A Performance Analysis of Task Scheduling Algorithms using Qualitative Parameters

Abhijit A. Rajguru,
SKNSCOE, Pandharpur,
Maharashtra, INDIA.

S.S. Apte,Ph.D
WIT, Solapur,
Maharashtra, INDIA.

ABSTRACT

A CPU is the very important part of the computer system; hence it must be utilized efficiently. When the demand for computing power increases, then scheduling problem becomes very important. The problem of task scheduling and load balancing are most important and challenging area of research in computer engineering. Task scheduling can be defined as allocating processes to processor so that total execution time will be minimized, utilization of processors will be optimized. Load balancing is the process of improving the performance of system through a redistribution of load among processor. In this paper, the performance analysis of various task scheduling algorithms based on different qualitative parameters is presented. The analysis indicates that task scheduling algorithms have some advantages as well as disadvantages. The main purpose of this paper is to help in design of new scheduling algorithms in future by studying existing task scheduling algorithms.

Keywords*:*
Scheduling algorithms, Classification, Qualitative parameters, Performance analysis.

1. INTRODUCTION

1.1 Scheduling in distributed systems:

Task Scheduling and load balancing has very important role in overall system performance and throughput. The main objective of the task scheduling, to distribute the tasks among processors to maximize throughput of the system, maintain stability of the system and efficient resource utilization. Load balancing is a process of improving the performance of system through a redistribution of load among processor.

1.2 Issues of load balancing and scheduling:

The load balancing and task scheduling mechanism has different issues as described below,

- Load balancing is very complex because processes may migrate from one system to another system even in the middle of execution to ensure equal work load. **[1]**
- An important problem is to decide how to achieve a balance in the load distribution between processors so that the computation is completed in the shortest possible time. **[2]**
- The large number and diverse nature of these computing resources and their users pose a significant challenge to efficiently schedule the loads and utilize the resources. **[3]**
- Algorithms for load balancing have to rely on the assumption that the on hand information at each node is accurate to prevent processes from being continuously circulated about the system without any progress. **[2]**
- One of the crucial aspects of the scheduling problem is load balancing. The challenge for a scheduling algorithm is that the requirements of fairness and data locality often conflict. **[4]**
- Scheduling in distributed operating systems is a critical factor in overall system efficiency because the distributed system is non-uniform and non-preemptive, that is, the processors may be different. **[5]**

1.3 Classification of scheduling algorithms:

Task scheduling algorithms are classified on the basis of different parameters.

1.3.1 Classification from user's point of view:

From the user's point of view, scheduling algorithms are classified into 3 categories.

1.3.1.1 Iterative Scheduling:

In this, scheduling of processes is done iteratively and the algorithms used for scheduling are known as iterative scheduling algorithms **[6].** Round Robin, shortest process next, lottery scheduling etc. are the examples of iterative scheduling algorithms.

1.3.1.2 Batch Scheduling:

In Batch scheduling, processes are queued together in a batch and scheduling is done in batches. FCFS, Shortest remaining time next, highest response ratio next are the examples of batch scheduling algorithms.

1.3.1.3 Real time scheduling: Real time tasks are those in which the accuracy of the outcome not only depend on the correctness of result but also depend on the time at which the results are produced. Rate monotonic and Earliest Deadline First (EDF) are examples of real time scheduling algorithms.

1.3.2 Classification based on the time of schedule:

In this, classification is done on the basis of time of scheduling the processes i.e. whether the processes are to be scheduled on the compile time or run time.

1.3.2.1 Static Scheduling:

In this technique, scheduling is done at compile time. For static scheduling, complete prior knowledge of task-set characteristics is required **[8].** Rate monotonic scheduling is the example of static scheduling used for scheduling real time tasks.

1.3.2.1 Dynamic Scheduling:
In this technique scheduling is done at run time **[7,8].** Dynamic schedulers are flexible. EDF is the example of dynamic priority scheduling algorithm used to schedule real time tasks.

1.3.3 Classification based on the site of scheduling:
This classification is based on whether the tasks are to be scheduled on a central site or on distributed sites.

