International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 2, April 2020, pp. 1805~1813
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i2.pp1805-1813

Reinforcement learning based multi core scheduling (RLBMCS) for real time systems

Dhruva R. Rinku[1], M. AshaRani[2]
[1]Department of Electronics and Communication Engineering, CVR College of Engineering, Hyderabad, India
[2]Department of Electronics and Communication Engineering, JNT University, Hyderabad, India

Article Info

Article history:

Received Mar 13, 2019
Revised Oct 29, 2019
Accepted Nov 6, 2019

Keywords:

Multi core scheduling
Multilevel feedback queue
Reinforcement learning
Symmetric multiprocessors

ABSTRACT

Embedded systems with multi core processors are increasingly popular because of the diversity of applications that can be run on it. In this work, a reinforcement learning based scheduling method is proposed to handle the real time tasks in multi core systems with effective CPU usage and lower response time. The priority of the tasks is varied dynamically to ensure fairness with reinforcement learning based priority assignment and Multi Core MultiLevel Feedback queue (MCMLFQ) to manage the task execution in multi core system

Corresponding Author:

Dhruva R. Rinku,
Department of Electronics and Communication Engineering,
CVR College of Engineering,
Hyderabad, India.
Email: rinkudhruva.ravi@gmail.com

1. INTRODUCTION

Scheduling is the process of allocation of tasks to a set of resources under certain constraints like deadline, utilization, response time etc. In RTOS (Real Time Operating Systems), task scheduling is a core component responsible for assigning resources to the task so as ensure time guarantee. The current task scheduling algorithms in RTOS can be classified into time-driven, priority-driven and hybrid-driven. Priority driven scheduling assign priorities to tasks in such a way that high priority task is selected for scheduling ahead of low priority tasks. The priority can be assigned in two ways – static and dynamic. In static priority scheme, the priority is assigned during task creation time and it does not change during the lifetime of the task. In dynamic priority schemes, the priority of task changes depending on system status and resource availability. Priority inversion is a problem in priority driven scheduling. The problem arises when tasks with different priorities share common data. Other issues like deadlock and multiple blocking of tasks is quite common in priority driven scheduling. Multiprocessor architectures (e.g., Symmetric Multiprocessors or SMPs, Single Chip Heterogeneous Multiprocessors or SCHMs) are increasingly popular to cost reduction and diverse applications that can be run on it. Scheduling in Multiprocessor system is more challenging as it must consider multi parameter optimization like effective resource utilization, task response time etc. The scheduling criteria for optimization in multi process system is given in Table 1.

The current scheduling algorithms fail for the cases of uncertain task arrivals and uncertain task nature. In these situations, there is an increase in deadline miss ratio and poor resource utilization. To overcome these challenges, the scheduling algorithm must be adaptive and must be self-learning. The scheduler must learn these dynamics from the environment and must adapt itself to ensure there is very low deadline miss ratio and higher resource utilization. This work is motivated by above problems and to design a self-learning scheduler. Previous works on self-learning scheduler are also available but no existing work has considered multi core systems and dynamic task arrivals with adaptation based on MLFQ. In this work, a reinforcement learning based dynamic priority scheme with multi level feedback queue is used for scheduling the tasks to multi processors. The priority to allocate for the task is modeled as continuous learning activity with reward and penalty score allocated based on the multi-objective parameter optimization. The proposed scheduler is implemented into Free-RTOS operating system and tested for its effectiveness with job mix of CPU and IO intensive tasks. To the best of our knowledge, this work is the first to consider multi objective realization using MLFQ and self-learning on MLFQ.

Table 1. Scheduling criteria

CPU Utilization	Keep the CPUs as busy as possible. Maximize the CPU Utilization
Throughput	Number of tasks completed per unit time. Maximize the throughput
Turn Around Time	Time to execute a particular task from submission to completion. Minimize the turnaround time
Waiting Time	Amount of time process is waiting in the ready queue. Minimize the waiting time
Response Time	Amount of time it takes from when a request was submitted until the first response is produced. Minimize the response time.

