European Journal of Applied Sciences 9 (5): 249-257, 2017
ISSN 2079-2077
© IDOSI Publications, 2017
DOI: 10.5829/idosi.ejas.2017.249.257

Scheduling IoT on to the Cloud : A New Algorithm

[1]*S. Balamurugan,* [2]*K. Amarnath,* [3]*J. Saravanan and* [4]*S. Sangeeth Kumar*

[1]Department of CSE, KIT- Kalaignarkarunanidhi Institute of Technology, Coimbatore, India
[2]Full Stack Java Developer, CaratLane Pvt. Ltd., Chennai, India
[3]Full Stack Web Developer, ProduleSystems Pvt. Ltd., Coimbatore, India
[4]Software Engineer, IBM Pvt. Ltd., Bengaluru, Karnataka, India

Abstract: Scheduling IoT objects on to the loud is a recent research topic under focus. In this paper, the Generic Round Robin scheduling has been studied under the aspect of dynamically varying quantum size. A number of variants are proposed for dynamically varying the time quantum. A simulator has been developed for recording the observations of the average waiting time of the process by dynamically changing the time quantum based on the burst time of the process in the Ready Queue (RQ) or Completed Queue (CQ). The observations are carried out for both the scenarios of zero arrival time and process arriving at different times. Among the proposed variants, the variant in which the time quantum is varied by employing the sum of the burst time of the process in the Ready Queue and the variant in which the time quantum is varied by adding up the maximum of burst time of process in ready queue tends to be nearly 40% more efficient when compared to static time quantum Round Robin scheduling. Other variants also produce considerable improvement when compared to static time quantum Round Robin scheduling. This paper would promote lot of research enhancements in the area of Scheduling IoT on to cloud.

Key words: Round-Robin Scheduling · Ready Queue · Completed Queue · Static Time Quantum · Dynamically varying Time-Quantum

INTRODUCTION

The need for efficient cloudallocation turns out to be the most important factor in today's IoT era. Especially with the advent of multitasking environment, the allocation of CPU to a process requires more careful attention, than in a uni-task, uni-user architecture. A good scheduling algorithm would assure unbiased fairness as well as prevent starvation or stagnation of process.

Need for an efficient scheduling mechanism in a multiprogrammed environment is very high, since the implementation of an operating system is itself, through one or more processes. The need for multiprogramming is high. Also the need to complete the interrupt or any of the I/O operation within a stipulated time is very high. Designing a novel process scheduling algorithm should be governed by the following operational criteria of maximizing the CPU utilization and throughput and minimizing the turnaround time, waiting time and response time [1].

Context switching is an important inevitable act in a processor especially in case of several tasks present in a computer system. Whenever a task requires an I/O operation to be performed another task, gets hold of CPU and save the context and could be resumed back later. It is to be noted that, during each scheduling decision, there is an occurrence of context switch, which means, that current process running on the processor will be pre-empted and put in a ready queue and the next desired process will be allocated the CPU. These context switches are a pure overhead, since the processor remains idle during the context switching, thereby reducing the CPU utilization factor which would be an undesired outcome of a good scheduling algorithm.

First Come First Served (FCFS) scheduling algorithm, which works under the scheme: the process that requests CPU first is allocated first. The compelling advantage of FCFS, is its simplicity, where the dispatching of processes are carried over, according to the arrival time of process with respect to the number of context switches, FCFS

Corresponding Author: S. Balamurugan, Department of CSE, KIT- Kalaignarkarunanidhi Institute of Technology, Coimbatore, India.

algorithm will exhibit minimal number of context switches, of O(1), when the number of processes to be scheduled is relatively less. The phenomenon does not hold good incase of multi-process, thereby reporting bad responsiveness. Since FCFS is non-preemptive, once a process acquires a CPU it runs to completion. This may not be suitable in a real-time scenario. This is caused because of the fact that long process may disrupt, short process to be completed well before [2, 3].

As against in priority scheduling, CPU is assigned to process based on the externally associated priority with the heuristic that the process with highest priority is 'run' first. Here the problem arises in choosing the priorities. It is to benoted that the entire functionality associated with the process is tobe known in advance, to assign priority to a process. A major problem encountered here is starvation, which would be solved by aging [1].

