm. For the evaluation of results, [1] did not take into account the variability of network bandwidth from user to user but instead used same network bandwidth for all the clients throughout the experiments. Further SRRT scheduling strategy was applicable only for static requests.

Most of the papers have considered the idea of SRPT based scheduling policies [5], [15], [13], [4],

[3], [1]. But these scheduling strategies do not prevent the starvation of large size requests in the presence of continuous arrival of short size requests. Further, all SRPT based policies are applicable only for the static HTTP requests where the file size is known in advance. But today in most of the practical scenarios, web servers use dynamic content as well (cgi and other non static requests). So, SRPT based policies cannot be applied in these scenarios.

Bender, Chakravati and Muthukrishnan [2] proved that SRPT will cause large files to have arbitrarily high response time. This paper raised an important point that while choosing a scheduling policy it is important to consider not only the scheduling policy's performance but also whether the policy is fair, i.e. some of the jobs have arbitrarily high response time. There have been some research papers which studies the types of unfairness caused by SRPT. For example, M. Gong and C. Williamson [6] investigates about two different types of unfairness (endogenous and exogenous) caused by Shortest Remaining Processing Time (SRPT) strategy.

In order to overcome the problems related to SRPT based policies, in this work we present simple but effective scheduling policies with dynamic priority adjustment which addresses the problem of starvation and keeping the mean response time low, both simultaneously. Further, the policy will be able to handle the dynamic HTTP requests where the file size is not known in advance.

6.3 Speed Up Scheduling

We present two scheduling algorithms named as *SSU (Static Speed Up)* and *DSU (Dynamic Speed Up)* for static (file size is known in advance) and dynamic environments (file size is not known in advance) respectively with a slight modification in the priority function defined in eq.4.1 based on implicit speed up. The algorithms are simple that assigns priority to the requests based on their specific characteristics. The request with lowest value of priority function is accelerated to the server and thus, is chosen for next service.

6.3.1 Static Speed Up (SSU) scheduling:

This algorithm is non preemptive. Let $AT(r)$, $FS(r)$ and $LB(r)$ respectively denote the arrival time of request r, file size requested by r and link bandwidth of the connection of the user sending request r to the server i.e. throughput of the user connection. The priority for request r is assigned as follows:

$$Priority(r) = \frac{AT(r) \times FS(r)}{LB(r)} \tag{6.1}$$

The request with minimum value of priority function is chosen next for service. If there are multiple such requests, then the one with shortest file size is chosen. The idea is to give priority to small size requests issued through faster connection in order to improve the mean response time but at the same time taking into consideration the arrival time of request in order to address the problem of starvation.

The *SSU* algorithm is applicable only for static requests arriving at web server since it includes the file size of request in its priority function.

6.3.2 Dynamic Speed Up (DSU) scheduling:

In order to schedule dynamic requests arriving at web server, we make use of LAS (Least Attained Service) policy since the request size is not known in prior. LAS is a preemptive scheduling policy which tries to predict the remaining job size by the amount of service, job has received so far and thus, mimics SRPT by giving preference to least attained service job. LAS will create starvation for large size requests since arriving jobs will always have low value of attained service and thus, jobs existing in queue for long period of time will not get time to be serviced. Thus, we also take into consideration the arrival time of jobs to address this problem of starvation.

Let $AT(r)$, $AS(r)$ and $LB(r)$ respectively denote the arrival time of request r, the attained service for r (amount of time it has received service so far) and link bandwidth of the user connection sending request r to the server. Then the priority for request r is assigned as follows:

$$Priority(r) = \frac{AT(r) \times AS(r)}{LB(r)} \tag{6.2}$$

The DSU scheduling algorithm is preemptive. We discuss the impact of the three parameters that we keep in the priority function on web requests :

- **Impact of Arrival Time in Priority:** The term $AT(r)$ in eq. 6.1 and 6.2 ensures that every request will eventually be served. This is because AT (r) keeps on increasing for the incoming requests with respect to time and we choose the request with minimum value of priority to get next service. Thus, the time will always come when the request waiting for too long will have the minimum value of priority function. Hence, the starvation for any request is impossible.

