A Hybrid Multiprocessor Scheduler for Soft Real-Time System with Processor Affinity Concept

Ms Jayna Donga¹ and Dr. M.S. Holia²
¹ PhD Scholar, Gujarat Technological University, Ahmedabad, Gujarat
² Assistant Professor, BVM Engineering College, Gujarat
E-mail: jaynambict@gmail.com

Abstract. Now a days numerous real-time OS like QNX, VxWorks, LynxOS and real-time extension of Linux practicing the processor affinity concept to schedule the real-time tasks and it offers more flexible approach instead of traditional approaches defined in the literature. Limiting the migration may reduce the context switch overhead and improve the cache performance but it largely influence on other parameters which are most important for the any real-time operating system. It degrades overall performance by decreasing the schedulability of tasks, increase the deadline miss ratio. This paper presents the problem in existing processor affinity based approach which confines the schedulability of the tasks and designed a novel processor affinity based algorithm to enhance the schedulability of the tasks and decreases the average deadline miss ratio by providing the flexible migration policy with the priority reassignment mechanism.

Keywords: Processor Affinity, Soft real-time System, Schedulability, Multiprocessor system, Dead-line Miss Ratio

1. Introduction
The system performance of any real-time system doesn’t depend only on correct outcome but it must had been generated within its deadline also [1-3]. The crucial part of any real-time system is to meet the deadline and RTOS can help to attain this aim by providing the scheduling mechanism. The part of an OS called scheduler which is responsible to take scheduling decision like resource allocation and de-allocation to particular task [4,5,9,10, 21]. The goals of scheduler in any general purpose OS is to attain fairness, maximize throughput, minimize response time and maximize resource utilization etc. In case of Real-time OS the main goal of scheduler is meeting the deadline [6, 14]. The scheduler can allocate and de-allocate the computing resources to the various tasks based on its priority(The task who has small deadline is having highest priority) in such a way so overall schedulability can be increased and reduce the deadline miss ratio.

2. Background Theory
In the current era, most of the systems are having multiprocessor architecture. In uniprocessor system the scheduler focus on only process selection that is which process should be executed on which processor in which order. In case of multiprocessor real-time system scheduler needs to address two different problems [11-13,18,22-33]: i) The Process selection issue (priority assignment): What priority should be assigned to each job/task in the ready queue.ii) The Processor selection issue: on which processor the task/job should be executed. The higher priority tasks/jobs must be scheduled first so deadline can be met both the problems; processor selection and priority assignment closely related to each other so one can combine any priority assignment approach with any processor selection approach[16]. There are three categories of priority assignment approaches; i) The task level fixed priority approach(e.g. Rate Monotonic Algorithm) [14] ii) job level fixed priority approach(e.g.
Earliest Deadline First Algorithm) [14] iii) job level dynamic algorithms (e.g. Least Laxity first Algorithm) [14]. As described above, multiprocessor real-time system addresses the processor selection problem along with process selection problem. Numerous research carried out on processor selection issue. All the processor selection approaches can be put in three categories; a) Global Scheduling approach: This approach allow the task/job to execute on any free processor [6, 16, 17], b) Partitioned Scheduling approach: The task can be executed on fixed processor no task migration is allowed [6, 16, 17], c) Hybrid Scheduling approaches: This approach allows a task to migrate within particular cluster [12, 13]. The literature describes the Semi-partitioned [8, 11] and restricted-migration approaches [7, 11] also. In semi partitioned approach most of the tasks had assigned a fixed CPU and only few are permitted to migrate.

3. Objectives

There are two main objectives for this proposed scheduling algorithm:
Increase the overall schedulability
Decrease the average deadline miss ratio

4. Real Time System Model

This section offers the real-time task model, definition of various terms and assumptions are taken into consideration for performing this analysis. This article presents the real-time system that includes the taskset of n periodic real-time tasks which can be scheduled on m identical processors. The task-set has been defined as $\tau = \{T_1, T_2, \ldots, T_n\}$ and processor set $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$. It is assumed that every task is periodic then $T_k = (e_k, d_k, p_k)$, where $e_k$ is worst-case execution time of the task, $d_k$ is the relative deadline and $p_k$ is the period of task. If the deadline of a task is not specified explicitly then by default equal to the period of that task else relative deadline can be given explicitly that would be less than the period of a task. The processor affinity of task $T_i$ is $\alpha_k$, where $\alpha_k \subseteq \pi$ and Utilization of the task is defined as $u_k = e_k/p_k$. The total utilization of a taskset would be $U = \sum_{i=0}^{n} u_i$. The task set called schedulable if the utilization factor becomes less than m for the multiprocessor system with m processors. In this article the assumption has been made that the processor affinity is set from the inception of the task during its execution and kept fix throughout the execution.

