Dynamic Priority-based Round Robin Algorithm: Design, Implementation and Performance Analysis

COEN 283 - Operating Systems
Santa Clara University

Jasjeet Dhaliwal
Sui Fung Alex Wong

Table of Contents

1. Abstract

The primary purpose of CPU utilization research has been to develop online scheduling algorithms that are O(1)-speed O(1)-competitive with the Optimal scheduling algorithm. The two most popular objectives for evaluation of a scheduling algorithm are fairness and job latency. Round Robin guarantees instantaneous fairness by allocating CPU time equally to every job at any given time instant while also minimizing variance and job latency. We have developed a scheduling algorithm that dynamically allocates CPU time to jobs based on their priority. We include a mathematical analysis comparing the performance of this algorithm to the Optimal scheduling algorithm and the Round Robin scheduling algorithm.

2. Introduction

Objective

To develop a Dynamic Priority-Based Round Robin scheduling algorithm that is O(1)-speed O(1)-competitive with the Optimal scheduling algorithm and the Round Robin scheduling algorithm.

What is the problem?

System performance is directly correlated with the efficiency of resource utilization, especially CPU. The problem of developing an online scheduling algorithm that optimizes CPU utilization is crucial in enhancing the performance of the system.

Why this is a project related to this class?

It focuses on developing a O(1)-speed O(1)-competitive scheduling algorithm, a topic crucial to the performance of any operating system.

Why other approach is no good? And why you think your approach is better?

The Round Robin algorithm does not assign any priority to jobs. This approach is potentially problematic when analyzing an interactive system. Therefore, our approach is better since it assigns priorities to jobs and ensures that jobs with a higher priority get a larger portion of CPU time.

Statement of the problem

The Round Robin scheduling algorithm is O(1)-speed O(1)-competitive with the Optimal scheduling algorithm on flow time and l_2-norm but does not assign priorities to jobs. This is a potential problem in interactive systems where certain jobs can be more urgent than others.

Area or scope of investigation

We aim to develop a self-sufficient scheduling algorithm that is:

- Dynamically assigns priorities to jobs.
- O(1)-speed O(1)-competitive with the Optimal scheduling algorithm on flow time and l_2-norm.
- O(1)-speed O(1)-competitive with the Round Robin scheduling algorithm on flow time and l_2-norm.

3. Theoretical Basis and Literature Review

Definition of the problem

Lack of dynamic priority based CPU time allocation in the traditional Round Robin scheduling algorithm. The traditional Round Robin scheduling algorithm does not account for allocating CPU time to jobs based on their importance to users of an interactive system.

Theoretical background

Developing a scheduling algorithm that optimizes CPU utilization is crucial in improving performance of a system. The traditional Round Robin scheduling algorithm guarantees instantaneous fairness by allocating CPU time equally to every job at any given time instant while also minimizing variance and job latency. However, it does not have the ability to assign priorities to jobs and complete those ones that have a higher priority first.

Related research to solve the problem

Below are a few of the research that we come across:

"A Task set based Adaptive Round Robin (TARR) scheduling algorithm for improving performance"
In this paper, the authors created a subset of the processes and called it task set. For the task set, the authors calculate a new time quantum in each iteration for all of the processes in the set, and having a valid range on the time quantum such that it will be a reasonable value among the task set.

"Temporal Fairness of Round Robin: Competitive Analysis for Lk-norms of Flow Time",
This paper provides an in-depth analysis of the mathematical analysis of the Round Robin algorithm in a multiple machine setting. It introduces the concepts of O(k)-competitiveness, flow time, and and l_k-norm. In addition, this is the first paper that successfully proves that the traditional Round Robin scheduling algorithm is O(1)-competitive with the Optimal scheduling algorithm based on the l_2-norm metric.

Advantage and Disadvantage of the above mentioned research

Advantage: By making the time quantum dynamic, the average turnaround time and the average waiting time can be reduced, since the value of the time quantum can be more appropriate for each process.

Disadvantage: The calculation for the time quantum value could be complicated, and it is possible to take extra time to compute the new value in each update.