  Consider a request r_1 with a large $FS(r_1)$, arriving at time $AT(r_1)$. Since requests keep coming, and arrival time of requests keep increasing, there exists a request r_2 , such that $\frac{AT(r_2)\times FS(r_2)}{LB(r_2)} > \frac{AT(r_1)\times FS(r_1)}{LB(r_1)}$, implying that r_1 will be served before r_2 . That is, if $AT(r_2) > AT(r_1)$ and $\frac{AT(r_1)\times FS(r_1)}{LB(r_1)} < \frac{AT(r_2)\times FS(r_2)}{LB(r_2)}$, r_1 is served before r_2. Since remaining file sizes are finite and decreasing, and link bandwidth is also constant, each request will definitely be served as priority of new requests keep increasing as their arrival time is increasing.

- **Impact of file size in Priority:** If there are various requests having near about similar arrival time issued through the same link bandwidth connection, then the request with least file size is preferred over others (like SRPT) in order to improve the response time.

  Consider a small size request r_1 with very small value of $FS(r1)$ arrived at time $AT(r_1)$. Let there be another request r_2 already present in request queue such that $AT(r_2) < AT(r_1)$ and

$FS(r_2)$ is much larger than $FS(r_1)$. If $\frac{AT(r_1)\times FS(r_1)}{LB(r_1)} < \frac{AT(r_2)\times FS(r_2)}{LB(r_1)}$, then r_1 will be served before r_2 . It indicates that smaller value of $FS(r_1)$ (resulting in lower priority function value for r_1 than r_2) results in preferring r_1 over r_2 for next service. Each request r will get served because there will be certain arrival time $AT(r\prime)$ (for some request $r\prime$) such that all the requests after that irrespective of their file size (> 1) will have higher value of priority function than r.

- **Impact of Link Bandwidth in Priority:** Link Bandwidth plays a very crucial role in influencing the response time for web clients. Greater the link bandwidth, lesser is the response time. If there are set of requests present in request queue having similar arrival time and near about same file size, then the one with the fastest link bandwidth is preferred over others which will result in lesser response time.

  Consider a request r_1 with arrival time $AT(r_1)$ issued through fast connection of say 100 Mbps. Then another request r_2 arrives with smaller value of $FS(r_2)$ as compared to $FS(r_1)$ issued through a relatively slower connection of say 1 Mbps such that $\frac{AT(r_1)\times FS(r_1)}{LB(r_1)} < \frac{AT(r_2)\times FS(r_2)}{LB(r_2)}$ (because $LB(r_1)$ is much greater than $LB(r_2)$), then r_1 (fast connection request) will be served before r_2 (slow connection request) showing that how link bandwidth can impact the value of priority function. Each request r (even low bandwidth request) will get served irrespective of other requests because there exist an arrival time beyond which their value of priority function is higher than request r assuming some maximum link bandwidth (which is finite).

6.4 Experiments and Results

We used a simple simulation model using trace-driven workload for the evaluation of results. We used a real one day logs of web requests arriving at International Institute Of Information Technology, Hyderabad proxy server collected on 9th September, 2012. Our workload consisted of a part of single day trace consisting of about 1 million requests. Each entry in the trace described a request made to the server and contained information about the request like arrival time of the request, URL requested, size of the request and other status information. The file size in our workload ranged from 56 bytes to 15 MB.

We used a *Scheduler Simulator* in order to simulate all the requests arriving at a web server present in the trace based workload. Each request present in the trace can be considered as a process whose burst time is directly proportional to the file size requested and is inversely proportional to the link bandwidth of the request connection. The RAM of the system is 4GB and processor speed is 3.3GHz. The requests were simulated as per their information in the trace based workload using the Scheduler Simulator.

For the purpose of assigning network bandwidth to each of the request, we partitioned the requests into following three types of classes :

Table 6.1 Overall mean and maximum response time (in sec) for both the scenarios for different scheduling strategies.

Algorithm	**Scenario1**		**Scenario2**	
	Mean	Max	Mean	Max
PS	2.37	4.44	1.03	6.16
SRPT	**1.58**	4.39	**0.5**	6.18
SSU	**1.33**	**4.18**	**0.6**	**5.98**
DSU	2.12	**4.26**	0.98	**6.08**

- *Small* files (0 - 50 KB): These smaller sized requests contributed towards 64% of the total workload.
- *Medium*-sized files(50 KB - 500 KB): These requests contributed towards 32% of the total requests present in our workload.
- *Large* files ($>$ 500 KB): These requests contributed towards 4% of the overall workload.

We performed our experiments for two different types of extreme scenarios.

