Fuzzy Burst Round Robin Scheduling

D. Pandy
C.C.S. University,
Meerut

Vandana
BIT, Meerut

Abstract: This paper designs and develops a variant of round robin policy which assigns weight to a process in dynamic environment on the basis of its remaining CPU burst after every round. The determination of weights is based on fuzzy considerations. Quantum size for every round of the process is determined dynamically in accordance with the assigned weight. The philosophy of our approach is; smaller the remaining CPU burst of a process, larger is the size of quantum assigned to it. This facilitates processes with shorter remaining time to leave the ready queue earlier, thus it increases the throughput and decreases the average waiting time. Proposed method is thus enabled to draw advantages of round robin policy as well as SJF with little more cost to be paid in terms of context switching. Even this cost can be reduced through a threshold value decided as minimum quantum size.

1. INTRODUCTION

Every scheduling algorithm has its own advantages and disadvantages. One advantage of round robin is that starvation is never a problem. It ensures that all processes in the ready queue share a time slice on the processor. But the main problem with this algorithm is that slice value must be correct. A very little value of the time slice proves to be expensive in terms of context switching, while a high slice value leads it to a non-preemptive character. In addition, round robin algorithms have low scheduling overhead of $O(1)$, which means scheduling the next process takes a constant time [1,2,3]. RR algorithms nevertheless are widely used in modern operating systems.

Weighted Round Robin (WRR) is a modified version of round robin policy that assigns some weight to each process P in proportion to its share of the CPU time. A process with larger weight will effectively get a larger proportion of quantum than a process with a smaller weight. WRR provides proportional share by running all process with the same frequency but adjusting the size of their time-quanta. The aim of proportional share schedular is to achieve proportional fairness for all the processes. Assuring fairness is essential especially in a dynamic environment where processes get blocked from using CPU. The process that has lost CPU time while it was blocked has to be compensated as it will try to gain more shares. Weighted round robin is simple to implement and schedules processes in the order $O(1)$. However, it has a relatively weak proportional fairness guarantee as its service ratio error can be quite large. Recently researches have been conducted on proportional-share analysis to achieve good proportional fairness in a dynamic environment while having a low scheduling overhead [4]. Helmy and Dekdouk [5], in 2007 has given burst round robin as a proportional share scheduling algorithm. Lottery scheduling, Group Ratio Round Robin Scheduling and Virtual Time Round Robin are few other examples of dynamic considerations.

This paper proposes the development of another variant of round robin policy which assigns weight to a process in dynamic environment on the basis of its remaining CPU requirement after every round. The determination of weights is based on fuzzy considerations. Quantum size for every round of the process is determined dynamically in accordance with the assigned weight. The philosophy of our approach is; smaller the remaining CPU burst of a process, larger is the size of quantum assigned to it. This facilitates processes with shorter remaining time to leave the ready queue earlier thus it increases the throughput and decreases the average waiting time. Proposed method is thus enabled to draw advantages of round robin policy as well as SJF with little more cost to be paid in terms of context switching. Even this cost can be reduced through a threshold value decided as minimum quantum size.

The contents of this paper are organized as follows: In section 2, we describe fuzzy weighting technique. The algorithm and implementation of the method is presented in section 3. Section 4 is devoted to the application of this algorithm to different randomly generated data-sets. The results have also been compared and discussed in this section. Section 5 presents conclusion.

2. FUZZY WEIGHTING TECHNIQUE

In conventional round robin scheduling, remaining CPU time requirement for a process keeps on reducing after completion of every round. Thus any dynamic weighting policy that assigns weight in proportion to the current CPU requirement of the process would keep on changing weights of the processes in every round. We use a weighting technique that assigns higher weights to shorter remaining bursts. Thus a process with higher weight means relatively higher time quanta.