Our solution to solve this problem

Our structure will be a doubly circular linked list to hold all the processes. When the CPU pointer is walking through the linked list, it calculates the time quantum on each node that it stops, then move to the next node after waiting on the current node for time quantum.
The calculated time quantum value will be bounded. This will prevent a process from holding the CPU as hostage for a long period of time, and it will also ensure an optimal swap rate of processes in order to optimize performance.

Where our solution different from others

Our algorithm is different from the traditional Round Robin scheduling algorithm because it dynamically assigns a time quantum based on the the priority of the job before every iteration.

Why our solution is better

Our algorithm introduces the ability to assign a time quantum to jobs based on their priority which directly enhances end user satisfaction in an interactive system.

4. Hypothesis

A scheduling algorithm that assigns CPU time based on dynamically changing priorities will be temporally fair, minimize job latency, and will be O(1)-speed O(1)-competitive with the Optimal scheduling algorithm as well as the Round Robin scheduling algorithm.

5. Methodology

How to generate/collect input data

We will randomly generate different sets of input with a list of processes. Each set of input contains a list of processes, the arriving time and total running time for each of the processes. Then we will parse the same input set to each algorithm we implemented.

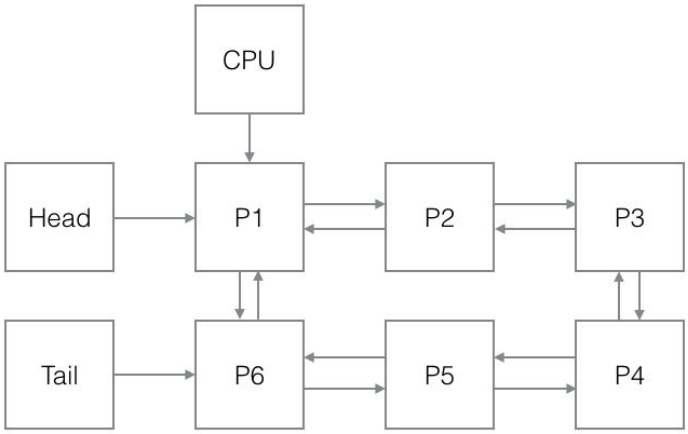
How to solve the problem

In order to solve the problem, first we need to design a doubly circular linked list to hold all the processes. When a process is getting into the system (the circular linked list), the process is inserted at the tail, and we will have the CPU pointer to walk through the linked list until the all the process procedures are executed.

When walking through the list, it calculates the new time quantum for the corresponding process, then “wait” for this amount of time, update the remaining time and move to the next process.

Algorithm Design

The CPU pointer will walk from head of the circular linked list, whenever it takes a step, it calculates the time quantum required for the corresponding process. Since it is a circular linked list, the CPU will walk until all the process are being executed (as they will be removed from the list). Newly arrive processes will be inserting at the tail of the list.



Formula to calculate time quantum (prior to a job entering the CPU):

$$DPRRA\ Time\ Quantum = min(Q1 + \frac{Process\ Waiting\ Time}{Total\ Waiting\ Time}(Q3 - Q1),\ Q3)$$

$$RRA\ Time\ Quantum = Q2$$

where

$$Q1 = 1st\ quartile\ of\ process\ burst\ time\ data\ set$$
$$Q2 = 2nd\ quartile\ of\ process\ burst\ time\ data\ set$$
$$Q3 = 3rd\ quartile\ of\ process\ burst\ time\ data\ set$$

This value is also bounded by:

$$Minimum\ Time\ Quantum = Q1$$
$$Maximum\ Time\ Quantum = Q3$$

After the process procedure is being executed, we will be calculating the following:
Average Flow Time among n processes:

$$\frac{1}{n}\sum_{i=1}^{n}(C_i - R_i)$$

L2-norm of the Average Flow Time among n processes:

$$\sqrt{\sum_{i=1}^{n}(C_i - R_i)^2}$$

where

$$C_i = Completion\ Time\ of\ Process\ i$$
$$R_i = Arrival\ Time\ of\ Process\ i$$

Language used

We will be using C++ for this project.