- In **Scenario 1**, we assigned 1 Mbps, 10 Mbps and 100 Mbps of network bandwidth to *Small*, *Medium-sized* and *Large* files respectively.
- In **Scenario 2**, we assigned 1 Mbps, 10 Mbps and 100 Mbps of network bandwidth to *Large*, *Medium-sized* and *Small* files respectively.

6.4.1 Scenario 1:Results

- *Small* files (0-50 KB) assigned to 1 Mbps connection.
- *Medium-sized* (50 KB-500 KB)files assigned to 10 Mbps connection.
- *Large* ($>$ 500 KB) files assigned to 100 Mbps connection.

Table 6.1 shows that both the SSU and DSU scheduling algorithms outperforms the existing scheduling algorithms in terms of maximum response time validating our idea that these algorithms provide less scope for starvation. The reason behind this decrease, is that our scheduling algorithms take into consideration the arrival time (aging) of request which do not allow any request to stay for a very long period of time. *SSU* scheduling provides the minimum response time in Scenario 1 even better than SRPT scheduling. The reason behind this improvement is that SRPT prefers the shorter size requests independent of their slow network bandwidth (1 Mbps) connection in this scenario. However, the actual response time suffered by the client is influenced by both the size of the requested file and the link bandwidth. Greater the link bandwidth, lower is the response time. *SSU* scheduling takes into consideration

Table 6.2 Scenario 1: Overall mean and maximum response time (in sec) for different types of requests for different scheduling strategies.

Algorithm	**Small**		**Medium**		**Large**	
	Mean	Max	Mean	Max	Mean	Max
PS	2.26	4.41	2.62	4.44	1.38	4.39
SRPT	**0.64**	2.71	3.13	4.22	4.3	**4.39**
SSU	1.23	4.10	1.55	4.18	0.77	**3.03**
DSU	2.04	4.19	2.59	4.26	1.67	**4.13**

both the size of requested file as well as the link bandwidth of the user connection. Note that, such improvements are not observed in DSU algorithm, because it does not know in prior the file size of request. It uses attained service measure for the estimation of file size. However, it outperforms PS scheduling (which also do not require file size) in mean as well as max response time. The major advantage of DSU scheduling is that it can be used for the scheduling of dynamic web requests (file size of request is not known in prior) whereas SRPT and SSU are not applicable in such cases.

Table 6.2 shows the mean and maximum response time for the different types of requests for different scheduling strategies for *Scenario 1*. The small size requests get a very low mean response time for SRPT (because of it's tendency to favor small requests), whereas on the other hand, large size requests (connected through 100 Mbps link bandwidth) suffer from very high mean response time of 4.3 seconds by SRPT. SRPT only takes into consideration the remaining size of requested file thereby making the large size jobs to wait for too long. Such high penalization for large requests is not observed for both SSU and DSU scheduling because along with the file size, they also consider the arrival time in priority to avoid starvation.

From the above results, it can be concluded that in *Scenario 1*, SSU strategy outperforms SRPT as well as PS scheduling in terms of mean response time without creating starvation like SRPT. DSU scheduling also provides less scope of starvation and outperforms PS scheduling with a major advantage of its applicability in serving of dynamic web requests.

6.4.2 Scenario 2:Results

- *Small* files (0-50KB) assigned to 100Mbps connection.
- *Medium-sized* (50-500KB) assigned to 10Mbps connection.
- *Large* files ($>$ 500KB) assigned to 1Mbps connection.

*T*he starvation caused by SRPT for large files (connected through 1 Mbps bandwidth) is visible from the *Table* 6.1 and 6.3 for *Scnenario 2*. SSU and DSU scheduling performs better than SRPT scheduling in terms of maximum response time indicating the reduction in starvation. However, in this scenario the overall mean response time is minimum for SRPT better than SSU as well as DSU scheduling. This is because in this scenario, smaller size requests are connected through faster link

Table 6.3 Scenario 2: Overall mean and maximum response time (in sec) for different types of requests for different scheduling strategies.

Algorithm	**Small**		**Medium**		**Large**	
	Mean	Max	Mean	Max	Mean	Max
PS	0.12	0.61	2.18	3.42	5.65	6.16
SRPT	0.006	0.09	1.05	2.9	4.48	**6.18**
SSU	0.07	0.2	1.17	2.9	4.4	**5.98**
DSU	0.27	0.68	2.12	3.88	5.4	**6.08**

bandwidth connection. Thus, by preferring only the smaller size requests, SRPT is also indirectly giving preference to faster connection requests (because smaller size requests are connected through faster connections). *SSU* scheduling gives preference to both shorter size requests as well as earlier arrived requests. Thus, it's response time is found to be slightly lower than SRPT. In this case, SRPT provides an optimal response time. We can say that this scenario is the best case scenario for SRPT scheduling.

