Burst Round Robin as a Proportional-Share Scheduling Algorithm

Tarek Helmy* Abdelkader Dekdouk**

* College of Computer Science & Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia.

** Computer Science Department, D'Oran Es-Sénia University, B.P.1524, EL-Mnaour, Algeria.

E-mails: helmy@kfupm.edu.sa & aek_dk@yahoo.com

Abstract* — *In this paper we introduce Burst Round Robin, a proportional-share scheduling algorithm as an attempt to combine the low scheduling overhead of round robin algorithms and favor shortest jobs. As being documented that weight readjustment enables existing proportional share schedulers to significantly reduce, but not eliminate, the unfairness in their allocations. We present a novel weight adjustment for processes that are blocked for I/O and lose some CPU time to assure proportional fairness. Experiments on the implemented simulator showed that quickly knocking away shortest processes achieves better turnaround time, waiting time, and response time. The advantage we gain is that processes that are close to their completion will get more chances to complete and leave the ready queue. This will reduce the number of processes in the ready queue by knocking out short jobs relatively faster in a hope to increase the throughput and reduce the average waiting time.

***Index Terms* - Proportional-share CPU scheduling, Quality of Services and Performance Management.**

I. INTRODUCTION

Modern operating systems (OS) nowadays have become more complex than ever before. They have evolved from a single task, single user architecture to a multitasking environment in which processes run in a concurrent manner. Allocating CPU to a process requires careful attention to assure fairness and avoid process starvation for CPU. Scheduling decision try to minimize the following: turnaround time, response time, and average waiting time for processes and the number of context switches [1]. Scheduling algorithms are the mechanism by which a resource is allocated to a client. In our research we restrict the concept of a resource to CPU time and clients to processes. A scheduling decision refers to the concept of selecting the next process for execution. During each scheduling decision, a context switch occurs, meaning that the current process will stop its execution and put back to the ready queue and another process will be dispatched. We define scheduling overhead as the incurred overhead when making a scheduling decision (other than context switches). Context switches are overheads, because CPU remains idle during context switches, which reduces CPU utilization. First-Come First-Serve (FCFS) algorithms will have a minimal amount of context switches as a process, once allocated to the CPU, will not release it until its completion. This means lower context switches of n–1, where n is the number of processes, and constant time scheduling overhead of O(1). But the responsiveness is bad when we have multiple tasks.

Shortest Job First (SJF) algorithm is known to be the algorithm with the least process turnaround time and process average waiting time. Turnaround time is the time it takes a process from its arrival time to the time of its completion, while waiting time is the amount of time a process waits. SJF, however, is not practical due to its low responsiveness in time sharing OSs. Moreover the scheduling overhead of a process in a ready queue is O(n), where n is the number of processes in the ready queue. Round Robin (RR) algorithms are widely used as they give better responsiveness but worse average turnaround time and waiting time [1, 6]. Considering a static set of n processes the number of context switches for only one round is n switches. The expected number of rounds = B/q, where B is the average CPU burst time for processes and q is a fixed time quantum. RR will have a higher average waiting and turnaround time (which is bad). However, RR algorithms are widely used in modern OSs like Linux, BSD and Windows. All use multi-level feedback queues with priorities and a RR scheduler over each process queue. In addition, RR algorithms have low scheduling overhead of $O(1)$, which means scheduling the next process takes a constant time [2, 3, 10]. A modified version of RR is the Weighted Round Robin (WRR) in which each process P has a specified weight that specifies its share of the CPU time. If a time quantum q is specified to be 10 time units (tu), and we have three processes A, B, and C having weights 7, 4, and 9, then the time quantum given to each process is proportional to the process weight. An example of one round of WRR will assign, for example, process A 70% of the time quantum. Similarly process B will receive

40% of the time quantum and process C will receive 90%. WRR is the simplest proportional-share scheduling algorithm. The aim of proportional-share schedulers is to achieve proportional fairness for all the processes, that is: if a process P has been promised a share S of CPU, it should get this share. Assuring fairness is essential especially in a dynamic environment where processes get blocked from using CPU. The process that lost CPU time while it was blocked has to be compensated as it will try to gain more shares. Recently, research has been conducted on proportional-share analysis to achieve good proportional fairness in a dynamic environment while having a low scheduling overhead [9]. We have surveyed Lottery Scheduling [5, 11], Group Ratio Round Robin Scheduling [2, 3, 10], and Virtual Time Round Robin [8, 7, 4, 12] and learned from their dynamic considerations. In those scheduling algorithms, the weight of a process when blocked for I/O gets updated as it moves back to the ready queue. Those papers use the term share to refer to the amount of the allocated resources that are dedicated to the process. The rest of this paper is structured as follows. Section II presents the proposed weighting technique for round robin scheduling algorithm. Section III discusses the dynamic consideration for implementation of our proposed scheduling algorithm. Section IV presents the results of our experimental evaluation. Section V presents the conclusions and directions for future work.