2. LITERATURE SURVEY

The existing solutions scheduling in RTOS is analyzed below, Seifedine Kadry et al. [1] applied dynamic quantum instead of static quantum in Multi Level Feedback Queue (MLFQ) based scheduling. The scheme was able to accommodate low priority tasks, which takes longer time for completion and maximize the resource utilization. Xiaojie Li et al. [2] proposed improved EDF algorithm which reduces the task switching overhead and number of task switch. Controlled preemption is proposed by Jinkyu Lee et al. [3], which regulates the preemption for better EDF. The approach applies heuristics to decide a better value of preemption time for each task in case of Uni processor systems.J. Lee et al. [4] proposed a scheduling policy to make effective use of contention free slots. Tasks in contending slots are migrated to contention free slots to reduce the number of competing tasks in contenting slots. J. Lee et al. [5] proposed, two important aspects: inter-job concurrency and job urgency are considered for scheduling. Analysis of LLF (Least Laxity First) scheduling on multi process platforms is considered in [6] by J. Lee et al. Laxity of a task at any time is defined as remaining time to deadline minus the amount of remaining execution. Algorithms employing Laxity for scheduling are called ZL-based scheduling algorithm.

S. K. Lee proposed, EDZL scheduling [7] employs the EDF strategy until tasks have zero-laxity and the ZL policy. M. L. Dertouzos et al.[8] proposed, Least Laxity First (LLF) scheduling is a ZL scheduling algorithm, which globally assigns a higher priority to a task with lower laxity. A. Easwaran et al. shown Cluster-based scheduling assigns each task to processor cluster. Tasks in each cluster are globally scheduled among themselves, and clusters in turn are scheduled on the multiprocessor platform. Clustering places a restriction on the amount of concurrency. [9]

Marko Bertogna et al. [10] proposed, A limited-preemption scheduling technique with benefits of both preemptive and non-preemptive scheduling. The runtime behavior of preemptive EDF scheduling is improved by avoiding preemptions that are not needed to satisfy the schedulability of the system. Algorithm for automatically setting a number of preemption points within the code of each task, to minimize the overall preemption cost under schedulability constraints is proposed by M. Bertogna et al. in [11]. A task decomposition algorithm is proposed by Abusayeed Saifullah et al.to decompose each parallel task into a set of sequential tasks with deadline distributed for the sequential tasks. EDF scheduling is executed for each of the sequential tasks [12].

An efficient scheduling approach is proposed by Kalyan Baital et al. [13] in the heterogeneous multicore system to map the tasks into appropriate cores based on the type of cores (big high power and small low power) as well as type of tasks (low power and high power). A run-time dispatcher dispatch different jobs into small cores. Task migration is used to migrate the rejected jobs from small cores into big cores through dispatcher. K. Baital et al. [14] presented a scheduling algorithm where random tasks generated at different time intervals with different periodicity and execution time can be accommodated into a system, which is already running a set of tasks, meeting the deadline criteria of the tasks. Using the concept of Pfair scheduling, random new tasks have been divided to fit into the idle times of the different cores of the system.

Genetic Algorithm based scheduling of tasks to multi-processor system is proposed by A. S. Radhamani et al. in [15] with the objective of minimizing the total completion time of all task and maximizing the utilization of processor. Fitness function is defined based on objective and genetic algorithm to use to search for the prospective solutions matching the objectives. J. Regehr in [16] show how to schedule two novel scheduling abstractions that overcome limitations of existing work on preemption threshold scheduling. The important weakness of MLFQ scheduler is the inability to define the optimized number of the queues and quantum of each queue also this algorithm has starvation for some process. These factors affect the response time directly. In [17] Parvar Mohammad et al., a new algorithm is presented to solve these problems and minimize the response time simultaneously. The proposed idea to solve the starvation problem while considering the processes priority too. In this algorithm Recurrent Neural Network has been utilized to find both the number of queues and the optimized quantum of each queue. K. Baital et al. [18] presents a scheduling algorithm where random tasks generated at different time intervals with different periodicity and execution time can be accommodated into a system, which is already running a set of tasks, meeting the deadline criteria of the tasks. A hybrid limited-preemption real-time scheduling algorithm is derived in [19] by M. Bertogna et al., that aims to have low runtime overhead while scheduling all systems that can be scheduled by fully preemptive algorithms. This hybrid algorithm permits preemption where necessary for maintaining feasibility, but attempts to avoid unnecessary preemptions during runtime.

