Improving the Quantum Time Dynamically in the Round Robin Scheduling Algorithm

Dino S. Cajic
dinocajic@gmail.com

***Abstract*—The Preemptive Round Robin Scheduling Algorithm is an important scheduling algorithm used in both process scheduling and network scheduling. Processes are executed for a predefined unit of time called a quantum. Once the CPU executes the process for the specified time slice, the process either terminates or returns to the back of the ready queue if the process has any remaining burst time left. Numerous proposals have been made to improve the static quantum time of the Round Robin Scheduling Algorithm; most research focuses on the optimization of the ready queue. In this paper, I proposed having predefined optimized quantum times for most process that can be retrieved whenever a new process enters the ready queue.**

Keywords—quantum time, scheduling algorithms, round robin scheduling algorithm, round robin quantum time, dynamic quantum, dynamic quantum time for round robin scheduling algorithm, CPU scheduling, Operating System algorithm, OS Scheduling Algorithm.

I. Introduction

Scheduling is one of the core functions of an Operating System; it is the method that assigns processes to the CPU so that they can be executed. When more than one process is waiting to be executed, the scheduler utilizes a scheduling algorithm to make the decision of which process to run next. In an operating system, many different processes compete for CPU time at any given moment. Numerous algorithms have been developed to optimize the CPU time utilization, such as:

- First-Come, First-Served Scheduling Algorithm
- Shortest-Job-First Scheduling Algorithm
- Priority Scheduling Algorithm
- Round-Robin Scheduling Algorithm
- Multilevel Queue Scheduling Algorithm
- Multilevel Feedback Queue Scheduling Algorithm [3]

The Round Robin Scheduling Algorithm is a type of scheduling algorithm mainly used by the operating system and "applications that serve multiple clients that request to use resources [1]." Each process is "arranged in the ready queue in a first-come first-served manner, and the processor executes the task from the ready queue on the basis of a time slice [2]." Even though the process might not have finished execution, once the time slice ends, the process is pushed to the back of the ready queue and the next process starts executing [3]. It does this repeatedly until the jobs are finished.

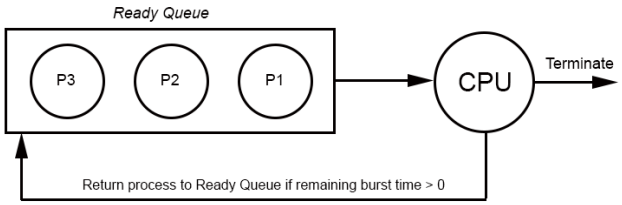

Fig. 1. Shows the Round Robin Algorithm taking processes from the ready queue and pushing them into the CPU for execution. When the quantum time slice expires, and the process is not complete, the process is pushed to the back of the ready queue [7]. If the remaining time burst is equal to zero, the process terminates.

In environments like the Real Time Operating System (RTOS), "the time slice should not be too small, since it could result in frequent context switches, and should be slightly greater than the average task computation time [2]." The Round Robin Scheduling Algorithm is also "limited by high waiting and turnaround times and low throughput" making it unsuitable for a system like the RTOS [1].

II. Current Research

Numerous proposals have been made to improve the static quantum time of the Round Robin Scheduling Algorithm. One such study proposes an algorithm called Priority Dynamic Quantum Time Round Robin Scheduling Algorithm (PDQT) [2]. The PDQT approach is performed by:

- prioritizing the process that enters the ready queue
- calculating a new quantum with a simple formula ($q = k + n - 1$), where q is the new quantum time, k is the old quantum time, and n is the priority of the processes in the ready queue
- setting different quantum times for processes based on their priority [2]

The researchers claim that the existing Round Robin Scheduler is "improved by reducing the context switches, as well as by reducing the waiting and turnaround times, thereby increasing throughput [2]."

Another area where the Round Robin Algorithm is used in is cloud computing. In cloud computing, load balancing is extremely important. Load balancing is the "mechanism of distribution of resources in a way that no overloading has occurred at any resource and optimal utilization has been performed [4]." The load balancer "uses the concept of time slices in the basic round robin algorithm [4]."