II. THE PROPOSED WEIGHTING TECHNIQUE

In this paper we implemented a new weighting technique for round-robin CPU scheduler as an attempt to combine the low scheduling overhead of round robin algorithms and favor short jobs. Higher process weights mean relatively higher time quanta. Shorter jobs will be given more time, so that they will be removed earlier from the ready queue. This aims to achieve better throughput, and waiting time, while trying to keep the context switches as low as possible.

We start by formulating a hypothesis that a process weight is inversely proportional to its CPU burst time:

$$Weight_i \propto \frac{1}{Burst_i}$$

The advantage we gain is that processes that are close to their completion will get more chances to complete and leave the ready queue. This will reduce the number of processes in the ready queue by knocking out short jobs relatively faster in a hope to increase the throughput and reduce the average waiting time. To assign specific weights to processes, we will need to normalize this equation and find a constant for this relation. The maximum CPU burst time is set to 100tu. We tried to classify the processes into five weight categories. Processes that have less CPU burst time will have higher weights as follows:

CPU Burst time (tu)	% of time quantum
1-10	100%
10-25	80%
25-50	60%
50-75	50%
75-100	40%

So a process that has a burst time of 20tu, for example, will have double the time given to another process that has a burst time of 80tu. This simple approach has shown a significant improvement in average turnaround time, average waiting time and responsiveness. Details of the experiment are shown in section IV. We gradually increase the number of classes until we reach 20, which have shown better result as compared to other results. We formulate the following for processes with CPU time greater than 5tu:

$$Weight_i = \frac{1}{\frac{Burst_i}{5}} \times 100$$

Because CPU burst time is on a scale of 100, dividing by 5, will generate 20 classes of weights. For processes that have CPU time less than 5tu, they will be served until they finish.

III. DYNAMIC CONSIDERATION

Usually when a set of processes are being executed, some processes go through a series of swapping in and out, especially if the memory is full. In this paper we consider processes that are blocked due to I/O requests only. When a process is blocked, it will be moved to a waiting queue to be further processed to an I/O device. After finishing its I/O, it will then move back to the ready queue. Those blocked processes have lost their share of CPU when they block for I/O [6, 9]. To achieve proportional fairness, the weights of those blocked processes will be updated according to the given formula:

$$New\_Weight_i = Old\_Weight_i \frac{process\_Waiting_i}{Avg\_processes\_Waitinig}$$

In this formula, we try to avoid skew-ness in average waiting time by giving I/O bound processes higher weights on their return to ready queue.

In addition, we purpose a selective preemption technique in which I/O bound processes can preempt currently running process if they have more waiting time than the average waiting time for all the processes. This is because the process has been waiting for I/O and should get higher weight than currently running processes.

IV. EXPERIMENTAL RESULTS

A. *CPU Bound Processes*

The simulator which is written in java was used to evaluate various scheduling algorithms. At first we focused on evaluating WRR with the proposed weighting technique against fixed time quantum round robin. We have used a job queue of 50 processes that arrive at different times and have lengths varying from 1 to 100tu. We call this set of 50 processes DATA1. The fixed time quantum was set to 10tu. Moreover, we have modified the simulator and added a *waiting* queue in which I/O bound processes are added when they are blocked. We have defined a scenario and a sequence of process arrivals and timed their blockage for I/O. For example process *A* is a CPU bound process and process *B* is I/O bound. Process *A* arrives at time 4 and has a CPU burst time of 45tu. Process *B* arrives at time 10 and has a CPU burst time of 23tu, but it blocks waiting for I/O after it executes for 8tu. Let us assume that the blocked process will be waiting for a certain amount of time before it comes back to the ready queue.

The BRR has shown a slight improvement over RR, when we had only five classes of weights. Table 1 shows results of running RR algorithm with a fixed time quantum of 10tu on DATA1. We show that assigning a higher weight to shorter processes has given us better waiting time, responsiveness and turnaround time. For a set of CPU bound process (i.e. with no I/O), the proposed algorithm achieved a reduction of 14.75% in average waiting time, 34% better response and 13.5% better turn around time. However the amount of context switches is 50.8% larger than RR, which is considered significantly large.

Table 1: RR results on DATA1 (50 CPU bound processes)

	Waiting	Response	Turnaround
Min	1	0	6
Mean	**511.38**	**47.3**	**556.68**
Max	1393	180	1492
S.Dev	432.27	45.23	457.13
Context Switched	**248**		

Table 2: BRR results on DATA1 using 5 classes of weights

	Waiting	Response	Turnaround
Min	0	0	4
Mean	**435.94**	**31.12**	**481.24**
Max	1363	104	1462
S.Dev	406.37	26.21	431.84
Context Switched	**374**		

We ran the experiment while increasing the amount of weight classes until we reached the 20-weights classing, which gave us a very appropriate result when compared with the fixed-time quantum RR algorithm. Table 3 shows an improvement of 27% in waiting time and 25% less average turnaround time. The amount of context switches is even much shorter than the first weight assignment used in Table 2.

