A

Dissertation

On

Persistence Priority Queue Based Job Scheduling

Submitted in partial Fulfillment of the requirement

For the award of the Degree of

Master of Technology

In

Computer Science & Engineering

Submitted By

Shubham Jain

University Roll No. 2K11/CSE/14

Under the esteemed guidance of

Dr. Kapil Sharma

Computer Engineering Department, DTU, Delhi

DELHI TECHNOLOGICAL UNIVERSITY

2011-2013

DELHI TECHNOLOGICAL UNIVERSITY

DELHI - 110042

CERTIFICATE

This is to certify that the dissertation titled "**Persistence Priority Queue Based Job Scheduling**" is a bonafide record of work done at **Delhi Technological University** by **Shubham Jain, Roll No. 2K11/CSE/14** for partial fulfilment of the requirements for degree of Master of Technology in Computer Science & Engineering. This project was carried out under my supervision and has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma to the best of our knowledge and belief.

Date: _____

(Dr. Kapil Sharma)
Project Guide
Department of Computer Engineering
Delhi Technological University

ACKNOWLEDGEMENT

First of all, let me thank the almighty god, my parents and my dear friends who are the most graceful and merciful for their blessing that contributed to the successful completion of this project.

I feel privileged to offer sincere thanks and deep sense of gratitude to **Dr. Kapil Sharma,** project guide for expressing his confidence in me by letting me work on a project of this magnitude and using the latest technologies and providing their support, help & encouragement in implementing this project.

I would like to take this opportunity to express the profound sense of gratitude and respect to all those who helped us throughout the duration of this project. **DELHI TECHNOLOGICAL UNIVERSITY**, in particular has been the source of inspiration, I acknowledge the effort of those who have contributed significantly to this project.

Shubham Jain

University Roll no: 2K11/CSE/14

M.Tech (Computer Science & Engineering)

Department of Computer Engineering

Delhi Technological University

Delhi – 110042

ABSTRACT

This thesis considers the problem of performance degradation of servers caused due the sudden increase in the requests resulting in request rejection and performance degradation. This overload condition can occur at any time without any predication. During this critical condition of overload, performance criteria such as response time and throughput get hampered which in turn affects user satisfaction level and can lead to imploded revenues.

Some users are important than others and hence they require special care under these overload conditions. We can distinguish these important customers by assigning priorities to them. Selecting higher priority user can be done using priority queue but simple priority queue implementation are non- persistent in nature and hence does not resist failures and it is difficult to recover or recovery may take a long time. Hence, we need a solution so that these requests become persistent in nature and do not get lost on failures and get easily recovered.

We have proposed the database implementation of queue such that it selects the highest unprocessed job for processing. Since requests are stored in database and this database can be made highly available resulting in approximate persistent system. To overcome starvation problem and to make scheduling algorithm more responsive while handling user requests with fixed priority, we have proposed a dynamic priority scheduling algorithm where the priority of the request is determined at execution time and it is weighted combination of priority scheduling and shortest job first with aging concept. Our algorithm is also motivated towards better profit for service provider while maintaining good response time and throughput.

Our simulation results show that our strategy can provide good profit to service providers. Our results also show that higher priority jobs are handled with special care and low priority jobs do not get starved.

Table of Contents

List of Figures

List of Tables

List of Abbreviations Used

FCFS	First Come First Serve
SJF	Shortest Job First
SPT	Shortest Processing Time
FIFO	First In First Out
SRPT	Shortest Remaining Processing Time
EDF	Earliest Deadline First
LLF	Least Laxity First
QoS	Quality of Service
LPT	Least Processing Time
OCC	Optimistic Concurrency Control
SQL	Structured Query Language
HA	Highly Availability
CPU	Central Processing Unit
IO	Input Output

Chapter One: Introduction

Internet is a rich source of information and its usage has gone changed from being source of communication, browsing information to a medium for e-services for conducting business and marketing of products. These changes are attracting more and more people to use e-services over internet which is creating problem for the servers to handle the large number of users. Sometime conditions become more problematic when a large number of users simultaneously try to access the server which results in excessive load on the server resulting in degradation in performance such as request rejection higher response time, lower throughput, starvation of user request, etc. There is a need to overcome this problem for better profit of service provider and better user satisfaction.

1.1 Motivation

Every server is characterized by its capacity up to which it can provide efficient services. If the capacity of server is reached and if server continues to accept requests then application performance and stability can be deteriorated and it may lead to server crash. To handle this problem, servers are configured such that if server's capacity is reached and the requests in the request queue exceed a predetermined number, then server will stop accepting requests. When this limit is reached, an error message is displayed. At such high load, Server may work inefficiently and response time may be very unacceptable by user. User requests are important in ecommerce and rejecting these requests may result in frustration among users and they will switch to other ecommerce website. Special care must be taken for the users with high priority so that they job get done without being delay, at the same time low priority requests should not get starved.

Higher priority requests must have proper resources for completing their completion (which is possible if low priority jobs are attended only after competition of high priority jobs). So, there is requirement of solution such that all user requests whether

of high priority or low priority must persist and high priority requests are processed quickly without starvation of low priority request and overall efficiency of the server should also be maintained. For server efficiency, we need to limit the number of requests it can handle at time.

For fulfilling above requirements, we need to design a dynamic priority queue that will efficiently handle high priority requests as well as store all incoming requests. One way is store the priority queue in system memory, but it is volatile in nature.. Another way of implementing dynamic priority queue is Database. Database makes it easier to handle and to add to it, failure recovery is also very quick.

1.2 Approach Use

In case of server overload, server may not be able to process all requests simultaneously rather it will process a predefined number of request at a time. So, which request should be processed first? In every application some requests are generally considered more important than others and should to be processed as soon as possible without delay. So, we can assign priority to every task that needs to be processed and based on priority, requests should be processed. Requests with higher priority should be processed first as compare to process with low priority. User with high priority should be special taken care of so that they get their job get done without being delay.

Higher priory requests must have proper resources for their completion. Since our resource are limited, so it better to assign priory to user/task and provide the resources to high priority users/task first and using this way high priority request may get performance isolation.

Priority scheduling can be static or dynamic based on requirements. Generally, most of the application use static priority scheduling but due to complex requirements of applications, dynamic priority scheduling is required for better performance.

We will use the following approach for handling under high user request:

1. Rather than using priority queue in system RAM, we store the incoming user request in the database along with various parameters such as arrival time, priority of job, job size etc. By querying database, we can select the current highest priority job and send it to the job executor for execution.

2. We have proposed a variant of priority scheduling which depends on weighted component of priority of job, size of job, and total waiting time of the job. Motive of proposed scheduling is to avoid starvation of jobs and increase the profit for service provider.

1.3 Organization of Thesis

This thesis work is organized as follows:

Chapter 2 presents the literature review of the existing scheduling policies, overloading problem in web servers, concurrency problems in databases and highly available databases.

Chapter 3 presents the concept of priority scheduling. It briefly explains the priority scheduling algorithm and describes the types of priority scheduling algorithm.

Chapter 4 presents the proposed system for solving the overloading problem. It consists of proposed system architecture which shows the overall design of the system. Then it discusses the various factors for selection of scheduling algorithm and concurrency control technique. We finally present the proposed scheduling algorithm for job scheduling.

Chapter 5 presents results based on the simulation performed using the scheduling algorithms. The performance comparison of four scheduling algorithms is done. These algorithms are compared on three different range of load for different performance factors. We finally present the overall performance summary of the four scheduling algorithms.

Chapter 6 discuss about the Conclusion and Future Scope of the work to further improve the proposed system.

Chapter Two: Literature Review

This chapter is devoted to the literature review of Scheduling, Overloading problem, Concurrency problems in databases and highly available databases.

2.1 Scheduling

Industrial operations are very crucial for fulfilling one or more objectives. Every operation consists of number of jobs that can be solved in a number of ways but there are some sequences of jobs execution that can fulfil the given requirements. Since resources available are generally less than required resources, they should be used in an optimal manner and that can be done using scheduling. Scheduling require some decision making process to deal with efficient allocation of resources. It deals with allocation of resources to jobs over the time and its goal is to optimize one or more objectives. Resources can be processing units and task can be may operation need to be done. Each job has a particular deadline, priority, and earliest possible starting time. Since our resources are generally less than required, resources should be used in optimal way and that can be done using scheduling. Scheduling has certain criteria associated with it such as efficiency, response time, throughput and fairness.

In past, many scheduling algorithms have been proposed based on different requirement and goals. In computer science, scheduling is generally used in scheduling jobs in CPU and servers. Simplest scheduling algorithm is first come first served, where all the jobs are processed in order of their arrival in the system. Jobs are placed in queue in arrival order. First job is removed from the queue and then that job is processed. Then next job is processed in the same manner and so on. FCFS treats all jobs equally so if all the equally important than we can use FCFS.

(Schwiegelshohn & Yahyapour, 1998) presented the analysis of the FCFS on the parallel system and proposed new pre-emptive FCFS algorithm and found that pre-emptive FCFS showed better performance as compared to non- pre-emptive FCFS for

particular parameters. (Lo & Mache, 2002) discussed the performance of FCFS for both the prime time and non-prime time job queues and their results showed that prime/non-prime time scheduling was very beneficial for batch scheduling, across short term scheduling algorithms such as FCFS, across workloads and varying levels of system load.

Smith (1956) addressed the single machine problem of minimizing the sum of the completion times. An optimal permutation was obtained by sequencing the jobs in non-decreasing order of their processing times; this rule is known as the Shortest Processing Time rule (SPT rule). Shortest job first scheduling is based on the execution time on the job. Jobs are executed on the basis of execution time where job with smallest execution time is processed first and then the job with next smallest execution time. Shortest job first have better response time than FCFS. SJF has been discussed in (Cherkasova, 1998), (Duquennoy & Grimaud, 2010).

Issues with SJF discussed by (Younas, Awan, & Chao, 2006) were as follows:

First, SJF policy cannot improve performance if all the requests are homogeneous requiring same service time. Second, it requires that the size of requests or the target data is to be known in advance. Third, most of the requests are of same size in terms of processing time. Fourth, there is possibility of starvation for jobs with longer execution time.

(Cherkasova, 1998) presented alpha scheduling strategy that allowed reordering the requests queue such that it significantly reduced the waiting time in a queue and resulted in significantly improved response time per request. Under the same load, it was found that the server which used alpha scheduling had shorter requests queue than the server which was operating under FIFO scheduling. Results showed better resources utilization and this resulted in improved throughput of the system. In was found that reducing the overall response time per request will lead to better system resources utilization and resulted in better system throughput.

McNaughton (1959) gave a simple algorithm that finds an optimal preemptive schedule: Scheduling was based on the least remaining execution time of the jobs. Job with least remaining execution time was processed first and then the job with next least

remaining execution time. Shortest remaining job first is pre-emptive, so if a new job arrives and it has less execution time than the remaining execution time of currently executing job then scheduler immediately switches to service newly arriving job. Shortest remaining time first has following drawbacks:

1. SRPT requires knowing the execution time of request in advance.

2. Starvation of the long execution time jobs as proved in (Bender, Chakrabarti, & Muthukrishnan, 1998).

SRPT has been a topic of research and was discussed by (Bansal & Harchol-Balter, 2001), (Harchol-Balter, Schroeder, Bansal, & Agrawal, 2003), (Duquennoy & Grimaud, 2010).

(Bansal & Harchol-Balter, 2001) investigated the problem of unfairness in SRPT scheduling and compared it with a variant of fair sharing, PS scheduling. They proved that for any job size distribution under moderate system load, all jobs preferred SRPT over PS. But as the load was increased, only a few job size distributions preferred SRPT. They also briefly discussed the comparison of SRPT with other scheduling algorithm such as FCFS, SJF, and LCFS.

(Harchol-Balter, Schroeder, Bansal, & Agrawal, 2003) urged that size of request can be well-approximated by the size of file. SRPT was proved to have lowest mean response time.

In 1973, Liu and Layland introduced the concept of fixed priority scheduling for periodic task. They introduced concept of Critical Instant Theorem (LIU & LAYLAND, 1973) regarding the feasibility of fixed priority task. A critical instant for a task is a release time for which the response time is maximized. The theorem stated that, for a set of periodic tasks with fixed priorities, a critical instant for a task occurred when it was invoked simultaneously with all higher priority tasks.

In priority scheduling, priority number is assigned to each task and task with highest (or lowest) priority are chosen to process. User with higher priority must be special taken care of so that they get their job done without being delay while low priorities users do not get starve. Priority scheduling is extensively used in various

applications and it was studied extensively in the past. Priority scheduling has also used in server scheduling algorithm to handle the complex requirements such as priority of users, size of the job, etc.

Priority scheduling is of two types.

1. Fixed priority job scheduling.
2. Dynamic priority job scheduling.

Fixed priority scheduling processes the jobs on the basis of given priorities and priority of job does not change once assigned. Fixed priority scheduling is also known as static priority scheduling. Fixed priority scheduling was studied in (Audsley, Burns, Richardson, Tindell, & Wellings, 1993), (Lo & Mache, 2002) , (Younas, Awan, & Chao, 2006), (Holton, Awan, & Younas, 2009).

(Audsley, Burns, Richardson, Tindell, & Wellings, 1993) has done analysis of system scheduled with static priority preemptive dispatcher at runtime where task were allowed to have internal blocking and released jitter. Formulae were designed for predicting the worst case interference a task could suffer from higher priority task and results were analysed using those formulas.

(Younas, Awan, & Chao, 2006) analysed the arrival process of distinct Ecommerce requests and employed a network-centric priority scheduling mechanism that gave special care to high priority requests. They emphasise the importance of vital request and aimed to improve the response time of such requests by reducing the queuing delay of vital requests at the network nodes. Experiments have shown a significant performance improvement of vital requests.

(Holton, Awan, & Younas, 2009) focused on the performance of Web portals in an E-commerce environment which involved the processing of a large number of users' requests. A class-based priority scheme was proposed which classifies users' requests into high and low priorities. Requirement for this classification was due to the reason that number of low priority requests (such as search and browse) was quite higher than high priority requests. Their objective was to reduce the response time for high priority requests without causing starvation for the low priority requests. They put a limit on the

number of consecutive high priority requests processed which prevents the starvation of low priority requests, and by varying that limit, along with adjusting queue lengths, system was tuned for various traffic conditions. Their results showed significant performance improvements in the processing of high priority requests. Problem with the given work was that the consideration of the simple priority scheme proposed was not satisfactory since it might lead to prospective clients leaving the site before making purchases due to delays in requests for information.

Dynamic priority scheduling process the request based on given priorities where priorities of the jobs are evaluated at run time. Here priority of job changes based on various factors such as total waiting of the job, deadline of the tasks etc. Popular dynamic priority scheduling algorithms are earliest deadline first (EDF) and least laxity first (LLF). Earliest Deadline First (EDF) is a dynamic deadline scheduler. It is dynamic because the priority of the task keeps changing throughout the execution cycle. It is called Deadline based because the scheduler considers the deadline of a task before deciding whether it needs to be scheduled. The task which is going to meet its deadline first is scheduler first (Mattihalli, 2010). Each EDF tasks needs three parameters to define the required scheduling behaviour of a task. They include 1.Cycle time which defines total period of the task. 2. Deadline which defines the deadline of the task with in the period. 3. Processing time which defines the total computation time of the task for completing the assigned activity in the given cycle and deadline.

(Oh & Yang, A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks, 1998) described Least-Laxity-First (LLF) as a scheduling algorithm that assigns a higher priority to a job which has the least laxity. The laxity is the time left by its deadline after the task is completed assuming that the task could be executed immediately without pre-emption. A task which has zero laxity can be scheduled right away and must be processed with no pre-emption in order to meet the deadline. A task which have negative laxity showed that the task will miss the deadline.

Dynamic priority scheduling has been discussed in (Ripoll, Fornes, & Crespo, 1996), (Chen, Gao, & Wang, 2007) , (Li, et al., 2010) , (Han, Ni, Sun, & Deng, 2010) , (Lee, Wang, & Zhou, 2011) .

(Chen, Gao, & Wang, 2007) discussed about busy business web system whose goal was revenue generation. They mentioned the criticality to increase the completion rate of transaction related requests and requests from VIP and introduced 'Multi-objective dynamic priority optimization scheduling' mechanism and algorithm was proposed based on reward-driven multi-dimension criterion for request classification and one-step-ahead overload estimation. Their goal was to increase the completion rate of the valuable requests. Their results showed decrease in average response time and increase in the completion rate of transaction request under overload condition when MODP strategy was used. But they paid less attention to the low priority request which resulted in starvation.

(Li, et al., 2010) proposed a dynamic job scheduling algorithm for process engine which both maximized the customer satisfaction and ensured quality of service (QoS) requirements. Priority was assigned to jobs based on weighted utility value which is a combination of predicted business value and the time left to execute the process. Priority assignment was done dynamically. Proposed priority scheduling considered only two priorities and QoS metric consisted of only response time. Results showed that by selecting appropriate values of parameters, proposed dynamic scheduling can provide an opportunity to balance between customer satisfaction and QoS. It was also found that better QoS can be achieved for performance optimization.