The problem of scheduling periodic real-time tasks on multiprocessors under the fork join structure used in OpenMP has been discussed by K. Lakshmanan et al.[20]. An Integer Linear Programming (ILP) formulation and two non-iterative heuristics for scheduling a task-based application onto a heterogeneous many-core architecture has describe by C. Tan et al.[21]. Pricopi M et al. proposed work [22] to prove that adaptive multi-core architectures can substantially decrease the makespan compared to both static symmetric and asymmetric multi-core architectures. A. Hatami et al. [23] proposed scheduling algorithm that able to perform better than traditional Earliest Deadline First (EDF) and minimize the overall completion time when the system in overloaded condition. T VAIRAM et al.[24]proposed work which presents a hybrid version of DPSO(Discrete Particle Swarm Optimization) which is a combination of DPSO and Cyber Swarm Algorithm (CSA). The efficiency of the algorithm is tested using the performance metrics such as makespan, mean flow time, reliability cost, fitness and resource utilization. The proposed Ant Colony Optimization (ACO) algorithms provide inherent parallelism, which is very useful in multiprocessor environments [25]. An optimized real time scheduling algorithm for tasks allocation in grid environment has proposed by Aghazarian V et al.[26]. Scheduling of shared memory with multi-core performance in dynamic allocator using arm processor has implemented by Venkata Siva Prasad Ch. et al. [27]. Survey of different scheduling algorithms have been discussed by Imran Qureshi [28].

3. RLBMCS SCHEDULING

Differing from previous solutions focusing only on deadline, the proposed RLBMCS scheduling, schedules tasks to multi cores in a way to optimize multiple objectives of,

- CPU Utilization(C)
- Throughput (T)
- Turnaround time (TA)
- Waiting time (WT)
- Response time (RT)
- Deadline meet ratio (DR)

3.1. Reinforcement learning

Reinforcement learning is a kind of machine learning technique where an agent learns and fine-tunes its behavior in environment based on the results of its past actions. It is goal-oriented algorithm guided by principle of reinforcement with rewards for right actions and penalty for wrong actions. By this way, it attains its objective. Reinforcement learning is modelled in terms of agents, environments, states, actions and rewards as shown in Figure 1.

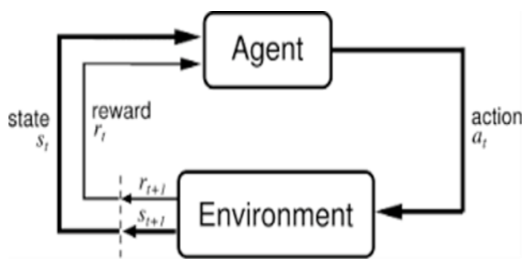


Figure 1. Reinforcement learning

- Agent: Agents takes actions.
- Action (A): A is the set of all possible moves the agent can make.
- Environment: The agent moves in this world.
- State(S): A state is a concrete and immediate situation in which the agent finds itself.
- Reward (R): A reward is the feedback by which we measure the success or failure of an agent's actions

Reinforcement learning represents an agent's attempt to approximate the environment's function, such that actions can be sent into the black-box environment that maximizes the rewards it spits out. It learns sequences of actions that will lead an agent to achieve its goal or maximize its objective function.

3.2. RLBMCS

The architecture of the proposed RLBMCS scheduling algorithm is given in Figure 2. Dispatcher runs in one core and it is responsible for scheduling of tasks to the multi-cores. A global multilevel feedback queue is used to place the tasks and multi-core retrieves the tasks from non-empty position of the Queue. The number of slots to be allocated to the task on the processor is dependent on the queue from where the task is fetched. By this way, the preemption interval is varied for each task. Most of the previous solutions, performed poorly because of too many preemptions and performance was improved if dynamic preemption period can be chosen based on task nature and preemption is controlled. From this point of view, the preemption period is made discrete in the proposed system by allocating different time quantum for each queue and processor allocate that time quantum to the task based on the queue from which the task is fetched. The incoming tasks and the preempted tasks are scheduled by the dispatcher to multilevel feedback queue based on the multi objective optimization realized using Reinforcement learning. The Reinforcement learning is mapped to RLBMCS as follows. Basically, reinforcement learning is based on the envoironment, agent, state, action and reward basis. To map the reinforcement learning application to our scheduling the same have been mapped as shown in Table 2.