Table 3: BRR results on DATA1 using 20 classes of weights

	Waiting	Response	Turnaround
Min	0	0	3
Mean	372.02	33.92	417.32
Max	1405	119	1504
S.Dev	397.91	28.24	423.62
Context Switched		293	

B. *Results with dynamic consideration (with I/O bound processes)*

We have created a new data set (DATA2) of 50 processes, which include I/O bound processes. We intend to compare the traditional RR algorithm, preemptive RR algorithm, BRR algorithm and preemptive BRR. The weight and preemption used in the BRR are newly proposed in this paper.

For the purpose of the experiment, we have set DATA2 sample to have a scenario of CPU process arrivals and burst times. We also declared that some process will be blocked for I/O at a given time of their execution and for a specified period of time. We expect higher turnaround, waiting and response times than a set of processes that do not include any I/O bound processes.

C. *Round Robin versus proposed approach:*

After running the fixed RR algorithm on DATA2, we have achieved the following listed in table 4:

Table 4: RR results on DATA2 with I/O bound processes

	Waiting	Response	Turnaround
Min	3	0	8
Mean	553.72	64.32	599.02
Max	1704	211	1799
S.Dev	453.39	53.97	474.04
Context Switches		496	

As expected, turnaround, response and waiting times all suffered from the existence of only 10% I/O bound processes. We note that turnaround time has increased from 556.68 to 599.02 with an increase of about 7%. Response and waiting times has suffered greatly and almost doubled. Now, when we run our weighted RR algorithm on the same dataset DATA2, we obtained interesting results:

Table 5: BRR results on DATA2 with I/O bound processes

	Waiting	Response	Turnaround
Min	0	0	9
Mean	422.9	49.62	468.2
Max	1437	146	1520
S.Dev	391.2	40.72	412.9
Context Switch		617	

The use of the proposed weighting technique improved the turnaround time from 599.02 to 468.2 with an improvement of 28%. Response time, average time both improved by about 23%.

D. Preemptive RR versus proposed approach with preemption

We now assume that newly added processes will preempt the currently running process when it comes back from I/O. Preemptive RR is compared with our proposed preemption technique, in which weight of blocked processes is increased according to the given formula. We first ran preemptive round robin on DATA2 and obtained the following results:

Table 6: Preemptive RR results on DATA2 with I/O bound processes

	Waiting	Response	Turnaround
Min	3	0	8
Mean	487.3	170.92	532.6
Max	1113	369	1208
S.Dev	351.56	117.67	370.6
Context Switched		585	

We note that preemption improves waiting and turnaround times on the expense of response time. The reason behind high response time is due to the fact that a returning process will preempt the currently running process, and jump over to the coming process that is appended to the end of the ready queue. Now, comparing preemptive RR with our proposed preemptive weighted RR algorithm, we achieve the following results:

Table 7: Preemptive BRR results on DATA2 with I/O bound processes

	Waiting	Response	Turnaround
Min	0	0	9
Mean	414.24	102.3	459.54
Max	1323	244	1406
S.Dev	363.23	77.95	385.34
Context Switched		622	

Our results show a significant improvement of 13% in turnaround time, 15% improvement in waiting time and 40% improvement in response time, with only little increase in context switches, Figures 1, 2, and 3.

