

Lecture Notes in Networks and Systems 1017


Kohei Arai *Editor*

Intelligent Computing

Proceedings of the 2024 Computing
Conference, Volume 2

 Springer

Series Editor

Janusz Kacprzyk , *Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland*

Advisory Editors

Fernando Gomide, *Department of Computer Engineering and Automation—DCA, School of Electrical and Computer Engineering—FEEC, University of Campinas—UNICAMP, São Paulo, Brazil*

Okyay Kaynak, *Department of Electrical and Electronic Engineering, Bogazici University, Istanbul, Türkiye*

Derong Liu, *Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, USA*

Institute of Automation, Chinese Academy of Sciences, Beijing, USA

Witold Pedrycz, *Department of Electrical and Computer Engineering, University of Alberta, Alberta, Canada*

Systems Research Institute, Polish Academy of Sciences, Warsaw, Canada

Marios M. Polycarpou, *Department of Electrical and Computer Engineering, KIOS Research Center for Intelligent Systems and Networks, University of Cyprus, Nicosia, Cyprus*

Imre J. Rudas, *Óbuda University, Budapest, Hungary*

Jun Wang, *Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong*

The series “Lecture Notes in Networks and Systems” publishes the latest developments in Networks and Systems—quickly, informally and with high quality. Original research reported in proceedings and post-proceedings represents the core of LNNS.

Volumes published in LNNS embrace all aspects and subfields of, as well as new challenges in, Networks and Systems.

The series contains proceedings and edited volumes in systems and networks, spanning the areas of Cyber-Physical Systems, Autonomous Systems, Sensor Networks, Control Systems, Energy Systems, Automotive Systems, Biological Systems, Vehicular Networking and Connected Vehicles, Aerospace Systems, Automation, Manufacturing, Smart Grids, Nonlinear Systems, Power Systems, Robotics, Social Systems, Economic Systems and other. Of particular value to both the contributors and the readership are the short publication timeframe and the worldwide distribution and exposure which enable both a wide and rapid dissemination of research output.

The series covers the theory, applications, and perspectives on the state of the art and future developments relevant to systems and networks, decision making, control, complex processes and related areas, as embedded in the fields of interdisciplinary and applied sciences, engineering, computer science, physics, economics, social, and life sciences, as well as the paradigms and methodologies behind them.

Indexed by SCOPUS, INSPEC, WTI Frankfurt eG, zbMATH, SCImago.

All books published in the series are submitted for consideration in Web of Science.

For proposals from Asia please contact Aninda Bose (aninda.bose@springer.com).

Kohei Arai
Editor

Intelligent Computing

Proceedings of the 2024 Computing
Conference, Volume 2

Editor
Kohei Arai
Faculty of Science and Engineering
Saga University
Saga, Japan

ISSN 2367-3370 ISSN 2367-3389 (electronic)
Lecture Notes in Networks and Systems
ISBN 978-3-031-62276-2 ISBN 978-3-031-62277-9 (eBook)
<https://doi.org/10.1007/978-3-031-62277-9>

© The Editor(s) (if applicable) and The Author(s), under exclusive license
to Springer Nature Switzerland AG 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

If disposing of this product, please recycle the paper.

Preface

It is with great pleasure that we introduce the proceedings of the Computing Conference 2024, held on July 11 and 12, 2024. This conference served as a platform for researchers and professionals from around the globe to convene and exchange ideas at the forefront of computing and its diverse applications. The enthusiasm and dedication displayed by participants underscored the significance of this event in fostering collaboration and advancing the field.

We received an overwhelming total of 457 contributions from esteemed scholars and practitioners. These submissions underwent a rigorous double peer-review process, facilitated by experts in their respective domains. After careful evaluation and deliberation, a total of 165 papers were selected for publication in these proceedings.

The diverse array of topics covered in these papers reflects the breadth and depth of contemporary computing research, spanning areas such as artificial intelligence, machine learning, cybersecurity, data science, and beyond. Each paper represents a valuable contribution to the collective knowledge base of the computing community, offering insights, innovations, and solutions to pressing challenges.

We extend our heartfelt gratitude to all authors, reviewers, organizers, and attendees whose efforts and contributions made this conference a resounding success. It is our hope that the insights shared and connections forged during this event will continue to inspire and propel advancements in computing for years to come.

Regards,
Kohei Arai

Contents

Optimised Round Robin with Virtual Runtime for CPU Scheduling	1
<i>Jagriti Bhatia, Sanskriti Mathuria, Vandana M. Ladwani, and Shobana Padmanabhan</i>	
Faster Lock-Free Atomic Shared Pointers	18
<i>Jörg P. Schäfer</i>	
CICO ₂ e: A Compute Carbon Footprint Estimation Tool Based on Time Series Data	39
<i>Christian Plewnia and Horst Lichter</i>	
Image Classification Method Based on Chaos Neural Network	56
<i>Kohei Arai</i>	
Human-Created and AI-Generated Text: What's Left to Uncover?	74
<i>Steven Salter, Phoey Lee Teh, and Richard Hebblewhite</i>	
AraXLM: New XLM-RoBERTa Based Method for Plagiarism Detection in Arabic Text	81
<i>Mona Alshehri, Natalia Beloff, and Martin White</i>	
Boosting Customer Retention in Pharmaceutical Retail: A Predictive Approach Based on Machine Learning Models	97
<i>Angel Espinoza-Vega and Henry N. Roa</i>	
Introducing Prediction Concept into Data Envelopment Analysis Using Classifier in Economic Forecast	118
<i>Guangzao Huang, Zijiang Yang, Grace Liu, and Guoli Ji</i>	
Investigating Machine Learning Techniques Used for the Detection of Class Noise in Data: A Systematic Literature Review	128
<i>Cindy van den Berg and Sunet Eybers</i>	
The Role of Chatbots in Data Analytics: An Evaluation of Functional Abilities	148
<i>Preeti Patel and Sowgol Shooshtarian</i>	
An Analytical Investigation into the Impact of Product Color on the Price of Retail Products and Purchasing Decisions of Consumers	169
<i>Zaid M. Altukhi and Nasser F. Aljohani</i>	