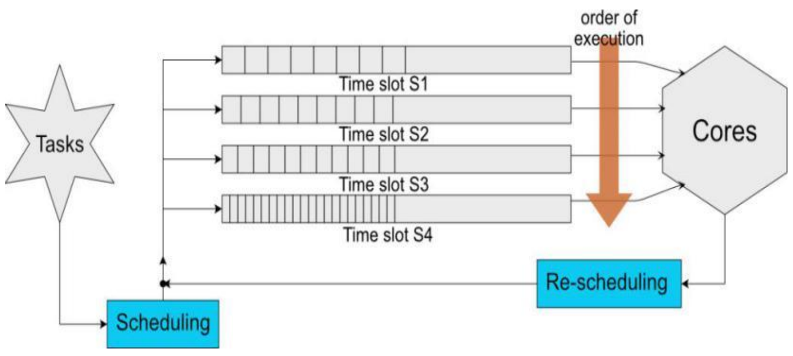


Figure 2. RLBMCS architecture

Table 2. Declaration of objects for reinforcement learning

Environment	Multicore Processor
Agent	Dispatcher
State	Four states are defined 1. Initial State(S1) 2. Objective Degradation stage(S2) 3. Objective Progression Stage(S3) 4. Objective Stabilization Stage(S4)
Action	Mapping the task to one of the N queues
Reward	The value of the multi objective function.

The multi objective function is modelled as,

$$O=w1*C+w2*T-w3*([TA/TAe]-1)-w4*([WT/WTe]-1)-w5*([RT/RTe]-1)+w6*DR$$

TAe is the SLA (Service level agreement) defined turnaround time
WTe is the SLA defined waiting time
RTe is the SLA defined response time

The weights values (w1, w2…w6) are assigned based on the importance to the parameters as defined by environment. Say if DR is important, the w6 value will be given a higher value compared to other weights. For each iteration, we get the next O value, it could be positive in case of better or negative in case of worse scenario, accordingly the state would be changed. The state transition is shown in Table 3. The value of M is the degree of confidence for migrating from S2 to S3. The value of P is the degree of tolerance from S3 to S2 and transition from S4 to S2. The value of T is the degree of stability for migrating from S3 to S4. There are N actions corresponding to allocation of task to any of the N queue. At each state, the action to be invoked is done with constraints on the task nature (its CPU/IO ratio), the time it has spent so far, deadline etc. Based on the constraints, action to place the task to corresponding queue is implemented, which are given in Table 4.

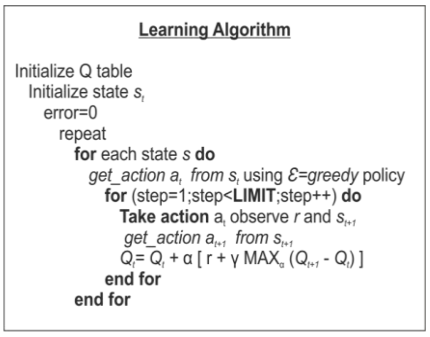
Table 3. State transition of tasks from one queue to another queue

	S1	S2	S3	S4
S1	X	Negative O value	Positive O value	X
S2	X	Negative O value	Positive O value over last M steps	X
S3	X	Negative O value over last P steps	Positive O value	Positive O value over last T steps
S4	X	Negative O value over last P steps	X	Positive O value

Table 4. Actions to be taken by tasks

A1	For a new task which just arrived allocate to Q0 (highest priority)
A2	For task preempted, allocate to one level below its last Queue allocated
A3	For task preempted, allocate to one level above its last Queue allocated
A4	For task preempted, allocate to same level of last Queue allocated

The actions are done in each state for allocation of tasks to the corresponding queue and the best possible actions to be taken for achieving the stable state of S4 is learnt for different nature of jobs and arrival in a test environment. The rules learnt are then used in production run for tasks in that corresponding environment. The pseudo code of the learning procedure is given below,



```
Learning Algorithm

Initialize Q table
   Initialize state $s_t$
      error=0
         repeat
            for each state $s$ do
               get_action $a_t$ from $s_t$ using $\mathcal{E}$=greedy policy
                  for (step=1;step<LIMIT;step++) do
                     Take action $a_t$ observe $r$ and $s_{t+1}$
                      get_action $a_{t+1}$ from $s_{t+1}$
                     $Q_t = Q_t + \alpha [ r + \gamma MAX_a (Q_{t+1} - Q_t) ]$
                  end for
            end for
```

