Critical Scheduling for Soft Real-Time Operating Systems

Ph.D. Synopsis

Submitted to

Gujarat Technological University

for the degree of

Doctor of Philosophy

in

Computer /IT Engineering

by

Jayna K Donga

Enrolment No: 179999913010

Supervisor

Dr. Mehfuza S. Holia,

Assistant Professor,

Electronics Department,

B.V. M. Engineering College, Vallabh Vidhyanagar, Anand

DPC Members

Dr. Jagdish Rathod, Professor, B.V. M. Engineering College, Vallabh Vidhyanagar, Anand

Dr. N. M. Patel, Professor, B.V. M. Engineering College, Vallabh Vidhyanagar, Anand

TABLE OF CONTENTS

Table of Contents

1. Abstract

The scheduling approaches have largely been analyzed so far, but the attention given to multi-processor environment and real-time operating system is very less. Mostly all the approaches which are optimal are appropriate for the general-purpose operating system which cannot accomplish the real-time systems requirements as the RTS is closely concerned about the constraints of timing. The correctness of output is extremely affected by the deadlines means the output must be generated within a given time-bound. Nowadays, the real-time systems are multi-tasking and multiprocessing, so the scheduler plays an important role for controlling the functioning into the system. Different researchers had designed numerous approaches for real-time task scheduling to fulfill the varying needs of different environments like uniprocessor systems and multiprocessor systems. The selection of scheduling approach is an essential task as the output is greatly affected by which approach is being used in which environment. Furthermore, existing schedulers designed for supporting real-time systems aren't absolutely optimal for multiprocessor system. The schedulers designed earlier had focused on process selection which decides only the execution order of a task in the task set and also they are application based. Now a days, we are entering into the era of the high-performance computing which demands for the multi-processor environment. The uniprocessor system and the multiprocessor system; both are having their individual requirements and challenges as per their architecture. In the multiprocessor system the scheduler has to address both the problems like process selection (Priority assignment) as well as processor selection (set Processor affinity).All the existing approaches for multi-processor real-time system described in literature are divided into two categories; 1) Global scheduling approach 2) Partitioned approach and they all are application based approaches. The concept of affinity is used widely in non-real-time environment to improve the cache performance but not in soft real-time systems. In current era, The concept of processor affinity for scheduling real-time tasks in hard real-time systems are used by the various RTOS like QNX, LynxOS and the real-time extended version of Linux which can give more flexible and generalized scheduler rather than the application based traditional algorithms described in literature. The cache performance of some tasks can be improved with the restricted migration on real-time task but the significant drawback is that it decreases the overall schedulability of the real-time system. Meeting the deadline for improving the schedulability by reducing the number of deadline miss is the very important aspect in any real-time system. 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. The work presented in this research has implemented G-JLFP-APA scheduler for soft real-time system which is global approach using processor affinity and full migration with job-level-fix-priority approach. This research also presents another P-FP-APA scheduler for soft real-time system which is partitioned approach using processor affinity and no migration with fix-priority assignment approach. After analyzing the performance of G-JLFP-APA and P-FP-APA schedulers to combine the advantages of both the approach; The proposed generalized scheduler PrAMS (multi-processor scheduler based on processor affinity) has been designed for the soft real-time system which focuses on three different aspects: i) processor allocation which is done using static affinity assignment to achieve generalized approach ii) Process selection (priority assignment) which is done using JLDP(Job Level Dynamic Priority) to improve the efficiency of algorithm iii)Flexible Migration policy which is done by task shifting with priority reprioritization mechanism for improving the system's schedulability and to solve the priority inversion problem. The parameters taken into consideration are Schedulabilty, No. of Context Switches, Tardiness, Deadline Miss Ratio and CPU_Utilization. The overall schedulability of the soft real-time system is improved by the proposed scheduler as compared to conventional scheduling algorithms, decreases the average deadline miss ratio, reduces the tardiness and increases the CPU_Utilization in the multiprocessor system.

2. Brief description on the state of the art of the research topic

This section presents a review related to prior work for this research. We first of all discuss the basic theory of real-time scheduling. Afterward the different types of real-time scheduling approaches are analyzed for uniprocessor as well as for multiprocessor which provides the base of this research work. We summarize the techniques and its limitations that can be modified and new work can be performed.

2.1 Background Theory

This section describes the basics of scheduling like real-time task set model, classification of scheduling approaches for uni-processor and multi-processor systems and different migration techniques.

2.1.1 Real-Time Task Model

In any real-time systems, the simultaneous computational events are done for that the

rightness of result doesn't depend on logical output only but it also depends on the time at which it is produced. The event is said to be tasks and during its life cycle every task can have a job sequence .Thus a sequential unit of work is called the job. Time constraints for the task is bounded by the deadlines that indicates the task execution must be completed before given time. This section presents the real-time task model, definition and assumptions that had been taken to perform the analysis. We assume the classical sporadic task model followed by the task [1][45]. Here it is assumed that the real time system contains n real-time tasks $T_1, T_2, \ldots, T_n$, that forms a taskset $\boldsymbol{\tau} = \{ \mathbf{T_1}, \mathbf{T_2}, \ldots\ldots \mathbf{T_n} \}$.Every task spawns a sequence of m jobs an instance of task $T_{i.1}, T_{i.2}, T_{i.3}, \ldots. T_{i.m}$.

