

AN EFFICIENT PROCESS SCHEDULING ALGORITHM: LONG DISTANCE ALGORITHM

Umesh Chandra Jaiswal

Department of Computer Science and Engineering
Madan Mohan Malaviya Engineering College
Gorakhpur- 273010 (UP)
INDIA

Abstract- This paper describes an improvement in scheduling scheme in batch processing system. The most popular scheduling algorithm is Shortest Job First (SJF). But SJF suffers with starvation problem especially with processes having longer burst time. This scheme seeks to minimize starvation by employing a new function for setting the priority of the processes. This new algorithm seeks to remove the problem with a new heuristic approach to set the priority function. The overhead is smaller than the some exiting approach such as HRRN and Rail Road Strategy. Calculation show that the function performance improves as the number of processes in the ready queue increases. This improvement is significant as compared to the exiting algorithms.

Approach: In this algorithm instead of waiting time as a deciding scheduling parameter, it uses a constant which increments every time for all the processes that had arrived and are in ready queue before the CPU time is allocated to the process waiting at top in the ready queue and their priority is updated but not for the processes which have arrived during the execution of the particular process under execution. In nutshell it is a SRPT (Shortest Remaining Processing Time) with modification but with some computational overhead.

Keywords- CPU scheduling, Shortest Job First, Highest Response Ratio Next, Starvation, Scheduling Overhead, Longest Distance Algorithm

I. INTRODUCTION

Process scheduling is a basic problem in operating system. This has been studied extensively and large numbers of researchers have proposed a number of algorithms for this essential purpose in the operating system. Scheduling algorithms determine which requests in the queue are serviced at any point of time, how much time is spent on each request on average, how the new request is handled by the existing algorithm if a new request arrives. The goal of scheduling algorithms is to minimize the mean turn around time of the requests and to behave in a fair manner to all the requests. The static algorithms are fastest solutions. They do not have the overhead in terms of decision making time. Static

algorithms can potentially make poor assignment decisions such as routing a request to a server through a node having a long queue of waiting requests while there are other almost idle routes available. Dynamic algorithms have the potential to outperform static algorithms by using some state information to help dispatching decisions. On the other hand, they require mechanism that collects, transmit, and analyze state information to calculate the overhead involved in the designed algorithm. The basic idea behind all these approaches is to assign each process a priority or order of execution which is followed while allocating the resources to the processes. These priorities guide the dispatcher to allocate processor in some order to avoid conflicts. A new priority function has been defined for the purpose. Shortest Remaining Processing Time (SRPT) a version of Shortest Job First (SJF) is popular approach to solve this kind of problem. This paper presents an improvement in this algorithm and managed to solve its starvation problem. The priority is calculated as a function of burst time, waiting time, and the distance of the process from the first process in the ready queue [1], and [2].

II. BACKGROUND AND LITERATURE REVIEW

Traditionally, the performance of scheduling policies has been measured using mean response time, and more recently mean slowdown and the tail of response time. These measures and policies that give priority to small job sizes (a.k.a. service requirements) at the expense of larger job sizes perform quite well. For example, Shortest-Remaining-Processing-Time (SRPT) is known to optimize mean response time. As a result, designs based on these policies have been suggested for a variety of computer systems in recent years. However the adoption of these new designs has been slow due to fears about the fairness of these policies. Specifically, there are worries that large job sizes may be “starved” of service under a policy that gives priority to small job sizes, which would result in large job sizes having response times that are unfairly long and variable.

These worries have recurred nearly everywhere size based policies have been suggested. A first example is the case of web servers, where recent designs have illustrated that giving priority to requests for small files can significantly reduce

response times. However, it is important that this improvement does not come at the expense of providing large job sizes unfairly large response times, which are typically associated with the important requests. It is in this setting that Bansal & Harchol-Balter provided the first study of the fairness of SRPT. The same trade-off has appeared across diverse application areas. For example, UNIX processes are assigned decreasing priority based on their current age – CPU usage so far. The worry is that this may create unfairness for old processes. Similar tradeoffs can be found in recent designs for routers, wireless networks, transport protocols, and so on.

In this context our algorithm tries to provide a fairness approach to the already existing algorithm HRRN (Highest Response Ratio Next).

Fairness is an amorphous concept, and nearly impossible to define in a universal way. The difficulty in defining the fairness of scheduling policies is best illustrated using a few simple examples:

- (i) Suppose jobs ‘a’ and ‘b’ are nearly of the same size, and ‘a’ enters the queue slightly before ‘b’.
- (ii) Suppose the job ‘c’ is very large and the job ‘d’ is very small, and job ‘c’ enters the queue slightly before job ‘d’.

Most of the people agree that it is fair to serve job ‘a’ before job ‘b’ and to serve job ‘d’ before job ‘c’. However, let us see how the fair service order changes depending on the setting being considered. If the queue in question is a ticket box office, then it is fair to serve job c before job d and if the setting is a hospital it is fair to serve the more urgent job, regardless of the sizes or arrival order. This simple example illustrates that we must have a clear idea of what is meant by the term “fair” in the application context before defining an appropriate notion of fairness. Thus, in our context “fairness” refers to the idea that it will be fair when it exhibits execution ordering of the process similar to that of FCFS (First Come First Serve). So our major motive is to have a better performance algorithm but with approximate fairness as the FCFS algorithm depicts. But in this paper we would only deal with the objective which is an issue and described in the following paragraph.