6.5 Discussion

We have considered two extreme case scenarios for the variability in network bandwidth for different users. In first scenario, *smaller* size requests and *larger* size requests are respectively connected through slower and faster link bandwidth connection while in the second scenario vice-versa.

In **Scenario 1**, *SSU* scheduling outperforms both PS and SRPT scheduling. SRPT scheduling blindly gives preference to shorter size requests irrespective of taking into account their slower network bandwidth in this scenario. It leads to the starvation of large size requests which are connected through faster link bandwidth connection. *SSU* scheduling takes into consideration both the size as well as link bandwidth thereby, improving the mean response time from 1.53 to 1.3 sec. *SSU* scheduling reduces the problem of starvation in terms of *maximum* response time as compared to SRPT scheduling from 4.39 to 4.18 sec because it takes into consideration the arrival time of request and thus, prevents the request from waiting for too long. DSU scheduling outperforms PS scheduling in terms of mean response time as well as max response time. DSU scheduling can further be applied for dynamic environments whereas SSU and SRPT cannot.

In **Scenario 2**, *small* size requests are connected through faster connection while the *large* size requests are connected through slower connection. SRPT scheduling while giving preference to *small* size requests is also indirectly giving preference to the requests with faster connection (because of the nature of this scenario). Thus, SRPT provides an optimal response time in this scenario (best-case scenario for SRPT). *SSU* scheduling suffers a drop from 0.5 sec (as in SRPT) to 0.6 sec in terms of mean response time as compared to SRPT scheduling. However, *SSU* scheduling again reduces the problem of starvation from 6.18 to 5.98 sec in terms of *maximum* response time as compared to SRPT. DSU scheduling outperforms PS scheduling in terms of mean as well as maximum response time in this scenario as

Table 6.4 Scenario 1: Percentage of speeded up and slowed down requests for different scheduling strategies.

Strategy	% of achieved speed-up	% of slow down
PS	49.8%	50.2%
SRPT	61.2%	37.6%
SSU	62.6%	37.2%
DSU	51.2%	48%

Table 6.5 Scenario 2: Percentage of speeded up and slowed down requests for different scheduling strategies.

Strategy	% of achieved speed-up	% of slow down
PS	85.25%	14.75%
SRPT	91.75%	8%
SSU	91%	7.5%
DSU	86.25%	13.75%

well. The reason behind this improvement is that PS scheduling do not give any preference to short size requests however, DSU tries to mimic SRPT by giving preference to least attained serviced request.

The *Table* 6.4 and 6.5 indicates the percentage of speeded up and slowed down requests *w.r.t.* FCFS scheduler for Scenario 1 and 2 respectively. SSU and SRPT scheduling dominates in terms of % of achieved speed up as compared to other scheduling algorithms for both the scenarios. DSU performs better than PS scheduling in both the scenarios.

6.6 User Abort Analysis

For starvation analysis, we used *Maximum Response Time* metric as a measure of unfairness till now. Now, we are going to analyze starvation for the different scheduling strategies using another metric, *i.e.* User Abort.

Today, users cannot wait for large amount of time for getting their request serviced. They might look for other alternatives if some particular website makes them wait for too long. Thus, we may assume that the user waits for certain specific amount of time (say *user threshold*) till his request gets served after which he aborts. This gives rise to a situation known as *User Abort*.

We set the value of user threshold to be slightly less than the maximum response time for *unbiased* PS scheduling and measure the percentage of users having response time greater than user threshold (percentage of user abort) for the PS, SRPT, SSU and DSU scheduling strategies.

The *Table* 6.6 clearly indicates that SRPT penalizes the large size requests whereas on the other hand, *SSU* and *DSU*, both strategies provide less scope of starvation thereby handling more number of

Table 6.6 Percentage Of User Abort in two scenarios for different scheduling strategies.

Strategy	Scenario 1	Scenario 2
PS	3.1%	2.1%
SRPT	3%	1.8%
SSU	**0.9%**	**0.25%**
DSU	**1.6%**	**1.1%**

users. *SSU* and *DSU* scheduling outperforms both PS and SRPT in terms of user aborts and thus, *these algorithms ensures more reliable service than either PS or SRPT*.