Shortest Job First (SJF) scheduling tends to provide an optimal (minimal) average waiting time. The major problem here is that, a precise knowledge about how long a process will run, is to be gained in advance, but this information is usually unavailable and unpredictable [4][5]. Moreover SJF, exhibits O(n) scheduling overhead, where 'n' is number of process in ready queue.

Round RobinAlgorithm (RRA), which is the main concern of this paper, is simple, fair and most widely used algorithm especially for time-sharing systems. RR is similar to FCFS, but the difference lies in pre-emption. A small unit of time quantum, 'Q', is defined and short-term scheduler goes round the process kept in a circular queue, allocating CPU to each process a time interval of one quantum. Wherever a new process arrives to be scheduled, it is placed at the tail-end of the ready queue. CPU scheduler picks the first process from the circular queue, sets the timer to interrupt after a single time quantum and then dispatches the process [6]. If the process, still run, even after the expiry of the time quantum, preemption of CPU occurs and the remaining processes added up at the tail of circular queue. If the execution of process is finished well before the end of time-quantum, the process will voluntarily release the CPU to the next process in the ready queue. The performance of RRA is vested upon, choosing the size of time quantum [7]. This paper presents a solution to the problem of choosing the time quantum, by dynamically varying the time-quantum based upon the burst time of the completed process and process in the ready queue. The solution tends to be more promising, when compared to the traditional RRA, with respect to the factor of Average waiting time of the process in the ready queue.

Literature Background: RRA is one of the simplest CPU scheduling algorithms, which assigns 'timespan' known as time slice to each of the process in the circular queue, thereby handling all the processes without priority. RRA ensures that each process that is required to complete a job gets ample amount of run-time and hence preventing starvation of process [1].

Consider a user of the computer system, simultaneously starts four applications, namely mp3 player, game application, word processor and a spreadsheet application. All the four applications are loaded into the main memory, as processes and each of the process is allocated the processor without any priority. The sharing of resources between processes is handled by employing the RRA. The RRA tends to perform well because; each application receives a certain time quantum, per processor cycle. Processor cycle is the amount of time; it takes to manage each process running one time [9].

With the example quoted above the applications provide a short cycle for the processor and more time could equally be allotted to each of these four processes, thereby making them appear to perform better with a end-users point of view. If RR strategy is not employed, the application which acquires the processor first, would monopolize the processor [9], there by providing a way for starvation of other processes. Hence the employment of RR strategy helps to keep track with end user and efficiently tackle all the four processes. RR strategy can also be applied for any real-time process scheduling and packet scheduling in computer networks.

If the size of the tasks varies in accordance with time, then the RR scheduling strategy would not be desirable to apply. The process that produces larger jobs would be favored as against other processes. In such cases of dynamically varying task/ job fair queuing would be preferred [11].

RR strategy would be an efficient promising alternative as against FCFS queuing in best effort packet switching and other statistical multiplexing techniques in wireless/wired networks. A multiplexer, which provides RR scheduling as a dedicated queue for every data flow, identified by address (source address and destination address). The algorithm allows every data packet in the queue, to take up turns in the transferring packets in a shared channel in a periodically repeated order. The scheduling tends to be non-conserving meaning that if one flow is out of packets; the next data flow will take up this place [10]. Choosing the time quantum of a RR scheduling is very important. If the time quantum is too small, the number of context switches increases tremendously, thereby lowering the efficiency of the processor [7]. It is to be noted that the context switch is a pure overhead, since no useful work is getting done when the processes fluctuated between CPU.

RRA tend to give good responsiveness but worst average waiting time and turnaround time [12] [13]. RRA have a low-scheduling overhead O(1) complexity, which means that scheduling the following process, takes up a constant time. Since performance of RR is largely dependent upon the time quantum, in this paper, efforts has been made to study the performance of the algorithm by varying the time quantum.

Proposed Methodology: As discussed in the previous section, emphasis is to be laid in choosing the time quantum, for the RR scheduling to yield fair results. So the proposed methods focuses on dynamically varying this time quantum based upon the worst time of the processes in the Ready Queue (RQ) or Completed Queue (CQ) (The processes that finishes the execution in CPU are queued up in a queue called completed queue) , This new time quantum determined by the burst time of processes in the ready queue or completed queue yields a real optimal value because time quantum variation is based upon the real burst time.