A Numerical Approach for the Fractional Laplacian via Deep Neural Networks	187
<i>Nicolás Valenzuela</i>	
RBF-SC: A Fast Community Detection Technique Using Radial Basis Functions	220
<i>Fang Hu, Jia Liu, Lina Wu, Xingang Fang, Mingfang Huang, and Haotian Liu</i>	
Improving Medication Prescription Strategies for Discordant Chronic Comorbidities Through Medical Data Bench-Marking and Recommender Systems	237
<i>Tom Ongwere, Nimbalkar Rutuja, and Tam V. Nguyen</i>	
Sentiment Analysis for Predicting the Variation Trend of Stocks: A Case Study of Vanke Co., Ltd.	251
<i>Qiuyi Jin and Jin Zheng</i>	
Suicide Ideation Prediction Through Deep Learning: An Integration of CNN and Bidirectional LSTM with Word Embeddings	271
<i>Christianah T. Oyewale, Ayodeji O. J. Ibitoye, Joseph D. Akinyemi, and Olufade F. W. Onifade</i>	
A Multi-clustering Unbiased Relative Prediction Recommendation Scheme for Data with Hidden Multiple Overlaps	284
<i>Avivit Levy, Michal Chalamish, B. Riva Shalom, Guy Sharir, Opal Peltzman, and Sivan Salzman</i>	
An Unsupervised Deep Learning Model for Aspect Retrieving Using Transformer Encoder	303
<i>Atanu Dey, Mamata Jenamani, and Arijit De</i>	
Personalized Student Performance Prediction Modeling for Student Digital Twins	318
<i>Sean Mondesire and Emmanuel Nsiye</i>	
A Social Profile-Based Recommendation Architecture for E-Learning Systems	330
<i>Xola Ntlangula and Wai Sze Leung</i>	
A Comparison of Student Engagement Across Three Teaching Modalities in an Introductory Statistics Course	344
<i>Sher B. Chhetri, Mario Toussaint, Nonhle C. Mdziniso, and Rebecca A. Hillman</i>	

Impact Factor Game Scoring Model for Positive Emotion Game Design 358
V. Sithira Vadivel

Cyber Safety Awareness Through a Massive Open Online Course (MOOC): Community Engagement Knowledge Transfer 365
Elmarie Kritzinger

Immersive Environments at School: “Stop Cyberbullying by Proximity” 377
Cristina Sánchez-Romero and Eva María Muñoz-Jiménez

Simulation and Analysis of Cyber-Attack on Modbus Protocol for Smart Grids in Virtual Environment 384
Shampa Banik, Rajesh Manicavasagam, Trapa Banik, and Shudipta Banik

Cybersecurity Governance in the Medical Ecosystem: An Orientation Guide with Specific Reference to the Merging of IT and OT Devices 402
Sebastian Von Solms and Jaco du Toit

The Impact of Utilising the Amazon AWS Hybrid Deployment Model on Assuring a Secure Migration of a Commercial Web Application into the Cloud 418
Khalied Koorowlay and Rafid Al-Khannak

Mitigating Cache Pollution Attack Using Deep Learning in Named Data Networking (NDN) 432
Mohd Maizan Fishol Hamdi, Zhiyuan Chen, and Milena Radenkovic

A Community Security Operations Centre (ComSOC) Model for SMMEs in Developing Countries 443
Nombeko Ntingi, Jaco du Toit, and Sebastian von Solms

Optimizing Energy States in Mobile Embedded Systems: A SIPN-Based Approach 457
Robert Stojic, Daniel Peters, and Florian Thiel

Secure Content Protection Schemes for Industrial IoT with SRAM PUF-Based One-Time Use Cryptographic Keys 478
Saloni Jain, Ashwija Reddy Korenda, Bertrand Cambou, and Chris Lucero

Feasibility Study with Actual Space Rockets Towards Information Theoretically Secure Radio Communication 499
Sumio Morioka, Satoshi Obana, and Maki Yoshida

FlexHi: A Flexible Hierarchical Threshold Signature Scheme	509
<i>Muhammed Ali Bingol, Sermin Kocaman, Ali Dogan, and Sibel Kurt Toplu</i>	
Autoencoder-Based Solution for Intrusion Detection in Industrial Control System	530
<i>Silvio Russo, Claudio Zanasi, Isabella Marasco, and Michele Colajanni</i>	
Determining the Segmentation Type Impact on an ID Card Fraud Detection System	544
<i>Daniel Benalcazar, Pamela Zurita, Diego Pasmíño, and Rodrigo Lara</i>	
Transforming Medical Waste Management Through IoT and Machine Learning: A Path Towards Sustainability	556
<i>Nahin Nasir, Mohammed Shahriar Hossain, Md. Saiful Islam, Md. Ariful Islam, and Md Manirul Islam</i>	
IntentRec: An Advanced Recommender System Leveraging User-Item Intent	576
<i>Abhishrut Vaidya and Niladri Chatterjee</i>	
Process for the Identification of Vehicle Functions for Cloud Offloading	596
<i>Martin Sommer, Daniel Baumann, Tobias Rösch, Falk Dettinger, Eric Sax, and Michael Weyrich</i>	
Smart Workplace Post Covid-19: Perceived Challenges and Potential Smart Solutions for University Work Environment	609
<i>Rabail Tahir and John Krogstie</i>	
Need for Cultural Sensitivity in the Design and Development of Technology to Aid in Dementia Care: A Review of Literature	625
<i>Arshia Khan, Sakina Rao, and Alfia Parvez</i>	
Sentiment Analysis of Post-COVID-19 Work-From-Home Culture: A Literature Review	637
<i>Joseph Kwame Adjei and Hannah Alhassan Suhuyini</i>	
Security Gaps in the Mobile Money System in Rwanda: Challenges, Risks and Mitigation	653
<i>Catherine Njogu, Furaha Benedict, Susan Muthoni, Marie Noelle Kanyamuneza, Evalyne Lwoba, Everlyn Musembi, Yussuf Papy, and Edwin Kairu</i>	

Comparative Analysis of Flow and Cardano Blockchains: Navigating Adoption Challenges, Computing Techniques and Implications for the Blockchain Landscape 665
Ayda Bransia, Bálint Molnár, and Simon Thompson

Author Index 671



Optimised Round Robin with Virtual Runtime for CPU Scheduling

Jagriti Bhatia¹(✉), Sanskriti Mathuria¹, Vandana M. Ladwani¹,
and Shobana Padmanabhan²

¹ Department of Computer Science and Engineering, PES University, Bengaluru,
Karnataka, India

{pesug20cs145,pes1ug20cs386}@pesu.pes.edu, vandanamd@pes.edu

² School of Computer Science and Engineering, RV University, Bengaluru,
Karnataka, India
shobanap@rvu.edu.in

Abstract. The Central Processing Unit (CPU) consists of the main circuitry to perform all the operations in a computer. One of the most important operations an Operating System performs is CPU Scheduling, which is responsible for managing all tasks and allocating CPU time to each task in an optimal order. This extends to applications in distributed systems since scheduling is required to maximise server utilisation in load-sharing mechanisms. One of the more commonly used scheduling algorithms for the mentioned application is Round Robin (RR), which is considered to be one of the most efficient. However, the efficiency of RR entirely depends on the chosen **Static Time Quantum**. In this paper, we have proposed an optimisation to the pre-existing RR algorithm, namely ‘Riti’, where Round Robin is implemented using **Virtual Runtime** and by calculating **Dynamic Time Quantum**. ‘Riti’ has achieved a 27.6% reduction in **average Turnaround Time** and 37.4% reduction in **average Waiting Time** as compared to Traditional Round Robin.