The proposed RLBMCS scheduler is realized as below. There are 5 queues defined [Q0, Q1, Q2, Q3, Q4]. All tasks which arrive as events are first put in Q0. Tasks in Q0 are taken by scheduler and priority of the task is calculated using the proposed RLMBCS priority calculation algorithm. The priority is from 1 to 4. Based on priority, the tasks are allocated to queues. Hence Q1 gets the tasks of priority 1, Q2 gets priority of 2, and so on. Each of Queue from Q1 to Q4 is allocated to one processor each. For each processor, there is a task routine which takes the tasks from their corresponding tasks and uses it for execution in Round Robin with a fixed quantum. The slots are allocated to processor of each queue as, 3 ms for Q1, 2.5 ms for Q2, 2 ms for Q3, and 1.5 ms for Q4. The tasks in Q1 are given more time slot for execution than tasks in Q4. After the time slice completion, the tasks are again sent to Q0 if not completed.

4. PERFORMANCE RESULT

The proposed RLBMCS scheduling is implemented on Free RTOS operating system and the performance of the solution is evaluated against MLFQ scheduling algorithm proposed in [1]. The performance is compared in terms of,
- Average turnaround time
- Completion time
- Response time

In Free RTOS, one thread is made to spawn tasks with different priority and execution time at different rates per seconds and these tasks are executed on the cores available in the machine. The test is conducted with following parameters as shown in Table 5 and the results obtained are tabulated in Tables 6, 7, and 8.

Table 5. Initialization

No of processors	2 to 4
Number of Tasks	100
Queue Size	10
SLOT Interval	10 milliseconds

Table 6. The performance result for the case of 2 CPU

X	RLBMC	MLFQ
Average turnaround time(s)	90	276
Completion time(s)	801	1231
Response time(ms)	365	489.3

Table 7. The performance result for the case of 3 CPU

X	RLBMC	MLFQ
Average turnaround time(s)	46.79	145.80
Completion time(s)	541	911
Response time(ms)	245	407.8

Table 8. The performance results for the case of 4 CPU

X	RLBMC	MLFQ
Average turnaround time(s)	19.2	90
Completion time(s)	411	721
Response time(ms)	181.2	337.2

The average turnaround time for different number of processors is plotted below in Figure 3. The average turnaround time in the proposed RLBMCS scheduling is far better than that of MLFQ. The reason for this performance is due to adaptive movement of the Tasks across the Queues in RLBMCS when compared to continuous downgrade of priority in MLFQ. The completion time of the tasks for varied number of processors is given in Figure 4.