Dynamically varying the time quantum based upon the burst time of the process in the completed queue or ready queue:

During the first cycle (cycle represents allocating the static time quantum to all the processes once) of scheduling, the static time quantum is adopted. At the end of first cycle, if any of the processes completes its execution, it is pushed on to a queue called completed queue. If no process completes its execution in the first cycle, the next cycle is carried out with the same static time quantum. The same is followed until a process arrives at the completed queue. Once the process arrives at the completed queue, the quantum is changed according to the variance of the burst time of the processes available in the completed queue, if the time quantum is varied dynamically, based upon the processes in the ready queue, the static time quantum is adopted during the first cycle of scheduling. From the second cycle, the time quantum is dynamically changed, according to the variants of burst time of process available in the ready queue. The variants are shown below:

Q: Static Time Quantum
MQ: Modified Time Quantum
BT: Burst Time
CQ: Completed Queue
RQ: Ready Queue

1. $MQ = \sum_{i=1}^{n} BT(P_i) \forall P_i \in$

2. $MQ = Q + \min(BT(P_i)) \forall P_i \in$

3. $MQ = Q + \max(BT(P_i)) \forall P_i \in$

4. $MQ = Q + avg(BT(P_i)) \forall P_i \in$

5. $MQ = Q + \left[\dfrac{\min(BT(P_i)) + \max(BT(P_i))}{2} \right] \forall P_i \in CQ$

6. $MQ = Q + recent(BT(P_i)) \forall P_i \in$

7. $MQ = Q + [\min(BT(P_i))] \wedge 2 \forall P_i \in$

8. $MQ = Q + [\max(BT(P_i))] \wedge 2 \forall P_i \in$

9. $MQ = Q + \left[\dfrac{\min(BT(P_i)) + \max(BT(P_i))}{2} \right] \forall P_i \in CQ$

The generic algorithm for the variants is shown below. It is to be noted that the algorithm works on iteratively, until all the processes in the ready queue gets executed.

The same variants are also applicable for processes in the ready queue.

Experimental Simulation: The proposed algorithmic variants were evaluated to meet the primary scheduling criteria: to minimize the average waiting time. T o evaluate the proposed variants, four different scenarios are studied. The scenarios are as follows:

Table 1: Algorithm: Dynamic Time Quantum Variation (DTQV)

```
n:Number of process to be executed
Pi: ith process
CQ: Completed Queue
RQ: Ready Queue
AT:Arrival Time
FT:Finish Time
ST:Start Time
T,TmpQ=Time
Input: n, Pi, AT(Pi)
Output :Average Waiting TimeTime Quantum is varied dynamically based
on burst time of process in CQ/RQ
Initialization: ST(Pi)=WT(Pi)=0,T,TmpQ,
fori is 1 to n
Begin
    Get BT(Pi);
GetAT(Pi);
    FT(Pi)=AT(Pi);
End
Get(Q);
while(RQ<>NULL)
begin
for I is 1 to n
    begin
if BT(Pi)>=Q
then
    ST(Pi)=T;
    BT(Pi)=BT(Pi)-Q;
    T=T+Q;
    WT(Pi)=ST(Pi)-FT(Pi);
    FT(Pi)=T;
Else
    ST(Pi)=T;
    BT(Pi)=0;
    T=T+Q;
Tmp Q=Q-BT(Pi);
T=T-TmpQ;
Wt(Pi)=ST(Pi)-FT(Pi);
FT(Pi)=T;
Enqueue(CQ,Pi);
Dequeue(RQ,Pi);
End If
End
Q=MQ
End
```

- Arrival time of all the process is equal to zero and the time quantum is varied dynamically, based upon burst time of process in completed queue (CQ)
- Process arrival at different arrival times and the time quantum is varied dynamically, based upon burst time of process in completed queue (CQ)
- Arrival time of all the process is equal to zero and the time quantum is varied dynamically, based upon burst time of process in ready queue (RQ)
- Process arrival at different arrival times and the time quantum is varied dynamically, based upon burst time of process in ready queue (RQ)

Scenario 1 and Scenario 3 are validated for three different cases:

Case:A
Process Burst Time: Small
Time Quantum: Small

Assume six processes arrive at time T=0. The burst time of the processes are assumed to be P_1=1, P_2=3, P_3=2, P_4=5, P_5=7 and P_6=6. The static time quantum is assumed to be 3. (all the time constraints are in milliseconds (ms))

Case: B
Process Burst Time: Large
Time Quantum: Small

Assume six processes arrive at time T=0. The burst time of the processes are assumed to be P_1=200, P_2=800, P_3=400, P_4=500, P_5=600 and P_6=1000. The static time quantum is assumed to be 3.