Tools used

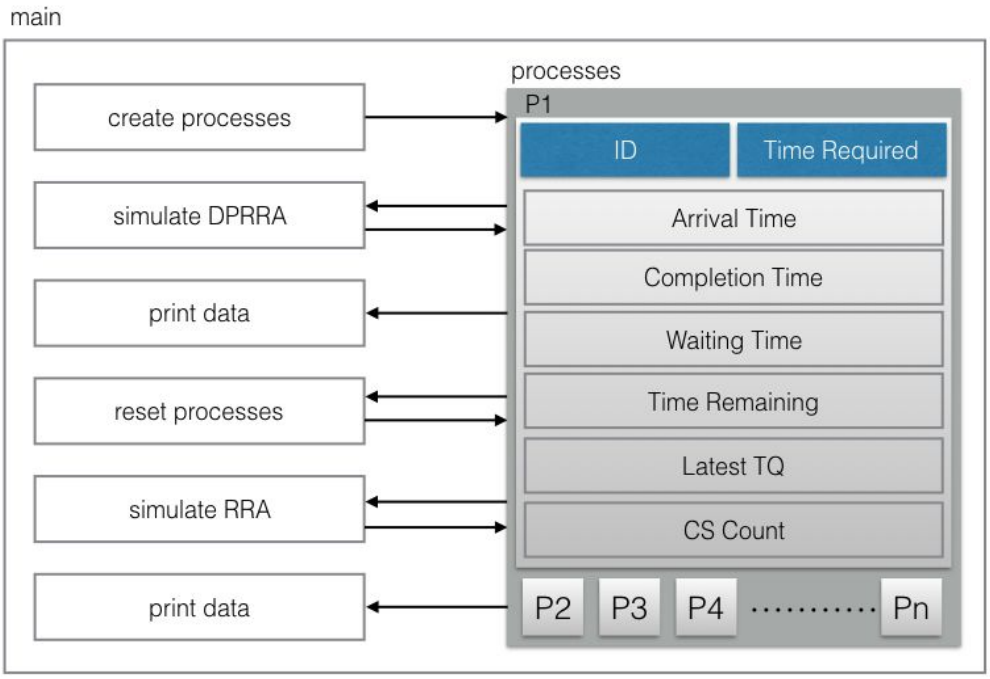
Terminal
gcc/g++
Makefile
Xcode

How to generate output

We will use multiple sets of process procedures, and use them for both our algorithm and the traditional round robin. The program will be able to log the waiting time for each process on each algorithm, then we can compare the average flow time and the L2-norm of the average flow time between the algorithms and obtain the result.