(Han, Ni, Sun, & Deng, 2010) discussed the problem of reducing the delay in the application start-up in multi-application operating environment. For solving above problem, author has proposed a dynamic priority scheduling algorithm based on interestingness evaluation of the application. Proposed model was based on the history records of users and using that user-application interestingness, a matrix is generated. Then, dynamic priority scheduling algorithm was used with the interestingness and an

implementation strategy are presented. Proposed algorithm was found more suitable for the resource constrained consumer electronics devices than the traditional ones.

(Lee, Wang, & Zhou, 2011) proposed a strategy dynamic priority scheduling algorithm (DPSA) based on the static priority scheduling algorithm. The purpose of the introduction of dynamic way was to avoid the long waiting of low priority tasks. The proposed technique has several queues based on the number of different priority of the task. Every arrived task was inserted into the queue based their pre-assigned priority. Proposed technique also used the concept of aging by shifting of low priority request from low priority queue to high priority queue. It seemed to solve the starvation problem

Some the other scheduling algorithms which are of area of interest are as follows:

Last come first serve scheduling is based on the sequence order where a most recent arrived job is processed first. All the incoming jobs are inserted first and then most recent job is picked for processing. Hence new job requests are handled with very little waiting time. Example, Let us consider a system where for four jobs J1, J2 ,J3 and J4 are in queue for execution with processing time say 12ms, 29ms, 13ms and 34ms. Let the request arrive in the order J1, J2, J3 and J4. Then the processing order will be first J4, then J3, then J2 and lastly J1. The Drawback of this scheduling technique is that job which enter early in the system may starve if new jobs are continuous arrived in the system.

In (Singhmar, Mathur, Apte, & Manjunath, 2004) discuss the use of LIFO in e-commerce for separating the browse request from revenue generating request and help in controlling overloading on servers. It was found that, at normal time FIFO is better than LIFO so they have used FIFO at normal time and LIFOPri at overload times.

Random scheduling is based on principle that each new job is chosen randomly from the queue with equal probability and then executed. So, each job has equal probability of being chosen and sometime a newly arrived job may be selected or some time an older arrived job me selected for execution. Example, Let us consider a system

where for four jobs J1, J2 ,J3 and J4 are in queue for execution with processing time say 12ms, 29ms, 13ms and 34ms. Let the request arrive in the order J1, J2, J3 and J4. Then the processing order will be will determine at runtime and since it can follow any order for processing of jobs independently.

(Hong, Huang, & Yu, 1995) described the Longest Processing Time First as **"**Given a set of k independent tasks, each with arbitrary execution time, and a set of n homogeneous processors or machines, the LPT scheduling algorithm assigns the task with the longest execution time (among those not yet assigned) to a free processor whenever this processor becomes free. For cases when there is a tie, an arbitrary tie-breaking rule can be assumed". It was found that Longest processing time scheduling in general, cannot attain a minimum completion time but this algorithm is still widely used because the results of scheduling are generally acceptable for error tolerable problem domains..

Aging technique is used in scheduling to increase the priority of the low priority jobs. This is done by adding some number to the priority of the job. In case of shortest job first, jobs with lowest execution are executed first then the job with next lowest execution time and so on. Priority is given to smaller execution time job resulting in starvation of higher execution time job. By increasing the priority of the longer execution time jobs, starvation of jobs may be controlled. Similarly, In case of priority scheduling, lower priority jobs get starved. So by increasing the priority of low priority jobs, we can prevent starvation of jobs. Aging techniques has been discussed in (Balajee, Suresh, Suneetha, Rani, & Veerraju, 2010).

(Balajee, Suresh, Suneetha, Rani, & Veerraju, 2010) discussed the scheduling of jobs with multiple queues. They analyzed that queue of low priority have chances of getting starved so they applied aging technique by increasing the priority of jobs with longer execution time. Using this method, they avoided starvation problem.

2.2 Overloading Problem

Internet is a rich source of information over the web and its usage has gone changed from source of communication, browsing information to a medium to a medium for e-services for conducting business and marketing of products. These changes are attracting more and more people to use e-services over internet which has created problem for the servers to efficiently handle the large number of users. The condition sometime become more problematic when a large number of user simultaneously try to access the server which result in excessive load on the server resulting in degradation in performance such as higher response time, lower throughput, starvation of user request, etc. This situation of excessive load is known as Overloading problem. Overloading of the applications result in degradation in server performance or it rejected the request from the user. To overcome this problem, lot of work has been done but still there is scope for improvement in this field.

Jobs enters the application at a greater rate than the capacity of the application which results in degradation of the performance due to various reason such as high response time, loss of request jobs, less throughput, etc. Websites slowdown has been discussed in many research papers and solutions have been proposed based on different requirements. It is generally seen that customers generally switch to other alternatives if they face the degraded performance from the server. So priority must be given to high priority user to complete they request as fast as possible while also preventing starvation of low priority users.

Overloading problem has been discussed in (Li, et al., 2001), (Bansal & Harchol-Balter, 2001) , (Carlstrom & Rom, 2002), (Elnikety, Nahum, Tracey, & Zwaenepoel, 2004), (Singhmar, Mathur, Apte, & Manjunath, 2004), (Zhang, Riska, & Riedel, 2005) , (Schroeder & Harchol-Balter, 2006), (Younas, Awan, & Chao, 2006), (Andersson, Höst, Cao, Nyberg, & Kihl, 2006), (Chen, Gao, & Wang, 2007) , (Farah & Murta, 2008), (Homayouni, Jahanbakhsh, Azhari, & Akbari, 2009), (Totok & Karamcheti, 2010) .

(Li, et al., 2001) discussed about the slow down at major websites during peak times. Due to large response time and downtimes, user generally left the website. They presented CachePortal technology for accelerating database-driven- e-commerce websites. Being evaluated and compared with many alternative solutions, it was urged that CachePortal can provide many e-commerce applications scalability and improvement in user response time.

(Carlstrom & Rom, 2002) presented algorithms and architecture for improving performance of session-admission control and request scheduling in web-servers. Author tried to divide session into stages with specific service requirements and transition probabilities and the updated structure was made aware to server. A stage wise queuing was proposed which divided the requests according to request stage and these classified requests were inserted into stage-specific queues. When a critical resource got freed, a request was selected, dequeued from queue and services were granted to that request. Resource sharing between stages was controlled which maximized the reward function for application.

(Elnikety, Nahum, Tracey, & Zwaenepoel, 2004) discussed about problem of overload and response time in e-commerce websites due to which customer switch to other ecommerce website which resulted in low revenue. A method was proposed for request scheduling and admission control for e-commerce websites to overcome the overload and response time problem. Admission control is based on the principle that a maximum load should be kept just below the capacity of an E-commerce system. This prevents system overload and also achieves high throughput. Proposed method was external to server and do not require modification to host system, web-server or database server. They have used SJF for scheduling the job for ensuring low response time and aging mechanism for preventing starvation of large jobs. Results showed that they have consistent performance during overload condition and there was increase in throughput of the system.

(Zhang, Riska, & Riedel, 2005) discussed the problem of overload problem on the servers and showed the overloading problem propagation in three tier e-commerce server.

It was found that overloading conditions occurred due to many factors such as increased in number of users and the number of updates required depending on nature of requests. Author has advocated the requirement of system which can detect the overloading condition at the lower level and apply some self-adaptable configurations for ensuring high availability of services. Author has also presented a model that did the self- adaptive resources management at lower level of system and showed how it can detect and handle overload conditions for overcoming the service unavailability problems.

(Schroeder & Harchol-Balter, 2006) analyzed the performance of web-server under persistence overload and transient overload, under various complex conditions from both client and server point of view. A kernel-level solution based shortest remaining processing time scheduling was proposed which aimed to improve the performance of web-servers under overload condition and results showed improvement in the performance various server and client related metrics. Authors also argued for replacing fair scheduling by unfair scheduling for servers that favour short requests without causing starvation of long requests and it also improved the response time and performance of the server.

(Younas, Awan, & Chao, 2006) analysed the arrival process of distinct Ecommerce requests and employed a network-centric priority scheduling mechanism that gave special care to high priority requests. They emphasise the importance of vital request and aimed to improve the response time of such requests by reducing the queuing delay of vital requests at the network nodes. Experiments have shown a significant performance improvement for vital requests.

(Andersson, Höst, Cao, Nyberg, & Kihl, 2006) discussed the problem of websites overloading during various crisis times such as terror attacks, natural calamities, etc. where people try to access the information over internet through websites which resulted in web server overloading. Authors discussed the problem of rejection of request and advocated that user request should not be rejected. A dynamic content adaptation scheme was described for handling overloading problem of web servers. This scheme adjusted the document byte count and keep throughput below the bandwidth. It was found that

proposed solution work efficiently under overload conditions and it was also found more efficient than other common admission overload methods.

(Farah & Murta, 2008) presented web server workload generator which can overload web server based on the maximum capacity of the system. Based on the requirement, generated load can be real or synthetic based on the capacity of the system. The results showed that the Web server experienced better performance when processing long peaks of overload than when processing impulses of overload.

(Homayouni, Jahanbakhsh, Azhari, & Akbari, 2009) discussed the performance of session initiated protocol(SIP) servers under overloaded conditions. Authors have proposed a window based distributed and adaptive overload control algorithm that tried to control overload by controlling the total calls to SIP server. Results showed that proposed algorithm was fast, stable and perform better than other commonly used overload control algorithm.

(Totok & Karamcheti, 2010) discussed Reward-Driven Request Prioritization (RDRP) technique that aimed to maximize the reward generated by the service by dynamically assigning higher execution priority values to the requests whose sessions was likely to bring more reward. Generally, it has been a difficult task for e-commerce services provider to meet the client Quality-of-Service expectations in case of web servers overloading conditions which resulted in increase in request rejections and higher response times. Under these conditions, generally users got frustrated and lower the current service usage which resulted in reduced revenue. These services were generally designed as software systems which consisted of several physical and logical tiers like database tier, web tier and application tier. These services generally accessed multiple backend data sources, application server architecture and the flow of a request through the system. Requests competed for two critical exclusively-held server resources: server threads and second Database connections. These resources were pooled by the web server and the application server respectively. Scheduling of requests to available threads and database connections was done according to the request priority set by the RDRP module. The request with the highest priority was served first and FCFS was used as a tiebreaking

policy. It was found that RDRP techniques yielded profit under both overload and under load conditions for better customer experience.

2.3 Concurrency Problem in Databases

E-services usage is very popular now a days and its popularity is even going day by day. Numbers of users using these services are increasing every day and these e-services have become major part of trading and marketing. These services require a transaction processing system for carrying out very crucial task such as mobile and web applications, online reservations, e-payments, etc. As the users are increasing, load on transaction system is also growing which may sometime leads to concurrency problems. (Bernstein, Hadzilacos, & Goodman, 1987) described concurrency control as "When two or more transactions execute concurrently, their database operations execute in an interleaved fashion. Operations from one transaction may execute in between two operations from another transaction. This interleaving can cause transactions to produce incorrect results, thereby leaving the database in an inconsistent state. In this case we say that the transactions conflict with each other. The conflict between two transactions is called a write/read conflict or a write/write conflict if the inconsistent database state results from improper interleaving of the write(z) operation of the first transaction with the read(z) or the write(z) operation, respectively, of the second transaction. The read/write conflict is defined similarly. Read operations can be interleaved in an arbitrary fashion, therefore, there is no read/read conflict".

(Ullman, 1988) also discussed about concurrency as "In a conventional locking concurrency control algorithm, read or write locks are employed to ensure serializability by preventing other transactions from accessing a database object until the transaction holding the lock unlocks the object." With concurrency control based on timestamps**,** a unique timestamp is assigned to each transaction; transactions are then processed in such a way that their execution is equivalent to a serial execution in the timestamp order of the transactions (Bernstein, Hadzilacos, & Goodman, 1987).

Locking and timestamp-ordering concurrency control algorithms are similar in the sense that both take care of conflicts as they occur. The optimistic concurrency control method (Kung & Robinson, 1979) differs since detection of conflicts and their resolution were deferred until commit time. The underlying assumption here was that such conflicts were rare. The goal of every concurrency control method is to preserve the consistency of the database. This is achieved by guaranteeing serializability of executed transactions. An execution is serializable if it is computationally equivalent to a serial execution. For databases, with concurrent updates and read operation, concurrency problem are generally occur if locking is not available or used.

(Concurrency Problems) has discussed problems that can occur due to concurrency control:

1. Lost update
2. Dirty read
3. Non Repeatable Read

Lost updates problem generally occurs if two or more transactions try to select the same row of a table and update the row on the basis of value which was originally selected. A transaction is generally unaware of the activities of other transactions. This resulted in overwrites in the updates made by the last transaction and this ultimately resulted in lost updates.

For example, server thread A and thread B read the same unprocessed job request job1 from the database, let thread A start processing the job1 first and update dispatch time as t1, but thread B unaware of the this operation also start processing of the job1 and overwrite the dispatch time as t2. Hence, the update from thread A is lost and it result in unnecessary processing of job1 twice.

Dirty Read problem generally occurs when a row selected by a transaction T1 is being updated by another transaction T2. In this case, the transaction T1 is reading data which is yet to be committed and may be altered by another transaction T2 which is updating the row.

For example, server thread A read the job1 from database and send the update the status as processing and dispatch time of the job1 in database but before committing this update in database, another thread B also read the job1as unprocessed and try to update the status and dispatch time.

Non-repeatable Read occurs when a transaction tries to access the same row many times and reads different data each time.

For example, server thread A read the job1 from database where status='New' and mean while another thread B also read the status of job as 'New'. Thread 'A' changes the status of job1 to 'Processing' and later thread B reads the status the same job as 'Processing' instead of 'New'. Later on completion of job thread A update status of job1 to 'Processed' and later thread B read the status of job1 as 'Finish' instead of 'Processing'.

Concurrency problem can be solved by various concurrency control techniques such as locking and optimistic control methods.

Lock based Protocol is a lock based technique to control the concurrent access to a data item. Data item can be locked in two modes:

A. Exclusive mode in which data item can be both readable as well as writable.

B. Shared mode in which data item can be read only.

A lock may be granted to a transaction on an item if the lock requested is found compatible with the locks already held by other transactions on the item. Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the item, no other transaction may hold any lock on the item. If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted. Drawbacks of this locking scheme are deadlock, starvation, resource blocking.

(Kung & Robinson, 1981) proposed Optimistic concurrency control (OCC) methods with the aim to eliminate locking. He described OCC transactions consisted of three phases:

1. Read Phase.

2. Validations Phase

3. Write Phase

Read Phase consists of all the write operations performed on the local copies of the object to be modified. In validations phase, it is generally determined that the transaction will not cause a loss of integrity. Write phase consists of operations where copies are made global by committing the local copies updates globally.

In e-commerce scenario, the first phase generally includes thinking and user input time. It may consist of searching a product and finally making an order. It may last for an unpredictable time period. The last two phases generally did not require user interaction. Usually, Validation and write phases may last for very short very period of time based on operation complexity and operations in these phases are very critical and they require exclusive access. Failing to do so could result in inconsistent data, e.g. lost update.

(Orji, Lilien, & Hyriak, 1988) discussed the comparison of timestamp-ordering concurrency control with optimistic concurrency control. The main difference between optimistic and timestamp concurrency control techniques is that timestamp technique check for conflicts as soon as they occur whereas optimistic technique delay the testing until the transaction ends. Another difference is that, optimistic concurrency control technique resolved conflicts by aborting the conflicted transactions whereas timestamp concurrency controls techniques by delaying the transactions which reduced the number of restart. It was found that optimistic technique was good for transaction dominated by retrievals. It was also found that for transaction dominated by updates, the optimistic concurrency control algorithm spent time for performing such operations that a high probability of getting aborted by some earlier conflicting operations. Problem found with timestamp techniques was the very high overhead for testing the conflicts as they occur in case of system where transitions were arriving at a very high rate.

(Soares & Borba, 2002) described the guidelines for web-based system whose major aim was to guarantee the performance an safety of the system without using the concurrency control techniques. It was found that the concurrency control provided by databases cannot provide the required level of concurrency control and hence for proper

consistency and functioning of system, there was a need to extend it using the programming level concurrency control. Author has also discussed some guidelines for improving concurrency of the system and suggested some guidelines for concurrency control techniques. Guidelines suggested the use of timestamp based techniques in which version information was added to object and object was allowed to update only if no newer version of the object was present in database. Else, the operation must be restarted.

(Laux & Laiho, 2009) described the problem of resource blocking due to database concurrency locking and recommended to use optimistic concurrency control. Authors has also discussed the problem of lost update problem and proposed new a Row Version Verifying (RVV) technique which can avoid the lost update problem and a kind of optimistic concurrency control was achieved for the DBMS which did not implicitly provide proper non-blocking concurrency control. Then author also discussed various SQL patterns which were related to lost update problem and provide scenarios for the updating databases. It was mentioned that the proposed SQL access patterns were found appropriate for all transactional applications with unreliable communication and low conflicting situations.