The weight of a process is determined by fuzzy membership functions. The range of CPU bursts of *n*-CPU bound processes is first categorized into several linguistic classes; such as short CPU-burst, medium CPU-burst and high CPU-burst etc. For this purpose the entire range is divided into as many equal/unequal parts as are the number of linguistic categories. A fuzzy set can be defined for CPU burst involving separate membership functions pertaining to each of these categories. If *q* denotes the quantum of round robin policy and $\mu(x_i) \times q$ gives the value of the membership function for a process P_i then $\mu(x_i) \times q$ will be the quantum assigned to the process. The quantum is calculated for every process after each round based on the remaining size of the process. Thus quantum size is dynamically updated after every round by using Fuzzy Inference Engine (FIE). Hence larger proportion of time quantum is assigned to the processes with shorter remaining CPU time requirements to enable them to leave ready queue quickly. This aims to achieve better throughput, waiting time and response time.

To demonstrate the functioning of the fuzzy weighting technique, we divide the whole CPU burst range into three equal parts corresponding to three linguistic categories: short CPU-burst, medium CPU-burst and high CPU-burst.

$$\text{Let} \quad a \approx \frac{1}{3} \max_{1 \le i \le n} \{x_i : x_i \text{ is CPU burst of process } P_i\} \qquad \ldots (2.1)$$

Expression (2.1) means that the approximate maximum size of the CPU-burst be assumed to be equal to 3a. Let us categorize the CPU-bursts of all processes into following classes:

Short CPU-burst :	$0 \le x_i \le a$;
Medium CPU-burst :	$a \le x_i \le 2a$
High CPU-burst :	$2a \le x_i \le 3a$.

Using b = (a/2), we can define following membership functions for above categories.

$$\mu_{\text{short-burst}}(x) = \begin{cases} 1, & 0 \le x \le b, \\ \dfrac{b+0.2\,(b-x)}{b}, & b \le x \le a. \end{cases} \qquad \ldots (2.2)$$

$$\mu_{\text{medium-burst}}(x) = \begin{cases} 0.8, & a \le x \le (3a/2), \\ \dfrac{b+0.2\,(2b-x)}{b}, & (3a/2) \le x \le 2a. \end{cases} \qquad \ldots (2.3)$$

$$\mu_{\text{high-burst}}(x) = \begin{cases} 0.6, & 2a \le x \le (5a/2), \\ \dfrac{b+0.2\,(3b-x)}{b}, & (5a/2) \le x \le 3a. \end{cases} \qquad \ldots (2.4)$$

Equations (2.2) – (2.4) can be extended to other ranges of CPU bursts and different number of linguistic categories. Using these equations, we can define the weight function W(*x*) as following:

$$W(x) = \begin{cases} \mu_{\text{short-burst}}(x), & \text{when the process has short-burst,} \\ \mu_{\text{mexium-burst}}(x), & \text{when the process has medium-burst,} \\ \mu_{\text{high-burst}}(x), & \text{when the process has high-burst.} \end{cases} \qquad \ldots (2.5)$$

Thus if $x_{i,r}$ denotes the remaining CPU burst of a process P_i after r^th round then in (r+1)^th round it is assigned a quantum of size $(W(x_{i,r}) \times q)$. To reduce the cost due to context switching, a threshold value may be decided to work as the minimum quantum size. However, no considerations for the context switching have been implemented in our work. The calculations in Fuzzy Burst Round Robin (FBRR) policy can be reduced by using a fixed defuzzified value for each of the three membership functions in (2.5). This would provide a fixed crisp value of the weights associated with each of the linguistic categories that is, short, medium and high bursts and thus save computation time for weight function evaluation for every process individually.

3. FBRR ALGORITHM AND IMPLEMENTATION

A general sequence of FBRR algorithm is listed below:

Step 1: Define the linguistic categories to be used and classify CPU bursts according to it.

Step 2: Define membership functions for each type in accordance with (2.2)-(2.4) and write the weight function W(*x*).

Step 3: Pick a process from ready queue in conventional round robin manner. Weight for this process be computed using $W(x)$, where x is the remaining time needed by the process on the processor.

Step 4: Time slices be assigned to it for that particular round in proportion to the weight and the process is given time on the processor according to its time slice.

Step 5: Update the remaining time of the process after completion of the round and send it to the rear end of the ready queue, if not zero.

Step 6: Go to Step 3 until ready queue is non-empty.