Case: C
Process Burst Time: Large
Time Quantum: Large

Assume six processes arrive at time T=0. The burst time of the processes are assumed to be P_1=200, P_2=800, P_3=400, P_4=500, P_5=600 and P_6=1000. The static time quantum is assumed to be 30.

Scenario 2 and Scenario 4 are validated for five different cases:

Case: 1
Process Burst Time: Small
Time Quantum: Small
Process Arrival Time: Small

Assume six processes, P_1, P_2, P_3, P_4, P_5 and P_6 arrive at time T=0, 1, 3, 6, 7 and 9 respectively. The burst time of the processes are assumed to be P_1=1, P_2=3, P_3=2, P_4=5, P_5=7 and P_6=6. The static time quantum is assumed to be 3.

Case: 2
Process Burst Time: Large
Time Quantum: Small
Process Arrival Time: Small

Assume six processes, P_1, P_2, P_3, P_4, P_5 and P_6 arrive at time T=0, 1, 3, 6, 7 and 9 respectively. The burst time of the processes are assumed to be P_1=200, P_2=800, P_3=400, P_4=500, P_5=600 and P_6=1000. The static time quantum is assumed to be 3.

Case: 3
Process Burst Time: Large
Time Quantum: Small
Process Arrival Time: Large

Assume six processes, P_1, P_2, P_3, P_4, P_5 and P_6 arrive at time T=50, 70, 100, 120, 128 and 300 respectively. The burst time of the processes are assumed to be P_1=200, P_2=800, P_3=400, P_4=500, P_5=600 and P_6=1000. The static time quantum is assumed to be 3.

Case: 4
Process Burst Time: Large
Time Quantum: Large
Process Arrival Time: Small

Assume six processes, P_1, P_2, P_3, P_4, P_5 and P_6 arrive at time T=0, 1, 3, 6, 7 and 9 respectively. The burst time of the processes are assumed to be P_1=200, P_2=800, P_3=400, P_4=500, P_5=600 and P_6=1000. The static time quantum is assumed to be 30.

Case: 5
Process Burst Time: Large
Time Quantum: Large
Process Arrival Time: Large

Assume six processes, P_1, P_2, P_3, P_4, P_5 and P_6 arrive at time T=50, 70, 100, 120, 128 and 300 respectively. The burst time of the processes are assumed to be P_1=200, P_2=800, P_3=400, P_4=500, P_5=600 and P_6=1000. The static time quantum is assumed to be 30.

RESULTS AND DISCUSSION

Arrival time of all the process is equal to zero and the time quantum is varied dynamically, based upon burst time of process in CQ/ RQ:

A simulator was designed, to evaluate the cases A, B and C. The performance of the variants is evaluated for each of the cases using the simulator based upon the factor of minimizing the average waiting time. The percentage of improvement of each of the variant when compared to traditional RRA, is shown in Table 1 (time quantum is varied dynamically based upon burst time of process in CQ) and Table 2 (time quantum is varied dynamically based upon burst time of process in RQ).

For Case A:

The average waiting time for all the variants tends to produce less average waiting time when compared to generic RRA. Simulation results show that the RRA with dynamically varying quantum (based upon burst time of process in RQ and CQ) is 3.921% more efficient when compared to static time quantum RRA. The same is inferred from Fig. 1.

For Case B:

The variant 1 and 3, applied on process chosen for m Ready Queue, yields very good result, nearly 42.2% more efficient when compared to static time quantum RRA. The other variants tend to be 25%-28% when compared to static time quantum RRA. The same is inferred from Fig. 2.

For Case C:

The variant 1 and 3, applied on process chosen for m Ready Queue, yields very good result, nearly 38% more efficient when compared to static time quantum RRA. The other variants tend to produce considerable improvements in the reduction of average waiting time, when compared to static time quantum RRA. The same is inferred from Fig. 3.

Process arrive at different arrival times and the time quantum is varied dynamically, based upon burst time of process in RQ/CQ

The percentage of improvement of each of the variant when compared to traditional RRA, is shown in Table 3 (time quantum is varied dynamically based upon burst time of process in CQ) and Table 4 (time quantum is varied dynamically based upon burst time of process in RQ).