As with the FCFS, though it offers considerable fairness but apparently at the expense of average waiting time. That is, as the name FCFS says that the process arrived gets executed as per their arrival that is the process that have arrived first gets executed first so on. So in this, scheduler offers CPU time to the process irrespective of the size of the processes arrived and it is obvious that the average waiting time will be greater as more the bigger sized process arrived before than the shorter and so they will be executed too and contributing to increase in average waiting time.

Hence in the proposed algorithm the CPU is allocated fairly to the processes to extent better than many existing algorithms like HRRN and Rail Road Strategy which had solely aimed their goal at getting less average waiting time thus more processes got executed that had short burst time

and though starvation is not occurred but the longer processes had to wait for their chance to get CPU for longer amount of time [2], [3], and [5].

III. PROPOSED ALGORITHM

A. Basic Strategy

Basis of our work LDA (Longest Distance Algorithm) is credited to the paper HRRN where few modifications are done, both in scheme and the implementation to meet the desired requirement of our proposed algorithm. These modifications made will provide considerable amount of fairness to the processes as well as better average waiting time than the FCFS. Our proposed algorithm is non-preemptive in nature as HRRN.

Long distance algorithm in this paper aims to solve the problem whereby long tasks tend to starve because they never get CPU time. In SJF, processes are prioritized on the basis of their burst time. It may happen that a process of long duration never runs because there is always a shorter task waiting for the CPU. As we are concentrating on the factor that fairness should be employed to some extent which was absent in other pre-existing algorithm so we present our proposed algorithm, Long distance algorithm, which fixes this by adjusting the priority of the processes. The equation (3.1) is used to calculate the priority (P):

$$P = 1 + (wt / bt) * f(d) \tag{3.1}$$

Where wt= waiting time, bt = burst time, f(d) = function of the distance of task from the front end of the ready queue. As the process far away from the front end of the ready queue is longer in duration if the list is sorted in non-decreasing order of the list. This is different from HRRN when a process starts starving moved forward in the priority list. This is illustrated with an example shown in Table 1 to Table 8.

TABLE 1. AN EXAMPLE OF PROCESSES

Process ID	Arrival Time	Burst Time
P1	0	5
P2	3	15
P3	4	6
P4	6	8
P5	13	12

TABLE 2. ALLOCATION AT TIME T=0 MILLISECONDS

Process Id	P1				
Distance(d)	1				

TABLE 3. ALLOCATION AT TIME T= 5 MILLISECONDS

Process Id	P1	P2	P3		
Distance(d)	Executed	1	2		

TABLE 4. ALLOCATION AT TIME T= 11 MILLISECONDS

Process Id	P1	P3	P2	P4	
Distance (d)	Execute d	Executed	1	2	

TABLE 5. ALLOCATION AT TIME T= 19 MILLISECONDS

Process Id	P1	P3	P4	P2	P5
Distance (d)	Executed	Execute	Executed	1	2

TABLE 6. ALLOCATION AT TIME T= 34 MILLISECONDS

Process Id	P1	P3	P4	P2	P5
Distance (d)	Executed	Executed	Executed	Executed	1

TABLE 7. ALLOCATION AT TIME T= 46 MILLISECONDS

Process Id	P1	P3	P4	P2	P5
Distance (d)	Executed	Executed	Executed	Executed	Executed

TABLE 8. GANTT CHART FOR PROCESSES

P1	P3	P4	P2	P5	
0	5	11	19	34	46

Thus our algorithm demonstrates that the main objective of our algorithm in each step is to deduce or execute a process in a way that some fairness is provided to it.

B. Methodology of Implementation

The implementation of this algorithm is as given in the following paragraph.

We have considered dividing the ready queue into two

different parts (multi-level queue scheduling scheme). The first queue is sorted in increasing order of the burst time and the dispatcher selects the next process from this queue. The second queue holds other processes which are waiting for CPU resources. As in Fig 1 the second part of the queue is prioritized by the distance of each process from the starting of the queue. The queue is sorted as in other scheduling algorithm schemes presently used for scheduling. When a new process arrives it is inserted at the end of the queue. In the next iteration this newly arrived process occupies its correct position in the queue sorted according to the given priority function. After the first queue becomes empty, small part of the second queue (processes having value greater than the threshold value, this is an adjustable parameter i.e. this threshold value can be adjusted according to the need of the situation where this algorithm is deployed) (Fig 1) [6].