To understand the FBRR algorithm well, we observe its implementation through the following example of ten processes. Processes with their CPU bursts requirements are listed in Table 1 in FCFS order.

Table 1. Processes in Ready Queue

Process	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
CPU	17	38	59	43	29	9	18	48	32	22

Step 1: We shall use three linguistic categories for CPU burst viz. short, medium and large. In Table 1, the maximum burst is 59 which can be approximated to 60. Hence from (2.1) we get a = 20. The range of CPU burst is now classified into three linguistic categories short, medium and large with their limits between 0-20, ~~20-40 and 40-60, respectively.~~

Step 2: Using (2.2) – (2.4), following membership functions can be defined for short, medium and large bursts, respectively:

$$\mu_s(x) = \begin{cases} 1 & 0 < x \leq 10 \\ (12-.2x)/10 & 10 < x \leq 20 \end{cases}$$

$$\mu_M(x) = \begin{cases} .8 & 20 < x \leq 30 \\ (14-.2x)/10 & 30 < x \leq 40 \end{cases}$$

$$\mu_L(x) = \begin{cases} .6 & 40 < x \leq 50 \\ (36-.6x)/10 & 50 < x \leq 60 \end{cases}$$

Thus the weight function can be defined as below:

$$W(x) = \begin{cases} \mu_S(x), \text{ when the process has short-burst,} \\ \mu_M(x), \text{ when the process has medium-burst,} \\ \mu_L(x), \text{ when the process has high-burst.} \end{cases}$$

Step 2 to 6 are worked in Table 2. It presents the quanta assigned to each process in each round by the fuzzy weighting technique and demonstrates the implementation of FBRR scheduling policy by showing the remaining CPU burst after every round.

4. RESULTS AND DISCUSSIONS

We worked on several randomly generated data sets (each having 50 processes with CPU bursts ranging 1-100 time-units) in an attempt to compare performance of fuzzy burst round robin policy with conventional RR policy in respect of average waiting time and average response time. Waiting time calculations for RR policy are based on a fixed quantum of 10 t.u., while FBRR has used dynamically varying time quantum based on five membership functions, that is, very short, short, medium, large and very large. These results are presented in Table 3. It can be observed that in every data-set, results of average waiting time and average response time in FBRR are better than RR. A major advantage that we gain in FBRR is that the processes that are closer to their completion get bigger portion of the quantum and hence complete relatively faster to leave the ready queue. This increases the throughput along with reduction in waiting time. The sacrifice in FBRR in comparison to conventional round robin is in terms of the cost of context switching.

Results of Table 3 in respect of average waiting time and average response time for RR and FBRR are presented in the form of bar-graph in Figure 1 and Figure 2 respectively, to facilitate the pictorial comparison of two scheduling strategies.

Table 2. Demonstration of Implementation of FBRR

Process		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
CPU burst		17	38	59	43	29	9	18	48	32	22
Quantum in round 1	Assigned	8.6	6.4	4.2	6	8	10	8.4	6	7.6	8
	Remaining	8.4	31.6	54.8	37	21	0	9.6	42	24.4	14
Quantum in round 2	Assigned	10	7.68	5.04	6.6	8	-	10	6	8	9.2
	Remaining	0	23.9	49.76	30.4	13	-	0	36	16.4	4.8
Quantum in round 3	Assigned	-	8	6	7.92	9.4	-	-	6.8	8.72	10
	Remaining	-	15.92	43.76	22.48	3.6	-	-	29.2	7.68	0
Quantum in round 4	Assigned	-	8.82	6	8	10	-	-	8	10	-
	Remaining	-	7.1	37.76	14.48	0	-	-	21.2	0	-
Quantum in round 5	Assigned	-	10	6.45	9.1	-	-	-	8	-	-
	Remaining	-	0	31.31	5.38	-	-	-	13.2	-	-
Quantum in round 6	Assigned	-	-	7.74	10	-	-	-	9.36	-	-
	Remaining	-	-	23.57	0	-	-	-	3.84	-	-
Quantum in round 7	Assigned	-	-	8	-	-	-	-	10	-	-
	Remaining	-	-	15.57	-	-	-	-	0	-	-
Quantum in round 8	Assigned	-	-	8.88	-	-	-	-	-	-	-
	Remaining	-	-	6.69	-	-	-	-	-	-	-
Quantum in round 9	Assigned	-	-	10	-	-	-	-	-	-	-
	Remaining	-	-	0	-	-	-	-	-	-	-