2.1.2 Classification of Scheduling Algorithms

Currently, the multi-tasking computer systems are used by us. In the multi-tasking system we can execute more than one task simultaneously. There are number of resources in our computer system that is either hardware or software. All the currently present tasks into the systems will share these resources. Which task will get which resource in which duration is the important decision in any computer system [2][9][39]. The part of OS which is responsible to take decision regarding allocation and De-allocation of resources during its execution is called scheduler [3][7][9]. Scheduling is the decision making process which performs the dissemination of different resources to various tasks as per their requirement.so a critical design issue in any operating system is to design an efficient scheduling algorithm. The scheduling approaches are designed with aim to balance the load, maximize resource utilization, minimize Tardiness, minimize deadline miss ratio, minimize no. of context switches and maximize schedulability[7][47]. The scheduling approaches designed for real-time systems so far can be categorized into two different classes based on the number of processor a real-time system contains.

i. Real-Time Scheduling in Uniprocessor System

In a single processor system, all the currently ready to run tasks will be executed on a single processor one by one in a particular order so that maximum tasks can be completed before their deadline. The scheduling approaches can be divided into two classes: 1) Static approach 2) Dynamic approach. In the first one, the task priorities are kept static which is assigned at the time of algorithm designing time whereas in second approach the priorities of the task is allocated at run time depending on various parameters and also it may be changed during its lifetime. Further the dynamic

scheduling can be divided into two categories fixed priority and dynamic priority. The sample for dynamic scheduling with static priority approach is the Rate Monotonic (RM) and Deadline Monotonic (DM) approach [6][10][48] whereas the Earliest Deadline First (EDF), Least Laxity First (LLF) are the cases for the dynamic scheduling approach with dynamic priority[8][13].

ii. Real-Time Scheduling in Multi-Processor System

In the multiprocessor system, more than one ready tasks in the system will be scheduled on different CPUs simultaneously in a particular order so that the maximum task complete its execution within its deadline [15]. So there should be appropriate approach for the selection of process and the processor amongst them. Two different issues addressed in the Multiprocessor scheduling are as following:

1) **The Processor selection problem**: It represents the task will be executed on which processor that is processor selection [4][14][30].

   Processor allocation problem further categories in three classes

   i) **No migration:** The task is fixed on particular processor and it must complete execution on the same processor no further migration will be allowed.

   ii) **Task-level migration:** Different jobs of the same task can be executed on different processors but every job can only be executed on the single processor.

   iii) **Job-level migration:** The migration of a single job is permitted and it can be executed on different processors.

2) **The Priority assignment / The Process selection problem:** It represents the order of execution for particular job of a task with reference to the other jobs of different tasks it should be executed.

   Priority assignment problem further can be classified into [5][11][14]

   i) **Fixed task priority:** Every task is having a single static priority which is applicable to all the jobs of that task. The example is Rate Monotonic (RM) scheduler

   ii) **Fixed job priority:** The different jobs of the same task can have different priorities but every job is having its single fixed priority. The example is the Earliest Deadline First (EDF) scheduler.

iii) **Dynamic priority:** A single job of a task can be assigned different priorities during its life-time. The example is Least Laxity First (LLF) scheduler.

There are various multiprocessor scheduling algorithms designed by the researchers but it can be divided into the two categories [7][10][41], any processor selection policy can be selected as per our application and it can be combined with any appropriate process selection algorithm. There are main two categories of the multiprocessor scheduling algorithm as described below:

i. Partitioned scheduling Approach [15]

First of all the tasks are partitioned amongst processors in partitioning scheduling approach and the task can be executed on fixed processor which is already assigned to it not on any other processors. In partitioned approach the number of migrations are zero as the tasks are not allowed to run on any other processors [16][32].

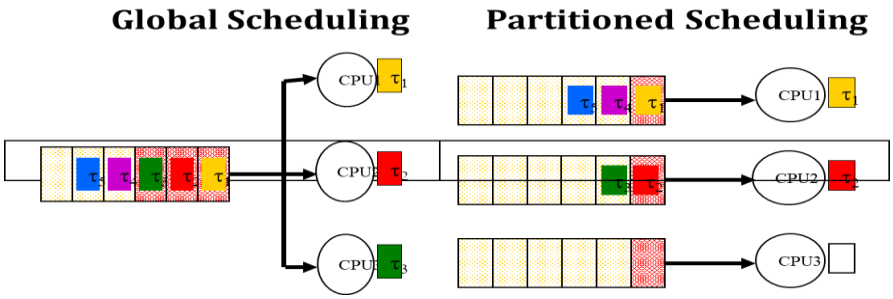

Figure 1.1: Difference between two Multiprocessor Scheduling Approaches [15]

ii. Global Scheduling Approach[12] [15][40]

The migration of the tasks on various processors in their execution duration is allowed in global scheduling and any task can be executed on any available processor [15][43].The figure 1.1 represents the architecture difference between partitioned approach and the global approach.

2.2 Literature Review

Now a days, the numerous real-time applications are using multiprocessor systems extensively. A semi-partitioned algorithm for multiprocessor soft real-time system for optimizing QoS has been presented by Behnaz Sanati et al. [34] whose aim is to decrease the

number of misses. A semi partitioned scheduling algorithm is the extension of partitioned scheduling approach with allowing migration to very few tasks of a task set thus it reduces the number of context switching. The greedy approach and load balancing two dynamic approximation approaches are used in proposed approach with an online semi partitioning policy in the soft real-time system with periodic task set. The aim is to reduce deadline miss ratio for improving the Quality of Service.

A multi-processor real-time Scheduler has been published by Hitham Alhussian et al. [31] has published in that complete relaxation given on the fairness rule. In the multiprocessor environment, Most of the present real-time scheduling approaches with optimal schedulability follow the fairness rule by providing no. of migrations and job pre-emptions that greatly affects the expectedness of the approach. This approach uses the global approach and provides fairness rule relaxation so that the no. of context switches and deadline miss ratio can be reduced.