Chapter 7

Applications: CPU Scheduling

7.1 Introduction

In real time uniprocessor multiprogrammed systems, only one process can run at a time. All the processes which are waiting for CPU resources and are ready to execute resides in ready queue. Whenever CPU becomes idle (either because executing process is waiting for I/O routine to complete or process terminates), a new process needs to be selected for execution among all the processes present in the ready queue. The task of selecting a new process among all the runnable processes present in the ready queue for execution is referred to as *CPU Scheduling*. The module that is responsible for giving control of the CPU resources to the selected process by *CPU Scheduler* is referred to as *dispatcher*. *CPU Scheduling* is a fundamental problem in terms of minimizing the mean wait time , ensuring fairness among processes and enhancing the overall performance of the system.

7.2 Overview of existing CPU Scheduling Algorithms

Many CPU scheduling algorithms such as FCFS, Shortest Job First, Round-Robin, Priority Scheduling etc. have already been proposed. First Come First Serve (FCFS) scheduling selects a job with least arrival time. This algorithm is fair to all and will not create starvation for any of the job. However in FCFS, the earlier arrived process with large duration time may unnecessarily delay the short jobs arrived later leading to high mean wait time. This phenomena is referred to as Convoy Effect. Shortest Job First algorithm selects a process with least duration time among all the runnable processes. SJF algorithm is provably optimal in giving minimal average wait time for all the processes. But the tendency of SJF algorithm to prefer shorter jobs might create starvation for large jobs. Thus, SJF algorithm might not be fair towards long processes in the presence of continuous arrival of short processes. Further, practically process duration time is not known in advance and thus, SJF algorithm cannot be applied directly for the purpose of CPU scheduling. There are heuristics like *exponential average* which predicts the next CPU burst time of a process depending upon its previous CPU burst time. Round Robin scheduling is similar to FCFS scheduling but preemption is added to allow switching between processes. In RR algorithm,

each of the process is given a fixed time slice known as *time quantum*. The process runs until the time quantum expires, then context switch takes place leading to the execution of the next arrived process and so on. Very large value of *time quantum* will make Round Robin exactly similar to FCFS algorithm. If the value of *time quantum* is too less, then most of the time would be spent in context switching between the processes. The average wait time under RR policy is often long. Priority based scheduling selects a process with highest priority to schedule next. This scheduling technique may create starvation for low priority processes if high priority processes keep coming with respect to time.

Wait time and Starvation (situation of waiting for indefinite time) are two of the most important factors for choosing a CPU Scheduling policy for processes. Various scheduling policies have already been proposed. However, existing scheduling techniques either reduce wait time or ensure that there is no starvation but not both. No existing scheduling algorithm aims at reducing mean wait time and avoiding starvation both simultaneously. Further, many existing scheduling algorithm make use of CPU burst time of a process which is not known to us in advance practically. We present two Speed Up scheduling techniques for the purpose of CPU Scheduling which aims at reducing mean wait time but at the same time ensuring less scope of starvation - *Static Speed Up Process Scheduling (SSUPS)* (assumes process burst time is known) and *Dynamic Speed Up Process Scheduling (DSUPS)* (process burst time is not known in advance).

7.3 Speed Up Process Scheduling Algorithms

7.3.1 Static Speed Up Process Scheduling (SSUPS)

This scheduling technique is non-preemptive. The priority for a process p is assigned as follows:

$$Priority(p) = AT(p) * BT(p) \tag{7.1}$$

where $AT(p)$ denotes the arrival time of process p and $BT(p)$ denotes the CPU burst time of process p. The process with least value of priority is chosen to schedule next. If there are multiple processes with minimum priority value, then the process with smallest burst time is chosen. This scheduling technique assumes that process CPU burst time is known in advance. The idea is to give preference to small processes but this preference takes into consideration the arrival time to avoid starvation.

7.3.2 Dynamic Speed Up Process Scheduling (DSUPS)

DSUPS scheduling technique is preemptive and is more practical than SSUPS because it does not make use of CPU burst time to assign priority but instead use attained service parameter to estimate the CPU burst time of the process. The priority for process p is assigned as follows:

$$Priority(p) = AT(p) * AS(p) \tag{7.2}$$

Table 7.1 Mean Wait Time (in time units) for different CPU Scheduling policies based on different system load values.