(Laiho & Laux, 2010) encouraged for the implement OCC at client side or middleware side because most database vendor provide only locking for concurrency control while some operations required optimistic concurrency control. Author discussed the problem with middleware caching and proposed a Row Version Verification (RVV) technique to implement an optimistic concurrency control technique at client side. Problem with the middleware was that they used their own caches to store data to avoid network load and reduce input output cost but this result in stale data. Since there was requirement of updated data for processing, it was required to update the data present in cache. To overcome this problem, Authors provided the RVV implementation using ORM Middleware and mentioned to use server side stamping for version control. They showed the problem of caching using code and updated RVV code which updates the code to avoid caching problem.

(Bai, Liu, & Hu, 2008) has compared proposed OCC algorithm with time interval based OCC algorithm. Generally, Time interval based optimistic concurrency control algorithm reduced the number of transaction restarts by dynamic adjustment of serialization order. It was found that there were some unnecessary restarts which could be reduced. Authors has proposed timestamp vector based optimistic protocol, a new optimistic concurrency control technique which can reduce more unnecessary restarts than time interval based OCC protocols.

2.4 Highly Available Databases

There are many applications where data stored in the database is of very high priority and it became very crucial for ensuring that database is available all the time. It was found that minutes of downtime generally result in loss of revenue and reputation. Highly available databases are discussed (Drake, et al., 2005), (A MySQL Strategy Whitepaper, 2013), (A MySql White Paper, 2012).

(A MySQL Strategy Whitepaper, 2013) discussed the causes of downtime and unavailability of database which are as follows:

1. System Failure: It consists of failure such as network faults, server faults, software crash, etc.

2. Physical Disaster: It include various events such as flood, fire, etc. that can cause failure of entire data centres.

3. Scheduled maintenance: It consist of downtown for maintenance works such as hardware or software upgrades, patches, etc.

4. Operator or user errors: It consists of various activities such as file deletion, malicious activities, etc.

A stable storage is the one that that will survive any failure but that one is practically impossible so we need look for approximate stable storage such as redundant copies, copies at different physical location and controlled updates in these copies. We want database to restore to most recent possible consistent state that exist before failure. All failure-masking activities (switchover, restart etc.) have to last at most single seconds

rather than minutes. Such databases are called Highly Available (HA). Highly available databases minimize the time during which the database is not available. (Drake, et al., 2005)

(A MySQL Strategy Whitepaper, 2013) discussed various reasons which created the need for highly available databases. Due to unavailable databases there is loss in revenue, customer unsatisfied and there is damage of brand image of an organization. Due to unavailability of database, customers cannot place orders; various financial tasks cannot be completed, etc.

(Drake, et al., 2005) discussed two methods of making databases highly available:

1. Process Redundancy
2. Data Redundancy.

Process redundancy in a highly available database allows the DBMS to continue operation in the presence of process failures. A process which is in the *active* state is currently providing (or is capable of providing) database service. A process which is in the *standby* state is not currently providing service but prepared to take over the active state in a rapid manner, if the current active service unit becomes faulty. This is called a *failover*. During failover usually standby processes replaces the active process and a special application will find the new active process out of all standby processes (Drake, et al., 2005).

Data redundancy is the process of creating multiple copies of the data so that the loss of a single copy of the data would render the database unavailable (Drake, et al., 2005). Data redundancy can be of two types:

1. Physical Data Redundancy
2. Logical Data Redundancy Using Replication.

Physical data redundancy refers to relying on software and/or hardware below the database to maintain multiple physical copies of the data. From the perspective of the database, there appears to be a single copy of the data. Some examples of physical data redundancy include: disk mirroring, RAID, remote disk mirroring, and replicated file systems (Drake, et al., 2005).

Logical data redundancy refers to the situation where the database explicitly maintains multiple copies of the data. Transactions applied to a primary database D are replicated to a secondary database D1 which is more or less up-to-date depending on the synchrony of the replication protocol in the HA Database. (Drake, et al., 2005)

Granularity is generally of two types: horizontal fragmentation or vertical fragmentation. Horizontal fragmentation refers to dividing tables by rows and vertical fragmentation means dividing tables by columns. Database can be replicated in two ways: fully replicated or partially replicated. In case of fully replicated database, the database exists in its entirety in each database process. In a partially replicated database the database fragments are distributed to database processes in such a way that copies of a fragment, hereafter called replicas, may reside in multiple database processes. In data replication, fragments can be classified as being primary replicas or secondary replicas. The primary replicas represent the actual data fragment and can be read as well as updated. The secondary replicas are at most read-only and are more or less up to date with the primary replica. Secondary replicas can be promoted to primary replicas during a failover.

Chapter Three: Priority Scheduling

3.1 What is priority scheduling?

In priority scheduling, a priority is assigned with each process and CPU is allocated to the process with the highest priority. Priority scheduling is very helpful in providing importance to the process or job which is more important than others. There is no limit on the value of priority but it should lie within a range and that should be properly divided into priority classes.

3.2 How priorities are defined?

(Silberschatz, Galvin, & Gagne, 2005) has discussed two ways for defining the priorities which are as follows:

1. Internal Defined priorities.

2. External defined priorities.

Internally defined priorities use some measurable quantity or quantities to compute the priority of the process. These technical quantities consist of memory usage and file/IO operations. External priorities are set by the criteria set outside the system such as importance and amount being paid for process access. For example, let us consider example of e-commerce websites which divide its user on three priority classes, namely Gold customers, Silver Customers and Bronze customers. User has to pay to join any priority class to access the website. Priority of the users is also determined based on previous history of users. A user with higher priority is given more importance and requests of high priority user are scheduled earlier than the request of lower priority user.

Here priority generated by previous history is internally defined and priority generated based on the group user joined is externally defined.

3.3 Types of Priority scheduling

3.3.1 Based on preemption of jobs

(Silberschatz, Galvin, & Gagne, 2005) discussed two types of priority scheduling:

1. Preemptive priority scheduling.
2. Non- preemptive priority scheduling.

When a process arrives at the ready queue, its priority is compared with the priority of the currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently arrived process. A non preemptive priority scheduling algorithm will simply put the new arrived process at the head of the ready queue.

3.3.2 Based on priority evaluation time of job

Based on the priority evaluation time, priority scheduling can be of two types:

1. Fixed priority scheduling.
2. Dynamic priority scheduling.

Fixed priority job scheduling: Fixed priority scheduling process the jobs based on given priorities which are fixed before the scheduling. Priorities are assigned to jobs once and for all. Fixed priority scheduling is also known as static priority scheduling.

Dynamic priority job scheduling: Dynamic priority scheduling process the request based on given priorities where priories of the jobs are evaluated at run time. Here priority of job changes based on various factors such as total waiting of the job, deadline of the tasks etc.

3.4 Priority scheduling with example

Let us consider a set following processes which are assumed to arrive at time 0, in the order P1, P2, P3, P4, and P5 with the length of CPU burst given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Table 1: Data table for processes (Silberschatz, Galvin, & Gagne, 2005)

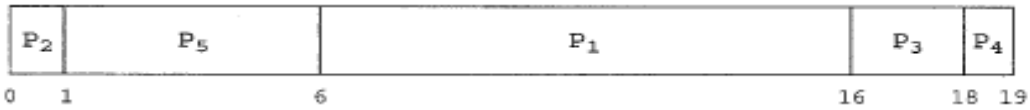

Figure 1 : Gantt chart (Silberschatz, Galvin, & Gagne, 2005)

This figure shows that process P2 is scheduled first with no waiting and after the execution of P2; P5 was selected for execution with waiting time of 1 millisecond. After that P1 was selected for execution with waiting time 6 milliseconds, after that P3 was selected with waiting time 16 millisecond and lastly P4 was selected with waiting time of 18 milliseconds.

Average waiting time: Waiting time of a job is time job spend in the queue before being selected for processing and average of waiting time of all the jobs present in system is average waiting time.

$$\text{Average waiting time} = \frac{(0 + 1 + 6 + 16 + 18)}{5}$$

$$= 8.2 \text{ milliseconds}$$

Average waiting time for the given set of processes is 8.2 milliseconds.

3.5 Special Cases of Priority scheduling

(Silberschatz, Galvin, & Gagne, 2005) has discussed two special cases of priority scheduling which are as follows:

1. First Come First Serve (FCFS) Scheduling.

2. Shortest Job First (SJF) Scheduling.

In FCFS scheduling, jobs are executed in the arrival order of the request. If priority associated with each process is same, then processes are scheduled in FCFS manner.

Process	Priority
P1	10
P2	5
P3	1

Table 2 : Data Table for FCFS

For example, let us consider a system with three processes P1, P2 and P3. Processes arrived in the system in order P1, P2, P3. If the jobs which arrive earlier has higher priority than the job which arrived later then Processes are executed in order of arrival i.e. FCFS scheduling order. Using priority scheduling, order of execution of processes is P1, P2, P3 which is same as execution of FCFS scheduling.

In SJF scheduling, jobs are execute the jobs in the order of the execution time of the jobs. Shortest job is processed first and then process with next smallest execution time is processed and so on. Shortest job first is like priority scheduling where the priority depends on the execution time. Smaller the execution time, higher is the priority.

Process	Execution Time (in milliseconds)	Priority
P1	10	5
P2	5	10
P3	12	1

Table 3 : Data Table for SJF

For example, let us consider a system with three processes P1, P2 and P3. Processes arrived in the system in order P1, P2, P3. Priority of the jobs are assigned such that job with smallest execution time has highest priority. Using priority scheduling, order of execution of processes is P2, P1, P3 which is same as the order of execution of jobs with SJF scheduling.

3.6 Problem with priority scheduling

Problem with priority scheduling are as following:

1. Indefinite blocking or starvation

In case of priority scheduling, jobs are executed in the order of priority assigned to the jobs. In a system where jobs are continuous arrived and jobs are scheduled using priority scheduling, there is possibility of starvation of low priority jobs. This happens due to the reason that in priority scheduling low priority jobs have to keep waiting in the queue while higher priority job are executing and if more jobs with higher priority also arrived in the system, then the waiting time of the low priority jobs will increased and there are changes that these low priority jobs may get starved.

2. Higher Average waiting time

In case of priority scheduling, lower priority jobs have to wait longer which increase the average waiting time of the jobs.

Solutions for problems in priority scheduling are as follows:

1. (Tanenbaum & Woodhull, 2006) has discussed the solution to prevent starvation of low priority process. It was mentioned to decrease the priority of the currently running process at each clock tick. If priority of currently executing job drop below that of next highest priority process, a process switch occurs and this changes the executing process to the currently highest priority process.

2. (Tanenbaum & Woodhull, 2006) has discussed another solution which is based on assigning a maximum time quantum to every process. This assigned time quantum is maximum time a process is allowed to run. When this quantum time get finished, the next highest priority process is scheduled to execute.

3. (Silberschatz, Galvin, & Gagne, 2005) discussed the solution to starvation problem of low priority processes by using is aging. Aging technique is use in scheduling to increase the priority of the low priority process. This is done through adding some number to the priority of the process. So, as the processes get older in waiting queue, their priority gets increased and hence the starvation of low priority jobs can be prevented.

3.7 Algorithms based on priority scheduling

Priority scheduling is widely used in many applications since it provide a way to provide special attention to important jobs, task and users. Based on the requirements, many variations are done on priority scheduling for achieving some particular targets. Some the algorithm based on priority scheduling are as follows:

1. Longest Processing Time (LPT) Scheduling

LPT scheduling is based on assigning the priority based on the execution time of the job such that job with largest execution time are executed first and then the task with next largest execution time are executed and so on. In case of tie-break an arbitrary rule can be used. This was found that LPT algorithm was not minimal but the computation time spent on LPT was much smaller than other optimal scheduling algorithms. This algorithm was discussed in (Hong, Huang, & Yu, 1995).

2. Earliest Deadline First (EDF) Scheduling

EDF is a dynamic deadline scheduler. It is dynamic because the priority of the task keeps changing throughout the execution cycle. It is called Deadline based because the scheduler considers the deadline of a task before deciding whether it needs to be scheduled. The task which is going to meet its deadline first is scheduler first. (Mattihalli, 2010) discussed that EDF tasks needs three parameters to define the required scheduling behaviour of a task. They include:

1. Cycle time which defines total period of the task.
2. Deadline which defines the deadline of the task with in the period.
3. Processing time which defines the total computation time of the task for completing the assigned activity in the given cycle and deadline.

3. The Least-Laxity-First (LLF) Scheduling

LDF is a scheduling algorithm that assigns a higher priority to a job which has the least laxity. The laxity is the time left by its deadline after the task is completed assuming that the task could be executed immediately without preemption. A task which has zero laxity can be scheduled right away and must be processed with no preemption in order to meet the deadline. Tasks which have negative laxity show that the task will miss the deadline (Oh & Yang, A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks, 1998).

4. Modified Least-Laxity-First (MLLF) Scheduling

Modified Least-Laxity-First (MLLF) scheduling solved the problem of LLF scheduling by significantly reducing the number of context switch which reduced the system overhead and resulted in better performance. Proposed algorithm worked like simple LLF scheduling as long as there is laxity-tie. In case of Laxity-tie, the executing task was allowed to execute without pre-emption till the deadline of other jobs are not missed. This algorithm was discussed by (Oh & Yang, A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks, 1998).

5. Multi-Objective Dynamic Priority (MODP) Scheduling

MODP optimization scheduling algorithm was based on reward-driven multi-dimension criterion for request classification and one-step-ahead overload estimation. Their goal was to increase the completion rate of the valuable requests. Their results showed that decrease in average response time and increase in the completion rate of transaction request under overload condition when they apply MODP strategy. This algorithm was proposed by (Chen, Gao, & Wang, 2007) for the systems which were designed for better revenue generation. They mentioned the criticality to increase the completion rate of transaction related requests and requests from VIP.

6. Fixed Task Priority (FTP) Scheduling

(Baruah & Fisher, 2008) described Fixed Task Priority as "Fixed Task Priority (FTP) scheduling algorithms are a subclass of the class of priority-driven algorithms for scheduling systems of recurring tasks, in which it is required that there is a unique priority associated with each task, and all the jobs generated by the task are assigned this priority". In FTP scheduling algorithms, all the jobs have the same priority which is generated by each recurrent task. Example of FTP scheduling algorithm include Deadline Monotonic (DM) scheduling algorithm which assigned priority to jobs on the basis deadline such that deadline is relatively small than assign high priority.

7. Nonpreemptive Priority (NPRP) Scheduling

Nonpreemptive Priority (NPRP) based job scheduling algorithm is a priority scheduling which is combination of backfill algorithm and Earliest Deadline based scheduling algorithm. In case of tie-breaking condition, FCFS scheduling is used. This algorithm was designed to reduce the turnaround time, waiting time and response time of the jobs present in the queue. In the priority algorithm, only non-preemption policy was considered because in preemptive scheduling, higher priority job pre-empt lower priority job. This algorithm was proposed by (Kannan & Selvi, 2010) and it was found that

turnaround time, waiting time and response time was minimized when compared with FCFS and SJF.

8. Dynamic priority scheduling algorithm (DPSA) Scheduling

Dynamic priority scheduling algorithm (DPSA) is based on the static priority scheduling algorithm where priority are assigned to job at run time. The purpose of the introduction of dynamic way was to avoid a task unit with low priority wait so long. The proposed technique has several queues based on the number of different priority of the task. Every arrived task was inserted into the queue based their pre-assigned priority. Proposed technique also used the concept of aging by shifting of low priority request from low priority queue to high priority queue. It seemed to solve the starvation problem but they have not discussed the condition when a newly arrived request may have high priority and it will go to the highest priority queue which may be has very large size or large execution time which may starve or increase waiting time of other job request in the queue which need to be process first. This technique was proposed by (Lee, Wang, & Zhou, 2011).

Chapter Four: Proposed System

Busy servers start rejecting requests under heavy overload which may consists of jobs which are very beneficial for the profit of the service provider. Target of any system is to first execute those jobs which are more important and profitable meanwhile taking care of other jobs so that no job gets starved. To handle performance under overload, we have proposed a method that will store all the jobs in database and execute jobs in priority order. A high overview of proposed technique includes:

1. Storing all the incoming user requests in the database along with various important job parameters such as arrival time, job priority, job size etc. By querying database, we can select the highest priority job and forward it to job executor for processing.

2. A dynamic priority scheduling which calculated priority dynamically based on weighted component of priority of job, size of job, and total waiting time of the job. Motive of proposed scheduling algorithm is avoiding starvation of jobs and increase in the profit for service provider.