The GPEDF scheduling approach presented by Li, Q. & Ba, W et. al.[33].At the time of priority assignment, in some cases the task set wants priority levels beyond the capacity of the system. This algorithm GPEDF addressed that issue by allocating the per group priority in that the group with the shortest deadline can have the maximum priority and it will be scheduled first. It is compared with conventional approaches EDF and the g-EDF. The outcomes proved that the GPEDF has less number of context switches, the short response time, reduced deadline miss ratio with very few priority levels.

The hybrid scheduler for weakly hard real-time system for multiprocessor environment has been presented by Habibah Ismail et.al.[17] that has less context switching overhead. The goal is to increase the global schedulability of a system by giving the hybrid approach with an effective assignment of jobs to the processor for taking benefit of both the approaches like partitioned and the global scheduling.

A Real-time scheduling algorithm CP-EDF(Controlled Pre-emptive EDF) has been presented by Keerthanaa C et. al. [32] in that restricted the number of pre-emptions for minimizing the number of context switches. The FP-EDF and Non-Pre-emptive EDF approaches compared with CP-EDF and it is proved that the CP-EDF gives improved results than existing two by reducing context switches, deadline miss ratio but it also decrease the CPU-Utilization.

R. Kalpana et. al. [36] has presented DC-gEDF policy for soft real-time system with zero context switching over head as it uses the non-pre-emptive policy. In this approach, based on task's domain description and its deadline requirements the clusters of the tasks will be created. After that the tasks will be scheduled which belongs to that group. The performance

analysis is done based on the deadline miss ratio of G-EDF and DC-gEDG. It is observed that that the improvement deadline miss ratio using proposed algorithm Domain-Cluster-gEDF.

The schedulability study of Linux scheduler based on APA concept in hard real-time system has been done by A. Gujarati et. al. [15] that mainly focuses on processor selection issue not on the process selection. They had formulated that the conventional approaches like the global, clustered, and partitioned scheduling can be strictly dominated by the APA-based scheduling and the paper proved that the processor_affinities are advantageous and it should be analysed properly. To design the enhanced analysis technique for scheduling using APA and strong scheduling approach by providing flexible migration policy can be an important area for the research.

The scheduling of soft real-time sporadic task systems under global EDF on an identical multiprocessor has been published by U.M.C. Devi et. al. [29].In the past research, the main focus was on hard real-time systems for global EDF but for the soft real-time systems the scenario is different as it tolerates bounded tardiness. The tardiness bounds for pre-emptive and non-pre-emptive global EDF in the multiprocessor system has been derived in this paper. It is proved that global EDF performs better than partitioned EDF for multiprocessor-based soft real-time systems. If any task completes a little bit late in soft real-time system then the result will not likely to be disastrous, but the tardiness must be bounded.

A scheduler G–EDF-like (GEL) has been presented by Jeremy P. Erickson [35] that discover the broader category of G-EDF-like (GEL) schedulers which is having equal overhead characteristics to G-EDF. They had shown that the selection of GEL schedulers gives the better tardiness than G-EDF.

A heuristic algorithm based on the earliness of deadline of tasks has been proposed by Xi Chen et. al. [37] to decrease the penalty of tardiness of tasks. The mobile edge computing system is taken into consideration as a soft real-time system and shown that how the task can be assigned and schedule to a server. The performance measure parameters which addressed here are deadlines miss ratio, throughput, and CPU_Utilization and Memory_Utilization.

Shiva Nejati et. al. [19] had analysed the CPU_Utilization limitations and presented a framework which is used to maximize the CPU_Utilization. They proposed a novel model for investigating the CPU utilization. The CPU_Utilization formulated by them as a constraint optimization problem and had given an implementation of their approach using optimization tool.

The problem of multiprogramming scheduling for Uniprocessor has been studied by C.L. LIU et. al. [28] which shows that for large task sets the CPU_Utilization has an upper

bound that is as low as 70% in optimum fixed priority scheduler. They concluded that the full CPU_utilization can be achieved by dynamic priority assignment based on urgency of their current deadlines and also it can be analysed for multi-processor real-time system.

Davis RI, Burns A et. al. [38] has done the survey for hard real-time schedulers and schedulability investigation techniques for homogeneous multi-processor environment which studied the key results in the real-time scheduling since 1960s to 2009. A taxonomy of the various scheduling approaches and consideration of different performance measure parameters are provided with comparison. A comprehensive survey has been given which covers partitioned, global, and hybrid scheduling approaches. The open issues, likely research directions and key research challenges are given by the review.

The schedulability analysis for multiprocessor sporadic task system on identical processors has been presented by Baker TP, Baruah SK et. al.[20] in that the investigation the regarding schedulability is done which can be called as feasibility and showed that the non-continuous-time schedule is equally effective as continuous time schedule.

Anderson JH et. al.[23] has published an EDF based multi-processor scheduler for soft real-time systems has been published by which proposed techniques and heuristics for reducing the tardiness. As per the information this paper is the first which investigating EDF scheduling in multiprocessor system for the soft real-time system. The esurience of bounded tardiness is the important contribution of a novel EDF based strategy. Here, the restricted per_task_utilizations but there is no need for restricting the overall utilization.

The Semi-partitioned scheduling of sporadic task systems on multiprocessors with arbitrary deadlines on identical multiprocessor environment has been presented by Kato S, Yamasaki N, Ishikawa Y et. al.[21] in that almost all the tasks will be assigned to particular fixed processors except very few tasks that can migrate on any available processor in the system. The priority assignment policy is EDF along with less number of context switches.