1.3.3.1 Centralized Scheduling:
In centralized scheduling all decisions are made at the central site **[9].** But failure of the central scheduler in the distributed system is a critical issue.

1.3.3.2 Distributed scheduling:
Scheduling of non-interactive processes in a network of computers comes under this category **[10].** It refers to the chaining of different jobs into a coordinated workflow that spans several computers.

1.3.4 Classification based on pre-emption:
Pre-emption means prediction of higher priority task. Depending upon, whether pre-emption is allowed or not, scheduling algorithms can be classified into two categories.

1.3.4.1 Pre-emptive Scheduling:
If the system can be interrupted during the execution of the process on the arrival of higher priority process then this type of system is known as pre-emptive system **[9,6]**. All real time scheduling algorithms are examples of pre-emptive scheduling algorithms.

1.3.4.2 Non Pre-emptive Scheduling:
If no interruption is allowed during the execution of process then such type of scheduling is known as non pre-emptive scheduling. First Come First Serve (FCFS) scheduling is non pre-emptive scheduling.

1.3.5 Classification based on the number of processes to be scheduled:
This classification is done by considering whether the scheduling is done on single processor or multiple processors.

1.3.5.1 Uniprocessor Scheduling:
If the scheduling is done on a single processor then it is known as uniprocessor scheduling**[11]**. Round Robin, RM scheduling etc are the examples of uniprocessor scheduling algorithms.

1.3.5.2 Multiprocessor Scheduling:
If the scheduling is done on multiple processors then it is known as multiprocessor scheduling. **[9,7,11]**. Global scheduling algorithms and partitioning scheduling algorithms are the examples of multiprocessor scheduling.

1.3.6 Classification based on different criteria:

1.3.6.1 Co-operative Scheduling:
In this case, system have many schedulers, each scheduler is responsible for performing certain activity in scheduling process towards common system wide range based on the cooperation of procedures, given rules and current system users.

1.3.6.2 Immediate/ Online Mode:
In this case scheduler schedules any recently arriving job as soon as it arrives with no waiting for next time interval on available resources at that moment **[6].**

1.3.6.3 Batch/ Offline Mode:
In this mode of scheduling the scheduler holds arriving jobs as group of problems to be solved over successive time intervals, **[6]**.

2. SCHEDULING ALGORITHMS

There are different types of CPU scheduling algorithms are available, each having its own characteristics, advantages and disadvantages. In this section some of the most important and significant scheduling algorithms are discussed, namely:

•First Come First-Served (FCFS) Scheduling Algorithm.
•Shortest Job First (SJF) / Shortest Remaining Time (SRT) Scheduling Algorithm.
•Priority-based Scheduling Algorithm.
•Round-Robin Scheduling Algorithm.
•Multi-level Feedback Queue Scheduling Algorithm**.**

2.1 First Come First Served Scheduling
In first-come, first- served (FCFS) scheduling algorithm, processes are assigned to the CPU in the order they request it. There is a single queue of ready processes. When a process enters the ready queue, its PCB is linked onto the tail of the queue. The average waiting time of FCFS scheduling, is ***quite long.*** Consider the following processes that arrive at time 0, and length of the CPU burst given in milliseconds:

Process	Burst Time	Waiting Time	Turnaround Time
P_1	48	0	48
P_2	6	48	54
P_3	6	54	60
Average	-	34	54

Table 1:

If the processes arrive in the order P_1, P_2, P_3, and are served in FCFS order, following result can be get.

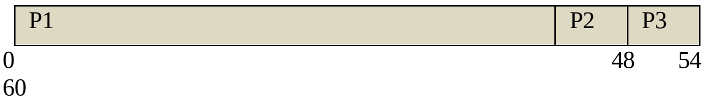

Figure 1:

If the processes arrive in the order P_2, P_3, P_1, the results will be as shown in the following chart:

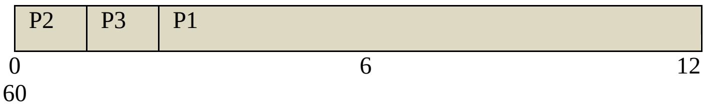

Figure 2:

The average waiting time is now (12 + 0 + 6)/3 = 6 milliseconds. The average waiting time of FCFS policy is generally ***not minimal.*** The FCFS scheduling algorithm is non-preemptive. The FCFS policy can't utilize resources parallel.