5. Real-Time Task Scheduling and Processor Affinity

The present-day real-time operating systems like QNX, RTEMS, Lynux and real-time extensions of Linux are using the idea of processor affinity in place of traditional approaches discussed in literature for implementing their scheduler which is more flexible way of task migration in multiprocessor systems. Many RTOS has applied the processor affinity concept for restricted migration. The set of processors on which the jobs of that task can be executed is called the Processor affinity of any task. Even though all the processors under job’s affinity are busy in higher priority job, the job cannot be executed on the free processor which is not in its affinity mask. Any task would be allowed to schedule only on those processors which are in its processor affinity defined by $\alpha_k$, where $\alpha_k \subseteq \pi$. As per the requirement of application[5-8]. Observant assignment of processor affinity can help to achieve better schedulability time for real-time system. One can change the processor affinity by using the kernel APIs; for example the Linux kernel which provides sched_getaffinity() and sched_setaffinity() for to read and to change the affinity set of a task respectively [19–21]. In this paper it is assumed that the processor affinity would not be changed during task’s whole life time. The real-time workload is considered for scheduling with processor affinity of task [19–21]. The processor affinity is used to present the conventional migration approaches described in literature. The
global approach can be applied if for every tasks the affinity is set to all processors. The partitioned approach can be applied if for every tasks the affinity is set to the only single processor. Any priority assignment approach can be used with processor affinity for processor selection. The concept of processor affinity can be understood by taking the example of Linux Push-Pull Scheduler in multiprocessor system. The load balancing requirements for run queue is also checked periodically by Linux scheduler so maximum CPU Utilization can be achieved. However, Linux scheduler is not flexible to move higher priority task on another processor for lower priority task which can’t be run elsewhere because of its processor affinity constraint; so controlled push/pull mechanism leads to failure from getting higher schedulability. In the article the scenario described in detail and presented an innovative approach for multiprocessor real-time system to overcome the limitations in existing system and increase the overall schedulability.

5.1. The Migration Opportunity Scenario for Real-Time Task

The priority inversion is the more common issue faced by uniprocessor and the similar problem also appears in multiprocessor system when the lower priority task blocks the higher priority task from its execution. Let’s understand this problem with example.

Consider the case with different migration opportunities in figure-1. We have one real time system which contains 3 processors and four tasks. The task priority order is $P_1 > P_2 > P_3 > P_4$ and processor affinity sets $\alpha_1 = \{\pi_1, \pi_2, \pi_3\}, \alpha_2 = \{\pi_2, \pi_3\}, \alpha_3 = \{\pi_1\}, \alpha_4 = \{\pi_2, \pi_3\}$. In figure-1(a), the processes $P_1, P_2, P_4$ are in running state and process $P_3$ is waiting. After some times the process $P_3$ comes in ready to run state but it will not be allowed to execute it on $\pi_4$ as it is already executing the higher priority process $P_1$. On the other side the task $P_4$ is being executed on processor $\pi_3$ which falls in the affinity set of process $P_1$. The solution may be to preempt process $P_4$ as the priority order is $P_3 > P_4$ and it can be shifted to processor $\pi_3$ to schedule process $P_3$ on $\pi_4$. One possible solution can be the preempting $P_1$ from the $\pi_3$ and scheduling the $P_1 (P_1 > P_3)$ from processor $\pi_1$ to processor $\pi_3$.

This push/pull will make feasible the execution of $P_3$ on $\pi_4$ and it will increase the overall schedulability of the system as shown in figure-1(b). One more solution is also possible as shown in figure-1(c); $P_1$ pushed to processor $\pi_2$ and $P_2$ pushed to processor $\pi_3$ so this solution required two context switches while in figure-1(b) require a single context switch so figure-1(b) can be cost effective solution in terms of switching overheads.
Figure 1 [12]: Figures (a)-(c) represents process scheduling state at different time with its affinity and possibilities of task shifting. (a) Initial state $t_0$ when $P_1$, $P_2$ and $P_4$ tasks are scheduled. (b) Probable state where $P_3$ is ready and scheduled on processor $\pi_1$ after shifting task $P_1$ on processor $\pi_2$. (c) Probable state where task $P_3$ is ready and scheduled on processor $\pi_1$ after shifting task $P_2$ to processor $\pi_2$ and shifting task $P_1$ to processor $\pi_2$.