Bado B et. al.[25] has presented a semi-partitioning algorithm on n identical processors for in parallel real-time system using EDF . In this approach a series of phases is used to define every periodic task which will be possibly parallelized. The semi-partitioned technique is used with migrations at native deadlines assigned to every phase They considered the phase parallelism which is extension of the common job parallelism. The well-known uni-processor EDF feasibility condition for asynchronous periodic tasks has been taken into consideration for deciding the schedulability of a Multi-Thread task.

Foong A et. al. [27] had done a comprehensive study to show the effect of processor affinity on network performance and they demonstrated that affinitization is having a

substantial effect on protocol processing efficiency and with the different affinitization, the performance bottleneck of the network receive process can be changed significantly.

Dorin F et. al.[24] has published the semi-partitioning hard real-time scheduler with constrained migrations for identical multiprocessor environment which gives feasible substitute between the two extremes global approach and partitioned approach for periodic tasks. The proposed approach use the idea of semi-partitioning scheduling which provides constrained migrations by not allowing jobs to migrate but two distinct jobs of a task are being allotted on various processors.

The Optimal virtual cluster-based multiprocessor scheduling with constrained deadline on multiprocessor platforms has been proposed by Easwaran A, Shin I, Lee I et. al [26] which gives more general approach called cluster-based scheduling to take advantages of both the conventional approaches partitioned and global scheduling. In the proposed algorithm each task instance will be assigned to a specific cluster of CPUs and it may migrate inside that cluster. They have given research direction for cluster scheduling with the goal of improving the CPU_Utilization bounds.

The feasibility analysis for multi-processor recurrent task systems using specified processor affinities has been presented by Baruah SK, Brandenburg BB et. al [22] that raises the question of finding the task system that may be implemented to always meet all deadlines, with affinity mask constraints. They proposed an algorithm which derived the answer of this question effectively that the runtime is in polynomial.

2.3 Findings of the Literature Survey

As per the analysis of the literature review done so far, it is seen that very less or negligible work for multi-processor scheduling is done using processor affinity based approach in soft real-time environment so with this findings in this research the proposed scheduler presented which is affinity based for soft real-time system.

In last five decades, many scheduling algorithms have been proposed but the main focus was uniprocessor scheduling and optimal algorithms exists for uniprocessor realtime system so still research scope for the multi processor scheduling.Multi-processor real-time schedulers are important for the applications of this computing era. The research work is proposed for multiprocessor scheduling.

Each existing scheduler has its own pros and cons, but they are trying to improve schedulability, reduce the deadline miss ratio, reduce the tardiness, reduce the number of context switches but no more focus on CPU_Utilization especially in the real-time environment

so I have tried to analyze all five performance measure parameters in this research.

Most of the algorithms in literature addressed the issue of process selection/priority assignments but very less focused on processor selection direction so in the proposed approach the main aim is to select appropriate processor so it will increase the overall schedulability. In existing multiprocessor schedulers P-FP and G-JLFP, the priority assignment taken into consideration is either fully static or job level fixed priority but it can be test with Job-Level-Dynamic priority assignment to improve the schedulability as well as CPU_Utilization. The traditional approaches like partitioned and global shown in literature are application based but modern RTOS supports the concept of arbitrary processor affinity which realizes any of the approach as per our application and it is more flexible and generalized than the existing one so the proposed scheduler uses the affinity concept in this research to make it generalized scheduler. The existing scheduler provides restricted migration which limits the overall schedulability of the system but this proposed research provides more flexible migration policy along with APA by task shifting using priority reprioritization and dynamic priority assignment which may improve the schedulability, improve the cpu_utilization, reduce the deadline miss ratio and reduce the tardiness.

3. Definition of the Problem

To design and develop a generalized scheduler for multiprocessor soft real-time operating systems using the concept of processor affinity with dynamic priority assignment which can increase the overall schedulabilty, maximize the CPU_Utilization, reduce the Tardiness and reduce the Deadline miss ratio by providing flexible migration policy which can be implemented with task shifting using priority reprioritization mechanism. The research title defined as:

"Critical Scheduling for Soft Real-Time System"

4. Objective and Scope of work

The research in real-time systems for multiprocessor scheduling is the important area which is still unconquered as real-time applications enforces various necessities on scheduler than the general purpose apps. In The RTS (real-time systems), the important requirement is the investigation of scheduling techniques that is used for deciding whether the real-time application's execution time can be bounded to meet the time constraints(deadline). The key objectives of this research work is that to design the generalized multiprocessor scheduler

using the concept of processor affinity and to provide flexible migration policy with priority reprioritization which optimize the different parameters of the multiprocessor scheduling algorithm for soft real time system and they are defined in the following section.

4.1 Scope of work for the Proposed Scheduler PrAMS

- Reduce the Deadline Miss Ratio
- Reduce the Tardiness
- Analysis of Switching Overheads(Number of Context switches)
- Improve the Schedulability
- Improve the Processor Utilization

5. Original contribution by the thesis

In this research, three different types of multiprocessor soft real-time schedulers are implemented using Litmus^{RT}. The details of each Multi-processor scheduler are given below:

5.1. Implementation of P-FP-APA scheduler with Processor Affinity and No migration at all

5.2. Implementation of G-JLFP-APA scheduler with Processor Affinity and Full migration

5.3. Designed and Implemented Multiprocessor soft real-time scheduler PrAMS along with the three approaches

i. Processor Selection Policy with Affinity Concept

ii. Process Selection(priority assignment) Policy with JLDP(Job-Level-Dynamic-Priority)

iii. Flexible Task Migration Policy with priority reprioritization mechanism (solve the priority inversion problem)

6. Methodology of Research, Results / Comparisons

In this section, a new processor affinity based scheduler is proposed which has been designed and implemented for multiprocessor soft real-time system. It has been named as **PrAMS** (Processor Affinity based Multiprocessor Scheduler).This scheduler has been designed to improve overall Schedulability, maximize CPU Utilization, minimize the Deadline miss ratio and Tardiness.