For Case 1:

The average waiting time for all the variants tends to produce less average waiting time when compared to generic RRA. Simulation results show that the RRA with dynamically varying quantum (based upon burst time of process in RQ and CQ) is 8% more efficient when compared to static time quantum RRA. The same is inferred from Fig. 4.

For Case 2:

The variant 1 and 3, applied on process chosen from Ready Queue, yields very good result, nearly 42.2% more efficient when compared to static time quantum RRA. The other variants tend to be 8.7%-27.4% when compared to static time quantum RRA. The same is inferred from Fig. 5.

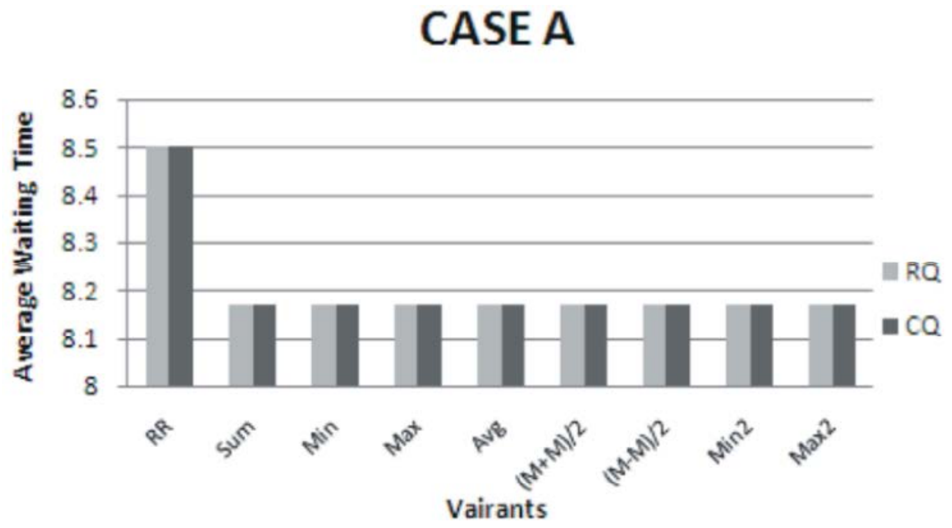

Fig. 1: Waiting time for Case A, when the time quantum is varied dynamically based on the burst time of the process in RQ and CQ

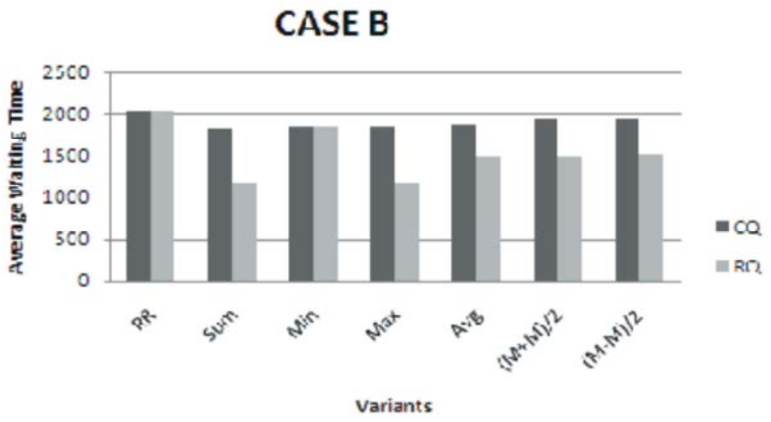

Fig. 2: Waiting time for Case B, when the time quantum is varied dynamically based on the burst time of the process in RQ and CQ

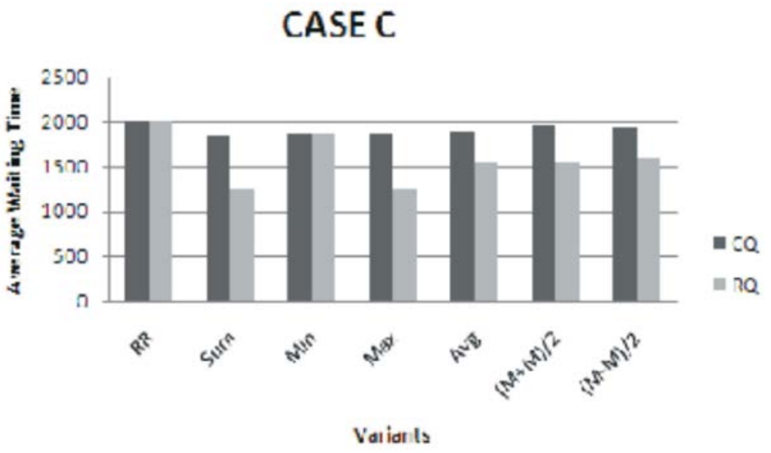

Fig. 3: Waiting time for Case C, when the time quantum is varied dynamically based on the burst time of the process in RQ and CQ