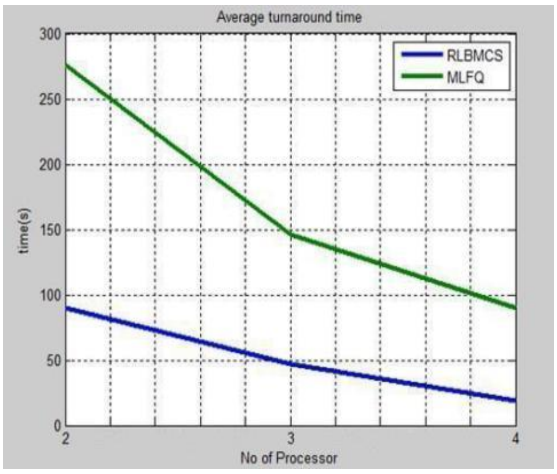


Figure 3. Comparison of turnaround time for RLBMCS with MLFQ

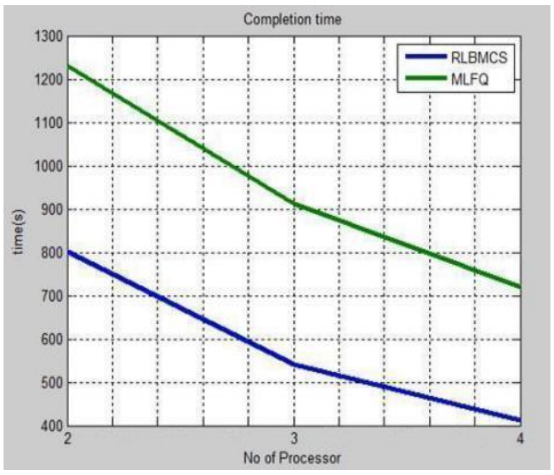


Figure 4. comparison of tasks completion time for RLBMCS with MLFQ

From the results, the completion time in RLBMCS is lower than that of MLFQ thereby increasing the processor utilization. The reason for reduction in completion time is due to preemption of the tasks when any IO is present and downgrading the priority on IO in case of MLFQ. The response time measurement for different processors is plotted below. From the result, the response time in the RLBMCS is far better than MLFQ. This reduction in response time is due to adaptive priority adjustment and movement in queue levels, so that fair response time is ensured. The response time is measured for different arrival of tasks and the result are shown above in Figures 5 and 6. The response time is low in case of RLBMCS compared to MLFQ. The context switch overhead in terms of CPU cycles is low in the RLBMCS compared to MLFQ Scheduler as given in Table 9. The CPU Utilization across all core is measured and plotted in Figure 7. The CPU Utilization is increased in RLBMCS scheduling compared to MLFQ Scheduler. The average makespan for different task arrival rate is shown in Figure 8. The average make span is reduced in RLBMCS Scheduling compared to MLFQ.

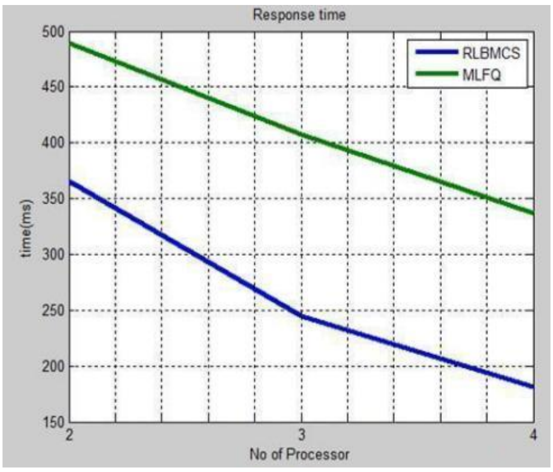


Figure 5. comparison of Response time for RLBMCS with MLFQ

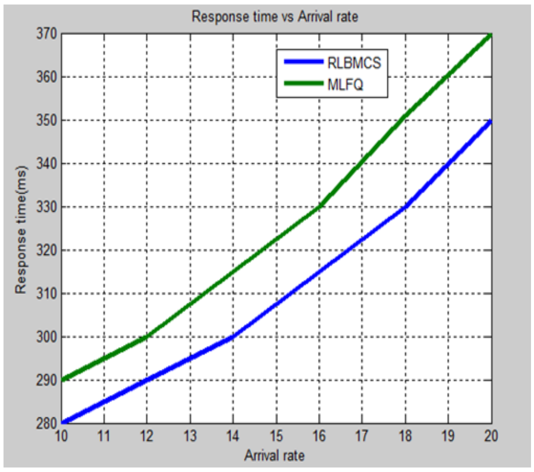


Figure 6. comparison of Response time for different arrival time for RLBMCS with MLFQ

Table 9. Context switch overhead

	RLBMC	MLFQ
10 tasks average	464	496
10 tasks standard deviation	32	33
30 tasks average	578	612
30 tasks standard deviation	102	102