6.1. Arbitrary Processor Affinity

The processor affinity means an arbitrary set of processors on which we can execute a

process. Figure 6.1 illustrates the processor affinity of a task T_i which is $\{P_8, P_9, P_{11}, P_{12}\}$; it means that the task T_i can be executed on any one of the processor which is free amongst its affinity set but not on any other processor out of its affinity.

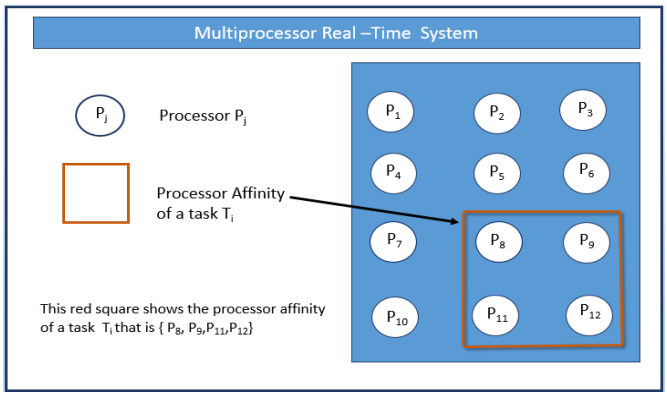

Figure 6.1: Illustration of Arbitrary Processor Affinity

6.1.1. Processor Affinity in Real-Time Task Scheduling

The contemporary real-time operating systems like QNX, RTEMS, Lynux and real-time extensions of Linux use the concept of processor affinity instead of traditional approaches described in literature to implement its scheduler which provides more generalized way of task migration in multiprocessor systems. Many such OS has applied the processor affinity mechanism for restricted migration on free processors. The processor affinity ($\boldsymbol{\alpha_k}$; where$\boldsymbol{\alpha_k \subseteq \pi}$ **)** of any task presents the set of processors on which the job of that task can execute. In this research it is assumed that the processor affinity will not be changed during task's whole life time. The conventional migration approaches described in literature can be presented with processor affinity. Any priority selection algorithm can be used with processor affinity for processor selection. We can understand the concept of processor affinity to select processor for the task by taking an example of Linux Push-Pull Scheduler in multiprocessor system. To accomplish scheduling quickness in priority based scheduling for real-time task the runqueue is divided into many number of linked list per priority one list. At the time of new task arrival, the incoming task is to be inserted into any one processor's runqueue based on its processor affinity and priority number [38][44]. When task gets completed, the task will be removed from the queue and runqueue will be restructured. If necessary, the task can be shifted from one CPU runqueue to another CPU runqueue with following the strict

affinity constraints of a task. Updating the runqueue of any processor generates various scheduling cases. When any lower priority task awakens on a runqueue that is running higher priority task or the lower priority task would be preempted by higher priority task then the scheduler push the priority task to another runqueue. The runqueue on which the task is to be pushed should have lower priority tasks. Whenever any currently running higher priority task completes its execution and lower priority task is going to be scheduled, the scheduler examines every processor's runqueue for pulling highest priority task from the other runqueue if any available. To push task on different processor's runqueue or to pull task from other processor's runqueue must not violate the affinity constraints [18]. The jobs are said to be suitable for pushing on the other queue if it is not scheduled currently. On the other hand the tasks that has been already scheduled cannot be pulled from any runqueue. The task in execution can be removed only when the higher priority task arrives. The running task will release its processor either voluntarily when execution gets complete or its time quantum is over in time sharing system. The load balancing requirements for runqueue is also checked periodically by Linux scheduler so maximum CPU Utilization can be achieved. However, Linux scheduler can't be flexible to move higher priority task on another processor for lower priority task which could not be executed elsewhere because of its processor affinity constraint; so restricted migration mechanism fails to attain higher schedulability. In this research this scenario explained in detail and presents novel processor affinity based real-time scheduling algorithm with flexible migration policy to overcome these limitations and improve the overall schedulability of the real-time system.

6.1.2. Illustration of Conventional and Proposed Migration Approach:

The degree of migration allowed to a task is the key parameter for classifying the multiprocessor real-time scheduler. The two excessive ends of this spectrum is the global scheduling approach and the portioned approach. In global scheduling, there will be a single common queue between all processors and ready to run tasks will be assigned dynamically to any available processor according to its priority while in partitioned scheduling, every task has assigned fixed processor on which it will be executed without any migrations. The hybrid approaches are also designed by the researchers but one of the prominent hybrid approach is clustered scheduling, in this approach the system

processors are classified into the separate clusters and every task will be assigned to a single cluster and "global" approach will be applied inside the cluster.

The recent RTOS like VxWorks, LynxOS, QNX, and the real-time extended version of the Linux are using the processor affinity concept which is providing more flexible migration and generalized algorithm instead of implementing the traditional methods described in the literature. Figure 6.2 illustrates the different migration strategies like global scheduling is with full Migration policy, partitioned scheduling is with no migration, clustered scheduling is with partial migration and the proposed approach with flexible migration for which schedulability analysis has been done in this thesis.