2.2 Shortest Job First Scheduling:
A shortest-job-first (SJF) scheduling algorithm associates with each process the length of the process's next CPU burst.

The CPU is assigned to the process that has the smallest next CPU burst. If the next CPU bursts of two processes are equal, FCFS scheduling policy is used. For example consider the following processes, with the length of the CPU burst given in

Process	Burst Time	Waiting Time	Turnaround Time
P1	12	6	18
P2	16	32	48
P3	14	18	32
P4	6	0	6
Average	-	14	26

milliseconds:

Table 2:

Using SJF scheduling, these processes are scheduled according to the following chart:

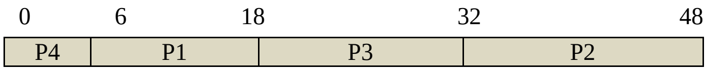

Figure 3:

The SJF scheduling algorithm provides the ***minimum average waiting time*** for a given set of processes. The problem of SJF algorithm is knowing the length of the next CPU request. The SJF algorithm cannot be implemented at the level of short-term CPU scheduling. The SJF algorithm can be pre-emptive or non-preemptive.

2.3 Priority Scheduling:

In priority scheduling a priority is assigned to each process. The CPU is allocated to the process with the highest priority. If two processes has equal priority then they are processed according to FCFS policy. Priorities are generally indicated by range of numbers, such as 0 to 20 or 0 to 5020. Here, low numbers represent high priority. Consider the following processes, assumed to have arrived at time 0, in the order P1, P2, . . . P5, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority	Waiting Time	Turnaround Time
P1	20	6	12	32
P2	2	2	0	2
P3	4	8	32	36
P4	2	10	36	38
P5	10	4	2	12
Average	-	-	16.4	24

Table 3:

Using priority scheduling, these processes are scheduled according to the following chart:

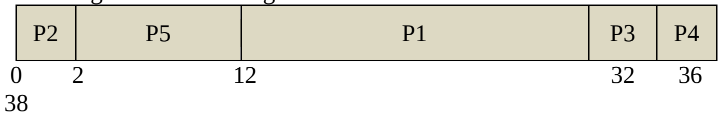

Figure 4:

Priority scheduling can be either pre-emptive or non-preemptive. A pre-emptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process. A non-preemptive priority scheduling algorithm will put the new arrived process at the head of the ready queue. A critical problem with priority scheduling algorithms is indefinite blocking, or starvation. A priority scheduling algorithm can leave some low priority processes waiting indefinitely.

2.4. Round Robin Scheduling:

It is one of the oldest, simplest, fairest and most widely used scheduling algorithms, designed especially for time-sharing systems. A small unit of time, called timeslice or **quantum**, is defined. All runnable processes are kept in a circular queue. The CPU scheduler goes around this queue, allocating the CPU to each process for a time interval of one quantum. New processes are added to the tail of the queue.

The CPU scheduler picks the first process from the queue, sets a timer to interrupt after one quantum, and dispatches the process. If the process is still running at the end of the quantum, the CPU is preempted and the process is added to the tail of the queue. If the process finishes before the end of the quantum, the process itself releases the CPU voluntarily. In either case, the CPU scheduler assigns the CPU to the next process in the ready queue. Every time a process is granted the CPU, a context switch occurs, which adds overhead to the process execution time. The average waiting time under the RR policy is ***often long.*** Consider the following set of processes that arrive at time 0, and the length of the CPU burst given in milliseconds: (a time quantum of 4 milliseconds)

Process	Burst Time	Waiting Time	Turnaround Time
P1	48	12	60
P2	6	8	14
P3	6	14	20
Average	-	11.33	31.33

Table 4:

Using round-robin scheduling, these processes are scheduled according to the following chart:

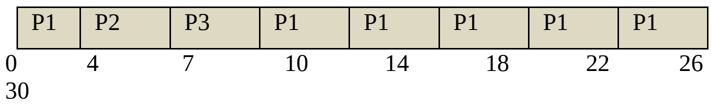

Figure 5:

In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row (unless it is the only runnable process). The RR scheduling algorithm is pre-emptive.