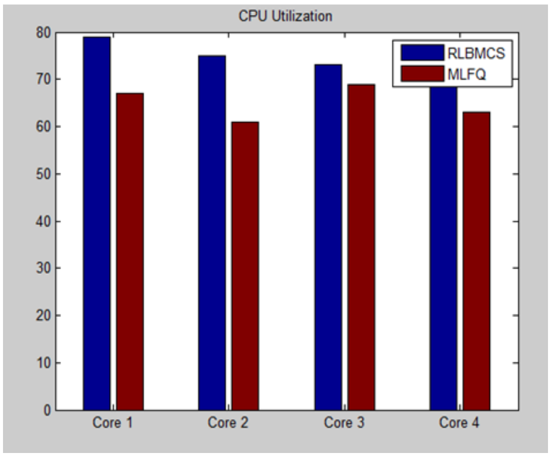


Figure 7. Comparison of CPU utilization for RLBMCS with MLFQ

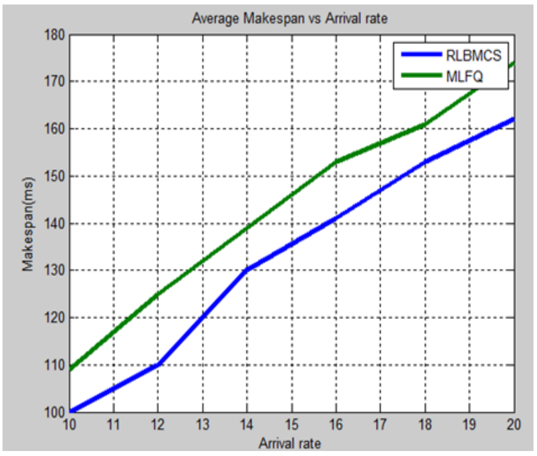


Figure 8. Comparison for Average Make span for RLBMCS with MLFQ

Summary of the results is given below:
- The response time is reduced in proposed RLBMCS compared to MLFQ
- The context switch overhead in terms of CPU cycles is low in proposed RLBMCS
- Average make span is low in proposed RLBMCS
- The task completion time and turnaround time is reduced in proposed RLBMCS

5. CONCLUSION

A Reinforcement learning based scheduling of tasks to multi core environment is proposed. The solution optimizes multi objectives in placement of tasks. Dynamic quantum for each task with reduction of preemption, Speedup or slow down based on the current load and the nature of task are all the salient features in this method of scheduling. The proposed scheduler was implemented on Free RTOS and the performance was evaluated against number of cores. The proposed solution performs better than MLFQ scheduling for multi cores.

ACKNOWLEDGEMENTS

We acknowledge the diligent efforts of Mr YVS Ravikanth in guiding towards implementation of this idea. Dhruva is thankful to Dr. Asharani for mentoring and her support and guidance for this work.

REFERENCES

[1] Seifedine Kadry, Armen Bagdasaryan "New Design and Implementation of MLFQ Scheduling Algorithm for Operating Systems Using Dynamic Quantum" *Statistics, Optimization And Information Computing*. Vol. 3, pp.189-196, June 2015.

[2] Xiaojie Li and Xianbo He "The improved EDF Scheduling Algorithm for Embedded Real-time System in the Uncertain Environment" ICACTE 2010.

[3] Jinkyu Lee and Kang G. Shin, "Preempt a Job or Not in EDF Scheduling of Uniprocessor Systems," *IEEE Transaction on computers*, vol. 63, no. 5, May 2014.

[4] J. Lee, A. Easwaran, and I. Shin, "Maximizing contention-free executions in multiprocessor scheduling," in *Proc. IEEE Real-Time Technol. Appl. Symp*. (RTAS), pp. 235–244, 2011.

[5] J. Lee, A. Easwaran, I. Shin, and I. Lee, "On optimal multiprocessor scheduling considering concurrency and urgency," in *RTSS Work-in-Progress Session*, pp. 21–24, 2010.

[6] J. Lee, A. Easwaran, and I. Shin, "LLF schedulability analysis on multiprocessor platforms," in *RTSS*, pp. 25–36, 2010.

[7] S. K. Lee, "On-line multiprocessor scheduling algorithms for real-time tasks," in *IEEE Region 10's Ninth Annual International Conference*, pp. 607–611, 1994.

[8] M. L. Dertouzos and A. K. Mok, "Multiprocessor on-line scheduling of hard-real-time tasks," *IEEE Transactions on Software Engineering*, vol. 15, pp. 1497–1506, 1989.

[9] A. Easwaran, I. Shin, and I. Lee, "Optimal virtual clusterbased multiprocessor scheduling," *Real-Time Systems*, vol. 43, no. 1, pp. 25–59, 2009.

[10] Marko Bertogna and Sanjoy Baruah, "Limited Preemption EDF Scheduling of Sporadic Task Systems," *IEEE Transactions on Industrial Informatics*, Vol. 6, no. 4, November 2010.

[11] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo, "Optimal selection of preemption points to minimize preemption overhead," in *Proc. 23rd Euromicro Conf. Real-Time Syst. (ECRTS'11)*, Porto, Portugal, Jul. 6–8, pp. 217–227, 2011.

[12] Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu "Multi-core Real-Time Scheduling for Generalized Parallel Task Models" *IEEE Real-Time Systems*, 2011.

[13] Kalyan Baital, Amlan Chakrabarti, "Dynamic Scheduling of Real-Time Tasks in Heterogeneous Multicore Systems" *IEEE Embedded Systems Letters*, 2018.