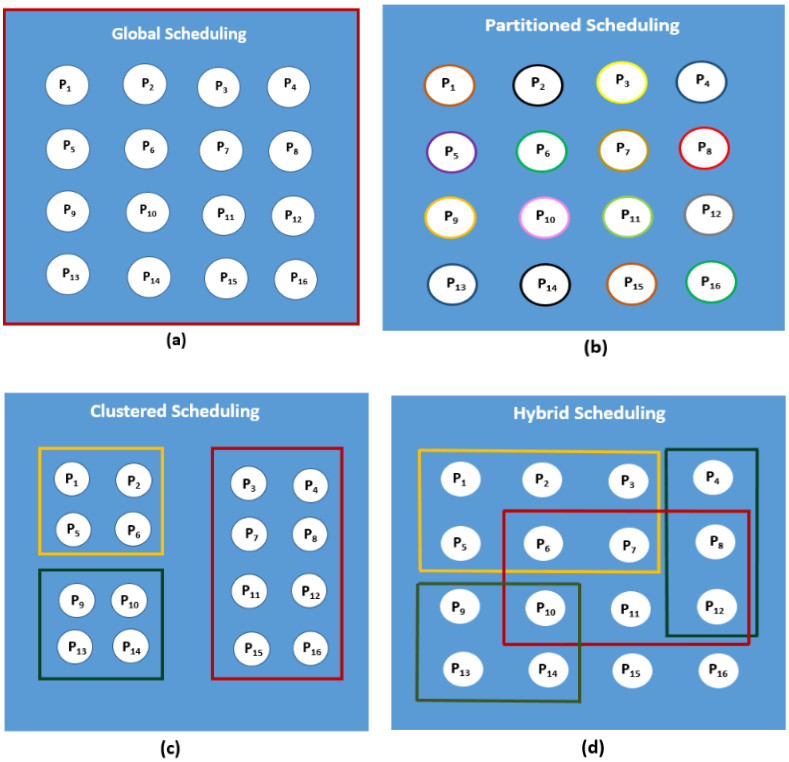

Figure 6.2: Illustration of different migration strategies: a) Full Migration policy with Global scheduling **b)** No migration with Partitioned Scheduling **c)** Partial migration with Clustered Scheduling **d)** Proposed Approach for which the schedulability analysis is done in this research

6.2. The Task Migration Scenario for Real-Time Tasks in Existing system and Proposed Solution

The uniprocessor system facing the problem of priority inversion which is the case when the higher priority tasks have to wait for the lower priority tasks for starting its execution. The same situation may occurs in multiprocessor system when the low priority job T_k holds a

processor, that is not in the affinity set α_j of task T_j, but another higher priority task T_i having common processor affinity set α_i with α_j and α_k (i.e. , also T_k holds processor that is common with T_i, where $p_i > p_j > p_k$). Let us understand the scenarios with different migration opportunities in figure- 1.A real time system with three processors and four tasks is having priority order $P_1 > P_2 > P_3 > P_4$ and processor affinity sets $\alpha_1 = \{\pi_1, \pi_2, \pi_3\}$, $\alpha_2 = \{\pi_2, \pi_3\}$, $\alpha_3 = \{\pi_1\}$, $\alpha_4 = \{\pi_2, \pi_3\}$.As shown in figure 6.3 (a) at an instance of time where task P_3 is waiting and tasks P_1, P_2, P_4 are running. Now P_3 becomes ready to execute but scheduler will not allow run it on π_1 as it is running P_1 which has higher priority than the P_3.On the other hand the task P_4 is running on processor π_3 which is in the affinity set of task P_1. There may be the solution to preempt task P_4 as the priority order is P_3>P_4 and we can shift P_1 the higher priority task to processor π_3 to schedule task P_3 on π_1.One likely solution will be the pulling P_4 from the π_3 and pushing the P_1 (P_1>P_3) from processor π_1 to processor π_3. This push-pull operation will make possible the execution of P_3 on π_1 and it will increase the overall schedulability of the system which is shown in figure 6.3 (b) and figure 6.3 (c); P_1 pushed to processor π_2 and P_2 pushed to processor π_3 hence the solution needs two preemption which causes two context switches. In figure 6.3 (b) needs only one context switch so it is cost effective proposed solution implemented using priority reprioritization.

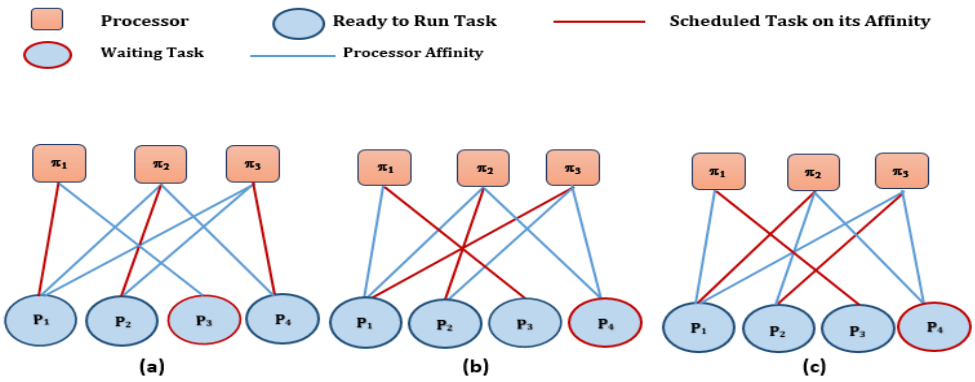

Figure 6.3: Problem in Existing approach and proposed solution

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 π_1 after shifting task P_1 on processor π_3. (c) Probable state where task P_3 is ready and scheduled on processor π_1 after shifting task P_2 to processor π_3 and shifting task P1 to processor π_2

The above case illustrates that due to the restrictive migration mechanism of existing processor affinity based scheduler they fails to get higher schedulability by not preempting the

higher priority running task and shifting it on another processor which is running lower priority task. Schedulability can be improved by pulling lower priority task and migrating higher priority task to that processor using proposed approach with flexible migration policy.