2.5 Multilevel Queue Scheduling:

In multilevel Queue Scheduling, processes are easily classified into different groups. A common division is made between foreground (interactive) processes and background (batch) processes.

- These two types of processes have different response-time requirements.
- In addition, foreground processes may have priority (externally defined) over background processes.
- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues (see Fig.).

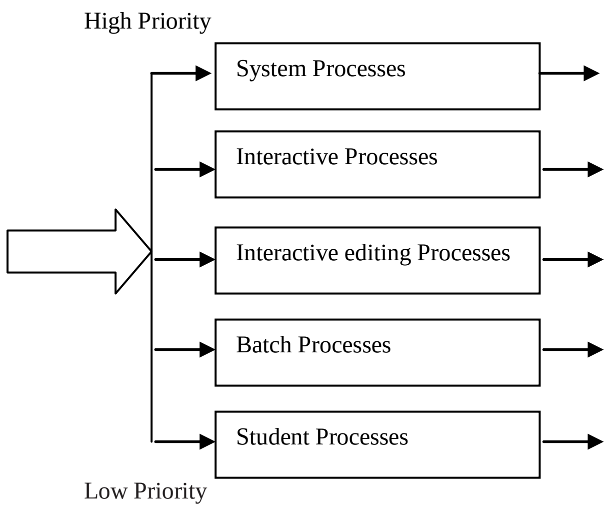

Figure 6: Multilevel queue scheduling.

The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.

Each queue has its own scheduling algorithm. The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.

2.6 Multilevel Feedback Queue Scheduling:

In multilevel queue scheduling algorithm, processes are permanently assigned to a queue when they enter the system. If there are separate queues for foreground and background processes, processes do not move from one queue to the other, since processes do not change their foreground or background nature. Due to this multilevel feedback queue scheduling having low scheduling overhead, but it is inflexible. The multilevel feedback-queue scheduling algorithm, allows a process to move between queues. To separate process following idea is used
- If a process uses more CPU time, it will be transferred to a lower-priority queue.
- A process that waits more time in a lower-priority queue may be transferred to a higher-priority queue.

This form of aging prevents starvation (see Fig.).
- A process entering the ready queue is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds.
- If it does not finish within this time, it is moved to the tail of queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds.
- If it does not complete, it is preempted and is put into queue 2.

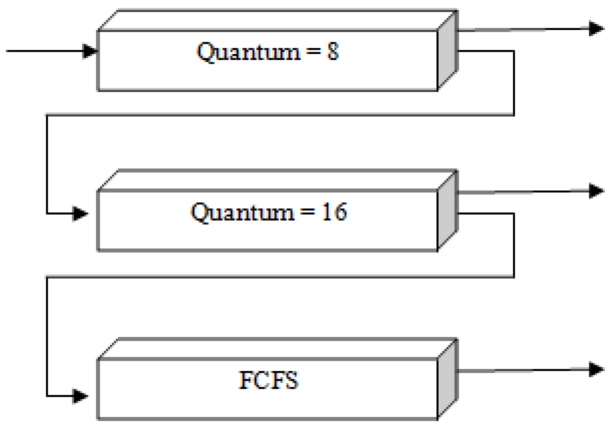

Figure 7: Multilevel feedback queues.

- Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty Multilevel feedback queue scheduling is most complex algorithm.

3. QUALITATIVE PARAMETERS

Following qualitative parameters are used to analyze performance of various task scheduling algorithms.

3.1. CPU Utilization/Efficiency:
Keep the CPU busy 100% of the time with useful work.

3.2. **Throughput**:
Maximize the number of jobs processed per unit time. Usually goal is to maximize the throughput.