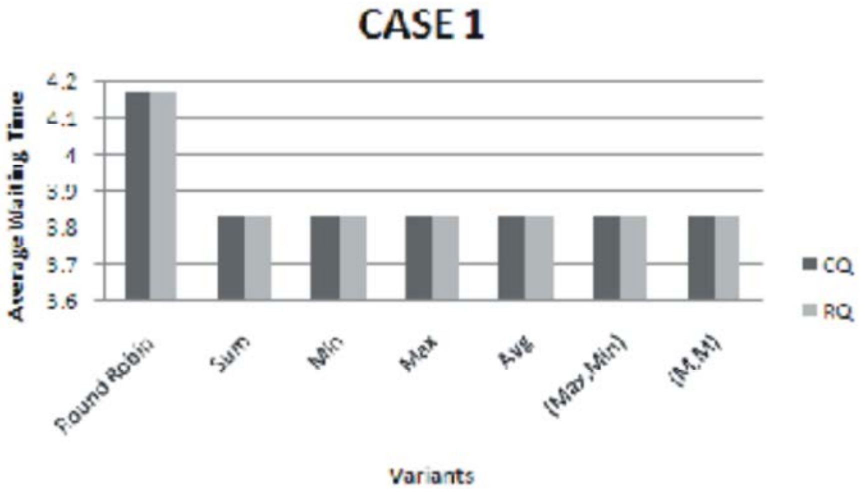

Fig. 4: Waiting time for Case 1, when the time quantum is varied dynamically based on the burst time of the process in RQ and CQ

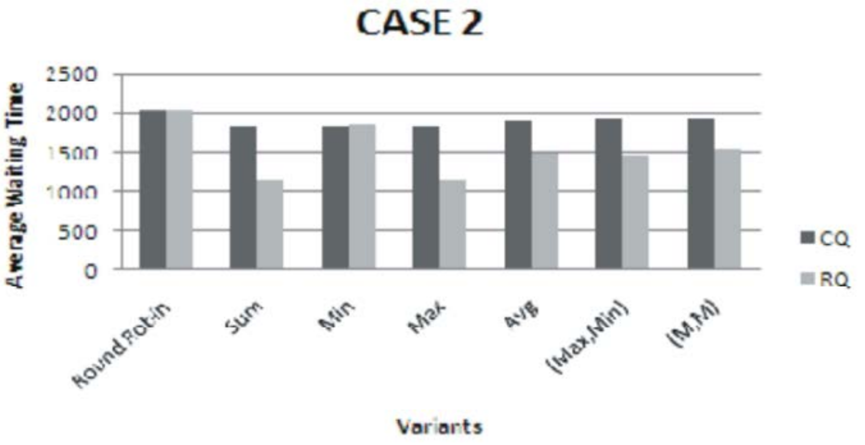

Fig. 5: Waiting time for Case 2, when the time quantum is varied dynamically based on the burst time of the process in RQ and CQ

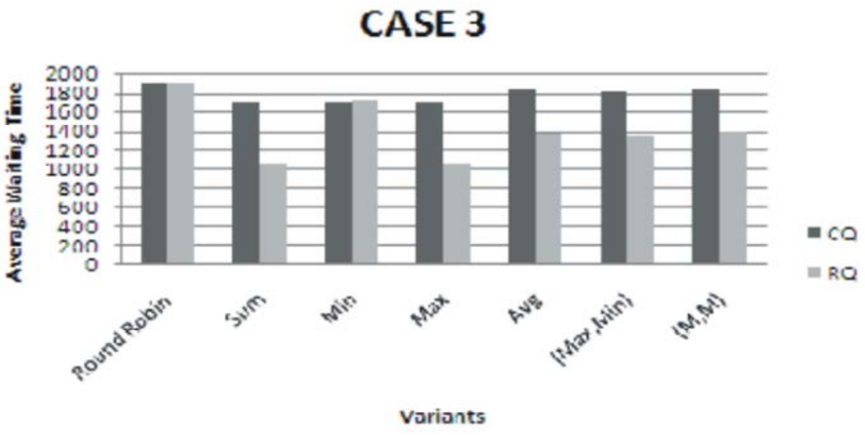

Fig. 6: Waiting time for Case 3, when the time quantum is varied dynamically based on the burst time of the process in RQ and CQ