Proposed system architecture

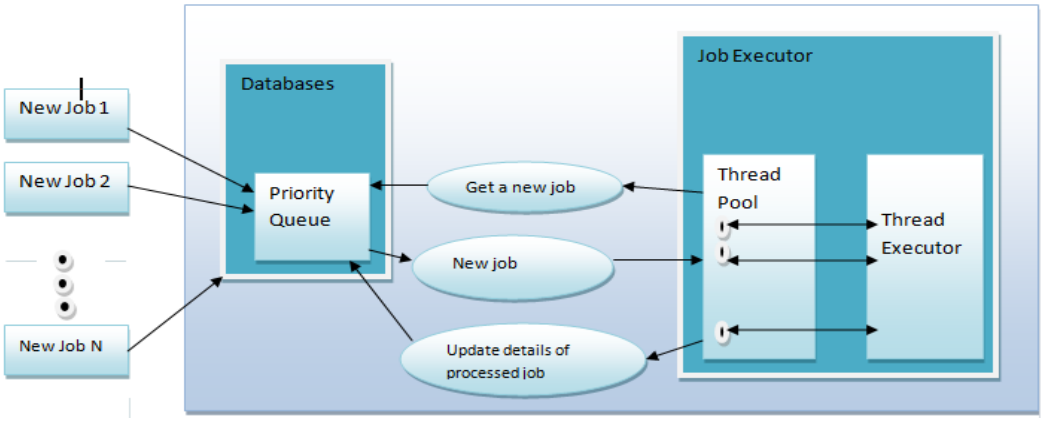

Figure 2: Proposed System Architect

Proposed System consists of following:

1. Queue Database Table
2. Job Executor
3. Set of jobs

Proposed system consists of databases which will act as a queue for storing jobs and a job executor which will actually execute the job to produce the desired results. Users will use the application and schedule the job. Job details are inserted into queue table. Job executor consists of thread pool and thread executor. Thread pool consists of one or more threads based on the capacity of the system. These threads will pick the highest priority job from the database and forward the selected job to thread executor which will process the job. Thread executor can perform functions based on the nature and type of the job. One example of thread executor is file generator which will generate the file of desired extension form content provided by the user.

Thread from the thread pool will poll the database and check if there is any unprocessed job present which need to be processed and if there is no such job then thread will sleep for some predefined time else if there exist unprocessed job that need to be processed then thread will select the currently highest priority job and forward it to the thread executor for processing of job. When processing of job gets completed, various details of the job such that completion time and job status are updated in database queue table.

4.1 Selection of job scheduling algorithm

Various scheduling algorithm exists like FCFS, SJF, Priority scheduling, etc. which are used for scheduling jobs. FCFS is the fair algorithm and it executes jobs in the order of arrival in the system. If all jobs are equally important then request arrival sequence will be the best criteria for priority. FCFS assign same priority to the all the tasks which ensure fairness but ignore priority assigned to the job.

But in our case, jobs arrive in system with some predefined priority such that high priority user's job is considered more important than low priority user's job.

There is a possibility that low priority jobs arrive early in system than high priority jobs and are processed early based on arrival time execution. So, FCFS cannot provide the highest priority job hence we have to look for other scheduling algorithms.

In Priority Scheduling, a priority number is assign to each task and task with highest (or lowest) priority are chosen for processing. Priority scheduling seems to fulfil our requirements since it seems to select the current highest priority job for processing. But it was found that priority scheduling has following problems:

1. Starvation of low priority jobs.

2. Higher average waiting time and response time.

So, simple priority scheduling may not provide optimal solution in complex conditions and hence we need to check other scheduling algorithms for better response time.

In SJF, Jobs are executed on the basis of execution time where jobs with smallest execution time are processed first and then the job with next smallest execution time and so on. Shortest job first has better response time than FCFS and priority scheduling. Although SJF improves the response time but still there are problem to select highest priority job and starvation of jobs with higher execution time. SJF alone cannot provide the required solution. Problems of priority scheduling can be minimized by using combination of existing algorithms such that problem of starvation and response time can be minimized. Problem of starvation can be minimized by using fair scheduling algorithm such as FCFS. Problem of response time and waiting time can be minimized by using SJF.

4.2 Preemption policy

Pre-emption policy determines if and when a higher-priority job can pre-empt a currently executing lower-priority job. There are two types of pre-emption policies: 1 Non-pre-emption policy and fully pre-emption policy (Lee & Shin, 2012). Fully preemption policy: In this policy, a higher priority job interrupts the execution of the currently processing job and starts its execution immediately provided there are no other

higher priority job present in the queue. Pre-empted job resumes its processing soon after the higher priority job is finished (Younas & Awan, 2003).

No pre-emption policy: No pre-emption policy disallows the preemption of the lower priority job by higher priority job. The job which is chosen for processing will be processed fully and its processing will not be interrupted by higher priority jobs.

Some drawbacks of pre-emption are:

1. Preemption of a job may lead to starvation of low priority jobs.

2. Delay incur due to preemption overhead of executing job (Bansal & Harchol-Balter, 2001), (Lee & Shin, 2012).

First problem is due to the fact that lower priority job will be preempted by the higher priority job and this may occur over large period of time resulting in starvation of lower priority job. Second problem is due to the context switch time required during the pre-emption of the job.

This overhead may even become more significant in case of databases since it may result in many unnecessary updates in the databases which are quite costly. So, we choose not to preempt the job.

4.3 Database implementation of scheduling algorithm

All the incoming request are inserted into the queue Database table. The order of processing of request is determined by the type of priority use:

4.3.1 FCFS

FCFS order can be achieved into the database by using the job arrival time in database. Unprocessed jobs are selected for processing in the order of arrival order.

Query = “select * from Queue where status=’New’ order by arrival_time”;

4.3.2 SJF

SJF order can be achieved into the database by using the size of the jobs. Unprocessed jobs are selected for processing in the order of job size.

Query=“Select * from Queue where job_size = (select min(job_size) from Queue where status=’New’) and status=’New’ order by arrival_time)”;

4.3.3 Priority Scheduling

Priority scheduling can be achieved in the database by using the priority of the jobs. Unprocessed jobs are selected for processing in the order of given priority of the job.

Query=”select * from Queue where priority = (”Select max (priority) from Queue where status=’New’) and status=’New’ order by arrival_time”;

4.4 Concurrency Control Technique in Database

Since our server is using more than one thread to process the request, problem related to concurrency occur if we do not use concurrency control mechanism. It was found that some of the tasks were processed more than once by different server processor threads. This leads to redundancy of work done at the cost system resources and waiting of other important users.

Concurrency problem can also lead lost update problem.

To handle concurrency control problem in database, we have following options:

1. Pessimistic concurrency control
2. Optimistic concurrency control

4.4.1 ***Pessimistic concurrency control***

It places the locks as one or more rows of database are accessed, so that two or users will not update the same row simultaneously. It assumes that same row will be updated by two or users simultaneously and by placing locks on row ensure that no conflict will occur.

Problem with Pessimistic concurrency control are follows:

1. Deadlock.
2. Starvation
3. Resource blocking due to locking
4. Overhead due to use of locking at every operation.

4.4.2 Optimistic concurrency control

Major benefits of optimistic concurrency control are follows:

1. Resources used to hold the locks are generally fewer during the update process.

2. Records are locked for a comparatively shorter period of time.

3. There is not blockage of resources hence records are generally available for other users to update.

4. It allows high concurrency.

5. Operations are generally faster as there is no overhead for locking for every operation.

Major disadvantage of optimistic concurrency control is:

1. Transaction needs to restart in case of conflict.

Based in the above analysis and our requirements which require high concurrency and no resource blockage, we choose optimistic concurrency control as concurrency control technique.

4.5 SQL patterns for concurrency control

There exist many SQL pattern to query database for selecting job for processing.

4.5.1 SQL pattern 1

Read Phase:

Select * From Queue where status=’New’ order by arrival_time

Write Phase:

Update Queue set status =’Processing’ , dispatch_time=’time’ where task_id=id

Problem with above SQL pattern is lost update problem.

This problem can be handles using locking based concurrency control as well as optimistic concurrency control. But we have decided to use optimistic concurrency control. So, we will concentrate on OCC technique to handle lost update problem.

4.5.2 SQL Pattern 2

Following the SQL pattern that use optimistic concurrency control using SQL statements.

Read Phase:

Select * From Queue where status=’New’ order by arrival_time

Validation and Write:

Update Queue set status =’Processing’ , dispatch_time = ’time’ where status = ’New’ and task_id = id

Read phase of SQL pattern 1 and SQL pattern 2 are both same but SQL pattern 2 make use OCC validation for checking the conflict in databases. At the validation stage update query is use which will first read the status of the selected job and if job status is not equal to ‘New’ which means some other user has already selected this job and forward this job to thread executor for processing. Hence there is conflict in this case. In this case, status and dispatch time will not be updated for this request and lost update situation is avoided.

4.6 Proposed job scheduling algorithm

As we have discussed earlier that we require a scheduling algorithm that have following characteristics:

1. It can schedule jobs based on the given priority of the jobs such that jobs with highest priority are processed first, then the job with next highest priority and so on.

2. Average response time of the jobs should be as small as possible.

3. No job should be starved.

Characteristic 1 can be achieved using priority scheduling. Characteristic 2 can be achieved using SJF and Characteristic 3 can be used using the concept of aging. Now, for handling these conditions we need to use a dynamic priority scheduling algorithm.

4.6.1 Proposed Dynamic Priority Scheduling algorithm

In Dynamic priority scheduling, priorities of task are calculated and assigned at the execution time. Dynamic priority scheduling is required to dynamically configure execution to current system conditions and take optimal steps to execute task.

For handling the above three Characteristic we need calculate priority at execution time which is weighted combination of the following:

1. Fixed priority of the job.
2. Size of the job.
3. Total waiting time in queue.

So, Dynamic priority can be calculated as weighted component of above three factors:

Dynamic Priority of job, DP = $\frac{a * fixed\ priority + c * total\ waiting\ time\ of\ job}{b * size\ of\ job}$

$$DP = \frac{a * FP + c * (CT - AT)}{b * JS}$$

Where FP = Fixed Priority

CT: Current System Time

AT: Arrival Time of job

JS: Job Size

a, b, c : Weighted factor for adjustment

4.6.2 SQL implementation of proposed algorithm

Select * from Queue where status='New' and ((a*priority + c * (current_time – arrival_time)) / b * job_size) = ((a*priority + c * (current_time – arrival_time)) / b * job_size) from Queue where status='New') order by arrival_time

Above algorithm can act as pure FCFS, Priority scheduling, SJF scheduling if required by setting the values of weighted factors a, b and c as follows:

A	B	C	Scheduling Algorithm
0	0	1	FCFS
0	1	0	SJF
1	0	0	Priority Scheduling

Table 4: Proposed Scheduling Algorithm Behaviour for different value of parameters values of a, b, c can be adjusted for maximizing profit.

4.7 Algorithm for job execution

User will access the application and send request. User request is inserted as jobs into the database. After that following steps are executed for the processing of jobs:

1. When a job executor thread is free

It polls database and see whether there is any request/job to process.

1.1 If there are jobs with status='New' in database, selects the dynamically generated highest priority job.

1.1.1 It checks the status of selected job again.

If the status of selected job = 'New'

1.1.1.1 Update status='Processing' and dispatch=current system time.

1.1.1.2. Send job to job executor for execution.

1.1.1.3. Job Executor execute the job

1.1.1.4. On job completion

1.1.1.4.1. Set job status='Finish'

1.1.1.4.2. Update completion time of finished job;

1.1.1.4.3. Goto step 2.

Else If status! ='New'

1.1.2.1 Restart the job.

1.1.2.2 Goto step 2.

1.2. If there are no jobs in database with status='New', Thread sleeps or do some other task for some predefined time.

After the user request are store as jobs in database, threads of thread pool present in job executor will select the job from queue table and send it to thread executor for processing. It consists of various other functions.

If any thread of thread pool is free then will poll the database to check whether there are job with status equal to 'New' exist in queue table or not. If no records found with status equal to 'New', thread will sleep for particular some particular period of time otherwise it selects the job with dynamically calculated highest priority unprocessed job. This consists of read phase of optimistic concurrency control algorithm.

Now status of the job is checked again and if status of job not equal to 'New' then current thread rolls back and new transaction is stated again. This is the transaction restart in optimistic concurrency control method and here validation fails so no write operation is performed. But if the status of the job is still 'New' then it updates the status of job to 'Processing' and dispatch time to current system time. This completes the validation and write phase of the optimistic concurrency control algorithm. Now job is sent to thread executor for processing of the job. Meanwhile other thread will concurrently follow the same steps to select a job for processing.

When the processing of the job is completed, then job status is updated to 'Finish' and completion time is updated to the job completion time. Now thread is free and it will poll the database again for a new job. This process will continue till there will no job with status equals to 'New'. Figure 3 shows the flowchart for proposed scheduling algorithm processing.

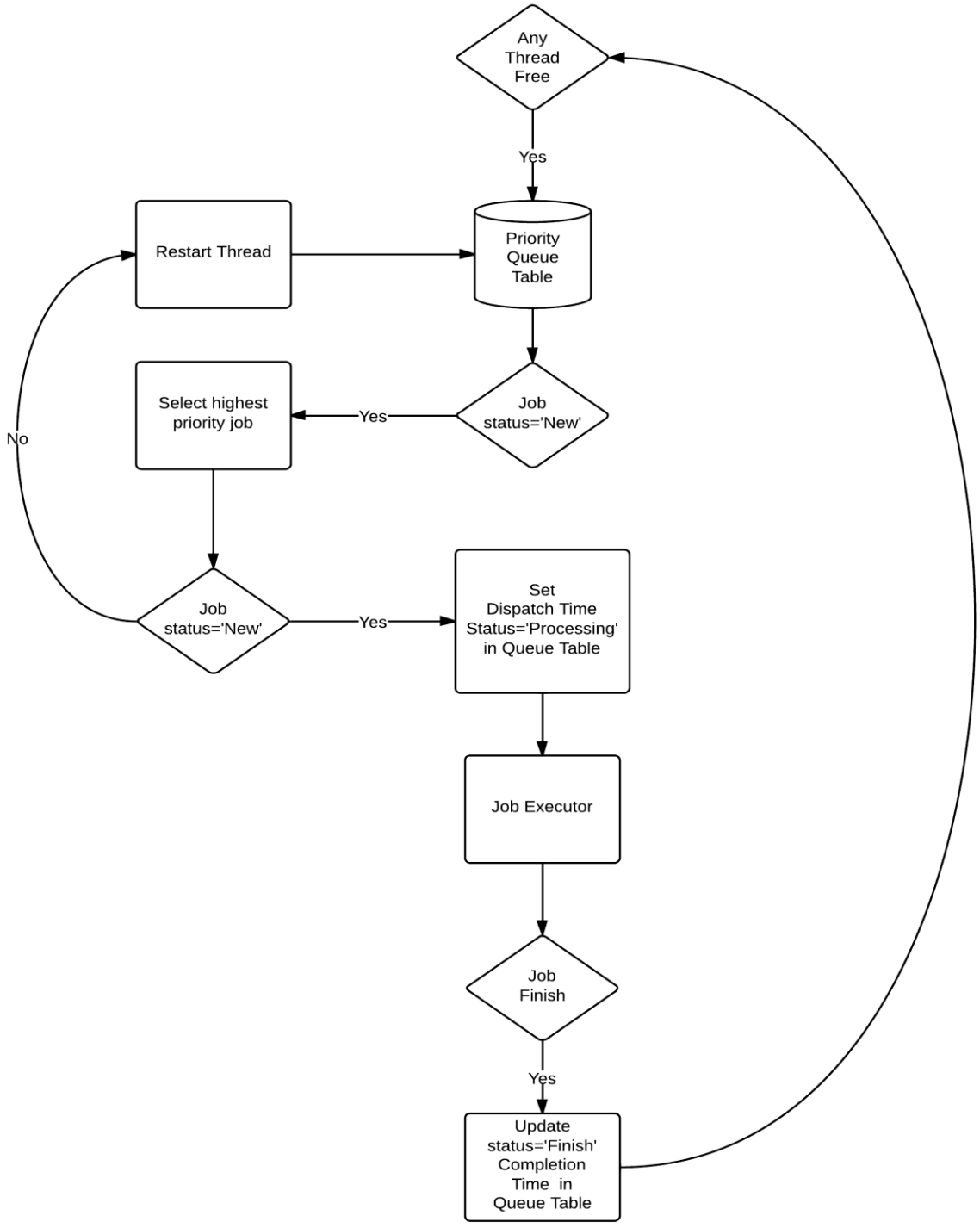

Figure 3: Proposed scheduling Algorithm

4.8 Summary

In this chapter, we have identified the weakness of existing scheduling algorithm and based on requirements, we have design and implement a scheduling algorithm for selecting job from database, such that jobs are selected on the basis of dynamically evaluated priority of the job which is weighted combination of fixed priority of the job, size of the job, and total waiting time of the job.

Chapter Five: Result and Discussion

5.1 Results

Here in this chapter, we have compared the proposed scheduling algorithm with first come first serve, shortest job first and pure priority scheduling algorithm on various performance factors.