6.3. Methodology of proposed APA based scheduler with priority reassignment

In the proposed approach as shown in figure 6.4, 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_j) is under the affinity of running higher priority task (T_i) so the ready to run task (T_k) with higher priority than T_j will be blocked that is called priority inversion problem.

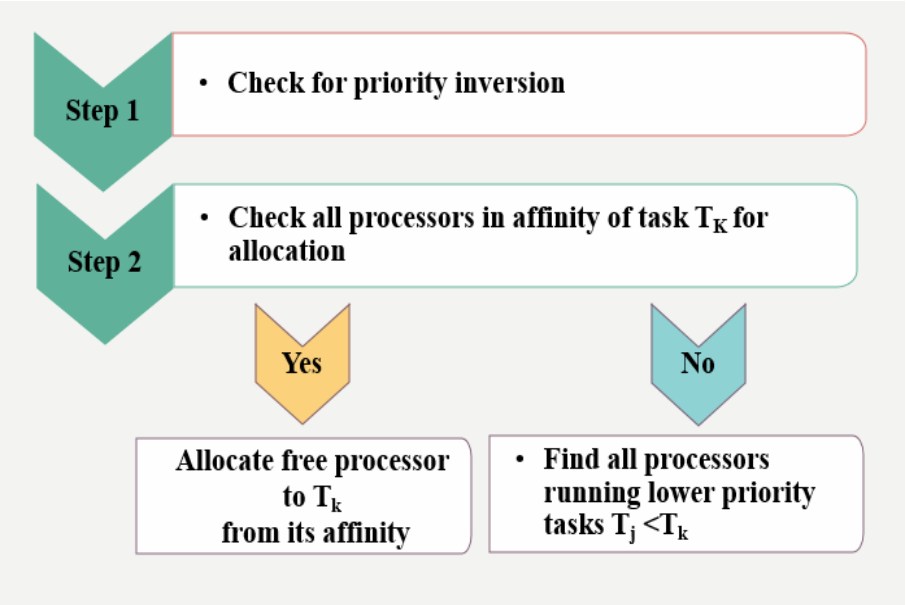

Figure 6.4: Approach-1

(Check for Priority Inversion)

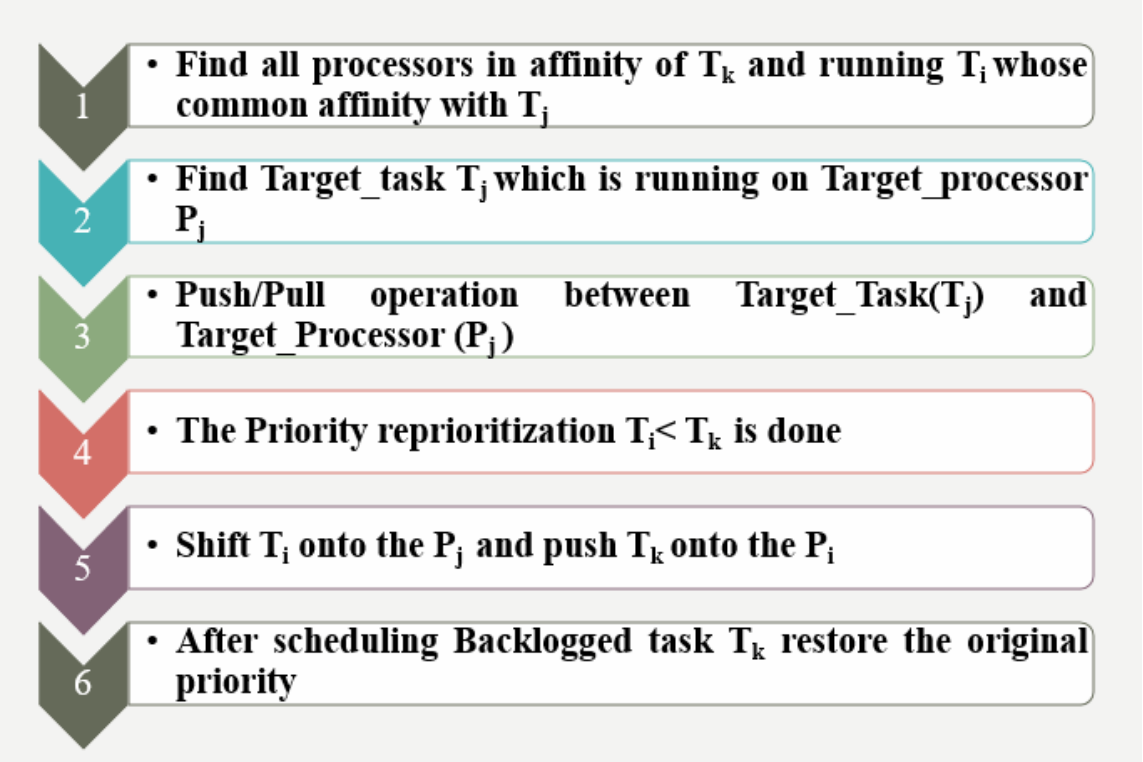

Figure 6.5: Approach-2

(Task Priority Reassignment)

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 6.5 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 $\{\pi_i, \pi_j, \pi_k\}$ and tasks on these processors are $\{p_i, p_j, p_k\}$ 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 π_j it only preempts p_j.

6.4. Results & Comparison

In the following section the experimental setup, task set generation criteria and results are described in detail with comparison and analysis along with five performance measure parameters.