The case shows that due to the migration constraints existing processor affinity based scheduler fails to get higher schedulability. The Schedulability can be increased by pulling lower priority task and migrating higher priority task to that processor. To determine which job should be migrated to achieve higher schedulability we use proposed approach to check task migration opportunities.

6. APA based scheduler with priority reassignment

The previous section presented the problem of existing processor affinity based scheduler. This section presenting a different approach for APA scheduler to overcome the problem appears in present scheduler and it decreases the deadline miss ratio and increases the overall schedulability.

The case described in figure-1 is taken into consideration to demonstrate the proposed algorithm in which $P_3$ is waiting but the Processor $\pi_1$ under its affinity is not free and running $P_1$ which has higher priority task than $P_3$. The $P_1$ can be pulled from the from $\pi_1$ if the priority of task $P_1$ becomes higher than $P_3$ and that can be done by reprioritization approach. The proposed algorithm set the priority of $P_3$ higher than $P_1$ so it can be pulled and $P_3$ can be scheduled on $\pi_1$, $P_1$ is ready task so it can be scheduled on processor $\pi_3$ as it is executing lowest priority task $P_3$ which can be preempted. After completion of this push/pull operation the task $P_1$ and $P_3$ would be set to their original priority. To solve the problem of priority inversion the priority ceiling protocol or priority inheritance protocol is used to swap the priority of the tasks. This solution must be guaranteed to execute all higher priority task successfully.

7. Flow of Proposed Algorithm

In the proposed approach as shown in figure-2, first the proposed algorithm check for the probability of the priority inversion problem. When any new task $T_k$ becomes ready to run scheduler checks that any processor in task’s affinity is free or not, if processor is free then no priority inversion problem and free processor is assigned to ready task $T_k$. But if priority inversion problem is found then task reprioritization would be done to decrease the deadline miss ratio and improve the overall schedulability. The priority inversion problem means the lower priority task blocking the higher priority process from occupying the computing resource. The priority inversion problem occurs if the task is ready to run but can’t be executed as the higher priority task is already running on the CPU in task’s ($T_k$) affinity and the CPU which is executing lowest priority task ($T_i$) is under the affinity of running higher priority task ($T_i$) so the ready to run task ($T_k$) with higher priority than $T_i$ will be blocked that is called priority inversion problem. Once the priority inversion found in the system the
proposed approach would recognize a Target_Processor, a Target_Task and then as per the requirement Target_Task push/pull to the target processor by reprioritization which is the priority of the ready to run task \( T_k \) is increased than \( T_i \) at kernel level. The reprioritization allows to integrate solution without altering any kernel API. The second approach as shown in figure-3 permits to change processor affinity of the ready to task \( T_k \) to Target_Processor so it pulls Target_Task. For example, if task to be scheduled has processor affinity \( \{p_i, p_j, p_k\} \) and tasks on these processors are \( \{T_i, T_j, T_k\} \) respectively, where \( p_i > p_j > p_k \). Now Target_Task is \( p_j \) and priority of task to be scheduled is set higher than \( p_j \). In that case, instead of preempting \( p_j \) it preempts \( p_k \). Setting affinity to only \( p_j \) it only preempts \( p_j \).

**Figure-2. Check for Priority Inversion**

**Figure-3. Task Priority Reassignment**
8. Experimental Setup and Result Analysis

This section presents the experimental setup and result analysis. In this article, result analysis is done on average deadline miss ratio and average schedulability analysis for multiprocessor real-time system. First of all the viability of proposed algorithm has been checked through the RTSim simulator for real-time system [7]. The SimSo API is used for Task set generation [6]. The Linux Testbed Litmus-RT is used for implementation and experimental work [14,15]. The proposed approach has been compared with P-FP and G-EDF. The DkC approach has been used for priority assignment [1] and period distribution is taken log-uniform [2]. This article presents a generalized processor affinity based scheduler gives realization of similar like partitioned, clustered and global tasks in a taskset and then assigns uniform processor affinity like high-utilization tasks had been given partitioned-like affinities and high-density tasks had been given global-like affinities so that they can easily migrate to other processors and the execution can be completed as early as possible. It has been taken into account that the overall utilization must be kept as minimum as possible so system load would be balanced. In case-1, calculated average deadline miss ratio for different task set size like 4, 8, 16, 32 and 64 scheduled on eight processors. In case-2, calculated average schedulability for different task set size like 4, 8, 16, 32 and 64 scheduled on eight processors. The case-3 presents average deadline miss ratio for different task set size on 16 processors. The case-4 presents average schedulability for different task set size on 16 processors. At the time of transient overload proposed algorithm gives better result than that of traditional approaches as our approach provides more flexible migration policy by priority reassignment. The result shows that proposed approach increases the overall schedulability and reduces the deadline miss ratio. The context switch analysis is not done here it may increase in some case.