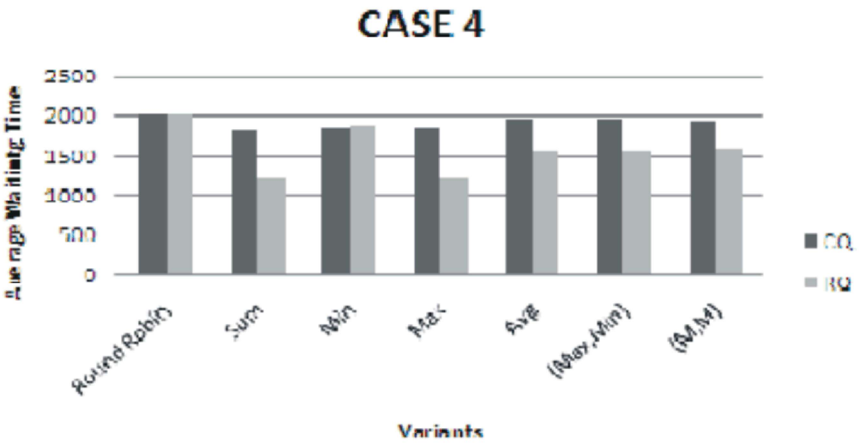

Fig. 7: Waiting time for Case 4, when the time quantum is varied dynamically based on the burst time of the process in RQ and CQ

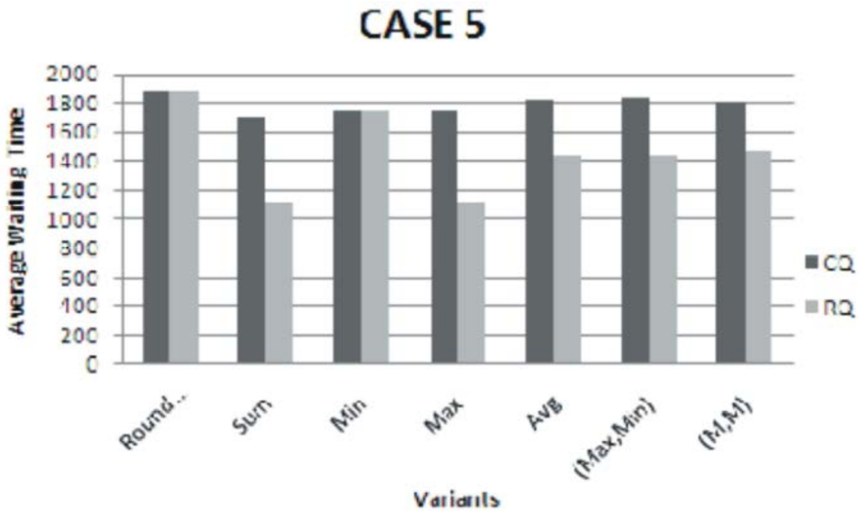

Fig. 8: Waiting time for Case 5, when the time quantum is varied dynamically based on the burst time of the process in RQ and CQ

For Case 3:

The variant 1 and 3, applied on process chosen from Ready Queue, yields very good result, nearly 45% more efficient when compared to static time quantum RRA. The other variants tend to be 9.3%-29.1% when compared to static time quantum RRA. The same is inferred from Fig. 6.

For Case 4:

The variant 1 and 3, applied on process chosen from Ready Queue, yields very good result, nearly 38.1% more efficient when compared to static time quantum RRA. The other variants tend to be 6.4%-22.4% when compared to static time quantum RRA. The same is inferred from Fig. 7.

For Case 5:

The variant 1 and 3, applied on process chosen from Ready Queue, yields very good result, nearly 40.56% more efficient when compared to static time quantum RRA. The other variants tend to be 6.8%-23.8% when compared to static time quantum RRA. The same is inferred from Fig. 8.