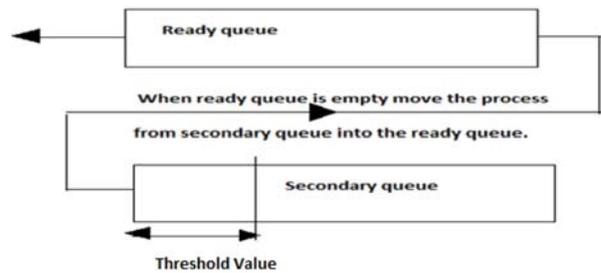


Figure 1. Basic Strategy

IV. COMPARISON AND PERFORMANCE

Initial sorting time for the queue using heap sort is $O(n \log n)$. This overhead is same as any other scheduling algorithm such as HRRN. Whenever a new process arrives, the process is placed in secondary queue at the end of the queue in $O(1)$ time. The other exiting algorithms such as HRRN, and SRP

TF takes $O(\log n)$ time to insert a new process. This is a significant achievement in the designed algorithm's performance. The new process need not to be inserted at its correct position because as the current process finishes its execution, the process queue is sorted and the new process finds its correct position after the sorting of the queue. The overhead of this algorithm is $O(1)$ when the process completes its execution and $O(1)$ when all the processes in the first queue gets completed. The HRRN algorithm has the overhead $O(n)$. The cost of this algorithm is in terms of one division and one multiplication for the computation of priority function of each process. Hence this algorithm takes one more multiplication operation than the HRRN algorithm. But removal of significant cost of $O(n)$ nullifies this multiplication operation. The performance graph have been plotted and shown in Figure 2 and 3. X-axis has burst time whereas y-axis has waiting time.

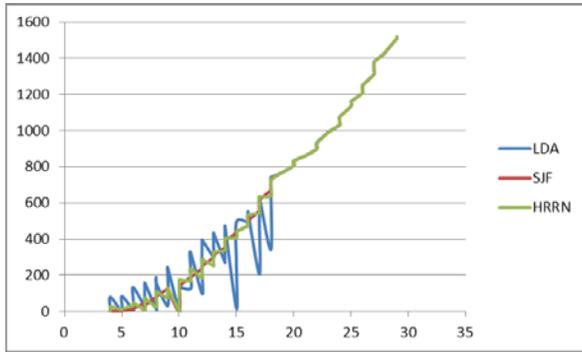


Figure 2. Comparison between LDA, SJF and HRRN

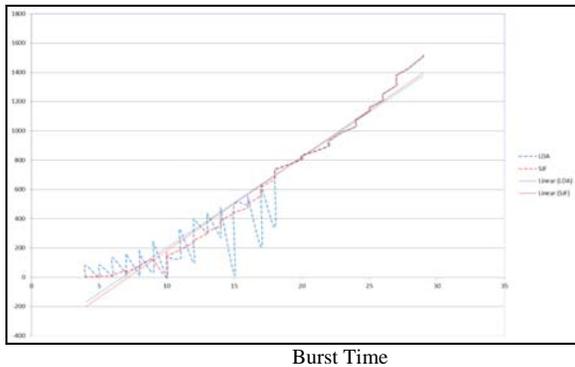


Figure 3. Comparison between Linear LDA, Linear SJF, LDA and SJF

V. CONCLUSION AND FUTURE WORK

Proposed algorithm focuses on providing a fair chance to some extent to all the processes present in the ready queue, which HRRN or SJF did not provide to such a level as our algorithm does. The algorithm succeeds in removing starvation problem of SJF. In bonus it provides fairness better than existing algorithms such as HRRN or Rail-Road strategy. Future work on this algorithm can be done on implementing and observing the result of this algorithm with preemptive technique. The multiple queue strategy can exploit the multi-processor architecture effectively. Each queue can be iterated on different processors separately.

REFERENCES

- [1] Silberschatz, A., Galvin, P.B., Gagne, G.: Operating system concepts 6th edition (ISBN 0-471-41743-2) Pg no.158-161
- [2] Wierman, A.: Fairness and Classification. ACM SIGMETRICS Performance Evaluation review archive Volume 34 Issue 4, March 2007. Pages 4 - 12
- [3] Hong, L., Ningyi, X., Zucheng, Z., Jihu, P.: N New HRRN in the Bus Arbitration of SoC Design. In: ASIC, Proceedings. 5th International Conference. (2003)
- [4] Aburas, A.A., Miho, V.: Fuzzy Logic based Algorithm for Uniprocessor Scheduling. In: Proceedings of International Conference on Computer & Communication. (2008)

- [5] Saboori, E., Mohammadi, S., Parsazad, S.: A New Scheduling Algorithm for Server Farms Load Balancing. In: Industrial and Information Systems (IIS), 2nd International Conference. (2010)
- [6] Schrage, L., Miller, L.: The queue M/GII with the Shortest Remaining Processing Time discipline, Operation Research 14(4), pp 670-684.



AUTHORS PROFILE

Dr Umesh Chandra Jaiswal is working as Associate Professor in the Department of Computer Science and Engineering, Madan Moahn Malaviya Engineering College, Gorakhpur, UP (INDIA). His area of interest is Natural Language Processing, Design and Analysis of Algorithms, Operating Systems, and Advanced Computing. He has published good number of papers in various journals, international and national conferences.

Email: ucj_jaiswal@yahoo.com