Keywords: CPU scheduling · Round Robin · Dynamic time quantum · Virtual runtime · Turnaround time · Waiting time · Convoy effect

1 Introduction

A process goes through multiple states from the time it is created. Figure 1 shows the primary states in the life cycle of a process and how a process may pass through all of these states.

At any given time, it is possible only for one process to run on a single-core CPU. Process scheduling is required for discarding a running/completed process to which the CPU is allocated and choosing another process from the waiting

J. Bhatia, S. Mathuria, V. M. Ladwani, and S. Padmanabhan—Contributed equally.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

K. Arai (Ed.): SAI 2024, LNNS 1017, pp. 1–17, 2024.

https://doi.org/10.1007/978-3-031-62277-9_1

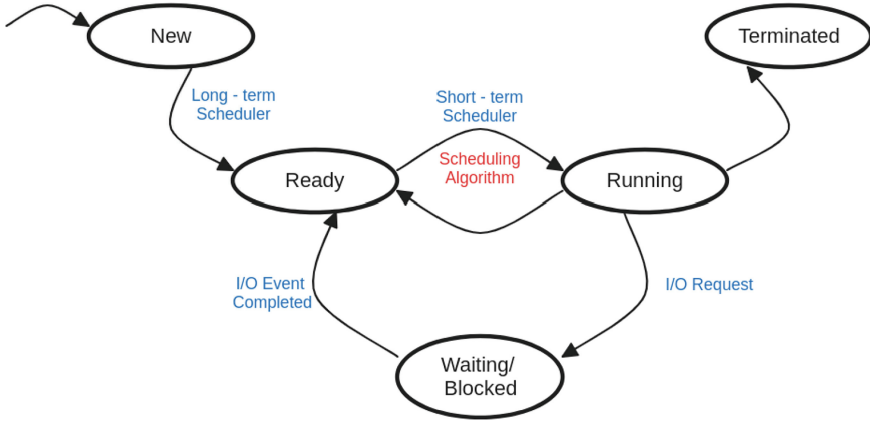


Fig. 1. Process States

queue based on a particular strategy. This allows the OS to allocate CPU time for each process, and hence keep the CPU busy at all times, enabling full use of the CPU and trying to avoid the possibility of deadlock. There are a variety of scheduling algorithms - Shortest Job First, First Come First Serve, Priority Scheduling, Round Robin, etc.

The efficiency of a scheduling algorithm is measured through various criteria.

1. CPU Utilization - The aim is to keep the CPU as busy as possible so that no CPU cycle goes to waste. Normal CPU utilization ranges from 50% to 100%.
2. Turnaround Time - The time a certain process takes, from being added to the ready queue to completion, is called Turnaround time (or TAT) for that process.

$$TAT = Completion\ Time - Arrival\ Time \quad (1)$$

3. Waiting time - The time spent by a process in the ready queue before it is completed is known as waiting time.

$$Waiting\ Time = TAT - Burst\ Time \quad (2)$$

This paper focuses on Round Robin. Round Robin is a very frequently used scheduling algorithm, however it has one major drawback - Convoy effect, due to which the waiting time for the processes is very high. To tackle this issue, we have proposed to optimise the Round Robin algorithm using ‘virtual runtime’.

The paper is structured in the following manner - Sect. 2 talks about the relevant research that has been conducted on Round Robin variations for CPU scheduling in the past. Section 3 provides a brief explanation of the basic concepts of the ‘Riti’ algorithm, that is built on. Section 4 gives an in depth explanation as to how the proposed algorithm works. Section 5 shows how ‘Riti’ performs as compared to other Round Robin algorithms present currently. To conclude, Sect. 6 highlights the benefits and limitations of the ‘Riti’ algorithm.

2 Related Work

The efficiency of a CPU depends on its ability to schedule processes and efficiently provide resources. Many different scheduling algorithms have been studied and worked upon to enhance the performance of CPU.

Alsulami et al. in their work talk about the necessity of a good scheduling algorithm for a good operating system performance. The authors have studied Round Robin with a dynamic time quantum. They state that the performance of the shortest job first scheduling of the dynamic time quantum depends on the method used to calculate it. The authors calculated dynamic time quantum using four different methods and have done a comparative study among them [4]. Various research papers use dynamic time quanta, such as Butangen et al. [6] which make use of the dynamic mean of the burst times of the running processes. Mishra and Mitawa too present an approach which sets the time quantum to the average burst time [12].

Pathak et al. display shortest job first scheduling modification in traditional Round Robin [16]. Even Srujana et al. [19] incorporate SJF in Round Robin. 'Riti' differs from these as it splits the processes into two queues and follows different scheduling strategies for both queues, still giving preference to shorter jobs by running the queue with shorter processes first.

Traditional Round Robin in the Cloud environment is not optimal as the cloud environment is constantly subject to changes and this can cause the waiting time of the processes to increase. Sanaj and Prathap propose that time slices should be assigned on the basis of the mean burst time of all the tasks in the waiting queue [18].

The idea of Recomputed Time Quantum [14] greatly influenced the calculation of dynamic time quantum. Aijaz et al. calculated dynamic time quantum for each cycle by adding the total burst time to the previous time quantum and dividing it by the number of processes [2]. We adopted this method in our proposed algorithm as well. However, *Riti* focuses on reducing the convoy effect and hence gives importance to executing the shorter processes first. *Riti* also implements a fair-scheduling method for the shorter processes, hence differentiating it from this paper.

Farooq et al. proposed that a suitable way of finding dynamic time quantum is to use $0.8 * \max(\text{burst times})$. The aim is to achieve the best results in efficiency without having to sort the processing according to burst times in ascending order. The author states that the goal is to lower the duration of an algorithm along with efficiency constraints [7].

Dynamic time quantum is also implemented by Alaa et al., however, this paper proposes to split the ready queue into two queues on the basis of the medium burst time. After arranging the processes based on their burst time in increasing order, they are split into two queues - light task queue and heavy task queue, depending on the medium burst time. The light task queue is executed before the heavy task queue and the time quantum for each queue is considered to be the burst time of the medium processes respectively [3]. 'Riti' also places processes into a light task queue or a heavy task queue. However, sorting based

on burst time is not performed since the value based on which the processes are split is not dependent on the burst times of the processes. Hence, avoiding sorting reduces the time complexity.

H. B. Parekh and S. Chaudhari's work [15] comes closest to 'Riti' in theory, as they make use of SJF and priority scheduling. 'Riti' differs from this work as it splits the ready queue into two based on the burst times, as compared to [15] which uses a single queue.