5.1.1 Various Result Terminologies

Scheduling consists of various performance criteria which are as follows:

1. Total Execution time
2. Average Response time
3. Average waiting Time
4. Weighed efficiency based on dispatch time
5. Weighed efficiency based on completion time
6. Longest time waiting job

1. Total execution time

It is the total time taken by the scheduler to complete the processing of all the jobs present in the system. Mathematically, it can be define as the difference of time of completion of the last job and time of start of execution of first job.

$$Total\ Execution\ Time = Completion\ time\ of\ \ of\ last\ job - Staring\ time\ of\ execution\ of\ first\ job$$

2. Response Time

Response time is the time taken to start responding to a given user request.

Mathematically, it is the difference between time of the submission of the job and time when first response is received.

$$Response\ Time = Time\ when\ user\ get\ the\ first\ response\ for\ requst - Time\ when\ request\ was\ made$$

3. Waiting Time

Waiting time of a job is time job spend in the queue before being selected for processing.

$$Waiting\ Time = Time\ when\ job\ was\ selected\ for\ processing - Time\ when\ job\ was\ submitted for\ processing$$

4. Average Waiting Time

Average of waiting time of the all the jobs present in system.

$$Average\ Waiting\ Time = \frac{Sum\ of\ waiting\ time\ of\ all\ the\ jobs}{Total\ number\ of\ jobs}$$

Or, $$Average\ Waiting\ Time\ for\ n\ jobs = \frac{\sum_{i=1}^{i=n}(Waiting\ time\ of\ jobi)}{n}$$

5. Average Response Time

Average of response time of the all the jobs present in system.

$$Average\ Response\ Time\ for\ n\ jobs = \frac{\sum_{i=1}^{i=n}(Response\ Time\ of\ jobi)}{n}$$

6. Weighed efficiency based on dispatch time:

Weighted efficiency based on dispatch time is efficiency in terms of profit on the basis of dispatch time of job. Higher the dispatch time, lower is the efficiency.

$$Weighted\ efficiency\ based\ on\ dispatch\ time\ for\ n\ jobs = \sum_{i=0}^{i=n} \frac{Priority\ of\ jobi}{Dispatch\ Time\ of\ jobi - Arrival\ time\ of\ jobi}$$

7. Weighed efficiency based on completion time:

Weighed efficiency based on completion time is the efficiency in terms of profit on the basis of completion time of job.

$$Weighted\ efficiency\ based\ on\ completion\ time\ for\ n\ jobs = \sum_{i=1}^{i=n} \frac{Priority\ of\ jobi}{Completion\ time\ of\ jobi - Arrival\ Time\ of\ jobi}$$

8. Longest time waiting job

It the longest time job spent waiting in the queue.

Longest time Waiting job = MAXIMUM (Waiting Time of all the jobs)

5.2 Result Analysis

We have carried out simulation using different load and have divide the load into three divisions: Minimum load, moderate load and maximum load. Minimum load contains jobs in the range 0 to 264; Moderate load contains the jobs in the range 265 to 528 and maximum load contains the jobs in the range 529 to 792. In our case, user gets response as soon as their request is selected so waiting time of job is same as response time.

5.2.1 Result Analysis under Minimum load

Under minimum load, the job arrives in the system at a rate lower than the job processing rate of the system.

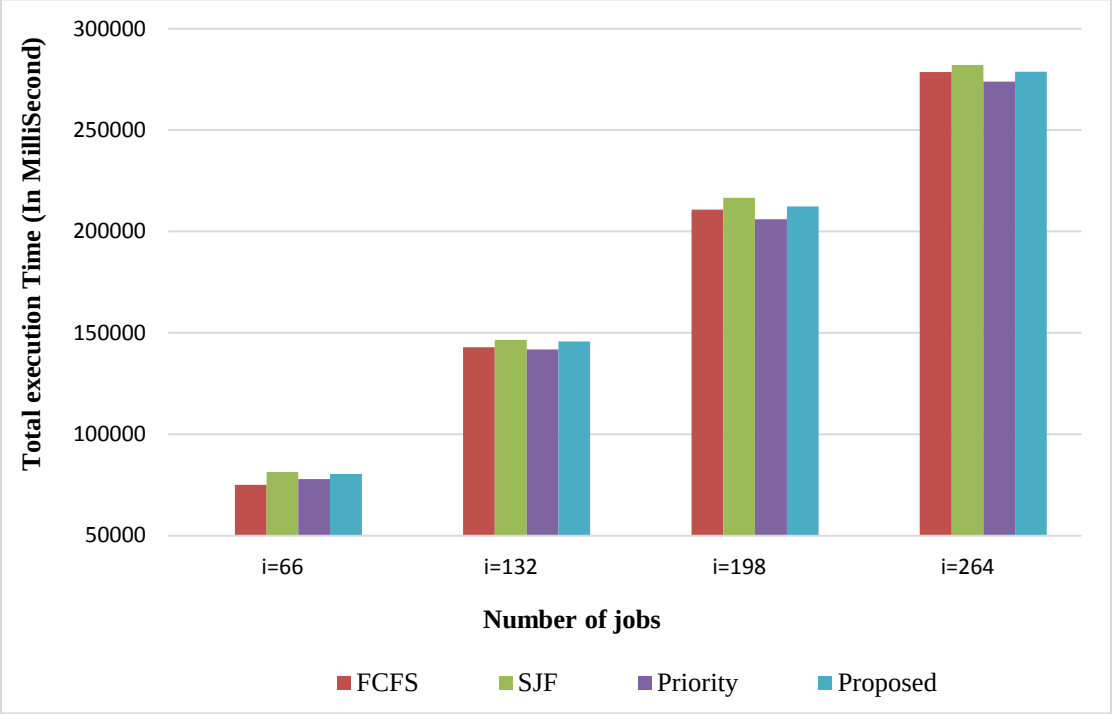

Figure 4 : Performance analysis on the basis of total execution time under minimum load

Figure 4 shows the performance analysis on the basis of total execution time under minimum load. Here, the vertical axis represents time in milliseconds taken to execute the given set of jobs, and the horizontal axis represents number of jobs to be executed. The number of jobs is in the range 0 to 264 which represent load on the system. Figure 4 shows that under minimum load pure priority scheduling seems to take lowest time of execution and all other algorithm take almost same time for execution. The difference in total time of execution between pure priority scheduling and other algorithms also seems to be very minor. This pattern is remains valid under the range of minimum load condition.

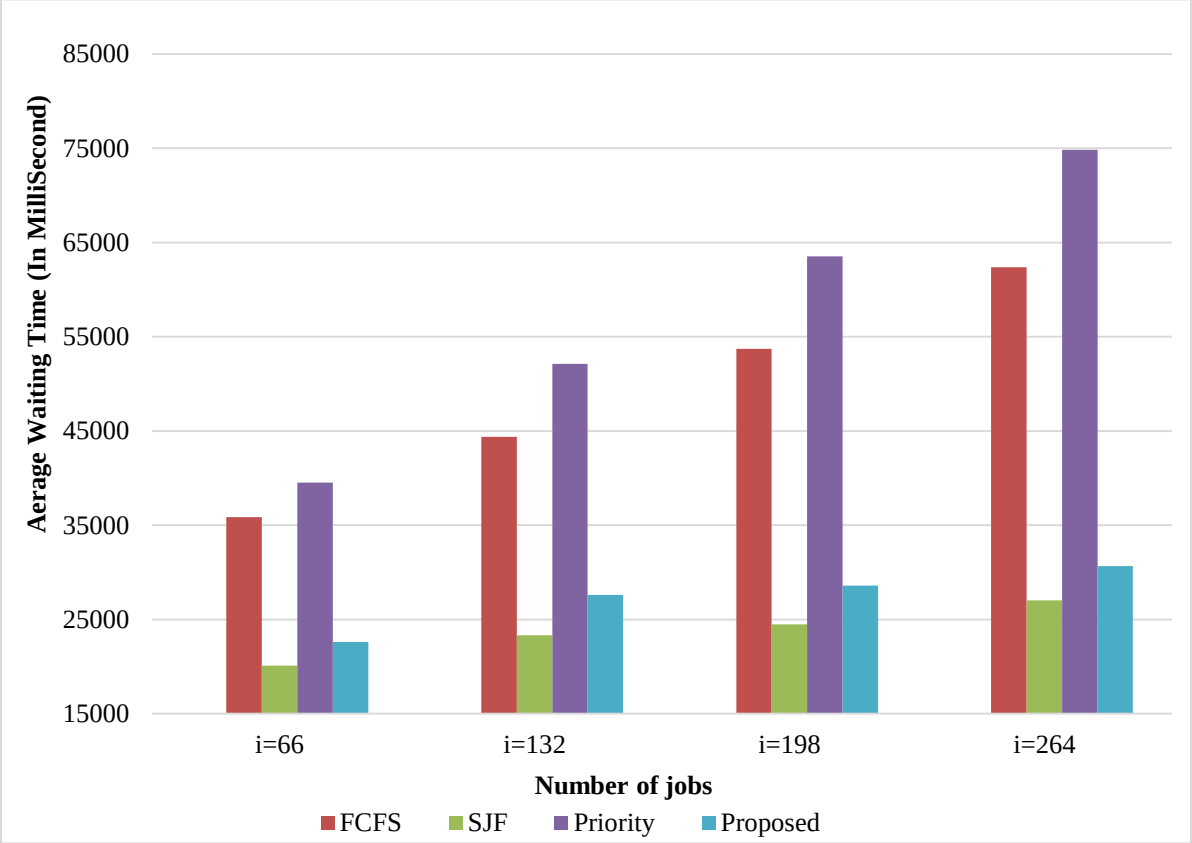

Figure 5: Performance analysis on the basis of average waiting time under minimum load

Figure 5 shows the performance analysis on the basis of average waiting time under minimum load. Here, the vertical axis represents average waiting time in milliseconds of the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 0 to 264 which represent load on the system. Figure 5 shows that under minimum load, SJF algorithm has the lowest average waiting time as expected and proposed scheduling algorithm has lower average waiting time than FCFS and pure priority scheduling algorithm. As the load is increased, there is comparatively less increase in the average waiting time in case of SJF and proposed scheduling algorithm. As the load increases, there is a gradual increase in the waiting time of FCFS and pure priority scheduling. Pure priority scheduling seems to take highest average waiting time among all other scheduling algorithm.

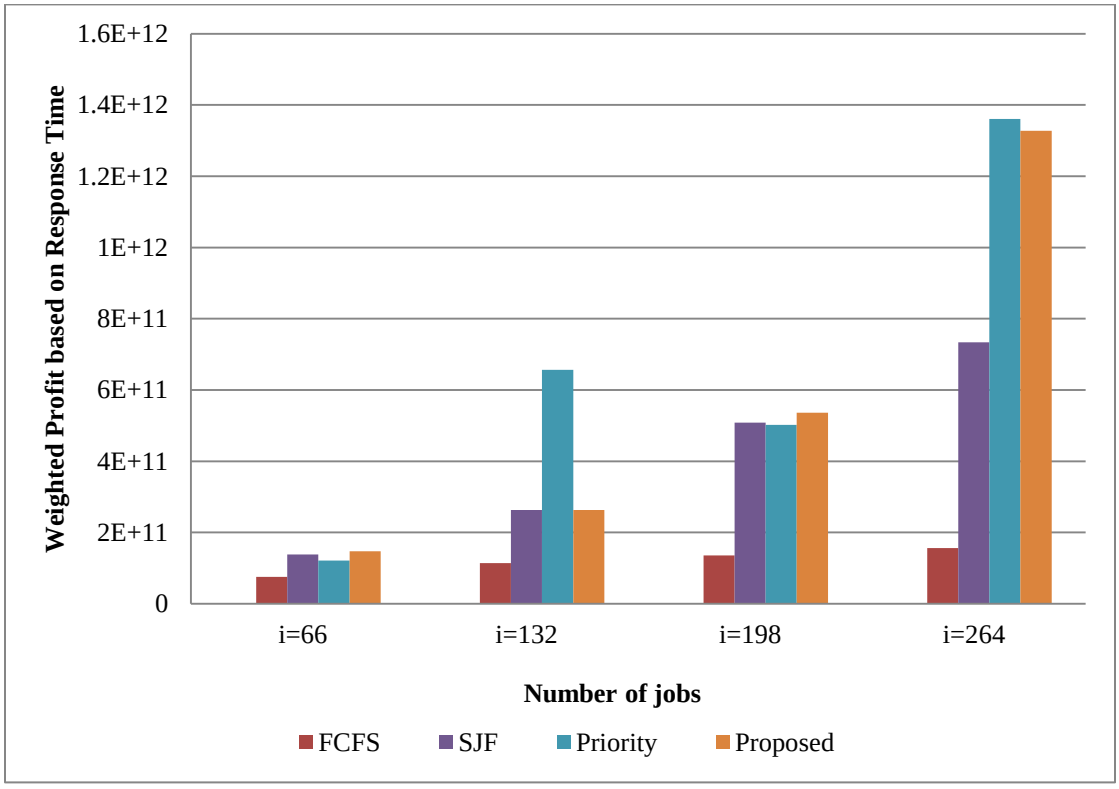

Figure 6: Performance analysis on the basis of weighted profit based on response time under minimum load

Figure 6 shows the performance analysis on the basis of weighted profit based on response time under minimum load. Here, the vertical axis represents the weighted profit based on response time while executing the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 0 to 264 which represent load on the system. Figure 6 shows that under minimum load except FCFS, all other compared algorithm has reached highest profit at different load. But as load increases both pure priority and proposed algorithm seems to be efficient. Pure priority scheduling seems to be more efficient since it handles high priority jobs first and hence its profit based on response time seems to be better. When the load is 264 jobs, the pure priority scheduling generate the maximum weighted profit based on response time.

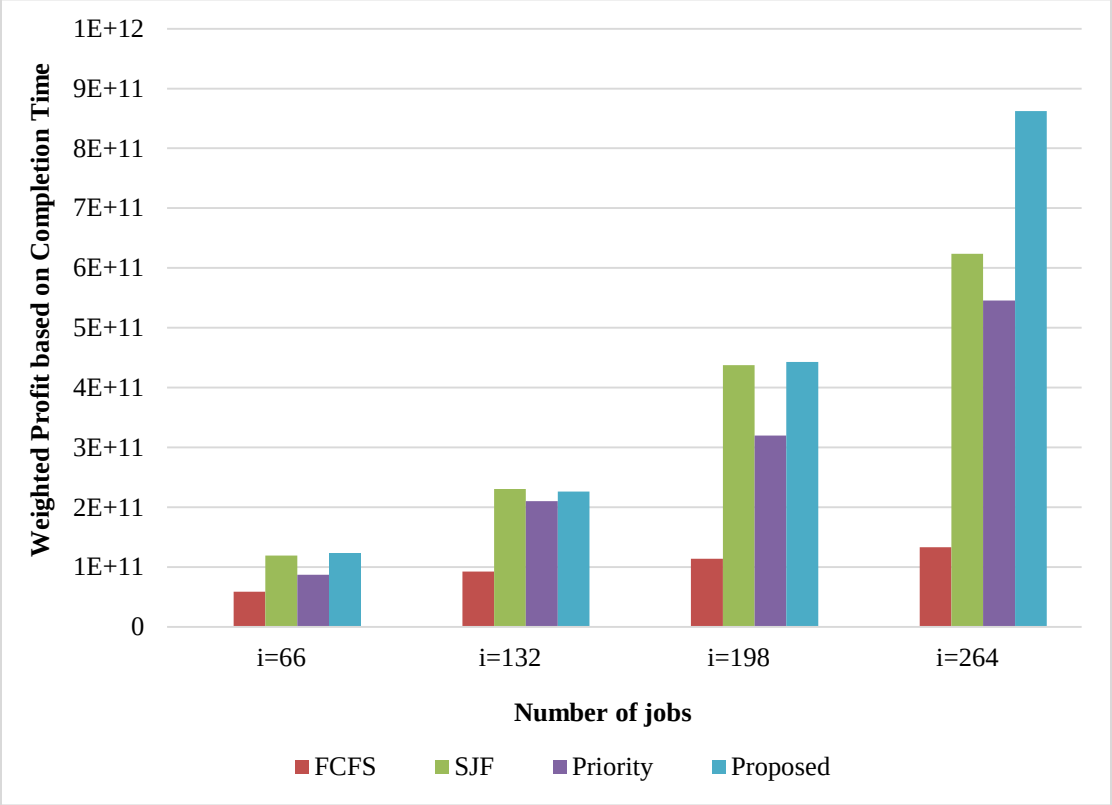

Figure 7: Performance analysis on the basis of weighted profit based on completion time under minimum load

Figure 7 shows the performance analysis on the basis of weighted profit based on completion time under minimum load. Here, the vertical axis represents the weighted profit based on completion time while executing the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 0 to 264 which represent load on the system. Figure 7 shows that under minimum load that proposed scheduling algorithm and shortest job first scheduling has almost same profit weighted profit on the basis of completion. As shown in figure 7, as load increase, weighted profit for proposed job scheduling algorithm was more than shortest job first scheduling algorithm. When load reaches 264 jobs, proposed scheduling algorithm has highest profit based on completion time.