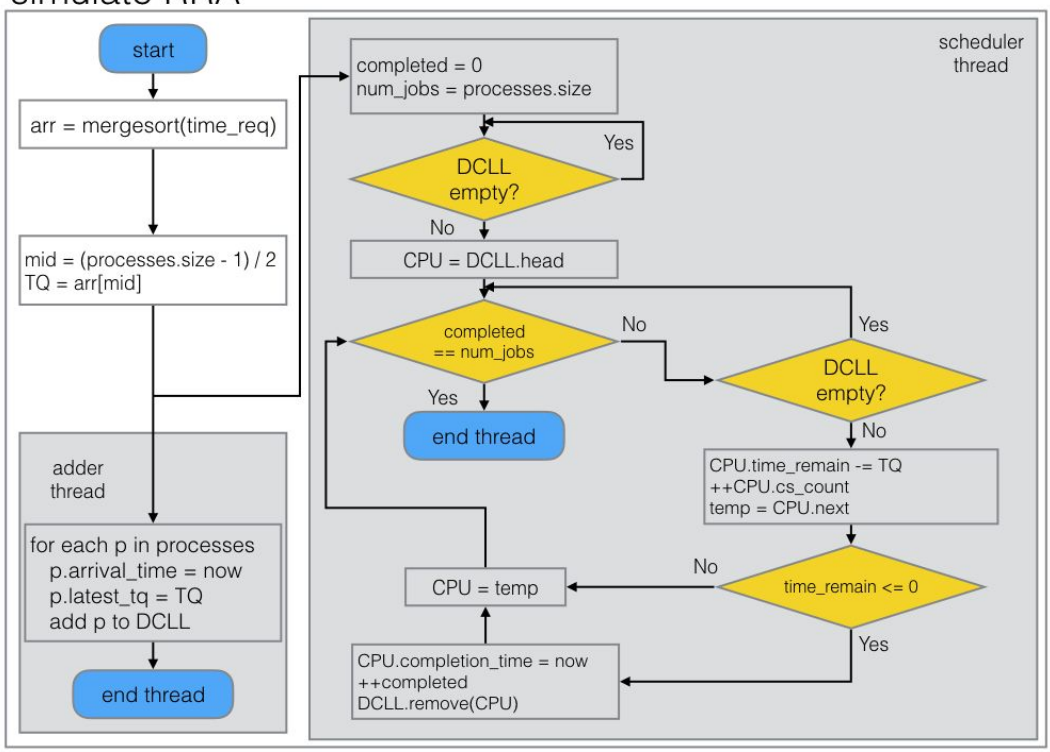
6. Implementation

Design Framework

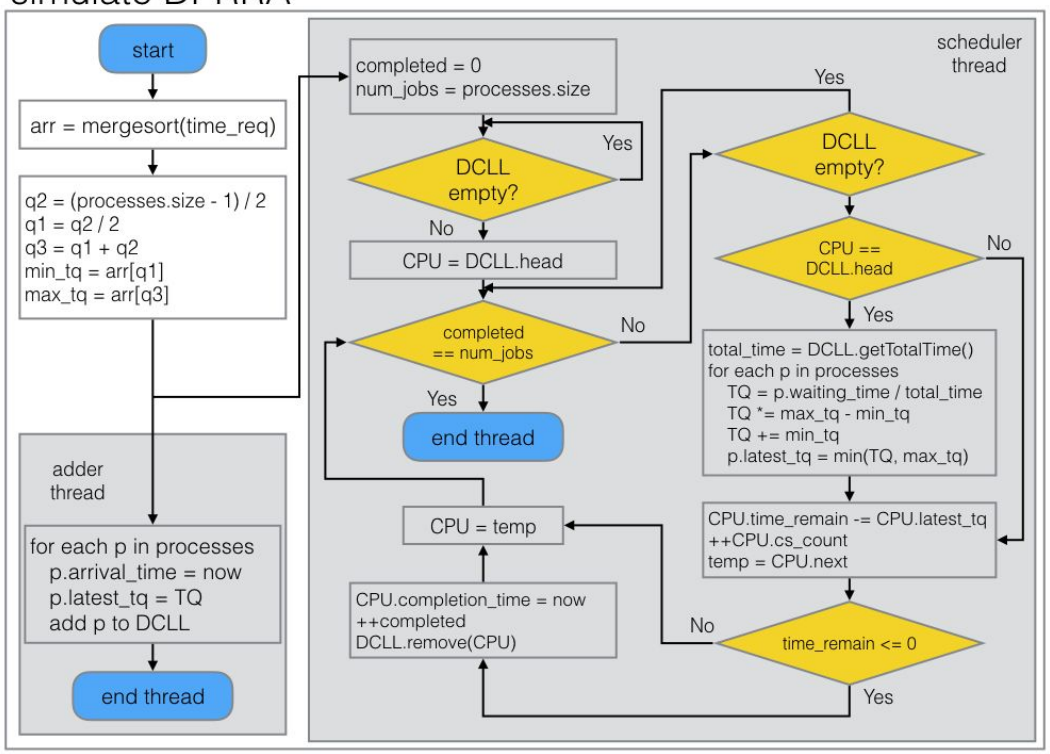


Flowchart

simulate RRA



simulate DPRRA



7. Data Analysis and Discussion

Analysis:
In order to validate the hypothesis our algorithm (DPRRA) needed to be O(1) speed, O(1) competitive with the traditional Round Robin Algorithm for the Average Flow Time and L2-Norm.