3 Background

3.1 Round Robin Scheduler

Round Robin, the preemptive process scheduling algorithm, is cyclic in nature. It is a widely used algorithm, mainly because it has a good average response time. Despite being considered a very fair algorithm, Round Robin does have its disadvantages. Varied burst times cause convoy effect and greatly degrade the performance of the algorithm. Priorities of the processes are also not considered.

The convoy effect is a phenomenon that slows down the entire Operating System due to the presence of a few large processes (CPU-intensive processes). Essentially, when large processes arrive in the queue before the smaller processes, they take a much larger time to execute and hence starve the smaller processes of their fair share of resources.

The time quantum used for the execution of Round Robin is vital. It controls the performance of the algorithm. If it is too small, there is a spike in the number of context switches and the overhead will increase. On the other hand, if the time quantum is too large, Round Robin will act as FCFS.

3.2 Completely Fair Scheduling (CFS)

The CFS scheduler attempts to implement fair-share scheduling but in an efficient manner. As seen in [8], CFS replaced $O(1)$ scheduler in Linux as the $O(1)$ had low throughput for background jobs and poor interactive performance. If the CPU following CFS switches too often, it means that the CPU is being more fair but at the same time results in an increase of context switches. At the same time, switching less frequently leads to less fairness. CFS uses two control parameters to take care of this problem:

- Sched_latency- This value is used for determining after how much time a switch should be considered by the CPU. The time to be allocated to a single process is $\frac{\text{sched_latency}}{\text{number of processes}}$; hence the time allocated changes dynamically, based on the workload on the CPU.
- Min_granularity: This is the minimum amount of time a time slice can be; to avoid many context switches it should not go lower than this value.

3.3 Virtual Runtime

This term comes up in the Completely Fair Scheduler (CFS). In this method, the CPU uses a counting-based technique known as virtual runtime to fairly divide itself among all competing processes. As each process runs, it accumulates this virtual runtime. When the CPU has to execute a process, it chooses the one with the least virtual runtime, thus ensuring equal distribution of CPU resources among all the processes.

3.4 Niceness

CFS allows the users/admins to assign priority to processes in the form of a ‘nice’ value. This value ranges from -20 to 19 . A positive value indicates that the process is ‘nice’ and hence means the process does not need the CPU as urgently. On the other hand, a negative value indicates that the process needs the CPU more urgently and thus the process has higher priority. This nice value can be used to calculate the weight of a process, which can be used in the calculation of time slice. The formula and conversion from nice value to weight are taken from [5].

The conversion from nice to static priority is defined by the `NICE_TO_PRIO` macro in the Linux Kernel. It is defined in `include/linux/sched/prio.h`. The relation is $prio = nice + 120$. The conversion of nice value to priority to weight is roughly equivalent to $\frac{1024}{1.23^{nice}}$. This is defined in `prio_to_weight` in `kernel/sched.c`

$$time_slice_i = \frac{weight_i}{\sum_{i=0}^{n-1} weight_i} * sched_latency \quad (3)$$

$$vruntime_i = vruntime_0 + \frac{weight_0}{weight_i} * runtime \quad (4)$$

4 Proposed Methodology

As mentioned earlier, Round Robin does not take into account the priority of the processes. In our proposed algorithm, ‘Riti’, we consider three things along with the process ID - burst time, arrival time, and the nice value associated with the process. Initially, the processes arrive and are placed into the queue. The ready queue is then split into two smaller queues as shown in Fig. 2. If the burst time is less than $min_granularity * number_of_processes$, then the process is assigned to Queue-1. Else, the process is assigned to Queue-2. This results in Queue-1 containing all the processes with smaller burst times, and Queue-2 containing all the processes with larger burst times. This is not a multi-scheduler algorithm, the concept of splitting the processes into two queues has been adopted so that the smaller processes are not starved of CPU time due to the larger processes.

The algorithm begins by running Queue-1 first so that the processes with a smaller burst time can be executed before the processes with a greater burst time, and are not starved. Queue-1 follows CFS concepts of scheduling, and uses

nice value to calculate the virtual runtime in order to schedule the processes based on their priority.

Once the execution of Queue-1 is completed, only then Queue-2 begins its execution. Here, a dynamic time quantum formula is applied [2].

$$TQ = \frac{LTQ + TBT}{number_of_processes} \tag{5}$$

TQ - Time Quantum LTQ - Last Time Quantum TBT - Total Burst Time

Here, Total Burst Time is the sum of the burst times of all the processes in the queue. The time quantum will change every loop, based on the value of the previous time quantum.

Finally, the values of average TAT and WT are calculated. These are compared to the average TAT and WT produced by running the same ready queue using the traditional Round Robin algorithm. It is noticed that there is a significant difference between the two, as the optimized algorithm performed much better.

Initially, to split the ready queue into Queue-1 and Queue-2, a time complexity of $O(no_of_processes)$ is required. The time complexity for the execution of Queue-1 is $O(no_of_processes^2)$, because the processes are looped through once each cycle and a second loop is required to accumulate the virtual runtime. The time complexity for the execution of Queue-2 is also $O(no_of_processes^2)$. The dynamic time quantum must be calculated every cycle, hence resulting in a $O(no_of_processes^2)$ complexity. As a result, the proposed algorithm's overall time complexity becomes $O(no_of_processes^2)$.