Case-1: Average Deadline Miss Ratio when different Task set scheduled on 8 processors

| Task Set Size | P-FP  | G-EDF | Proposed |
|---------------|-------|-------|----------|
| 4             | 0.2306| 0.0532| 0.007025 |
| 8             | 0.50  | 0.11  | 0.035    |
| 16            | 0.347 | 0.787 | 0.0213   |
| 32            | 0.347 | 0.503 | 0.0821   |
| 64            | 0.416 | 0.335 | 0.0124   |

![Figure-4.Average Deadline Miss Ratio when different Task set scheduled on 8 processors](image)
Case-2: Average Schedulability when different Task set scheduled on 8 processors

| Task Set Size | P-FP | G-EDF | Proposed |
|---------------|------|-------|----------|
| 4             | 65%  | 88%   | 92%      |
| 8             | 78%  | 82%   | 89%      |
| 16            | 57%  | 76%   | 85%      |
| 32            | 63%  | 84%   | 90%      |
| 64            | 66%  | 73%   | 81%      |

![Figure-5](image-url) Average Schedulability when different Task set scheduled on 8 processors

Case-3: Average Deadline Miss Ratio when different Task set scheduled on 16 processors

| Task Set Size | P-FP | G-EDF | Proposed |
|---------------|------|-------|----------|
| 4             | 0.227| 0.222 | 0.007025 |
| 8             | 0.33 | 0.41  | 0.0981   |
| 16            | 0.519| 0.850 | 0.0213   |
| 32            | 0.396| 0.304 | 0.0821   |
| 64            | 0.432| 0.287 | 0.0124   |

![Figure-6](image-url) Average Deadline Miss Ratio when different Task set scheduled on 16 processors
Case-4: Average Schedulability when different Task set scheduled on 16 processors

| Task Set Size | P-FP  | G-EDF  | Proposed |
|---------------|-------|--------|----------|
| 4             | 54%   | 72%    | 79%      |
| 8             | 47%   | 85%    | 91%      |
| 16            | 62%   | 78%    | 87%      |
| 32            | 60%   | 83%    | 89%      |
| 64            | 45%   | 69%    | 76%      |

Figure 7. Average Schedulability when different Task set scheduled on 16 processors

9. Conclusion and Future scope
After the sufficient result analysis it is concluded that the proposed affinity based scheduler improves the overall schedulability and decreases the deadline miss ratio as compared to the traditional scheduling approaches like global approach and partitioned approach. The proposed approach uses the hard pre-emption migration policy with priority reassignment which may increase switching overhead. Context switch analysis is not considered here so one can try for context switches analyses. This approach is kept limited to dynamic priority as well as also kept fixed affinity but one can extend it to dynamic affinity assignment.