Output Generation:
We scheduled a set X of processes first on DPRRA and then on RRA. We collected the following performance metrics of each algorithm:

1. Average Flow Time
2. L2-Norm
3. Average Context Switches
4. Average Throughput
5. Average Time Quanta

Each metric requires the measurement of a time unit. The time unit we used was **clock ticks.** Below are the bounds that we used on each set:

Max CPU Time Required = 5000 clock ticks
Min CPU Time Required = 1 clock tick

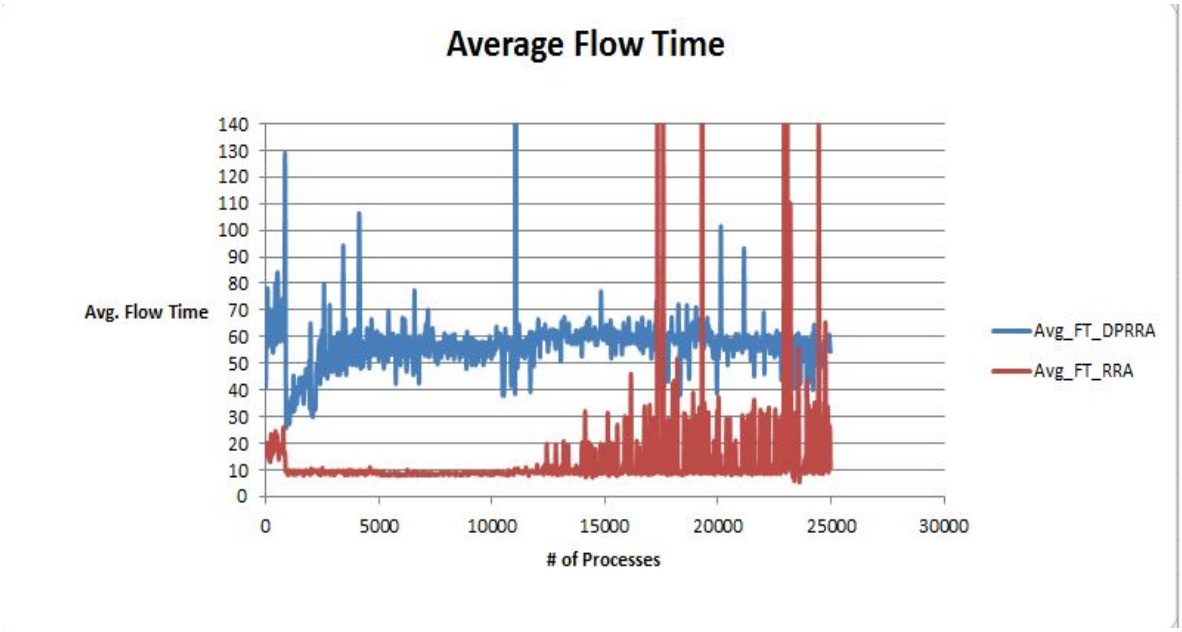
Max # of Processes in a set Xi = 24980
Min # of Processes in a set Xi = 5

Total # of iteration of the above test = 1000

The output was collected in .csv format and plotted using scatter plots. The results are displayed below.