System load (ρ)	FCFS	SJF	SRPT	RR	SSUPS	DSUPS
0.8	74	**37**	**27**	78	**38**	77
0.9	129	**57**	**45**	129	**59**	126
1	135	**60**	**49**	143	**63**	132
1.1	672	**188**	**170**	628	**194**	600
1.2	932	**259**	**248**	868	**273**	890
1.3	1543	**443**	**436**	1451	**520**	1497
PLD	3425	**987**	**951**	2819	**1122**	2911

Table 7.2 Maximum Wait Time (in time units) for different CPU Scheduling policies based on different system load values.

System load (ρ)	FCFS	SJF	SRPT	RR	SSUPS	DSUPS
0.8	455	**1448**	**1461**	1059	**1420**	1802
0.9	589	**1554**	**1713**	1417	**1520**	2397
1	695	**1823**	**1823**	1548	**1802**	2051
1.1	1796	**7887**	**7894**	6759	**6453**	8380
1.2	2351	**8979**	**8979**	7170	**6658**	**8395**
1.3	3371	**10671**	**10671**	9233	**7262**	**8774**
PLD	7053	**15431**	**16529**	15495	**13051**	**14186**

where $AT(p)$ denotes the arrival time of process p and $AS(p)$ denotes the attained service duration for process p. The process with minimum priority value is chosen.

7.4 Experiments and Results

We used a simulation based model for conducting experiments and getting results. We assumed that processes are arriving in Poisson distribution and their burst times are exponentially distributed. The number of processes for experiments were 10 thousand. We conducted experiments for different values of process load ρ and present here the results for the same. We also used real process logs dataset (PLD) used in Chapter 5 for comparison between different scheduling algorithms.

Experimental results show that SSUPS scheduling algorithm provides far better mean wait time as compared to FCFS and Round Robin scheduling but slightly higher than SJF and SRPT (which are optimal algorithms for providing minimal mean wait time). Further, our SSUPS algorithm reduces the problem of starvation in terms of maximum wait time as observed in SJF and SRPT (preemptive version of SJF) for all values of system load since our algorithm takes into consideration the arrival time to avoid large delay for any of the process. On the other hand, SJF and SRPT makes large size process to wait for too long resulting in high value of maximum wait time. Low value of maximum wait time guarantees that all the processes get good service. DSUPS algorithm is also successful in

Table 7.3 Standard Deviation of wait time for different CPU Scheduling policies based on different system load values.

System load (ρ)	FCFS	SJF	SRPT	RR	SSUPS	DSUPS
0.8	**94**	**107**	110	144	**105**	224
0.9	190	**152**	**171**	250	**149**	351
1	197	**168**	**180**	268	**165**	349
1.1	879	**781**	**832**	1125	**714**	1474
1.2	1208	**1010**	**1018**	1474	**920**	1879
1.3	1835	**1526**	**1536**	2080	**1311**	2154
PLD	**2265**	2826	**2782**	3293	**2253**	3610

providing better mean wait time as compared to FCFS and Round Robin for majority of system load values and provides lesser maximum wait time as compared to SJF and SRPT under overload situation ($\rho >= 1.2$). FCFS algorithm provides the least maximum wait time since it is fair to all the processes but at the expense of high mean wait time. By examining the experimental results from *Table* 7.1 and 7.2, we can conclude that our process scheduling algorithms (SSUPS and DSUPS) are successful in providing less mean wait time but at the same time ensuring less scope of starvation. Thus, our proposed algorithms addresses the trade-off between wait time and starvation. SSUPS algorithm performs much better than DSUPS algorithm in terms of both mean as well as maximum wait time. The reason is that SSUPS algorithm assumes that process burst time is known in advance whereas DSUPS algorithm tries to estimate the process burst time using attained service parameter. But DSUPS algorithm is more practical than SSUPS algorithm since process burst time is not known in advance.

From [16], regarding interactive systems (such as time sharing systems), it is stated that minimizing the *variance* measure for wait time is more important than to minimize the *mean* measure. A system with reasonable and predictable waiting time may be considered more desirable than a system that is faster on average but is highly variable. However, little work has been done on CPU-scheduling algorithms that minimize variance. *Table* 7.3 shows the value of standard deviation (square of standard deviation is *variance*) in wait time for different CPU scheduling algorithms based on different values of system load. We experimentally find that *SSUPS algorithm provides the minimal variance* in wait time and thus, *outperforms all the existing algorithms in terms of variance minimization*.

7.5 Speed-up/Slow-down Analysis

In this section, we compare and contrast different scheduling algorithms based on their speed-up/slow-down characteristics.