3.3. Turnaround time:
From the time of submission to the time of completion, minimize the time batch users must wait for output.

3.4. Waiting time:
Sum of times spent in ready queue, minimize this.

3.4. Response Time:
Time from submission till the first response is produced, minimize response time for interactive users.

3.5. Fairness:
Make sure each process gets a fair share of the CPU.

Each scheduling algorithms having their own advantages and disadvantage. Following table shows comparative analysis of different scheduling algorithms.

Algorithms / Parameters	FCFS algorithm	SJF Scheduling algorithm	Priority Scheduling algorithm	RR algorithm	Multilevel Feedback Queue Scheduling
CPU Overhead	Low	Medium	Medium	High	High
Throughput	Low	High	Low	Medium	High
Turnaround Time	High	Medium	High	Medium	Medium
Response Time	Low	Medium	High	High	Medium
Implementation	Simple	Hard	Hard	Simple	Hard
Scheduling Criterion	CPU executes job that arrived earliest	CPU executes job with shortest time remaining to completion*	CPU executes process with highest priority	like FCFS, but with limited time slices	like multilevel queue, except that jobs can migrate from one queue to another
Fairness	Not fair	Not Fair	Fair	Fair	Not Fair
Preemptiveness	Non- Preemptive	Non- Preemptive Or Preemptive	Non- Preemptive Or Preemptive	Preemptive	Non- Preemptive Or Preemptive
Advantages	Easy to implement	Average waiting time is minimum, Maximum throughput in most of cases	It uses RR technique to make sure that a single job does not hog resources of the processor	Fair with every process, good average response time	Highly efficient, Flexible
Disadvantages	Average waiting time is long, Not useful for time sharing system.	Need advanced knowledge of estimation and preemption, Not useful for time sharing system.	Starvation problem, May not give best average waiting time.	Average waiting time is poor, It assumes that all processes are equally important, thus each process receives an equal portion of CPU, More context switch overhead.	Inefficient in single core system, Hard to implement.

Table 5: Comparative analysis of Task scheduling algorithms.

4. CONCLUSION

The purpose of this paper was to compare different task scheduling algorithms based on identified qualitative parameters. In this paper the analysis of different scheduling algorithms are carried out, various parameters are used to check the results. The shortest job first (SJF) algorithm is recommended for the CPU scheduling problems of minimizing either the average waiting time or average turnaround time. Also, the first come first serve (FCFS) algorithm is recommended for the CPU scheduling problems of minimizing either the average CPU utilization or average throughput. Round robin scheduling algorithm is fair to every process. Multilevel feedback scheduling algorithm is highly efficient and low scheduling overhead.

In future work, more and more real experimentation are needed to choose good scheduling algorithm.

5. REFERENCES

[1] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma, “Performance Analysis of Load Balancing Algorithms”, World Academy of Science, Engineering and Technology, 2008.

[2] Veeravalli Bharadwaj, Depasish Ghose and Thomas G. Robertazzi, “Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems”, Journal of Cluster computing, Vol-6, Issue-1, 2003.

[3] Sivakumar Viswanathan, Bharadwaj Veeravalli and Thomas G. Robertazzi, “Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems”, IEEE Transactions on Parallel and Distributed Systems, Vol-18, 2007.

[4] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar and Andrew Goldberg, “Quincy: Fair Scheduling for Distributed Computing Clusters”, Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09), 2009.

[5] M. Nikravan and M. H. Kashani, “A Genetic Algorithm for Process Scheduling in Distributed Operating Systems Considering Load balancing”, Proceedings 21st European Conference on Modeling and Simulation (ECMS), 2007.

[6] Hermann Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications", Springer, second edition.

[7] Peter Brucker, "Scheduling Algorithms", Springer, fifth edition.

[8] Giorgio C. Buttazzo, "Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications", Springer, Third edition.

[9] Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel", O'Reilly Online Catalogue, October 2000.

[10] Hermann Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications", Springer, second edition.

[11] Ishan Khera Ajay Kakkar “ Study of Scheduling Algorithms for Real Time Environment” International Journal of Computer Applications , April 2012