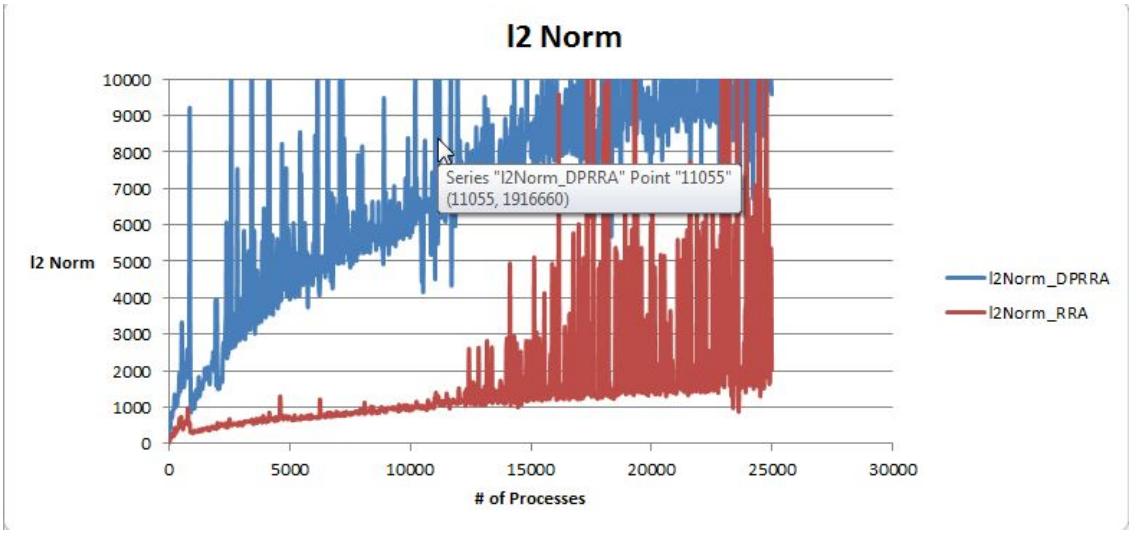
1. Average Flow Time:

DPRRA is O(1) speed O(5.96) competitive with RRA.



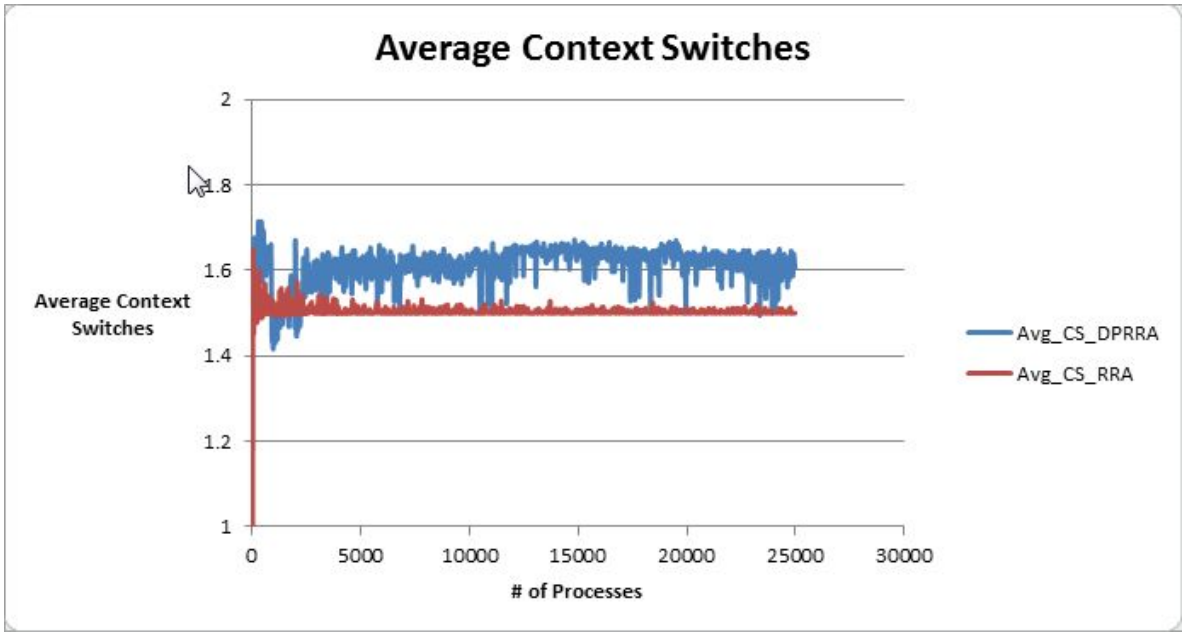
2. L2-Norm

DPRRA is O(1) speed O(7.20) competitive with RRA.