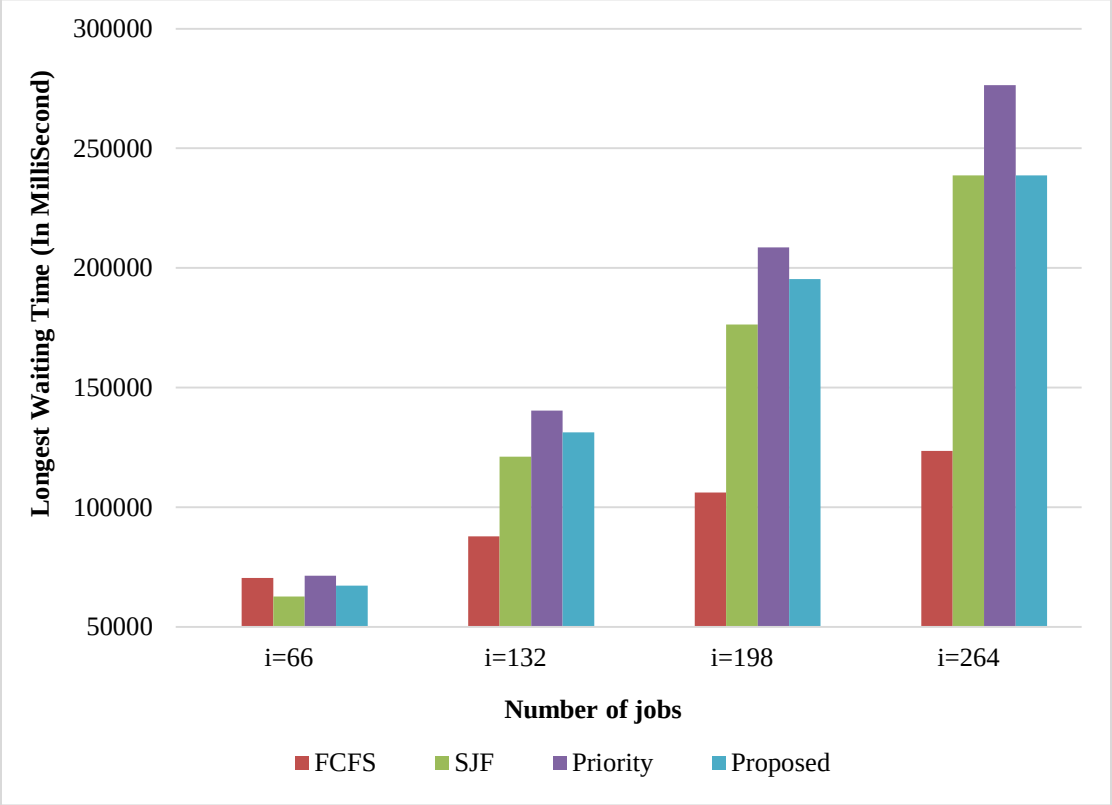

Figure 8: Performance analysis on the basis of longest waiting time under minimum load

Figure 8 shows the performance analysis on the basis of longest waiting time job under minimum load. Here, the vertical axis represents the longest waiting time of the job, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 0 to 264 which represent load on the system. Figure 8 shows that under minimum load, FCFS scheduling has lowest waiting time job since FCFS is fair algorithm whereas pure priority scheduling has the longest waiting time of the job. As the load increases, job in the case pure priority scheduling, shortest job first and proposed scheduling algorithm seems to wait longest since these are unfair scheduling algorithm. Figure 8 shows that longest waiting time for job using pure priority scheduling is increasing at the highest rate as compared to other scheduling algorithms.

5.2.2 Result Analysis at Moderate load

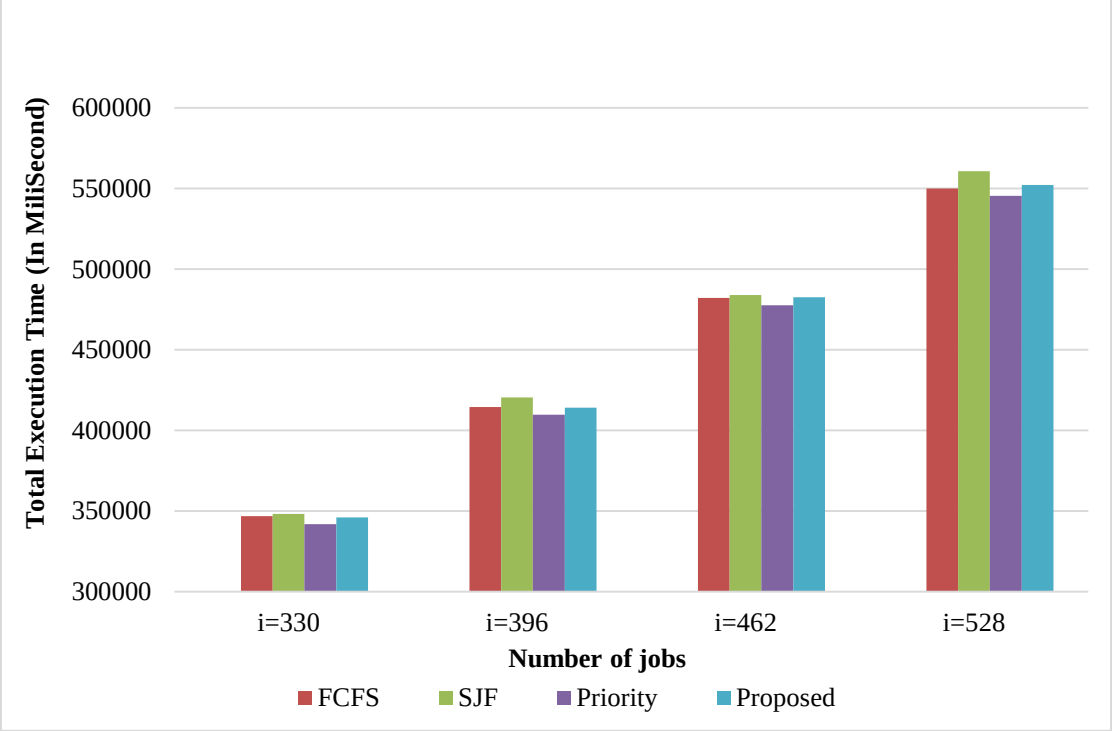

Figure 9: Performance analysis on the basis of average waiting time under moderate load

Figure 9 shows the performance analysis on the basis of total execution time taken to process the given set of jobs under moderate load. Here, the vertical axis represents the total execution time taken to process the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 265 to 528 which represent load on the system. Figure 9 show that under moderate load all the scheduling algorithms have almost same total execution time. Pure priority scheduling seems to take least total execution time whereas shortest job first seems to take longest total time of execution but difference in total execution time all scheduling algorithm seems to be negligible.

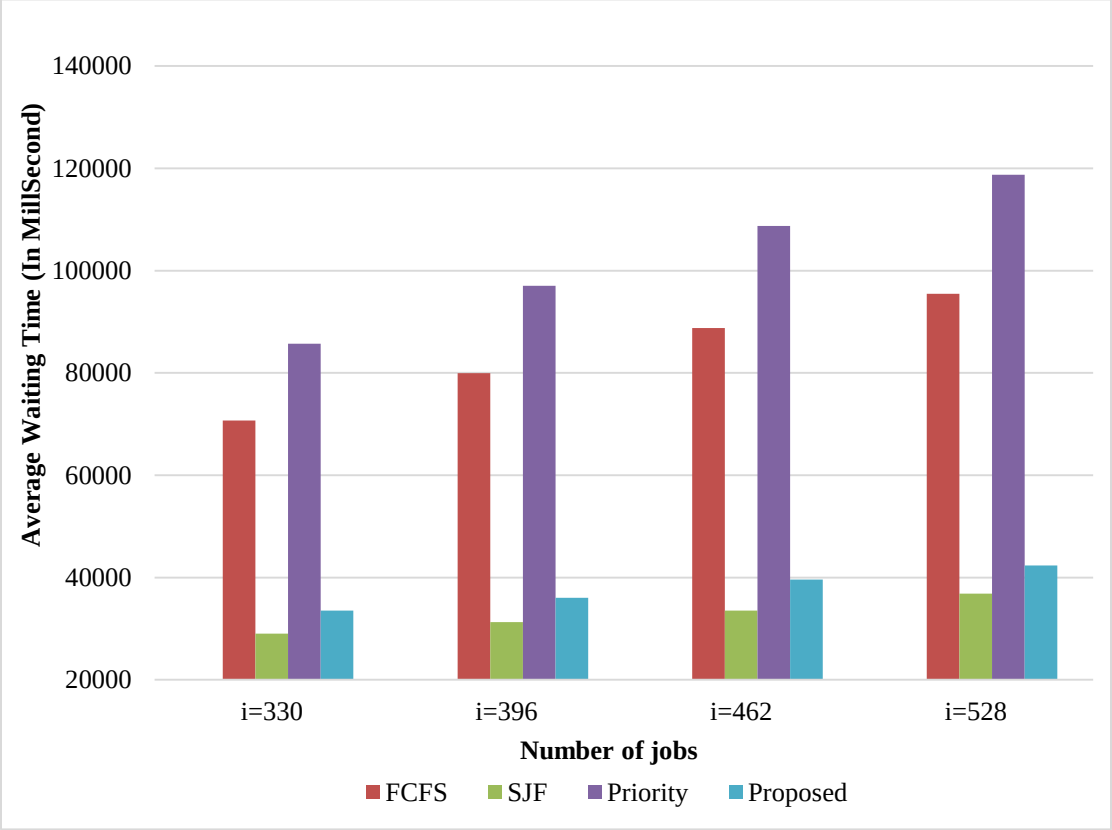

Figure 10: Performance analysis on the basis of average waiting time under moderate load

Figure 10 shows the performance analysis on the basis of average waiting time under moderate load. Here, the vertical axis represents average waiting time in milliseconds of the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 265 to 528 which represent load on the system. Figure 10 shows that under moderate load, SJF have the lowest average waiting time where priority scheduling algorithm has the highest average waiting time. Proposed scheduling algorithm has the lower average waiting time than FCFS and pure priority scheduling. Proposed scheduling algorithm considers both priority and size of the job which results in lower average waiting time than FCFS and priority scheduling. Shortest job first favours jobs with shorter execution time and hence results in lowest average waiting time.

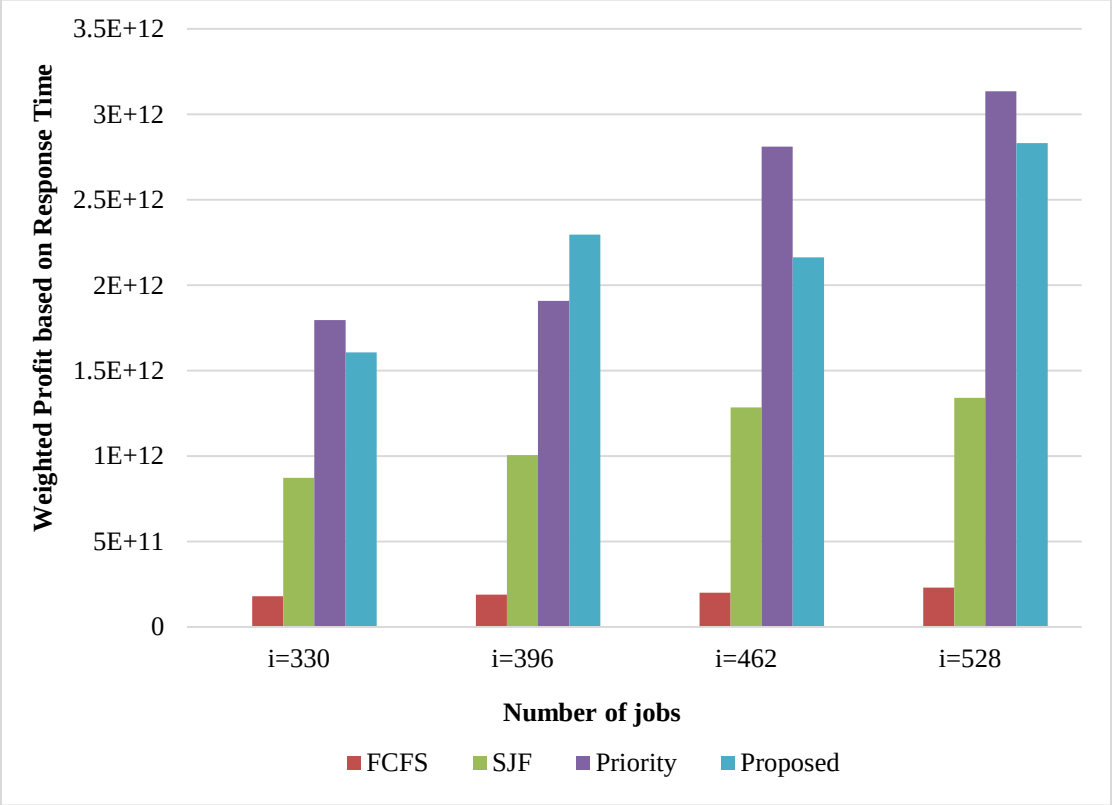

Figure 11: Performance analysis on the basis of weighted profit based on response time under moderate load

Figure 11 shows the performance analysis on the basis of weighted profit based on response time under moderate load. Here, the vertical axis represents the weighted profit based on response time while executing the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 265 to 528 which represent load on the system. Figure 11 shows that under moderate load, FCFS has the lowest profit based on response time whereas pure priority scheduling has the highest weighed profit based on response time. Profit of proposed scheduling algorithm is less than profit of profit of pure priority scheduling (except for load of 396 jobs) whereas proposed scheduling algorithm has more profit than shortest job first and first come first serve. Reason behind high profit of priority scheduling is that it favours job with highest priority and do not consider other factors such as expected execution time of job, etc.

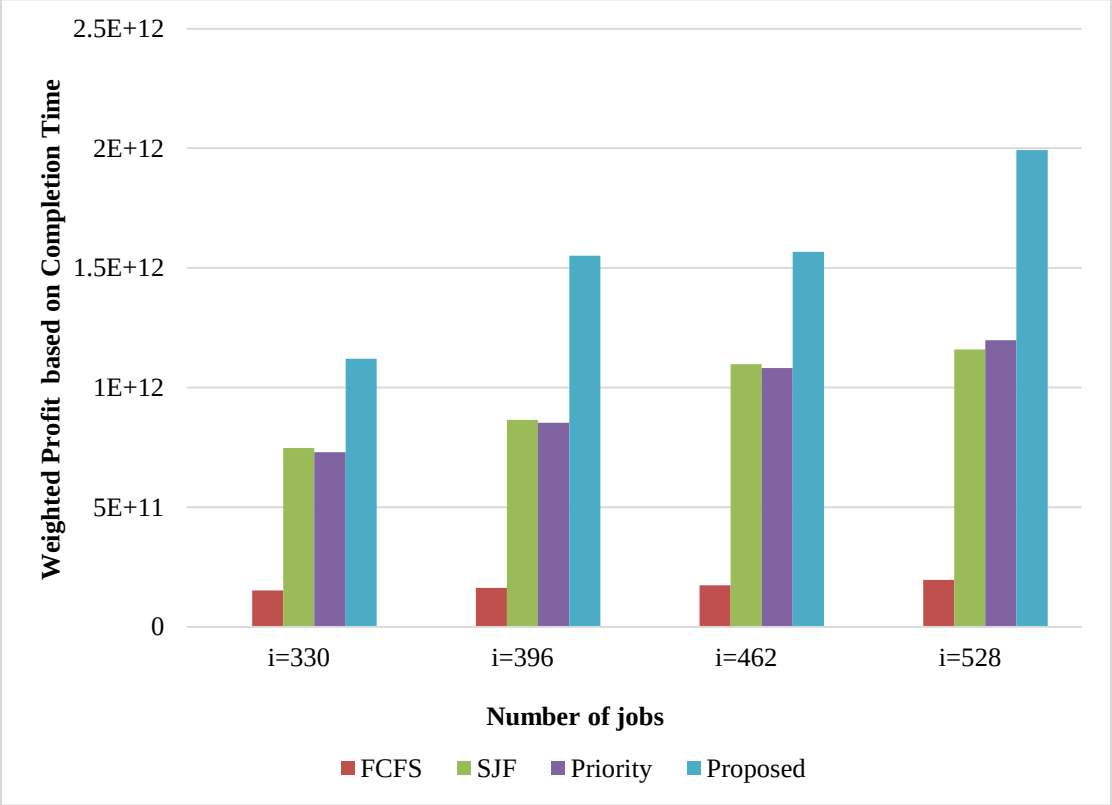

Figure 12: Performance analysis on the basis of weighted profit based on completion time under moderate load

Figure 12 shows the performance analysis on the basis of weighted profit based on completion time under moderate load. Here, the vertical axis represents the weighted profit based on completion time while executing the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 265 to 528 which represent load on the system. It shows that under moderate load, proposed scheduling has the highest profit on the basis of completion time whereas the FCFS has the least profit based on execution time. Since proposed scheduling algorithm is dynamic in nature and favours jobs with higher priority and smaller size and avoids starvation, expected profit seems to be high. It shows that weighted profit based on completion time of proposed scheduling grows faster with increases in load than other scheduling algorithm.