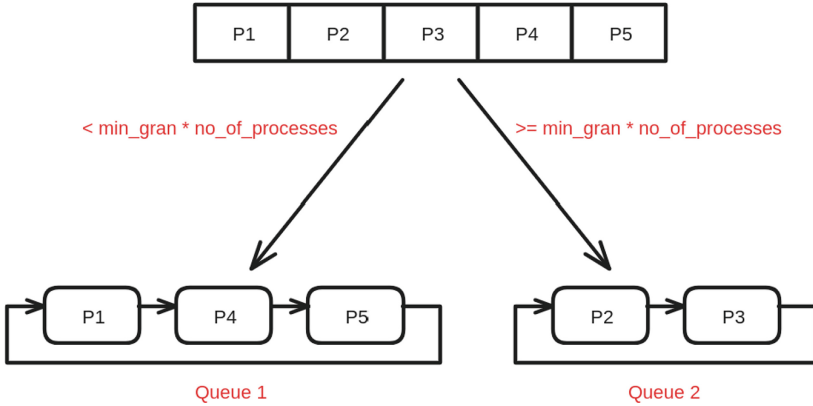


Fig. 2. Method to split the ready queue

Algorithm 1. Pseudo code for *Riti*

```

N ← number_of_processes
Q1 ← light_load queue
Q2 ← heavy_load queue
min_gran ← 6
sched_latency ← 48

for each process i in ready queue do
  if Burst_Timei ≤ min_gran * N then
    | Q1.append(process)
  else
    | Q2.append(process)
  end
end
// Q1 execution
for each process i in Q1 do
  runtimei ← 0 vruntimei ← 0 weighti =
  prio_to_weight[nice_valuei + 20]time_slicei =
   $\frac{weight_i}{\sum_{j=0}^{n-1} weight_j} * sched\_latency$ 
end
for each cycle in Q1 do
  for each process i in Q1 do
    | runtimei = runtimei + time_slicei if runtimei ≥ burst_timei
    | then
    | | Q1.remove(processi)
    | else
    | | vruntimei = vruntimei +  $\frac{weight_0}{weight_i} * runtime_i$ 
    | end
  end
  sort Q1 according to vruntime in ascending order
end
// Q2 execution
TQ ←  $\frac{TBT}{N}$  // TBT - total burst time
for each cycle in Q2 do
  for each process i in Q2 do
    | runtimei ← TQ if runtimei ≥ burst_timei then
    | | Q2.remove(processi)
    | end
  end
  new_TQ ←  $\frac{LTQ+TBT}{N}$ 
  TQ ← new_TQ
end

```

‘Riti’ works best when the ready queue processes have varied burst times. For example, if there are large CPU-bound processes in the queue but an I/O process

that has a short CPU burst time enters the queue, it is prioritised and finished first. Even within multiple I/O processes with short CPU burst times, there is fairness in the scheduling of the I/O processes. The main focus of ‘Riti’ is to ensure the smaller processes get completed smoothly without getting disrupted by the larger processes, making virtual runtime an important aspect.

5 Observation

The proposed methodology has been depicted with the help of two examples - (1) with common Arrival Time, and (2) with different Arrival Times. The `min_granularity` is considered to be 6 in the following examples, and the `sched_latency` is considered to be 48.

5.1 Example of Processes with Same Arrival Time

Here, the Arrival Time of all the processes is considered to be zero. It is assumed that all the processes have already arrived before the scheduling begins. The process details are given below in Table 1.

Table 1. Parameters and values used throughout the experiment

p_id	AT	BT	Nice Value
P1	0	35	3
P2	0	43	-12
P3	0	16	17
P4	0	41	-16
P5	0	14	0

In this example, the processes arrive at the same time but have randomly generated Burst Times and Nice Values.

Table 2. Optimized round robin

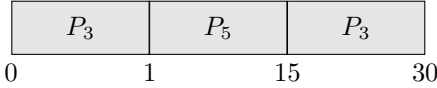
p_id	TAT	WT
P1	65	30
P2	147	104
P3	30	14
P4	149	108
P5	15	1

When the scheduling of processes is according to the proposed algorithm, the results observed are shown in Table 2, and the following values are observed as well:

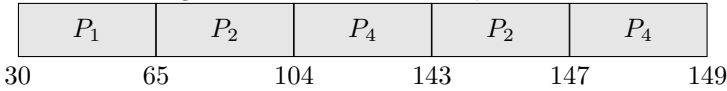
Avg TAT = 81.2

Avg WT = 51.4

P_3 and P_5 , being process with smaller burst times will join Queue-1. Shown below is the Gantt chart for Queue-1.



The process that will join Queue-2 are P_1 , P_2 and P_4 , as they have larger burst times. The following is the Gantt chart for Queue-2.



Calculating TQ for traditional Round Robin: $TQ = \frac{Sched_latency}{number_of_processes}$

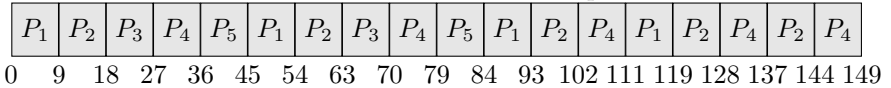
Hence, $TQ = floor(\frac{48}{5}) = 9$

Table 3. Traditional Round Robin

p_id	TAT	WT
P1	119	84
P2	144	101
P3	70	54
P4	149	108
P5	84	70

By applying the Traditional algorithm, the values in Table 3 are observed, along with the following results: Avg TAT = 113.2 Avg WT = 83.4

The Gantt chart for traditional Round Robin is depicted below.



Figures 3 and 4 show the comparison of TAT and WT for this example.

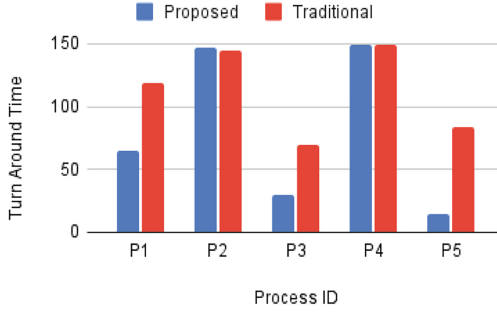


Fig. 3. Comparison of turnaround time (same arrival time)

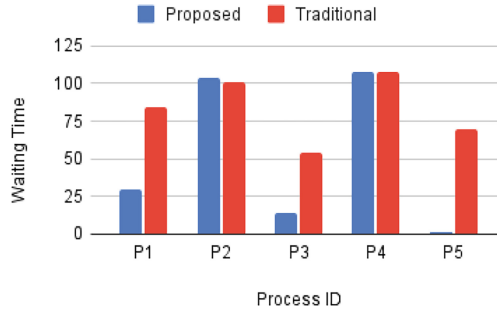


Fig. 4. Comparison of waiting time (same arrival time)

From Figs. 3 and 4, in the case of P_1 , P_3 and P_5 , a large change is noticed in TAT and WT for Proposed and Traditional RR methodologies. This shows a significant change in average TAT and average WT values.

5.2 Example of Processes with Different Arrival Time

In this example, the Arrival Times of the processes are not the same, they all arrive at different times (see Table 4). The Burst Times and Nice Values for all the processes have been randomly generated.

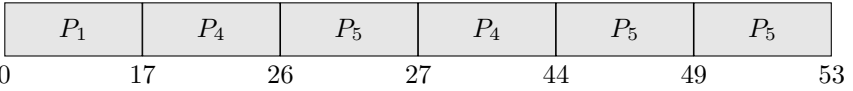
Table 4. Parameters and values used throughout the experiments

p_id	AT	BT	Nice Value
P1	0	17	-14
P2	1	39	15
P3	2	31	-5
P4	3	26	-8
P5	4	10	1

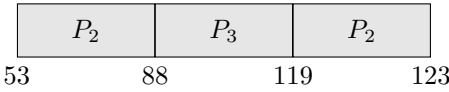
Table 5. Optimized round robin

p_id	TAT	WT
P1	17	0
P2	122	83
P3	117	86
P4	41	15
P5	49	39

Using the proposed methodology, the following values are obtained from the observed values in Table 5: Avg TAT = 69.2 Avg WT = 44.6 As shown in the Gantt chart the lighter processes - P_1 , P_4 and P_5 join Queue-1.