Table 3. Comparison of Average Waiting and Response times

Data Set No.	Average Waiting Time		Average Response Time	
	Round Robin	FBRR	Round Robin	FBRR
1.	1346.80	1084.757344	38.437	30.63
2.	1862.84	1484.823750	38.550	27.870
3.	1597.80	1253.3381	36.88	26.487
4.	985.220	876.1527	37.206	30.456

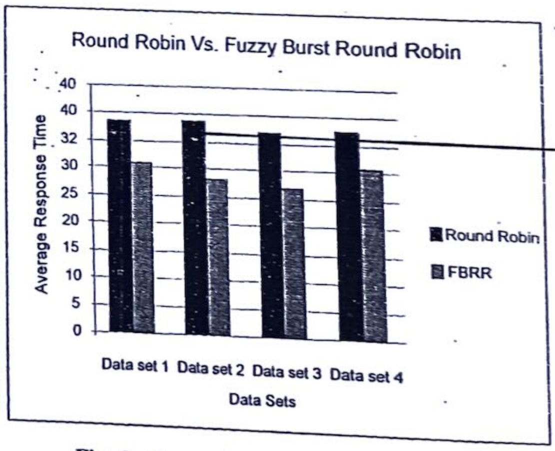

Fig. 2. Comparison of Response times

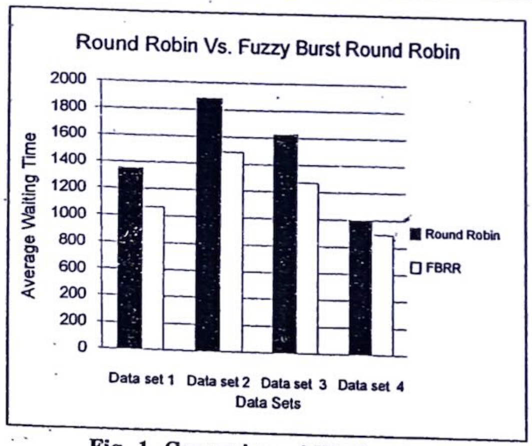

Fig. 1. Comparison of Waiting times

To have a little deeper study, it would be better if instead of comparing the average waiting time and average response time, we look into the behaviour of individual processes towards the waiting and response times. For this purpose we prefer to work on a smaller data-set of ten processes given in Table 1. We evaluate the waiting time and response time for each of the ten processes. In case of RR policy we use a fixed quantum of 10 t.u. as in the earlier data-sets. However, the functioning of FBRR is implemented with three linguistic categories viz. short-burst, medium-burst and large-burst in order to obtain the weights. Corresponding membership functions are given in section 3. Keeping the quantum size 10 time units, the weights have been used to assign dynamically varying portion of this quantum to each process in each round. Results of waiting times and response times for each individual process in the two scheduling policies are presented in Table 4. It can be observed from the table that waiting time and response time both, have smaller values in FBRR for all the processes except P_3, where these are equal.

Table 4. Waiting and Response times of Individual Processes

Process	CPU burst	Waiting Time (RR)	Waiting Time (FBRR)	Response Time (RR)	Response Time (FBRR)
P_1	17	89	63.60	6.23	4.74
P_2	38	215	203.56	6.66	6.36
P_3	59	256	256.00	5.34	5.34
P_4	43	255	235.23	6.93	6.47
P_5	29	194	189.78	7.69	7.54
P_6	9	50	33.2	6.55	4.69
P_7	18	136	99.52	8.56	6.53
P_8	48	258	251.43	6.38	6.24
P_9	32	253	202.46	8.90	7.33
P_{10}	22	223	170.36	11.136	8.74
Average		192.9	170.51	7.4376	6.3

Figures 3 and 4, present individual performances of the processes for waiting time and response time respectively for both scheduling strategies.