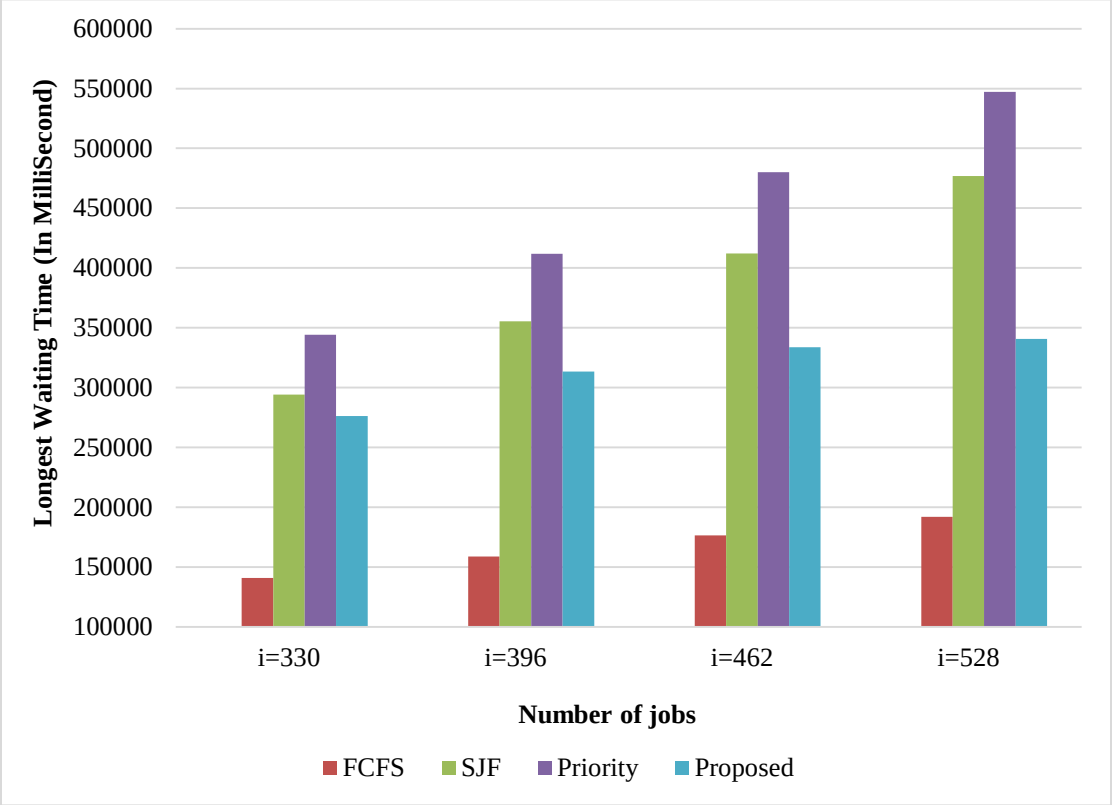

Figure 13: Performance analysis on the basis of longest waiting time job under moderate load

Figure 13 shows the performance analysis on the basis of longest waiting time job under moderate load. Here, the vertical axis represents the longest waiting time of the job under particular job scheduling algorithm, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 265 to 528 which represent load on the system. Figure 13 shows that under moderate load, FCFS has lowest longest waiting job where as pure priority scheduling has the highest longest waiting time of the job. Proposed algorithm show better results as compared to SJF and pure priority scheduling by reducing the longest job waiting time. This result is observed due to the reason that proposed scheduling algorithm uses the concept of aging which prevent the changes of starvation of low priority jobs and hence reduce the longest waiting time of job.

5.2.3 Result Analysis under Maximum load

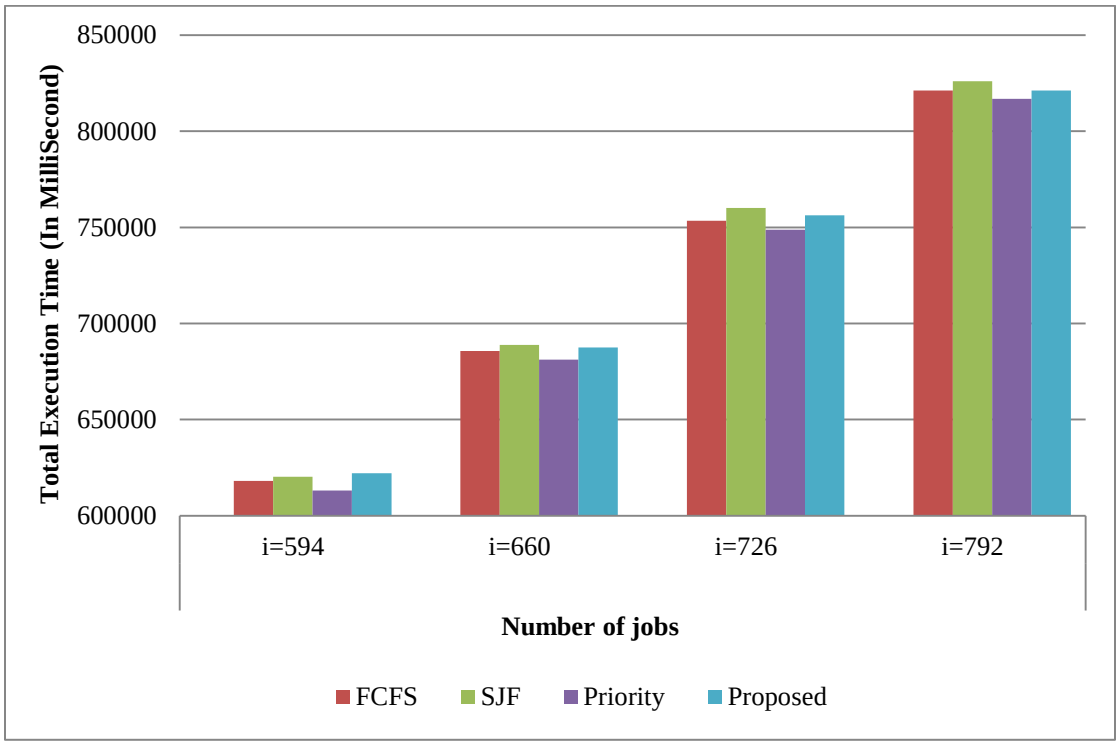

Figure 14: Performance analysis on the basis of total execution time under maximum load

Figure 14 shows the performance analysis on the basis of total execution time taken to process the given set of jobs under maximum load. Here, the vertical axis represents the total execution time taken to process the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 529 to 792 which represent load on the system. Figure 14 shows that under maximum load, all the Scheduling algorithms take almost same time for completing the jobs. Pure priority scheduling has taken the least time to process all the jobs whereas the SJF has taken the longest time to execute given set of jobs. Difference in the total execution time of all the scheduling algorithms seems to be very small.

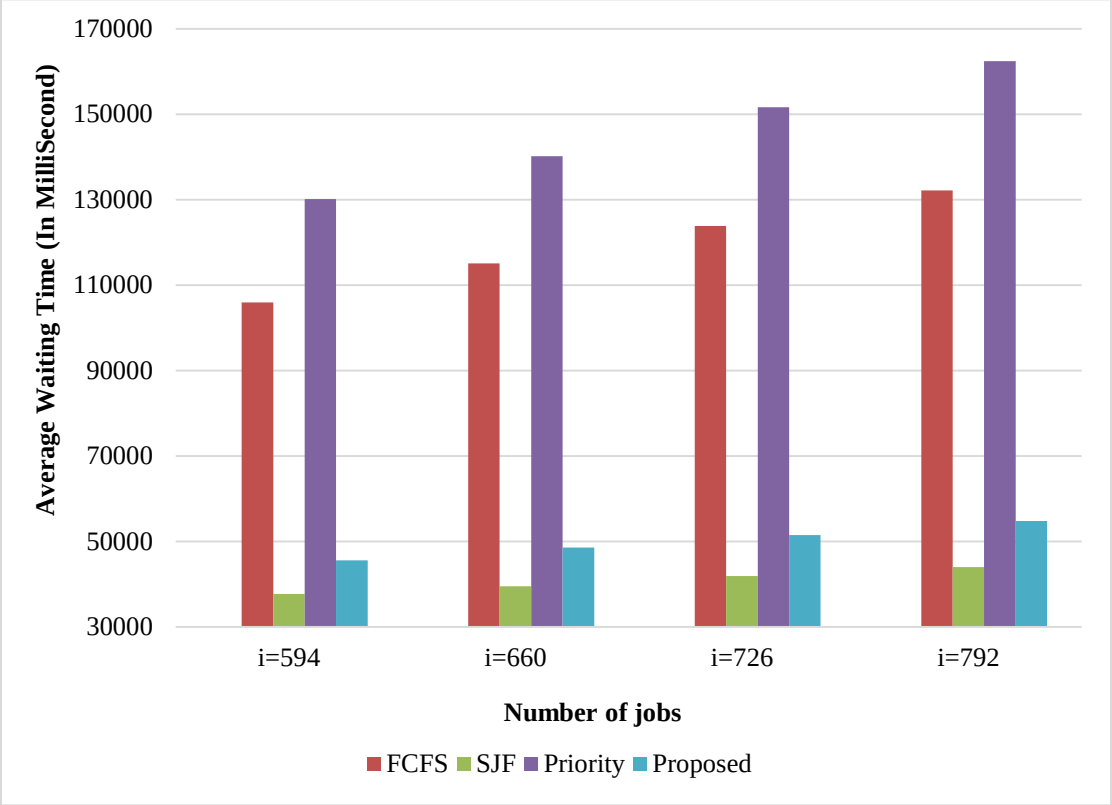

Figure 15: Performance analysis on the basis of average waiting time under maximum load

Figure 15 shows the performance analysis on the basis of average waiting time under maximum load. Here, the vertical axis represents average waiting time in milliseconds of the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 529 to 792 which represent load on the system. Figure 15 shows that under maximum load, SJF have the lowest average waiting whereas pure priority scheduling seems to have highest average waiting time for the given set of jobs. Proposed scheduling algorithm shows lower average waiting time than FCFS and pure priority scheduling. There is huge difference in the average waiting time between pure priority scheduling and proposed scheduling algorithm.

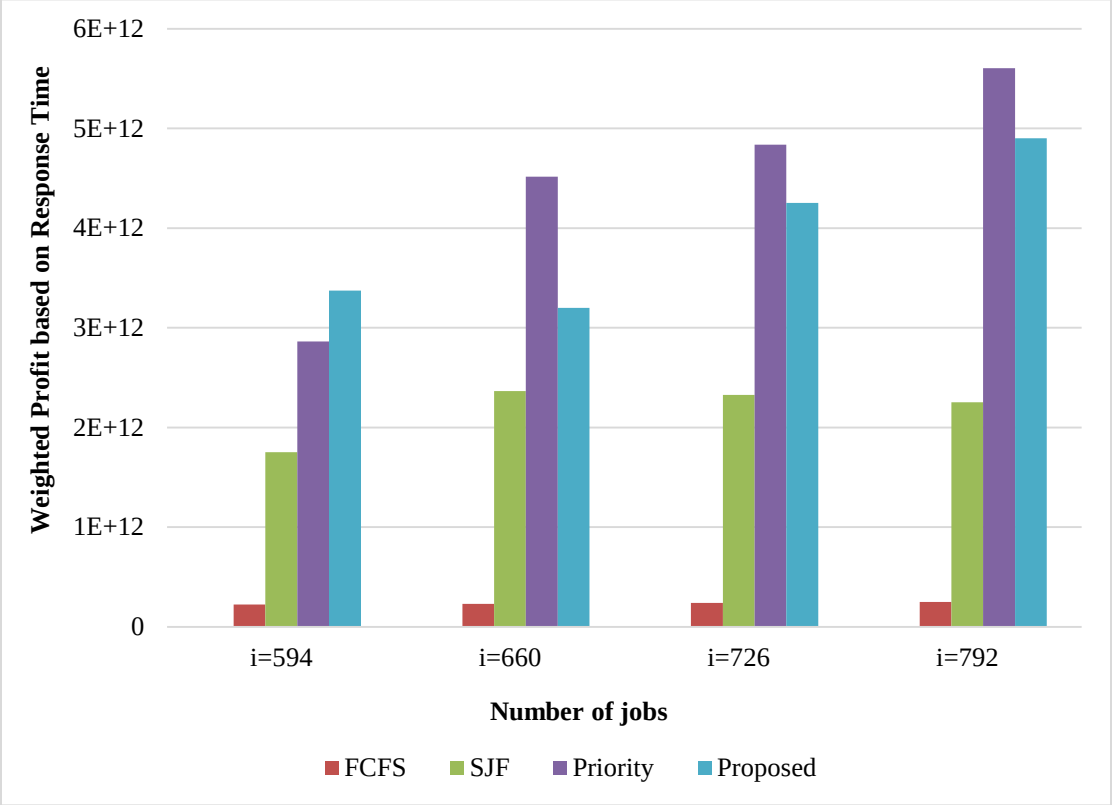

Figure 16: Performance analysis on the basis of weighted profit based on response time under maximum load

Figure 16 shows the performance analysis on the basis of weighted profit based on response time under maximum load. Here, the vertical axis represents the weighted profit based on response time while executing the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 529 to 792 which represent load on the system. It shows that under maximum load, pure priority scheduling has the highest weighted profit based on response time whereas FCFS has the lowest weighted profit based on response time. Proposed scheduling algorithm shows better results than FCFS and SJF. Pure priority scheduling and proposed scheduling algorithm favours high priority request due to which high priority user get better response time and hence results in higher profit based on response time.

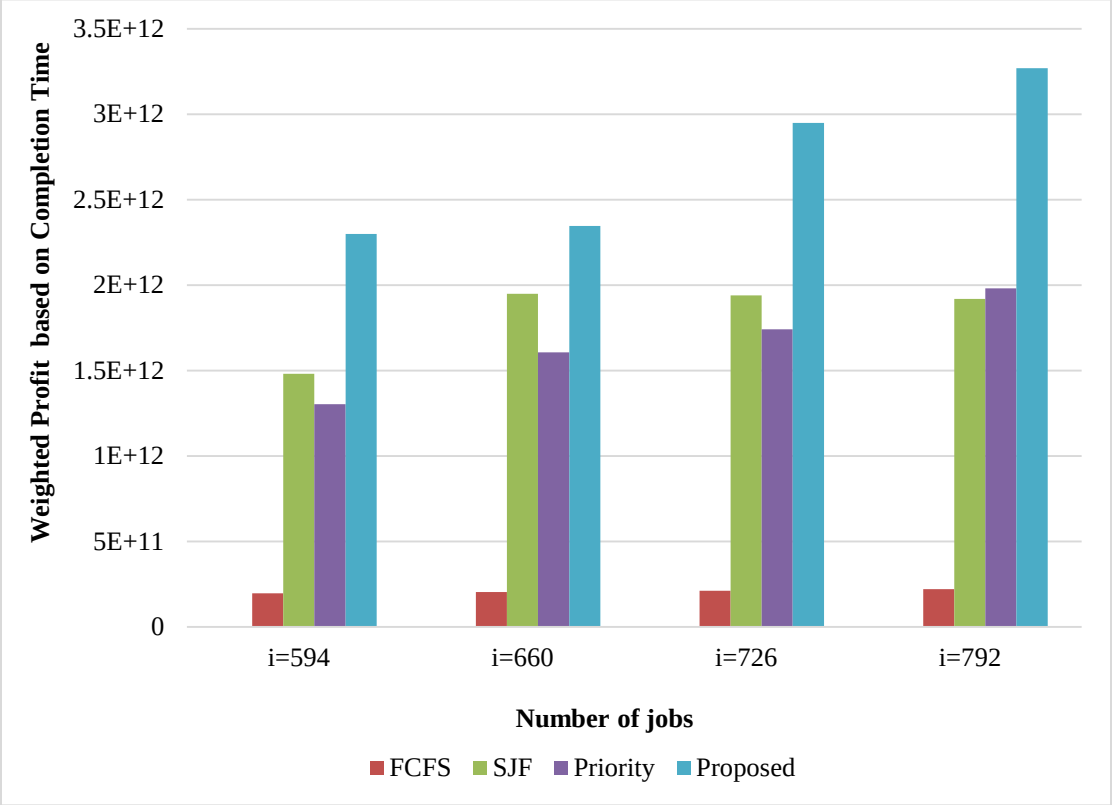

Figure 17: Performance analysis on the basis of weighted profit based on completion time under maximum load

Figure 17 shows the performance analysis on the basis of weighted profit based on completion time under maximum load. Here, the vertical axis represents the weighted profit based on completion time while executing the given set of jobs, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 529 to 792 which represent load on the system. Figure 17 shows that under maximum load, proposed scheduling algorithm shows better profit based on completion time than all other compared scheduling algorithm. FCFS has the lowest profit based completion time. Proposed priority scheduling algorithm dynamically evaluates priority of jobs and hence a better sequence of job processing takes place. FCFS which scheduled the jobs according to the arrival order seems to generate lowest revenue where as SJF and pure priority scheduling show variable behaviour which not seems to be very promising for better profit generation when compared to proposed scheduling algorithm. This shows the

efficiency of proposed scheduling algorithm to generate better revenue as compared to other scheduling algorithm.