P_2 and P_3 , having larger burst times will join Queue-2 as shown below.



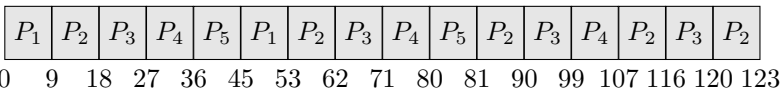
As seen in the previous example, the TQ for traditional Round Robin will be 9.

Table 6. Traditional round robin

p_id	TAT	WT
P1	53	36
P2	122	83
P3	118	87
P4	104	78
P5	77	67

With the Tradition Round Robin, the result is as follows, compiling the observed values in Table 6: Avg TAT = 94.8 Avg WT = 70.2

The Gantt chart for traditional Round Robin is as follows.



Comparison of TAT and WT for example B (processes with different Arrival Time) is depicted through Figs. 5 and 6.

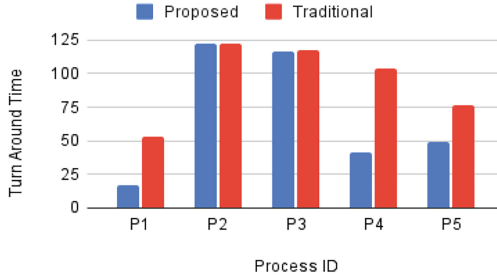


Fig. 5. Comparison of Turnaround Time (different Arrival Time)

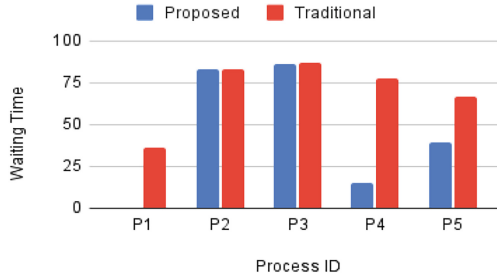


Fig. 6. Comparison of Waiting Time (different Arrival Time)

A large difference is seen between TAT and WT respectively, for Proposed and Traditional RR methodologies for processes P_1 , P_4 and P_5 , as shown in Figs. 5 and 6.

5.3 Comparison with Other Round Robin Variations

To prove the effectiveness of the proposed methodology, ‘Riti’ has been compared with other Round Robin variations, namely Efficient Dynamic Round Robin (EDRR) [7], New Median-Average Round Robin (NMARR) [17], Modified Round Robin CPU Scheduling Algorithm With Dynamic Time Quantum (RRDTQ) [1], Efficient Round Robin Algorithm (ERRA) [2], An Improved Round Robin CPU Scheduling Algorithm with Varying Time Quantum (IIRVQ) [11], and Modified Median Round Robin Algorithm (MMRRA) [13]. Some of these algorithms have also been compared in [9]. Three groups were created based on the range of burst time of the processes -

- Group - 1: Burst time ranging between 5–100
- Group - 2: Burst time ranging between 5–150
- Group - 3: Burst time ranging between 5–250

Ten processes were randomly generated for each group. All the Round Robin variations along with Traditional Round Robin, and ‘Riti’ were run against each other for all the processes. It can be observed from the below graphs (see Figs. 7,

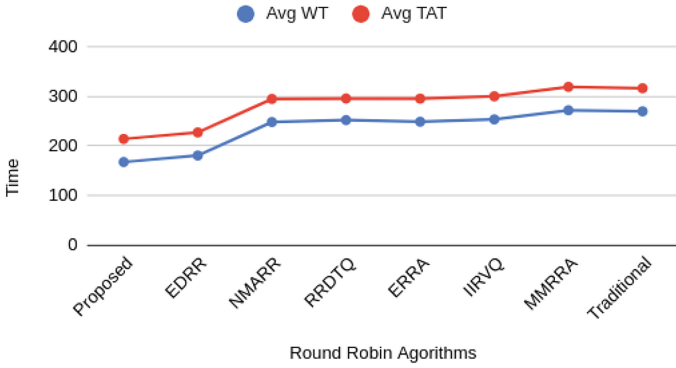


Fig. 7. Group - 1 Results

Table 7. Comparison of Riti in Group - 1

Algorithm	TAT Comparison (in %)	WT Comparison (in%)
ERRA	6.129	7.26
MMRRA	32.88	38.38

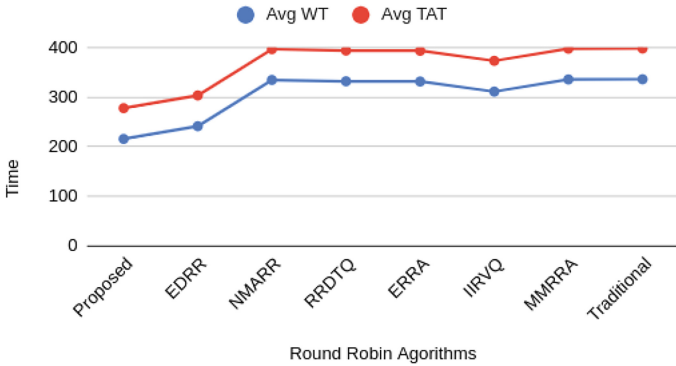


Fig. 8. Group - 2 Results

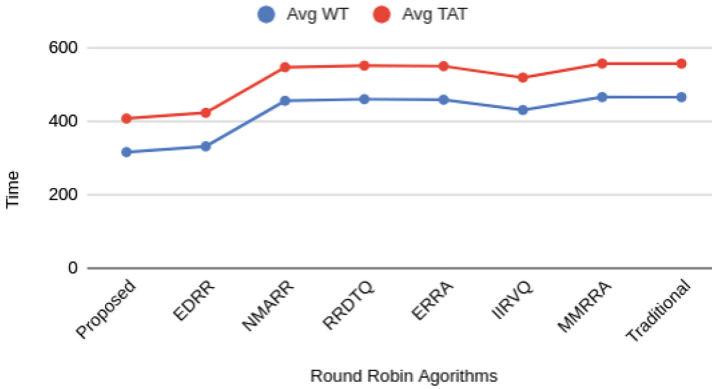
8 and 9) that ‘Riti’ had the least Waiting Time and Turn Around Time. A comparison of the same is also depicted in Tables 7, 8 and 9.

From the graphs depicted above, it can be concluded that the proposed algorithm, ‘Riti’, leads to a significant difference in average Turnaround Time and average Waiting Time as compared to the traditional algorithm.