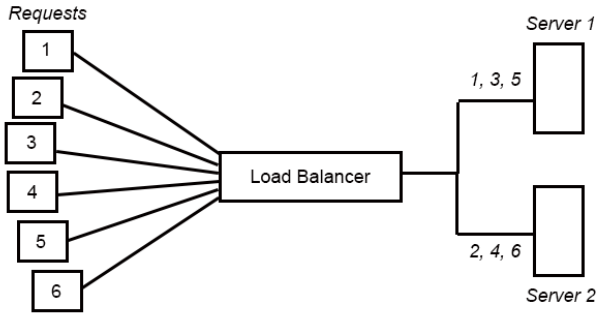

Fig. 2. The load balancer will transfer each request using the load balancer. Once the first request arrives, "the load balancer will forward that request to the 1st server. When the 2nd request arrives (presumably from a different client), that request will then be forwarded to the 2nd server [6]."

To increase the effectiveness of the load balancer, researchers have proposed "an algorithm where the smaller processes are considered first from the service queue and given them more time quantum (TQ) so that the load balancer can execute its task earlier [4].

III. Optimized Round Robin Quantum

The primary objective of this research is to generate an optimized Round Robin time slice for each queue variation. Each quantum is stored as a value in a three-dimensional array. The first dimension represents the total process count inside the ready queue. The second dimension portrays the total remaining process burst time. The third dimension depicts the longest process burst time inside the ready queue. Once the data is available, it can then be retrieved by the algorithm whenever the ready queue is updated.

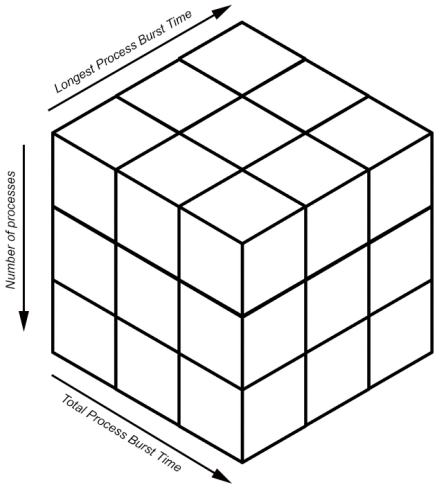

Fig. 3. The three dimensional data structure holding the optimized time slice for each ready queue scenario. The array is scalable to suit the needs of the particular operating system and hardware configuration.

A. Algorithm for Generating the Dynamic Quantum Data Structure

Depending on the number of desired options, a three-dimensional array of any size can be generated as long as it is within the hardware and operating system specifications. Note that the size may have adverse effects on devices utilizing smaller memory.

The initial three-dimensional array is generated by the following algorithm:

1. Initialize the highest expected ready queue values for each of the three dimensions.
    - i Set the highest process count that might be expected inside the ready queue for the first dimension.
    - ii Set the largest total process burst that might be expected inside the ready queue for the second dimension.
    - iii Set the largest expected single process burst time for the third dimension.
2. Loop through each dimension. If a scenario is not possible, set it equal to zero.
    - i The total time cannot be less than the number of processes.
    - ii The longest process burst time cannot be larger than the total time remaining.
    - iii Longest process time cannot be greater than total time remaining – (number of processes – 1).
3. Create a randomly generated array for the specified three-dimensional option.
    - i Set the first array element as the longest burst time of the ready queue.
    - ii If the total number of processes is equal to one, return the process.
    - iii Take the longest burst time and subtract it from the total time.
    - iv Divide the remaining time by the remaining number of processes. The result will be used in generating random burst times for each other process.
    - v Generate a random burst time between 1 and the high value for n -1 processes, where n is equal to the number of remaining processes inside the ready queue.
    - vi Take the total burst time for n-1 and subtract it from the total burst time for n processes. This will generate a value for the n^{th} remaining process.
    - vii Return the newly generated random queue.
4. Generate the best time slice for the given sequence consisting of values generated in step 3.
    - i Initialize the best average waiting time as a high value.
    - ii Initialize the optimum quantum as 1.
    - iii Check for the best time slice by testing each quantum value starting at 1 and ending at the longest process burst time for that particular ready queue.
    - iv Simulate CPU execution for each process.