Consider a set of processes $< p_1, p_2, p_3 , ... , p_n >$. Let their corresponding wait times under scheduling policy X and Y be denoted by $< w_{x_1}, w_{x_2}, w_{x_3} , ... , w_{x_n} >$ and $< w_{y_1}, w_{y_2}, w_{y_3} , ... , w_{y_n} >$ respectively.

Table 7.4 Percentage of achieved speed up (SU) and slow down (SD) for different CPU scheduling strategies with respect to other scheduling strategies.

Algorithm	**FCFS**		**RR**		**SJF**	
	SU	SD	SU	SD	SU	SD
FCFS	0%	0%	34.8%	61.4%	9.4%	**82.8%**
RR	61.4%	34.8%	0%	0%	8.8%	**87%**
SJF	**82.8%**	9.4%	**87%**	8.8%	0%	0%
SRPT	**88%**	8%	**93.8%**	1.8%	**75.4%**	14.4%
SSU	**81.6%**	10.6%	**87.2%**	8.2%	16.6%	42.2%
DSU	**72.8%**	24.4%	**76.8%**	19.6%	**48%**	**47.8%**

Table 7.5 Percentage of achieved speed up (SU) and slow down (SD) for different CPU scheduling strategies with respect to other scheduling strategies.

Algo.	**SRPT**		**SSU**		**DSU**	
	SU	SD	SU	SD	SU	SD
FCFS	8%	88%	10.6%	81.6%	24.4%	72.8%
RR	1.8%	93.8%	8.2%	87.2%	19.6%	76.2%
SJF	14.4%	75.4%	42.2%	16.6%	47.8%	48%
SRPT	0%	0%	**78.2%**	13.8%	**60.8%**	8.8%
SSU	13.8%	78.2%	0%	0%	47.4%	48.6%
DSU	8.8%	60.8%	48.6%	47.4%	0%	0%

Definition 11. *A scheduling algorithm X is said to speed up process p_i w.r.t scheduling algorithm Y iff $w_{x_i} - w_{y_i} < 0$. Similarly, X is said to slow down process p_i w.r.t. Y iff $w_{x_i} - w_{y_i} > 0$.*

Table 7.4 and 7.5 presents the comparison analysis for different CPU Scheduling algorithms based on their speed-up/slow-down characteristics for $\rho = 1.1$. Round Robin speeds up 61.4% of the processes w.r.t. FCFS scheduling. All the four scheduling algorithms SJF, SRPT, SSUPS and DSUPS speeds up nearly 70-90% of the processes w.r.t. FCFS and Round Robin Scheduling. SRPT scheduling speeds up majority of the processes w.r.t. any other scheduling strategy. SSUPS scheduling w.r.t. DSUPS scheduling speeds up 47.8% processes and slows down 48.6% processes. In this manner, we can compare between any two scheduling algorithms based on their speedup/slowdown characteristics.

7.6 Discussion

Speed Up concept provides a new mechanism to compare and contrast different scheduling algorithms. Speed Up can be implemented by various techniques. In [9] and [10], it was implemented by location table and swapping of jobs. In this paper, we use priority function to seamlessly and certainly perform speed up. Any scheduling algorithm that uses ranking function to select next job can potentially speed up that job.

From speed up perspective, we have assumed that no process is requesting for speed up. We extended our implicit priority functions of speed up problem for the purpose of CPU Scheduling. Experimental results show that our proposed techniques are successful in reducing the mean wait time and at the same time providing less scope of starvation, thus addressing the trade-off between wait time and starvation. Further, SSUPS algorithm provides minimal *variance* in wait time as compared to all the other existing scheduling algorithms.

Chapter 8

Conclusions

Today in flexible and dynamic application environments, user might request for faster execution of some already executing instances. In such cases, the system should be able to respond to such on-line requests for speed up. The problem of scheduling of speed up requests at run-time has not been adequately studied in literature before. In this thesis, we modeled the Speed Up Scheduling problem *without acquiring additional resources* to handle on-line speed up requests and analyzed two different aspects of it - *Speed Up with no constraints* and *Fine Grained Selective Speed Up*. We provided efficient implicit techniques to address speed up problem where the notion of speed up is incorporated in the priority function. Experimental results show that our proposed algorithms are able to provide almost the similar achieved speed up as compared to existing speed up algorithms. Further, our speed up algorithms outperforms the existing algorithms in terms of slow down caused to non-urgent jobs thereby providing remedies for the delayed jobs alike existing algorithms where speed up is achieved for urgent jobs at the expense of high slow down for the rest of the jobs. Overall mean wait time is improved dramatically by our speed up algorithms owing to the design of the priority function. We provided GPSU algorithm to address more specific case of speed up problem - *Fine Grained Selective Speed Up* (each of the urgent job could request for specific percentage of speed up at run-time) which has not been yet addressed. Our algorithms are computationally efficient than existing speed up algorithms where there is an overhead of maintaining a *location table*.