Conclusion and Future Work: In this paper, the generic Round Robin scheduling has been studied under the aspect of dynamically varying quantum size. Observations are made for calculating the average waiting time of the process by dynamically changing the time quantum based on the burst time of the process in the ready queue or completed queue. The observations are carried out for both the scenarios of zero arrival time and process arriving at different times. A number of variants are proposed for dynamically varying the time quantum. Among the proposed variants, the variant in which the time quantum is varied by replacing the sum of the burst time of the process in the ready queue and the variant in which the time quantum is varied by adding up the maximum of burst time of process in ready queue tends to be nearly 40% more efficient when compared to static time quantum Round Robin scheduling. Other variants also produce considerable improvement when compared to static time quantum Round Robin scheduling.

As a future work, heuristic techniques may be applied to dynamically choose the variant for dynamically varying the time quantum specific to an application. Extensive research can also be carried out to apply the modified Round Robin algorithm for real-time scheduling applications.

REFERENCES

1. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, 20056. “Operating system concepts”, John Wiley and sons, 6th Edition, 2005.
2. Place, J., 1989. FCFS.A novel scheduling policy for a tightly coupled parallel computer systems proceedings of the ACM Annual Conference on Computer Science, Feb 21-23, ACM Press, Louisville, Kentucky, PP.188-194, DOI:10.1145/75427.75450.
3. Zhao, W. and J.A. Stankovic, 1989. Performance Analysis of FCFS and improved FCFS scheduling algorithm for dynamic real-time computer systems proceedings of IEEE Real-time computer systems symposium, Dec 1989, pp: 156-165.
4. Lupetti, S. and D. Zagorodaov, 2006. Data popularity and shortest job first scheduling of network transfers. Proceedings of the International conference on Telecommunications, August 29-31, IEEE computer society USA, pp: 26. DOI:10.1109/ICDT.2006.28.
5. Sandmann, W., 2006. Benefits of alternating FCFS/SJF service order. Proceeding of the 6th WSEAS International conference on Applied Informatics and communications, August 18-20, Word scientific and Engineering Academy and Society. Stevens Point, Wisconsin, USA:194-199.
6. Back, D.S., K. Pyun, S.M. Lee, J. Cho and N. Kim, 2007. Hierarchical deficit Round-Roubin scheduling algorithm for a high level fair service, Proceedings of the International Symposium in Information Technology Convergence, Nov. 23-24, IEEE Computer Society, Washington D.C., USA, pp: 115-119.
7. Rami J. Matarneh, 2009. “Self Adjustment Time Quantum in Round Robin Algorithm depending on burst time of the now-running processes”, American Journal of Applied Sciences, 6(10): 1831-1837.
8. Helmy, T. and A. Dekdouk, 2007. Burst Round Robin as a Proportional Share Scheduling Algorithm, IEEE GCC, KingFahed University.
9. http://ww.wisegeek.com/what-is-round-robin-scheduling.htm.
10. http://en.wikipedia.org/wiki/Round-Robin-Scheduling#Process_Scheduling.
11. Shreedhar M. and G. Verghese, 1995. “Efficient Fair Queueing using Deficit Round Robin”, in proceedings of ACM SIG COMM ’95, 4(3): 231-242.
12. Chaskar, H.M. and U. Madhow, 2003. “Fair Scheduling with tunable latency: a round-robin approach”, IEEE/ACM Trans. Net, 11(4): 592-601.
13. Luca Abeni, Giuseppe Lipari and Giorgio Buttazzo, 1999. “Constraint Bandwidth vs. Proportional Share Resource Allocation”, in Proceedings of IEEE International Conference on Multimedia Computing and Systems, Florence, Italy.
14. Wang Y. and A. Merchant, 2006. “Proportional Service Allocation in distributed storage systems”, Technical Report HPL- 2006-184, HP Laboratories.
15. Balamurugan Shanmugam, Visalakshi Palaniswami, 2013. Modified Partitioning Algorithm for Privacy Preservation in Microdata Publishing with Full Functional Dependencies”, Australian Journal of Basic and Applied Sciences, 7(8): 316- 323.
16. Balamurugan Shanmugam, Visalakshi Palaniswami, R. Santhya and R.S. Venkatesh, 2014. Strategies for Privacy Preserving Publishing of Functionally Dependent Sensitive Data: A State-of-the-Art Survey: Australian Journal of Basic and Applied Sciences, 8(5).
17. Balamurugan, S. and S. Charanyaa, 2014. "Principles of Social Network Data Security" LAP Verlag, Germany, ISBN: 978-3-659-61207-7.