v During each CPU cycle, decrement the remaining quantum and the remaining burst time for the process, and increment the total time elapsed.

vi Once the process exits the CPU, check to see if the process finished. If it did, set the finishing time for that process to equal the value stored in the total time elapsed variable.

vii Keep moving the processes from the ready queue into the CPU until the longest process burst time is equal to zero.

viii Compute the waiting times for each process. The arrival time is set to zero for each process. The turnaround time is equal to the time that the process was terminated. The waiting time is computed by subtracting the burst time for the process from the turnaround time.

$$turnaround = completion - arrival \quad (1)$$

$$waiting = turnaround - process\ burst \quad (2)$$

ix Compute the average waiting time for the particular quantum; if it's less than the current best average waiting time, update the quantum value.

x Repeat the process until the quantum time being tested is greater than the longest burst time.

xi Return the best quantum value for the particular ready queue.

B. Retrieving Dynamic Quantum

Once the data structure is generated, the optimum quantum should be retrievable for most circumstances. The algorithm should be provided with three arguments: number of processes in the ready queue, the total remaining process burst time, and the longest process time.

1. The algorithm starts by performing three checks to verify that the data provided is accessible within the array.
   i If the process count exceeds the first-dimensional array count, the number of processes is set as the last index value of the first dimension.
   ii If the total time remaining exceeds the second-dimensional array count, the total time remaining is set as the last index value of the second dimension.
   iii If the longest process time exceeds the third-dimensional array count, the longest process time is set as the last index value of the third dimension.
2. Return the stored optimum quantum value from the array.

C. Updating the Dynamic Quantum Data Structure

The dynamic quantum data structure was designed so that it can be updated with the data that each specific machine produces, especially if similar processes in the ready queue keep reoccurring. The current data structure's optimum quantum was created by first simulating the longest process burst time inside a ready queue and then generating random burst times for the remaining processes. The update feature can be turned off after the learning phase completes and turned on again after a specific segment of time.

IV. Results

The dynamic quantum was retrieved for each randomly generated ready queue and the waiting times were compared with the waiting times that were obtained by having a static quantum. Prior to running the simulation, the dynamic quantum data structure was initialized. Each three-dimensional array initialization parameter was set to 20, meaning that the quantum array is good for retrieving the time quantum for up to 20 processes inside the ready queue. The total time of the processes inside the ready queue can be up to 20 units and the longest process time can be set up to 20 units.

TABLE I. Average Waiting Times for Dynamic Quantum and Static Quantum Set To 3

Process Count	Average Waiting Time			
	Ready Queue Process Burst Times	***Optimum Quantum***	***Dynamic Waiting***	***Static Waiting***
4	4,2,1,12	4	3.0000	5.0000
5	3,1,2,1,10	3	3.8000	3.8000
6	5,1,1,1,1,3	5	10.3333	10.8333
7	2,1,1,1,1,1,6	2	5.4286	5.5714
8	3,1,1,1,1,1,1,9	3	7.1250	7.1250
9	2,1,1,1,1,1,1,1, 7	2	7.4444	7.5556
16	3,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,2	3	18.0625	18.0625
16	4,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1	1	18.2500	20.0000

Fig. 4. The table displays the total number of processes in the ready queue, the generated ready queues, the optimum quantum that was retrieved from the dynamic quantum array, the average waiting time for the dynamically obtained quantum, and the average waiting time produced by the static quantum. The static quantum was set to 3 CPU bursts.

Looking at the results obtained in Table I, the dynamically obtained quantum outperformed the statically generated quantum in 5 out 8 tests, or 62.5% of the time. For the remaining 37.5% of the time, the dynamic quantum was equally as good as the static quantum.