For web scheduling, the performance of SRPT degrades dramatically in an environment where there is a high variability in link bandwidth. Using our implicit techniques for speed up problem, we provided two web scheduling algorithms SSU and DSU for static and dynamic environments respectively. We established the usefulness of our algorithms with a simulation based model using trace driven workload. We extended our algorithms for CPU Scheduling and experimentally finds that our algorithms are successful in reducing the mean as well as variance in wait time and at the same time providing less scope for starvation.

Speed Up concept provides a new mechanism/metric based on which we can compare and contrast different scheduling algorithms based on their speed-up/slow-down characteristics. In this thesis, we analyzed the existing scheduling algorithms from the perspective of speed up. *Speed Up provides a*

new way of addressing urgent jobs at run-time, provides a different evaluation criteria for comparing scheduling algorithms and has practical applications.

8.1 Scope for future work

This thesis provides a framework for addressing speed up problem. But still there are several issues which could be addressed :

- We explained how jobs requesting for speed up at run-time can be scheduled under the assumption that *no additional resources can be acquired*. An extension of our work could be to examine how such on-line requests for speed up can be handled if multiple servers and other additional resources are available so as to maximize the number of speeded up urgent jobs but at the same time ensuring efficient resource utilization.

- We also assumed that jobs are independent of dependence constraints throughout this work. It would be interesting to see how we can speed up jobs requesting for it where that job is either dependent on some other non-urgent job or on another job requesting for speed up ?

- An extension of our work could to be to consider a case where there exists multiple servers each having its own queue for arriving jobs and the jobs could move from one queue to another. How to handle speed up requests in such case where *load-balancing* could also be used in achieving speed up.

Bibliography

[1] A. AlSa'deh and H. Adnan. Shortest Remaining Response Time Scheduling for Improved Web server Performance . *WEBIST*, 2008.

[2] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continous job streams. . *In Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[3] E. W. Biersack, B. Schroeder, and G. U. Keller. Scheduling in practice. *ACM Sigmetrics*, March 2007.

[4] L. Cherkasova. Scheduling Strategy to Improve Response Time for Web Applications. . *Proceedings Of High Performance Computing and Networking, HPCN*, April 1998.

[5] M. Crovella and R. Frangioso. Connection scheduling in web servers. . *USENIX Symposium on Internet Technologies and Systems*, 1999.

[6] M. Gong and C. Williamson. Quantifying the properties of SRPT scheduling. . *MASCOTS*, 2003.

[7] D. Gross, J. Shortle, J. Thompson, and C. Harris. Fundamentals of Queueing Theory. 2008.

[8] M. Harchol-Balter, B. Schroeder, M. Agrawal, and N. Bansal. Size based scheduling to improve web performance. . *ACM Transactions on Computer Systems (TOCS)*, 2003.

[9] E. Kafeza, D. Chiu, and K. Karlapalem. Improving the Response Time of Business Processes: An Alert-based Analytical Approach . *HICSS*, 2006.

[10] E. Kafeza and K. Karlapalem. Speeding Up Electronic Commerce Activities Using CapBasED-AMS. *WECWIS*, 2000.

[11] E. Kafeza and K. Karlapalem. Speeding up CapBasED-AMS activities through multi- agent scheduling. *In Multi-Agent-Based Simulation Springer*, pages 119–132, 2001.

[12] A. King. Speed up your site: web site optimization. . *New Riders Indiana, 1st edition*, 2003.

[13] C. Murta and T. Corlassoli. Fastest Connection First: A new Scheduling Policy for Web Servers. . *ITC-18*, 2003.

[14] L. Schrage. A proof of optimality of the shortest remaining processing time discipline. . *Operations Research*, 1968.

[15] B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. . *ACM (TOIT)*, 2006.

[16] A. Silberschatz and P. Galvin. Operating System Concepts, 8th Edition, International Student Version. 2009.

[17] Y. Tay. Analytical Performance Modeling for Computer Systems. 2010.