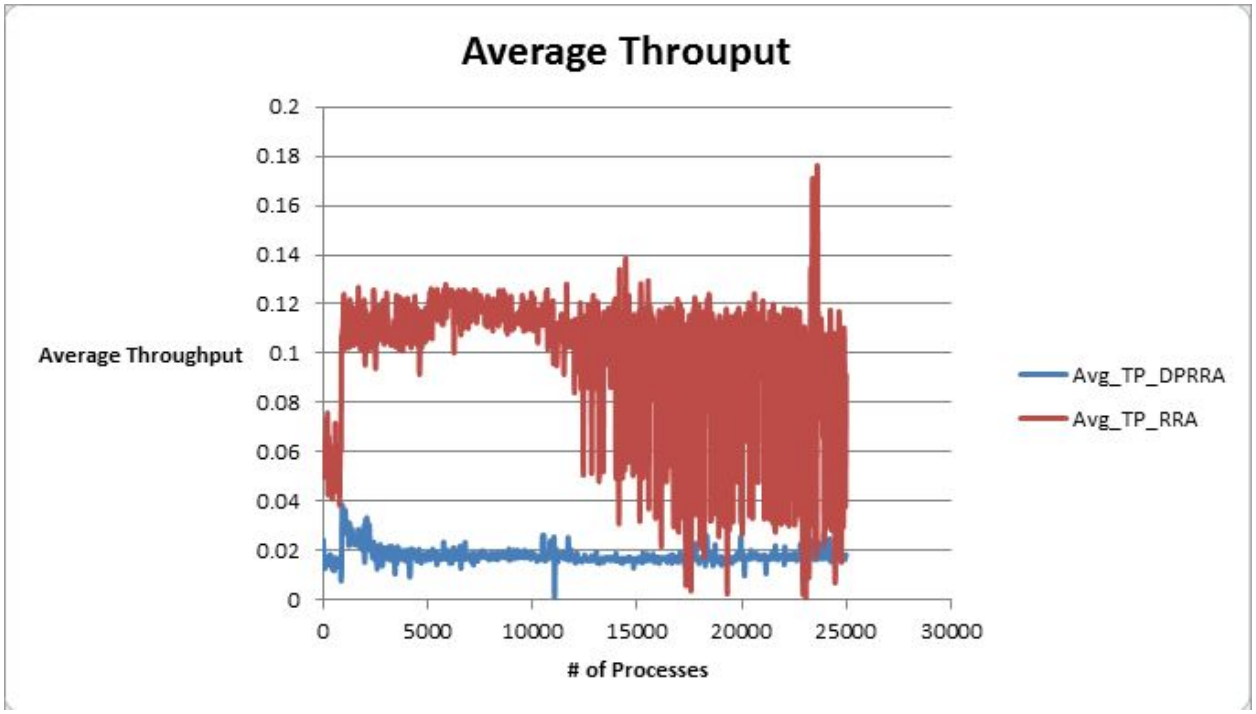
3. Average Context Switches

DPRRA is O(1) speed O(1.07) competitive with RRA.



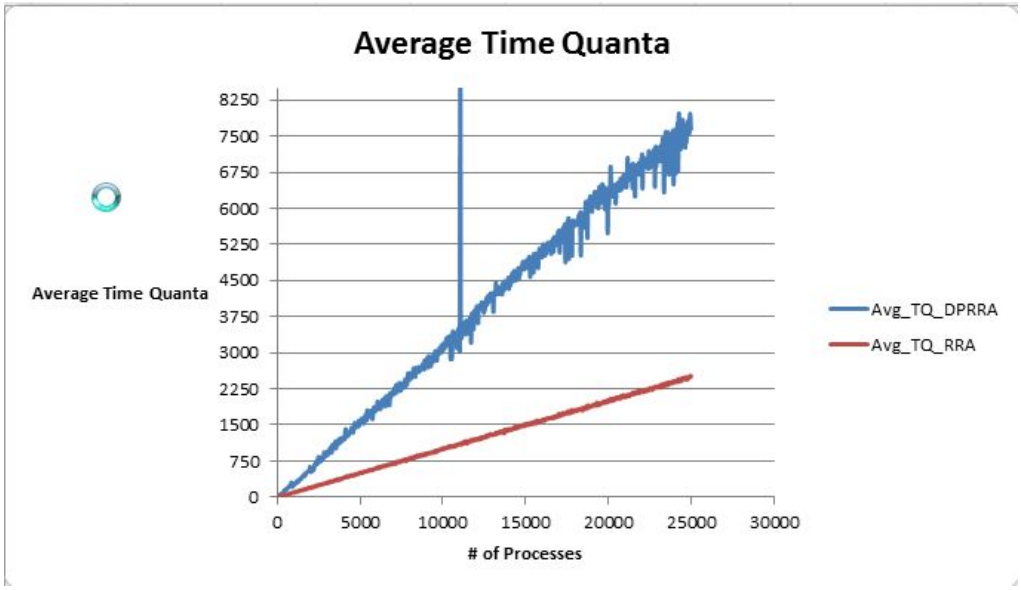
4. Average Throughput:

DPRRA is O(1) speed O(5.96) competitive with RRA.



5. Average Time Quanta:

This is not a competitiveness comparison metric.



8. Conclusion

Summary and Conclusion

Based on the output that we gathered, Dynamic Priority-based Round Robin Algorithm scatter plot has a similar shape in average flow time, L2-norm flow time, average context switches and average time quanta to the traditional Round Robin Algorithm.

However, Dynamic Priority-based Round Robin Algorithm is does not perform in O(1)-speed O(1)-competitive in the comparison metrics, as it is required to run an extra loop in order to update time quantum for each processes within the ready list. Even though this calculation is performed only once every loop through the list of ready processes, the overhead involved in calculating a new time quantum and updating the time quanta for every process severely impacts the performance of the algorithm.

Recommendation for Future Studies

Dynamic Priority-based Round Robin Algorithm can be improved by addressing two problems: a) Overhead associated with calculating dynamic time quanta b) Calculating Optimal Time Quanta to be allocated to each process.

The overhead associated with calculating time quanta can be improved by experimenting with the algorithm used for calculation as well as with the frequency of the calculation.

Calculating the optimal time quanta might be an intractable problem but getting close to optimal time quanta value might be achievable using experimentation and mathematical tools for comparison. Thus, improving the equation for the Time Quantum is might make this approach viable, but it needs to be treated carefully, as the calculation might introduce lots of overhead.

Our original approach was to calculate the time quantum at each node, which introduced lots of overhead, as the total waiting time was calculated with O(n) speed, and had to be done at each node.

With our current approach, the calculation of the time quanta was reduced to the head node only. Even though it helped us reduce the amount of calculation overhead it also reduced our accuracy on the calculation of the optimal time quanta, as the waiting time and total waiting time lose precision “incorrect” with respect to the “current time” on other nodes.

9. Bibliography

N. K. Rajput, A. Kumar, “A Task set Based Adaptive Round Robin (TARR) scheduling algorithm for improving performance”, 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), Feb 2015.

R. Racu, Li Li, R. Henia, A. Hamann, R. Ernst, “Improved Response Time Analysis of Tasks Scheduled under Preemptive Round-Robin”, 2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codedesign and System Synthesis (CODES+ISSS), Sep 2007.

Sungjin Im, Janardhan Kulkarni, Benjamin Moseley, “Temporal Fairness of Round Robin: Competitive Analysis for Lk-norms of Flow Time”, 2015 27th ACM Symposium on Parallelism in Algorithms and Architecture (SPAA), Jun 2015.

Tong Li, Scott Hahn, Dan Baumberger, “Efficient and Scalable Multiprocessor Fair Scheduling Using Distributed Weighted Round-Robin”, 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming(PPoPP), Feb 2009.

Razvan Racu, Li Li, Rafik Henia, Arne Hamann, Rolf Ernst, “Improved Response Time Analysis of Tasks Scheduled under Preemptive Round-Robin”, 2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Sep 2007.

Luciano Lenzini, Enzo Mingozzi, Giovanni Stea, “Tradeoffs Between Low Complexity, Low Latency, and Fairness With Deficit Round-Robin Schedulers”, IEEE/ACM Transactions on Networking, Aug 2004

10. Appendices

Program Source Code, Documentation and Output Files

They can be found on the following Github repo:
https://github.com/jasjeetIM/DPRAA