References
[1] R. Davis, A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems, Real-Time Systems, vol. 47, no. 1, 2011, p. 1-40
[2] P. Emberson, R. Stafford, R. Davis. Techniques for the synthesis of multiprocessor tasksets, 1st Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, 2010
[3] Dertouzos ML, Mok AK. Multiprocessor online scheduling of hard-real-time tasks. IEEE Transactions on Software Engineering, 1989, 15(12):1497 –1506
[4] Davis RI, Burns A. A survey of hard real-time scheduling for multiprocessor systems. ACM Computing Surveys, 2011. 43(4):35:1–35:44
[5] Dorin F, Yomsi PM, Goossens J, Richard P. (2010) Semi-partitioned hard real-time scheduling with restricted migrations upon identical multiprocessor platforms. CoRR abs/1006.2637
[6] “SimSo : Simulation of Multiprocessor Scheduling with Overheads,” project web site http://projects.laas.fr/simso/
[7] C. Liu and J. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” Journal of the ACM, vol. 20, no. 1, pp. 46–61, 1973.
[8] Burns A, Davis RI, Wang P, Zhang F. (2012) Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme. Real-Time Systems 48:3–33
[9] Kato S, Yamasaki N, Ishikawa Y. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In: Proceedings of the 21st Euromicro Conference on Real-Time Systems, ECRTS, 2009, p. 249 –258
[10] Anderson JH, Bud V, Devi UC. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In: Proceedings of the 17th Euromicro Conference on Real-Time Systems, ECRTS, 2005. p. 199–208
[11] Calandrino JM, Anderson JH, Baumberger DP. A hybrid real-time scheduling approach for large-scale multicore platforms. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems, ECRTS, 2007. p. 247–258
[12] Ms.JaynaDonga Dr.MehtuzaHolia, "Processor Affinity based Multiprocessor Scheduling Algorithm for Soft Real-Time System", Solid State Technology, ISSN:0038-111X, Volume-63, Issue:5, December 2020.
[13] Baker TP, Baruah SK. Schedulability analysis of multiprocessor sporadic task systems. In: Handbook of Realtime and Embedded Systems, CRC Press, 2007
[14] LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM, ACM, New York, NY, USA, v. 20, n. 1, Jan. 1973. ISSN 0004-5411. p. 46–61
[15] The original LITMUSRT paper: J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson, "LITMUSRT: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers ", Proceedings of the 27th IEEE Real-Time Systems Symposium, pp. 111–123, December 2006.
[16] The description of the current version: B. Brandenburg, “Scheduling and Locking in Multiprocessor Real-Time Operating Systems”, PhD thesis, UNC Chapel Hill, 2011.
[17] H.-C. Jang, H.-W. Jin, “Hoti”. High Performance Interconnects, In Proceedings of the 17th IEEE Symposium, 2009
[18] Foong, J. Fung, D. Newell, S. Abraham, P. Irelan, A. Lopez Estrada. Architectural characterization of processor affinity in network processing, In ISPASS, 2005.
[19] Foong, J. Fung, D. Newell. An in-depth analysis of the impact of processor affinity on network performance, in ICON, 2004.
[20] Gujarati, F. Cerqueira, B. B. Brandenburg. Multiprocessor real-time scheduling with arbitrary processor affinities: from practice to theory. Real-Time Systems, 51(4), 2015. p. 440–483
[21] S. Baruah, B. B. Brandenburg. Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities. In RTSS, 2013. p. 160–169
[22] Gujarati, F. Cerqueira, B. B. Brandenburg. Schedulability analysis of the Linux push and pull scheduler with arbitrary processor affinities. In ECRTS, 2013. p. 69–79
[23] E. Markatos, T. LeBlanc. Using processor affinity in loop scheduling on shared-memory multiprocessors. In: IEEE Trans. on Parallel and Distributed Systems, 5(4), 1994. p. 379 –400
[24] N. Kim, S. Tang, N. Ottermess, J. Anderson, F.D. Smith, and D. Porter, "Supporting I/O and IPC via Fine-Grained OS Isolation for Mixed-Criticality Real-Time Tasks", Proceedings of the 26th International Conference on Real-Time Networks and Systems, pp. 191-201, October 2018.
[25] M. Chisholm, N. Kim, S. Tang, N. Ottermess, J. Anderson, F.D. Smith, and D. Porter, "Supporting Mode Changes while Providing Hardware Isolation in Mixed-Criticality Multicore Systems", Proceedings of the 25th International Conference on Real-Time Networks and Systems (RTNS 2017), October 2017.
[26] N. Kim, M. Chisholm, N. Ottermess, J. Anderson, and F.D. Smith, "Allowing Shared Libraries while Supporting Hardware Isolation in Multicore Real-Time Systems", Proceedings of the 23rd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2017), April 2017.
[27] B. Brandenburg and M. Gül, "Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations", Proceedings of the 37th IEEE Real-Time Systems Symposium (RTSS 2016), December 2016
[28] D. Compagnin, E. Mezzetti, and T. Vardanega, "Experimental evaluation of optimal schedulers based on partitioned proportionate fairness", Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015), July 2015.
[29] B. Brandenburg, "A Synchronous IPC Protocol for Predictable Access to Shared Resources in Mixed-Criticality Systems", Proceedings of the 35th IEEE Real-Time Systems Symposium, 196-206, December 2014
[30] J. Kwon, K.-W. Kim, S. Puik, J. Lee, and C.-G. Lee, "Multicore Scheduling of Parallel Real-Time Tasks with Multiple Parallelization Options", Proceedings of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2015), 232–241, April 2015.
[31] L. Bonato, E. Mezzetti and T. Vardanega, "Supporting Global Resource Sharing in RUN-scheduled Multiprocessor Systems", Proceedings of the 22nd International Conference on Real-Time Networks and Systems (RTNS 2014), October 2014.
[32] G. Gracioli, "Real-Time Operating System Support for Multicore Applications", PhD thesis, Federal University of Santa Catarina, Florianópolis, Brazil, 2014
[33] B. Brandenburg, "Scheduling and Locking in Multiprocessor Real-Time Operating Systems", PhD thesis, UNC Chapel Hill, 2011.