[14] K. Baital and A. Chakrabarti, "An Efficient Dynamic Scheduling of Tasks for Multi-core Real Time Systems", in *Advances in Computing Applications, Chakrabarti A., Sharma N., Balas V. (eds)*, Singapore: Springer, pp. 31-47, 2016.

[15] A. S. Radhamani and E. Baburaj, "Performance Efficient Heterogeneous Multicore Scheduling Strategy based on Genetic Algorithm", *ARPN Journal of Engineering and Applied Sciences*, vol. 8, no. 1, pp. 26-32, 2013.

[16] J. Regehr, "Scheduling tasks with mixed preemption relations for robustness to timing faults", *Proc. 23rd IEEE Real-Time Syst. Symp.*, pp. 315-326, Dec. 2002.

[17] Parvar Mohammad, M.E. Parvar, and Safari Saeed, "A Starvation Free IMLFQ Scheduling Algorithm Based on Neural Network," *Int.J. Computat. Intell. Res*., vol. 4, no.1, 27–36, 2008.

[18] K. Baital, A. Chakrabarti, A. Chakrabarti, N. Sharma, V. Balas, "An efficient dynamic scheduling of tasks for multicore real-time systems" in Advances in Computing Applications, Singapore:Springer, pp. 31-47, 2016.

[19] M. Bertogna and S. Baruah, "Limited preemption EDF scheduling of sporadic task systems*," IEEE Trans. Ind. Informat.*, vol. 6, no. 4, pp. 579–591, Nov. 2010.

[20] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *2010 31st IEEE Real-Time Systems Symposium,* San Diego, CA, pp. 259-268, 2010.
[21] C. Tan, T. S. Muthukaruppan, T. Mitra, L. Ju, "Approximation-aware scheduling on heterogeneous multi-core architectures", *Proc. ASP-DAC/IEEE*, pp. 618-623, 2015.
[22] Pricopi M, Mitra T., "Task scheduling on adaptive multi-core," *IEEE Trans Comput* 63(10):2590–2603, 2014.
[23] A. Hatami, S. Chuprat, H. Md Sarkan, N. Firdaus Mohd Azmi "Hybrid Real-Time Task Scheduling Algorithm in Overload Situation for Multiprocessor System", *JTEC,* Vol 9, No 3-4, 2017.
[24] T. Vairam, S. Sarathambekai "Multiprocessor task scheduling problem using hybrid discrete particle swarm optimization",Sådhanå, 43:206, December 2018.
[25] A. Shah and K. Kotecha, "ACO based dynamic scheduling algorithm for real-time multiprocessor systems," *International. Journal.of Grid and High Performance Computing,* vol. 3, no. 3, pp. 20–30, 2011.
[26] Aghazarian V, Ghorbannia A, Motlagh NG, Naeini MK, "RQSG-I: an optimized real time scheduling algorithm for tasks allocation in grid environments," 2011 IEEE 3rd International Conference on Communication Software and Networks, Xi'an, pp. 205-210, 2011.
[27] Venkata Siva Prasad Ch. and S. Ravi "Scheduling Of Shared Memory With Multi - Core Performance In Dynamic Allocator Using Arm Processor" *ARPN Journal of Engineering and Applied Sciences,* Vol. 11, No. 9, May 2016.
[28] Imran Qureshi, "CPU Scheduling Algorithms: A Survey" *Int. J. Advanced Networking and Applications,* Vol. 05, Issue 04, Pages: 1968-1973, 2014.

BIOGRAPHIES OF AUTHORS

Dhruva R. Rinku was born in Ahmedabad, India in 1973. She received her M.Tech. degree in digital systems & Computer Electronics from JNTU, Hyderabad, India in 2009.Presently she is pursuing her research work on schedulers for RTOS on multicore processors. She has been working as Associate Professor in CVR College of Engineering. Hydeabad and is responsible for microprocessors and embedded systems related subjects in the department of Electronics and Communication Engineering.

Dr. M. AshaRani was born in Srikakulam, India. She received her M.Tech. degree in digital systems & Computer Electronics from JNTU, Hyderabad, India in 1997. She received her PhD in Electronics & Communication Engineering from JNTU Hyderabad in 2008. Presently she is working as Professor in College of Engineering, JNTU, Hyderabad in the department of Electronics and Communication Engineering with over 26 years of teaching experience. Her research interests include Fault Tolerant Systems, Design for Testability, Real Time Embedded System Design and Multi-core Embedded System Design.