Figure 1: Waiting time for the tested algorithms

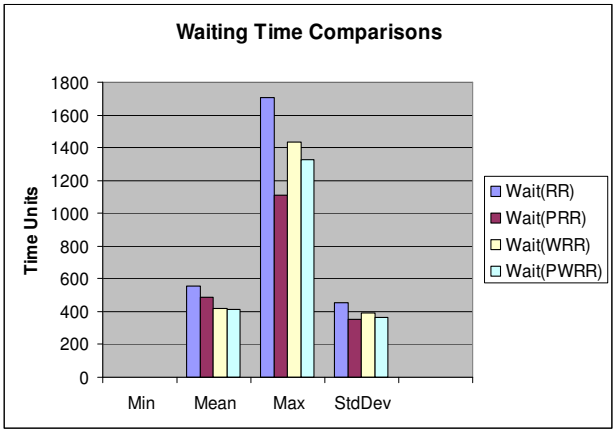


Figure 2: Response time for the tested algorithms

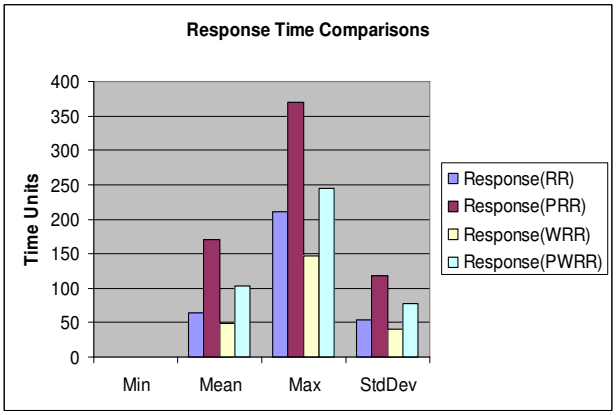
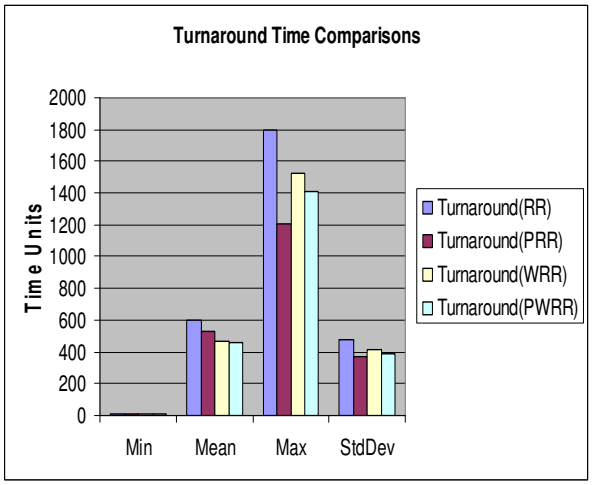


Figure 3: Turnaround time for the tested algorithms



V.CONCLUSION AND FUTURE WORK

In this paper, we have proposed a weighting technique for round-robin scheduling algorithm based on processes CPU burst time. This technique showed promising results when ran on the scheduling simulator. Using this weighting technique on a mix of processes

that include I/O bound processes has shown a good improvement over the running of a fixed time quantum RR. We suggest that this should be further analyzed with actual computation of CPU burst time to investigate whether the CPU burst time computation does not overwhelm the algorithm.

ACKNOWLEDGEMENT

We would like to thank King Fahd University of Petroleum and Minerals for providing the utilized computing facilities. Special thanks to the anonymous reviewers for their fruitful comments.

REFERENCES

1. Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne, "Operating Systems Concepts," *John Wiley and Sons*. 6Ed 2005.
2. Bogdan Caprita, Wong Chun Chan, Jason Nieth, Clifford Stein, and Haoqiang Zheng, "Group ratio round-robin: O(1) proportional share scheduling for uni-processor and multiprocessor systems," *In USENIX Annual Technical Conference*, 2005.
3. B. Caprita, W. C. Chan, and J. Nieh, "Group Round-Robin: Improving the Fairness and Complexity of Packet Scheduling", *Technical Report CUCS-018-03, Columbia University*, June 2003.
4. H. M. Chaskar and U. Madhow, "Fair scheduling with tunable latency: a round-robin approach", *IEEE/ACM Trans. Net.*, 11(4):592–601, 2003.
5. Ion Stocia, Hussein Abdel-Wahab, and Kevin Jeffay, "On the duality between resource reservation and proportional share resource allocation," *In Multimedia Computing and Networking*, 1997.
6. Jason Nieh and Chris Vaill and Hua Zhong, "Virtual-Time Round-Robin: An O(1) Proportional Share Scheduler," *In Proceedings of the 2001 USENIX Annual Technical Conference*, June 2001.
7. John Regehr, "Some guidelines for proportional share CPU scheduling in general-purpose operating systems", *In The 22nd IEEE Real-Time Systems Symposium (RTSS 2001)*, London, UK, December 3-6 2001.
8. John Regehr,"Using Hierarchical Scheduling to Support Soft Real-Time Applications on General-Purpose Operating Systems*", PhD thesis, University of Virginia*, May 2001.
9. Kevin Jeffay, F. Donelson Smith, and James Anderson Arun Moorthy, "Proportional share scheduling of operating system services for real-time application," *In Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.
10. Luca Abeni, Giuseppe Lipari, and Giorgio Buttazzo, "Constant bandwidth vs. proportional share resource allocation", In *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, Florence, Italy, June 1999.
11. M B. Jones and J. Regehr, "CPU Reservations and Time Constraints: Implementation Experience on Windows N"*, In Proceedings of the Third Windows NT Symposium, Seattle*, WA, July 1999.
12. Y.Wang and A. Merchant, "Proportional service allocation in distributed storage systems", *Technical Report HPL- 2006-184*, HP Laboratories, Dec. 2006.
13. W. C. Chan and J. Nieh, "Group Ratio Round-Robin: An O(1) Proportional Share Scheduler", *Technical Report CUCS-012-03, Columbia University*, Apr. 2003.