The following tables show the performance of ‘Riti’ as compared to ERRA, which was the best performing amongst the Round Robin Optimizations and MMRRA,

Table 8. Comparison of Riti in Group - 2

Algorithm	TAT Comparison (in %)	WT Comparison (in%)
ERRA	8.46	10.6
MMRRA	30.17	35.753

**Fig. 9.** Group - 3 Results**Table 9.** Comparison of Riti in Group - 3

Algorithm	TAT Comparison (in %)	WT Comparison (in%)
ERRA	3.67	4.69
MMRRA	26.8	32.14

which performed the most poorly. It can be observed that ‘Riti’ is significantly better than both in terms of TAT and WT.

6 Conclusions

6.1 Benefits of ‘Riti’

As compared to the traditional Round Robin algorithm, ‘Riti’ shows a significant reduction in both, average Turnaround Time as well as the average Waiting Time, as shown in Figs. 10 and 11. It is also ensured that the smaller processes are scheduled first by splitting them into a different queue. In this manner, convoy effect is avoided.

From Table 10, it can be inferred that the average TAT for traditional RR is 113.2, and for ‘Riti’ is 81.2, leading to a difference of a significant 32 units. And the difference for WT between the two methodologies is 32 units. This large difference can be visualised in the graph - Fig. 7.

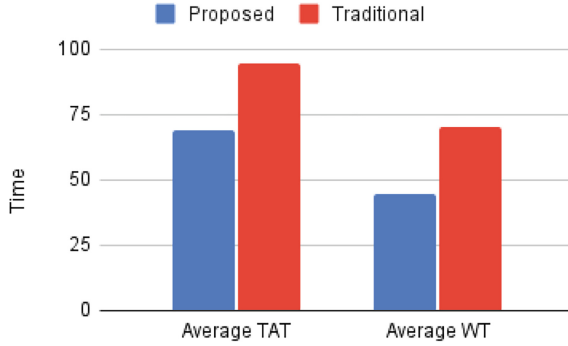


Fig. 10. Comparison with traditional round robin (different arrival time)

Table 10. Traditional round robin vs optimized round robin (example a - processes with same arrival time)

Algorithm	TAT	WT
Traditional RR	113.2	83.4
Proposed RR (<i>Riti</i>)	81.2	51.4

The data in Table 11 indicates the difference between average TAT and WT for both the proposed and traditional RR methodologies respectively. This result can be visualised by the graph depicted in Fig. 8. Thus the proposed algorithm ‘Riti’ achieves significant performance improvement over the existing Round Robin scheduling algorithm.

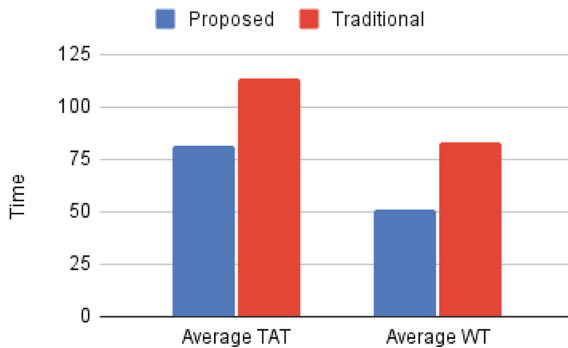


Fig. 11. Comparison with traditional round robin (same arrival time)

Table 11. Traditional round robin vs optimized round robin (Example B - Processes with different arrival time)

Algorithm	TAT	WT
Traditional RR	94.8	70.2
Proposed RR (<i>Riti</i>)	69.2	44.6

6.2 Limitations of ‘Riti’

While comparing with other Round Robin variations, it was observed that ‘Riti’ tends to make more context switches while scheduling, which can result in the reduction of the efficiency of the CPU with respect to scheduling processes. Also, while this paper has used a simulation of processes to test out the proposed algorithm, an extensive study has not been carried out yet using real-time processes.

6.3 Future Scope

1. ‘Riti’ can be tested on more processes in a real-time scenario.
2. This algorithm has the scope to be applied as a load balancing algorithm in distributed computing schedulers. Currently, Round Robin is being used as a static load balancing algorithm. The modifications made to traditional Round Robin as shown in ‘Riti’ has the potential to be used as a dynamic load balancing algorithm. It can predict which server the load can be assigned to such that there is optimal usage of all servers.
3. Similar applications as the above, such as being used by scalable multiprocessors [10], or by network routers to allocate bandwidth to different data packets. It can even be applied by database systems to distribute queries to different database servers show that this algorithm has a lot of scope to be used in real-life situations.

References

1. Sohrawordi: A modified Round Robin CPU scheduling algorithm with dynamic time quantum. *Int. J. Adv. Res.* **7**, 422–429 (2019)
2. Aijaz, M., Tariq, R., Ghorri, M., Rizvi, S.W., Qazi, E.F.: Efficient round robin algorithm (ERRA) using the average burst time. In: 2019 International Conference on Information Science and Communication Technology (ICISCT), pp. 1–5 (2019)
3. Fiad, A., Zoulikha, M.M., Hayat, B.: Improved round robin scheduling algorithm with varying time quantum. In: 2020 Second International Conference on Embedded and Distributed Systems (EDiS), pp. 33–37 (2020)
4. Alsulami, A.A., Al-Haija, Q.A., Thanoon, M.I., Mao, Q.: Performance evaluation of dynamic round robin algorithms for CPU scheduling. In: 2019 SoutheastCon, pp. 1–5 (2019)
5. Arpaci-Dusseau, R.H., Arpaci-Dusseau, A.C.: *Operating Systems: Three Easy Pieces*, 1.00 edn. Arpaci-Dusseau Books, August 2018