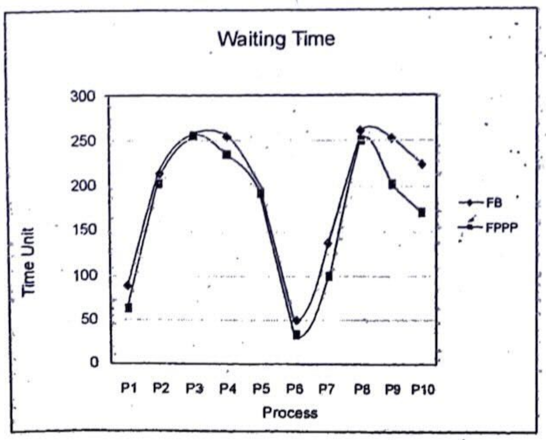

Fig. 3. WT Behaviour of Individual Processes

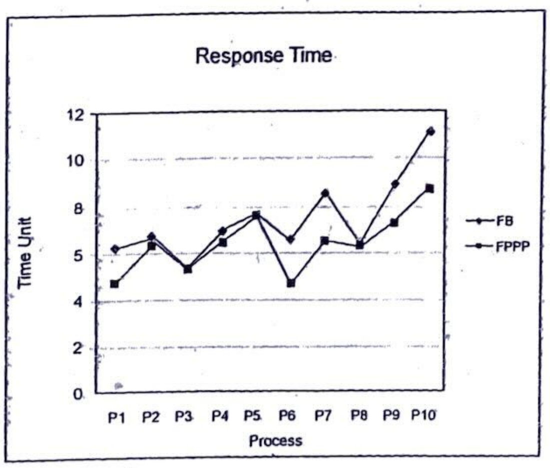

Fig. 4. RT Behaviour of Individual Processes

5. CONCLUSIONS

In this paper we have proposed a fuzzy weighting technique for round robin scheduling based on processes CPU burst time. The technique assigns different percentages of time quantum according to the remaining burst of the process after every round of its execution. This dynamically varying percentage of time quantum for a process keeps on increasing with shorterning of remaining burst. Thus it facilitates a near completing process for faster completion and will work positively towards increasing the throughput.

In fuzzy waiting technique, the value of weight function lies between zero and one. Hence it assigns ~~some percen~~tage of the quantum to every process after it completes a round. The maximum quantum can be 100% of the round robin quantum. The whole process is easily implementable through a fuzzy engine. Every process passes through this engine after completion of each round and gets its fresh time quantum.

FBRR scheduling algorithm draws the advantages of round robin as well as priority based algorithms. Normally a scheduling algorithm is tested on three areas: fairness, responsiveness and efficiency. Our algorithm is successful in the areas of fairness and responsiveness. We have certainly sacrificed in the area of efficiency due to increase in context switches, but the faster finish of the near completion processes makes up for some loss in efficiency.

REFERENCES

1. Abeni Luca, Lipari Giuseppe and Buttazza Giorgio, "Constant Bandwidth Vs. Proportional Share Resource Allocation", Proceedings of the IEEE International Conference of Multimedia Computing

and Systems, Florence, Italy, pp 107-111, June 1999.

2. Chandra A., Adler Micah, Goyal Pawan and Shenoy Prashant, "Surplus Fair Scheduling: A Proportional share CPU Scheduling Algorithm for Symmetric Multiprocessors", Proceedings of the 4th Symposium on Operating System Design & Implementation, San Diego CA., pp 45-58, October 2000.

3. Chaskar Hemant M. and Madhow Upamanyu, "Fair Scheduling With Tunable Latency: A Round Robin Approach", IEEE/ACM Transactions On Networking, Vol. 11, No. 4, August 2003.

4. Kay J. and Lauder P., "A Fair Share Scheduling", CACM, 31(1) pp 44-55, January 1988.

5. Helmy Tarek and Dekdouk Abdelkader, "Burst Round Robin as a Proportional Share Scheduling Algorithm", Proceedings of 4th IEEE-GCC Conference on Towards Techno-Industrial Innovations, Behrain, pp 424-428, November, 2007.

□