Changing the static quantum from 3 to 5, and generating a random ready queue with the same number of processes, produces similar results. From the results displayed in Table II, the dynamic quantum ouperforms the static quantum in 6 out of 8 tests, or 75% of the time. For the remaining 25% of the time, the waiting times for both the dynamic and static quantum were equal.

TABLE II. AVERAGE WAITING TIMES FOR DYNAMIC QUANTUM AND STATIC QUANTUM SET TO 5

Process Count	Average Waiting Time			
	Ready Queue Process Burst Times	***Optimum Quantum***	***Dynamic Waiting***	***Static Waiting***
4	4,1,2,12	4	2.7500	3.0000
5	3,2,1,1,10	3	4.0000	4.4000
6	5,1,1,1,1,3	5	10.3333	10.3333
7	2,1,1,1,1,1,6	2	5.4286	5.8571
8	3,1,1,1,1,1,1,9	3	7.1250	7.3750
9	2,1,1,1,1,1,1,1, 7	2	7.4444	7.7778
16	3,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,2	3	18.0625	18.0625
16	4,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1	1	18.2500	19.0000

Fig. 5. The table displays the total number of processes in the ready queue, the generated ready queues, the optimum quantum that was retrieved from the dynamic quantum array, the average waiting time for the dynamically obtained quantum, and the average waiting time produced by the static quantum. The static quantum was set to 5 CPU bursts.

V. CONCLUSION

Unless the static quantum is already the optimum time slice for the ready queue, retrieving the optimum time slice from the dynamic quantum array produces lower average process waiting times for the Round Robin Scheduling Algorithm.

REFERENCES

[1] "What is Round Robin Scheduling (RRS)? - Definition from Techopedia," Techopedia.com. [Online]. Available: https://www.techopedia.com/definition/9236/round-robin-scheduling-rrs. [Accessed: 27-Jun-2019].

[2] M. A. Mohammed, M. AbdulMajid, B. A. Mustafa and R. F. Ghani, "Queueing theory study of round robin versus priority dynamic quantum time round robin scheduling algorithms," 2015 4th International Conference on Software Engineering and Computer Systems (ICSECS), Kuantan, 2015, pp. 189-194. doi: 10.1109/ICSECS.2015.7333108.

[3] A. Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts. John Wiley & Sons, Inc, 2013.

[4] S. Ghosh and C. Banerjee, "Dynamic Time Quantum Priority Based Round Robin for Load Balancing In Cloud Environment," 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), Kolkata, India, 2018, pp. 33-37. doi: 10.1109/ICRCICN.2018.8718694

[5] A. Yasin, A. Faraz and S. Rehman, "Prioritized Fair Round Robin Algorithm with Variable Time Quantum," 2015 13th International Conference on Frontiers of Information Technology (FIT), Islamabad, 2015, pp. 314-319. doi: 10.1109/FIT.2015.62

[6] J. C. Villanueva, "Comparing Load Balancing Algorithms," Comparing Load Balancing Algorithms. [Online]. Available: https://www.jscape.com/blog/load-balancing-algorithms. [Accessed: 27-Jun-2019].

[7] P. Krzyzanowski, "Process Scheduling," Process Scheduling, 18-Feb-2015. [Online]. Available: https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html. [Accessed: 27-Jun-2019].

[8] A. Yasin, A. Faraz and S. Rehman, "Prioritized Fair Round Robin Algorithm with Variable Time Quantum," 2015 13th International Conference on Frontiers of Information Technology (FIT), Islamabad, 2015, pp. 314-319. doi: 10.1109/FIT.2015.62

[9] A. Alsheikhy, R. Ammar and R. Elfouly, "An improved dynamic Round Robin scheduling algorithm based on a variant quantum time," 2015 11th International Computer Engineering Conference (ICENCO), Cairo, 2015, pp. 98-104. doi: 10.1109/ICENCO.2015.7416332

[10] M. U. Farooq, A. Shakoor and A. B. Siddique, "An Efficient Dynamic Round Robin algorithm for CPU scheduling," 2017 International Conference on Communication, Computing and Digital Systems (C-CODE), Islamabad, 2017, pp. 244-248. doi: 10.1109/C-CODE.2017.791893