6. Butangen, A.K.G., Velasco, C.E., Codmos, J.C.B., Bayani, E.F., Baquirin, R.B.: Utilizing dynamic mean quantum time round robin to optimize the shortest job first scheduling algorithm. In: Proceedings of 2020 6th International Conference on Computing and Data Engineering, ICCDE 2020, pp. 14–18. Association for Computing Machinery, New York (2020)
7. Farooq, M.U., Shakoor, A., Siddique, A.B.: An efficient dynamic round robin algorithm for CPU scheduling. In: 2017 International Conference on Communication, Computing and Digital Systems (C-CODE), pp. 244–248 (2017)
8. Jose, J., Sujisha, O., Giles, M., Bindima, T.: On the fairness of Linux O(1) scheduler. In: 2014 5th International Conference on Intelligent Systems, Modelling and Simulation, pp. 668–674 (2014)
9. Joshi, A., Goyal, S.B.: Comparison of various round robin scheduling algorithms. In: 2019 8th International Conference System Modeling and Advancement in Research Trends (SMART), pp. 18–21 (2019)
10. Li, T., Baumberger, D., Hahn, S.: Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin. In: Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2009, pp. 65–74. Association for Computing Machinery, New York (2009)
11. Mishra, M., Rashid, F.: An improved round robin CPU scheduling algorithm with varying time quantum. *Int. J. Comput. Sci. Eng. Appl.* **4**, 1–8 (2014)
12. Mishra, R., Mitawa, G.: Improved round robin algorithm for effective scheduling process for CPU. In: 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), pp. 590–593 (2021)
13. Mora, H., Abdullahi, S.E., Junaidu, S.B.: Modified median round robin algorithm (MMRRA). In: 2017 13th International Conference on Electronics, Computer and Computation (ICECCO), pp. 1–7 (2017)
14. Oyam, N.A., Pidlaon, L.J., Baquirin, R.B., Bayani, E.F., Fronza, R.J.: Refining the round robin algorithm using a recomputed time quantum: a comparison. In: Proceedings of 2020 6th International Conference on Computing and Data Engineering, ICCDE 2020, pp. 1–4. Association for Computing Machinery, New York (2020)
15. Parekh, H.B., Chaudhari, S.: Improved round robin CPU scheduling algorithm: round robin, shortest job first and priority algorithm coupled to increase throughput and decrease waiting time and turnaround time. In: 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), pp. 184–187 (2016)
16. Pathak, P., Kumar, P., Dubey, K., Rajpoot, P., Kumar, S.: Mean threshold shortest job round robin CPU scheduling algorithm. In: 2019 International Conference on Intelligent Sustainable Systems (ICISS), pp. 474–478 (2019)
17. Sakshi, C.S., Sharma, S., Kautish, S., Alsallami, S.A.M., Khalil, E.M., Mohamed, A.W.: A new median-average round robin scheduling algorithm: an optimal approach for reducing turnaround and waiting time. *Alexandria Eng. J.* **61**(12), 10527–10538 (2022)
18. Sanaj, M.S., Prathap, P.M.J.: An enhanced round robin (ERR) algorithm for effective and efficient task scheduling in cloud environment. In: 2020 Advanced Computing and Communication Technologies for High Performance Applications (ACCTHPA), pp. 107–110 (2020)
19. Srujana, R., Mohana Roopa, Y., Datta Sai Krishna Mohan, M.: Sorted round robin algorithm. In: 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), pp. 968–971 (2019)



Faster Lock-Free Atomic Shared Pointers

Jörg P. Schäfer^(✉)

German Aero Space Center, Institute of Transportation Systems, Berlin, Germany
joerg.schaefer@dlr.de

Abstract. CPU's don't increase in speed anymore, as Moore's Law has claimed for so long. Although, "the free lunch is over" (Herb Sutter), parallel algorithms can gain more throughput and reduce latency, which is crucial to complex real-time applications like audio and video processing, robotics, or real-time sensor data processing in embedded hardware. Parallel algorithms, however, come with the price of concurrency and synchronization. For example, the priority-inversion is a problem, where low-priority threads can block high-priority threads due to locking data structures used by both threads. Lock-free data structures, on the other hand, use atomic CPU instructions to avoid these problems. They, however, are hard to implement and even harder to prove correct. Atomic shared pointers have been proposed as a (part of a) solution to making lock-free algorithms easier to write and verify. Since they are a fundamental tool in the toolbox of parallel algorithms, their run-time performance has a huge trailing impact. So far, there is just a hand full of existing implementations to atomic shared pointers. This work contributes an improved implementation to atomic shared pointers, a formal proof of its correctness, and an extensive performance evaluation in comparison to other implementations showing that it outperforms others in most use-cases.

Keywords: Concurrency · Algorithm · Atomic shared pointers · Lock-free · Real-time implementation

1 Introduction

Real-time applications such as in robotics or video and audio processing increase their workload and processing capabilities by using multiple threads. In such applications, there are often multiple threads running with different priorities, e.g., at least one low-priority and one high-priority thread. Such threads usually communicate via common variables and data structures. To avoid race-conditions, most of these data structures ensure thread-safety using exclusive locking variables (*mutexes*).

Using a mutex for synchronization comes with the priority inversion problem, where low priority threads slow down high priority threads by locking such a common mutex: When the low priority thread is sent to sleep while holding the lock,

With thanks to the German Federal Ministry of Digital and Transport (BMDV) for funding this work through project "EDDY" (Förderkennzeichen 19F2208B).

the high priority thread needs to wait until the low priority thread is finished. A second disadvantage of using mutexes comes with the CPU's over subscription. The problem is effectively the same as before, i.e., threads are more likely prone to suspension while locking a mutex, which blocks other threads accessing the same mutex. These situations are exactly, what real-time applications need to avoid.

On the other hand, modern CPU's provide atomic operations on single (or double) words, e.g., exchange and compare-and-swap (*CAS*) to name the most important ones. Several simple data structures have already been implemented *lock-free*, i.e., without using mutexes and instead relying on these atomic instructions. However, this area of research is an open field because it is particularly hard to develop even simple data structures lock-free [7, 11, 15]. Several new problems arise, one of which is the ABA problem, where concurring threads might change an *atomic variable* forth and back to the value last seen, making other threads believe that nothing happened during their last visit of this atomic variable. This false assumption might lead to algorithms with undefined behaviour in this situation.

Herb Sutter proposed to use lock-free atomic shared pointers [13, 14], as they solve several problems including the ABA problem. In his proposal, he could show that a singly connected list is implementable in a few lines of (C++) code without any further ado. At that time, however, no implementation for atomic shared pointers existed.

A *shared pointer* is a concept in programming languages that usually leave memory management to the language's user. Here, shared pointers are smart pointers that help managing the lifetime of dynamically allocated memory and the object that resides in there. When a shared pointer is created to manage the lifetime of a dynamically allocated object, it takes ownership of it, i.e., the last living copy of the shared pointer is responsible for its proper destruction. From here on, we focus on C++ (up to C++23) as programming language.

The ubiquitous way to implement shared pointers is by additionally allocating a *control block* for a particular object, which contains a reference counter. Thus, shared pointers referencing the same object also share the same control block including this shared reference counter. Copying a shared pointer increments this reference counter. On the other hand, when a shared pointer dereferences its object, e.g. by being destroyed or being made referencing another object, it needs to decrement its reference counter. Upon reaching zero, the object is destroyed and the memory deallocated by the last accessing thread.

Now, when a shared pointer dereferences an object, it needs to write to at least two different memory addresses: the pointer to the control block and the reference counter within the control block, which cannot be done atomically in most of the current CPU's.¹ One way to achieve atomic operations on shared pointers is guarding them with an exclusive locking variable. At least GNU-C++ 12.3.0 and Clang 14.0.0 use this approach for the implementation of `std::atomic<std::shared_ptr>`.

¹ TSX enables CPU-level transactions, but substantial reasons exist, to avoid it [9].