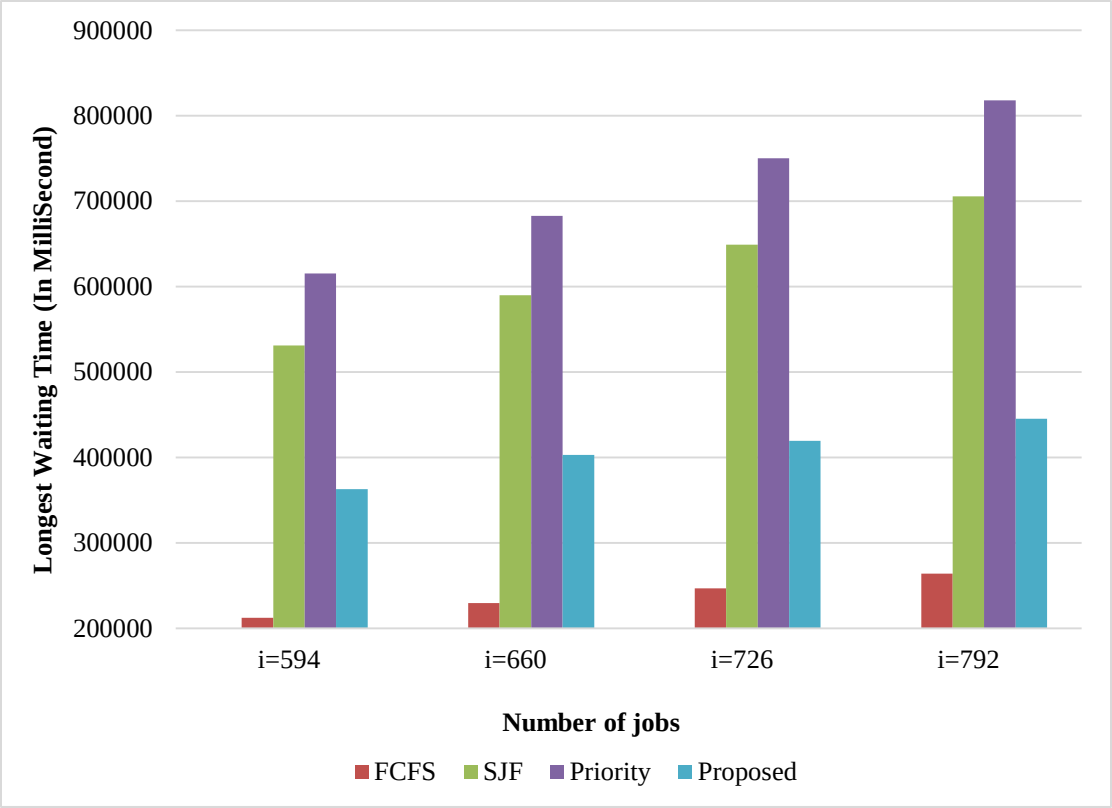

Figure 18: Performance analysis on the basis of longest waiting time job under maximum load

Figure 18 shows the performance analysis on the basis of longest waiting time job under maximum load. Here, the vertical axis represents the longest waiting time of the job under particular job scheduling algorithm, and the horizontal axis represents number of job to be executed. The number of jobs is in the range 529 to 792 which represent load on the system. It shows that under maximum load, FCFS has lowest longest waiting job as assumed due to its fairness. Results also show that pure priority scheduling has longest waiting time for a job followed by shortest job first since these algorithms are unfair in nature and do not consider waiting time of the jobs. Proposed algorithm show better results as compared to SJF and pure priority scheduling by reducing the longest job

waiting time. Proposed scheduling algorithm uses aging concept for scheduling due to which low priority jobs do not get starved and hence some level of fairness is seen in results. Result shows that pure priority scheduling and SJF have job with longer waiting time and the possibility of starvation of jobs.

5.3 Summary

5.3.1 First come first serve

Our simulation result shows that FCFS has lower total execution time than shortest job first and proposed scheduling (except for few cases), but has higher execution time than priority scheduling. Average waiting time of FCFS is less than pure priority scheduling but is more than shortest job first and proposed scheduling algorithm. FCFS scheduling algorithm has lowest profit based on response time and profit based on based on completion time. This result is due to the fact that some jobs with larger job size and lower priority are executed early than higher priority job with comparatively lower execution time are executed later and hence lower profit on basis of dispatch time.

5.3.2 Pure priority scheduling

Our simulation result shows that priority scheduling has smallest execution time but very high waiting time and hence high response time. Pure priority scheduling has the highest profit based on job response time and this is due to the fact that high priority jobs are executed early and has high weightage of profit but it is found that job with low priority have high changes of starvation and hence response time and waiting time may be higher.

5.3.3 Shortest job first

Our simulation result shows that shortest job first takes longest total execution time but it has lowest average waiting time. Shortest job first do not have good profit based on response time as well as profit based on completion time. This is because some the jobs with high priority are larger in size and hence are processed after very long wait

and hence has larger dispatch time. Our results also show the possibility of starvation of jobs under SJF scheduling as the longest waiting time of job is quite high.

5.3.4 Proposed scheduling algorithm

Our simulation result shows that proposed scheduling algorithm has execution time which is more than priority scheduling but less than shortest job first. Proposed scheduling algorithm has good profit based on response time as well as profit based on completion time. These result seems due to the fact that job to be processed is selected on the basis of dynamic priority which picks job with some average of given priority and size of job. Also low priority jobs are also promoted by the time so there are no chances of long waiting time which avoids starvation of job.

Chapter Six: Conclusion and Future Scope

This chapter presents the conclusion of the thesis and future work that can be done on this thesis.

6.1 Conclusion

In this thesis, a new technique for handling jobs in overload condition is proposed. The method involves databases as queue for persistent storage of jobs so that system can resist failures and request rejection problem. All the user request are stored in database and on the basis of priority of job and other factors such as job size and waiting time, jobs are selected for processing.

We have proposed database implementation of queue which will avoid rejection of user request in overload condition, provide good profit to service provider and maintains user satisfaction. Based on the capacity of the server, job processing threads are allotted to the server. Applications can choose the scheduling policy according to their system load. Result shows that at minimum load, proposed scheduling algorithm provide equivalent performance as compared scheduling policies but as the load increases, proposed algorithm seems to be better than compared algorithm.

Simulation results also show that higher priority jobs are handled efficiently and lower priority jobs are not starved. Average waiting time of the jobs is improved as compared to pure priority scheduling and FCFS. Proposed method of job scheduling seems to fulfil the main aim of any application i.e. better profit and user satisfaction.

6.2 Future Scope

The following directions are worthy of consideration for future work:

1. Proposed method of scheduling use simple method for determining job execution time based on job size. More generalized methods can be used in future to determine the job execution time.

2. We have followed non-preemptive scheduling in this thesis and it can be extended to be preemtive scheduling by minimizing the overhead cost for preemption.

3. Proposed scheduling algorithm should be analyzed with real time scheduling and under real time overloading conditions so as to further analyze the behaviour of proposed technique and making further improvements in proposed technique.

4. We have used a single priority queue for job processing. Proposed method can be extended to multiple priority queues for different priority classes so as to further improve the processing of higher priority jobs while preventing starvation of low priority jobs.

References

A MySQL Strategy Whitepaper. (2013). *MySQL:A Guide to High Availability.* Oracle.

A MySql White Paper. (2012). *Oracle and Intel Testing:1 BILLION Writes per Minute.*

Andersson, M., Höst, M., Cao, J., Nyberg, C., & Kihl, M. (2006). Design and Evaluation of an Overload Control System for Crisis-Related Web Design and Evaluation of an Overload Control System for Crisis-Related Web Server Systems. IEEE.

Audsley, N., Burns, A., Richardson, M., Tindell, K., & Wellings, A. J. (1993). Applying new scheduling theory to static priority pre-emptive scheduling.

Bai, T., Liu, Y., & Hu, Y. (2008). Timestamp Vector based Optimistic Concurrency Control Protocol for Real-Time Databases. IEEE.

Balajee, M., Suresh, B., Suneetha, M., Rani, V. V., & Veerraju, G. (2010). PREEMPTIVE JOB SCHEDULING WITH PRIORITIES AND STARVATION CUM CONGESTION AVOIDANCE IN CLUSTERS. *Second International Conference on Machine Learning and Computing* (pp. 255-259). IEEE.

Bansal, N., & Harchol-Balter, M. (2001). Analysis of SRPT Scheduling: Investigating Unfairness. *ACM SIGMETRICS international conference on Measurement and modeling of computer systems. 29*, pp. 279-290 . New York: ACM.

Baruah, S., & Fisher, N. (2008). Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems. *ICDCN* (pp. 215-226). Heidelberg: Springer-Verlag.

Bender, M. A., Chakrabarti, S., & Muthukrishnan, S. (1998). Flow and stretch metrics for scheduling continuous job streams. *Discrete algorithms .* ACM.

Bernstein, P. A., Hadzilacos, V., & Goodman, N. (1987). *Concurrency control and recovery in database systems.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Carlstrom, J., & Rom, R. (2002). Application-aware Admission Control and Scheduling in Web Servers. *IEEE INFOCOM* (pp. 506-515). IEEE.

Chen, M., Gao, Z., & Wang, W. (2007). Research on Multi-Objective Dynamic Priority Request Scheduling of Business Web System. IEEE.

Cherkasova, L. (1998). Scheduling Strategy to improve Response Time for Web Applications. *International Conference and Exhibition on High-Performance Computing and Networking*, (pp. 305-314).

Concurrency Problems. (n.d.). Retrieved from Accessing and Changing Relational Data: http://technet.microsoft.com/en-us/library/aa213029%28v=SQL.80%29.aspx

Drake, S., Hu, W., McInnis, D. M., Sköld, M., Srivastava, A., Thalmann, L., . . . Wolski, A. (2005). Architecture of Highly Available Databases. In *Service Availability* (Vol. 3335, pp. 1-16). Springer Berlin Heidelberg.

Duquennoy, S., & Grimaud, G. (2010). Efficient Web Requests Scheduling Considering Resources Sharing. *Modeling, Analysis and Simulation of Computer and Telecommunication Systems.* IEEE.

Elnikety, S., Nahum, E., Tracey, J., & Zwaenepoel, W. (2004). A method for transparent admission control and request scheduling in e-commerce web sites. *13th international conference on World Wide Web.* New York: ACM.

Farah, P. R., & Murta, C. D. (2008). A Transient Overload Generator forWeb Servers. (pp. 119-126). IEEE.

Han, H., Ni, H., Sun, P., & Deng, F. (2010). A Dynamic Priority Scheduling Algorithm based on Interestingness Evaluation. *Second International Conference on Computer Engineering and Technology* (pp. 214-217). IEEE.

Harchol-Balter, M., Schroeder, B., Bansal, N., & Agrawal, M. (2003). Size-based Scheduling to ImproveWeb Performance. *21*(2).

Holton, D. R., Awan, I. U., & Younas, M. (2009). Priority Scheduling of Requests toWeb Portals. *Advanced Information Networking and Applications.* IEEE Computer Society.

Homayouni, M., Jahanbakhsh, M., Azhari, V., & Akbari, A. (2009). Overload Control in SIP Servers: Evaluation and Improvement. *17th International Conference on Telecommunications* (pp. 666-672). IEEE.

Hong, T.-P., Huang, C.-M., & Yu, K.-M. (1995). A Fuzzy LPT Algorithm for Scheduling. (pp. 2588-2592). IEEE.

Kannan, G., & Selvi, D. T. (2010). Nonpreemptive Priority (NPRP) based Job Scheduling Model for Virtualized Grid Environment. *3rd International Conference on Advanced Computer Theory and Engineering* (pp. V4-377- V4-381). IEEE.

Kung, H. T., & Robinson, J. T. (1979). On optimistic method for concurrency control. *fifth international conference on Very Large Data Bases* (pp. 351-351). ACM.

Kung, H., & Robinson, J. T. (1981). On Optimistic Methods for Concurrency Control. *Transactions on Database Systems* (pp. 213-226). ACM.

Laiho, M., & Laux, F. (2010). Implementing Optimistic Concurrency Control for Persistence Middleware using Row Version Verification. *Second International Conference on Advances in Databases, Knowledge, and Data Applications* (pp. 45-50). Menuires: IEEE.

Laux, F., & Laiho, M. (2009). SQL Access Patterns for Optimistic Concurrency Control. *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns* (pp. 254 - 258). Athens: IEEE.

Lee, J., & Shin, K. G. (2012). Preempt a Job or Not in EDF Scheduling of Uniprocessor Systems. *PP*(99).

Lee, Z., Wang, Y., & Zhou, W. (2011). A dynamic priority scheduling algorithm on service request scheduling in cloud computing. *Electronic & Mechanical Engineering and Information Technology.* IEEE.

Li, L., Tian, Q., Tian, R., Liu, L., Gao, Z., & Chen, Y. (2010). A QoS Constrained Dynamic Priority Scheduling Algorithm for Process Engine. (pp. 11-16). IEEE.

Li, W.-S., Candan, K. S., Hsiung, W.-P., Po, O., Agrawal, D., Luo, Q., . . . Yilmaz, C. (2001). CachePortal: Technology for Accelerating Database-driven E-commerce Web Sites. *VLDB Conference.* Roma, Italy: VLDB.

LIU, C. L., & LAYLAND, J. W. (1973). Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *20*(1).

Lo, V., & Mache, J. (2002). Job Scheduling for Prime Time vs. Non-prime Time. *Cluster Computing.* IEEE Computer Society.

Mattihalli, C. (2010). Designing and Implementing of Earliest Deadline First scheduling algorithm on Standard Linux. *Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing* (pp. 901-906). IEEE.

Oh, S.-H., & Yang, S.-M. (1998). A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks. *Fifth International Conference on Real-Time Computing Systems and Applications* (pp. 31 - 36). Hiroshima: IEEE.

Orji, C. U., Lilien, L., & Hyriak, J. (1988). A performance analysis of an optimistic and a basic timestamp-ordering concurrency control algorithms for centralized database systems. *Fourth International Conference on Data Engineering* (pp. 64 - 71). Los Angeles: IEEE.

Ripoll, I., Fornes, A. G., & Crespo, A. (1996). Optimal Aperiodic Scheduling for Dynamic-Priority Systems. (pp. 294-300). IEEE.

Schroeder, B., & Harchol-Balter, M. (2006). Web servers under overload: How scheduling can help. *6*(1), 20-52 .

Schwiegelshohn, U., & Yahyapour, R. (1998). Analysis of first-come-first-serve parallel job scheduling. *ninth annual ACM-SIAM symposium on Discrete algorithms .* ACM.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2005). *Operating System Concepts.* John Wiley & Sons. INC.

Singhmar, N., Mathur, V., Apte, V., & Manjunath, D. (2004). A Combined LIFO-Priority Scheme for Overload Control of E-commerceWeb Servers. IISW.

Soares, S., & Borba, P. (2002). Concurrency control with Java and relational databases. *Computer Software and Applications Conference* (pp. 843 - 849). IEEE.

Stallings, W. (2012). *Operating Systems Internals And Design Principles* (Seventh ed.). Prentice Hall.

Tanenbaum, A. S., & Woodhull, A. S. (2006). *Operating Systems Design And Implementation* (Third ed.). PEARSON Prentice Hall.

Totok, A., & Karamcheti, V. (2010). RDRP: Reward-Driven Request Prioritization for e-Commerce web sites. *9*(6), 549-561.

Ullman, J. D. (1988). *Principles of database and knowledge-base systems.* New York, NY, USA: Computer Science Press, Inc.

Younas, M., & Awan, I. (2003). Efficient Commit Processing of Web Transactions using Priority Scheduling Mechanism. *Proceedings of the Fourth International Conference on Web Information Systems Engineering* (pp. 291 - 294). IEEE.

Younas, M., Awan, I., & Chao, K.-M. (2006). Efficient Scheduling of Vital E-Commerce Requests. *e-Business Engineering.* IEEE.

Zhang, Q., Riska, A., & Riedel, E. (2005). Workload Propagation – Overload in Bursty Servers. *Second International Conference on the Quantitative Evaluation of Systems* (pp. 1-10). IEEE.