6.4.1. Experimental Setup

This research presents a generalized processor affinity based scheduler which gives realization of similar like partitioned, global tasks in a taskset and then assigns fixed 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 within deadline. The proposed Scheduler PrAMS has been compared with existing Partitioned and Global schedulers implemented using affinity. In this research, experiment and result analysis is done for five parameters that is deadline miss ratio, tardiness, number of context switches, schedulability analysis and CPU_Utilization for multiprocessor real-time system. Table 6.1 presents task set generation criteria and the value taken into consideration. The Linux Testbed Litmus-RT is used for implementation and experimental work [16][42][46]; The research presents the result analysis for various cases in which proposed approach compared with existing methods for various test cases (fixed number of processors, fixed taskset size, both variable) for 100 tasksets with either fixed(n=4,8,16,32,64) taskset size or random(m+1 to 4m) and No. of processors(m=2,4,8,12,16). It is not possible to accommodate all the results here so in this section I have kept few sample results for two test cases which is having fixed taskset size with m=2,4 processors and random taskset size(m+1, 4m) with more number of processors for all five parameters.

Table 6.1: Task Set Generation Criteria and the value taken into consideration

Criteria	Value
Priority Assignment	JLDP
Task set Count	100
Running time	60 seconds
Period Range	10ms to 100ms
Task set Generation Algorithm	Random Vectors with fixed Sum algorithm
No of processors	2,4, 8, 12,16
No of Tasks in a task set range	(4,8,16,32,64) OR (m+1, 4m)
Task Utilization	For all Taskset more than 75% of m
Affinity Assignment	Fixed Affinity Assignment
Relative Deadline	$d_i < p_i$

There are five parameters considered for the results.

Parameter 1: Average Deadline Miss Ratio should be minimum.

Parameter 2: Average Tardiness (ns) should be as low as possible.

Parameter 3: Average No. of Context Switches should be minimum.

Parameter 4: Average Schedulability (%) should be as high as.

Parameter 5: Average CPU_Utilization (%) should be maximum.

Here, following three different schedulers have been compared.

1. Partitioned Approach with affinity implementation
2. Global Approach with affinity implementation
3. Proposed Approach(along with affinity and novel migration policy)

All the tests have been taken on Litmus^{RT} for soft Real-Time Systems. The results mentioned here is the average of 100 Tasksets. One Taskset can contains any no. of tasks either fix or random and many jobs of each task can appear within the given duration as per the given period. Task generation detail is given in Table 6.1.

6.4.2. Results of All Schedulers in All Test cases for implementing processor affinity based scheduler without flexible migration (PrASWM) with the average of 100 task sets

In this case we have taken results for 100 taskset which has been tested on no. of processors (m=2, 4) and the taskset size (n=4, 8, 16, 32, 64).The taskset size is kept fixed which is 2^i (where the $1< i < 7$, m is no. of processors and n is taskset size).

There are two different test categories have been selected.

1. The test has been taken with two Processor and taskset size (n=4, 8, 16, 32, 64).
2. The test has been taken with four Processor and taskset size (n=4, 8, 16, 32, 64).

As shown in the Table-6.2, proposed approach performs better concerning all other schedulers (P-FP-APA, G-JLFP-APA) for all the parameters like Schedulability, Deadline Miss Ratio, Tardiness and CPU_Utilization as in proposed approach the JLDP is combined along with processor affinity. The context switch analysis is also shown for all the schedulers.

Table 6.2: Results of All Schedulers in All Test cases with the average of 100 task sets with fixed task set size (n=4,8,16,32,64) and number of processors(m=2,4)

		Case 1 (m=2)					Case 2 (m=4)				
Parameter	**Approach**	**n=4**	**n=8**	**n=16**	**n=32**	**n=64**	**n=4**	**n=8**	**n=16**	**n=32**	**n=64**
Average Deadline Miss Ratio	**P-FP-APA**	0.0068	0.072	0.3904	0.5902	0.8041	0.0047	0.0509	0.2632	0.4065	0.6504
	G-JLFP-APA	0.0045	0.0515	0.2165	0.4366	0.5954	0.0032	0.0265	0.189	0.3248	0.4068
	PrASWM	0.0018	0.0351	0.0146	0.2165	0.4437	0.0009	0.037	0.0608	0.1822	0.2389
Average Tardiness	**P-FP-APA**	25.65	29.56	34.98	42.09	48.61	22.47	26.04	31.78	39.85	44.56
	G-JLFP-APA	21.63	26.7	30.66	34.29	39.52	19.51	24.22	28.48	31.63	37.25
	PrASWM	10.85	17.2	19.67	26.54	31.55	8.58	14.53	17.07	22.63	28.45
Average No. of Context Switch	**P-FP-APA**	67.25	98.38	123.31	147.34	177.58	99.25	112.13	124.5	164.19	180.66
	G-JLFP-APA	99.25	102.13	145.81	183.25	211.45	126.25	138.25	163.56	194.22	269.02
	PrASWM	92.75	95.25	168.81	205.25	195.52	119.5	153.13	156.38	213.25	247.02
Average Schedulability	**P-FP-APA**	71.5	67.13	64.88	62.09	57.48	73.5	68.75	67.19	64.03	60.33
	G-JLFP-APA	81.5	78.13	76.56	73.16	64.47	84.25	80.88	78.06	74.06	67.89
	PrASWM	84	82.5	79.63	76.06	72.02	86.25	83.13	80.31	78.13	74.05
Average CPU Utilization	**P-FP-APA**	51.46	54.25	57.33	61.4	65.47	48.25	51.78	54.34	57.54	61.33
	G-JLFP-APA	53.96	59.53	63.36	64.32	71.45	50.22	56.21	58.17	61.98	65.67
	PrASWM	55.29	63.11	64.28	68.71	73.77	53.43	59.59	61.92	64.57	70.64

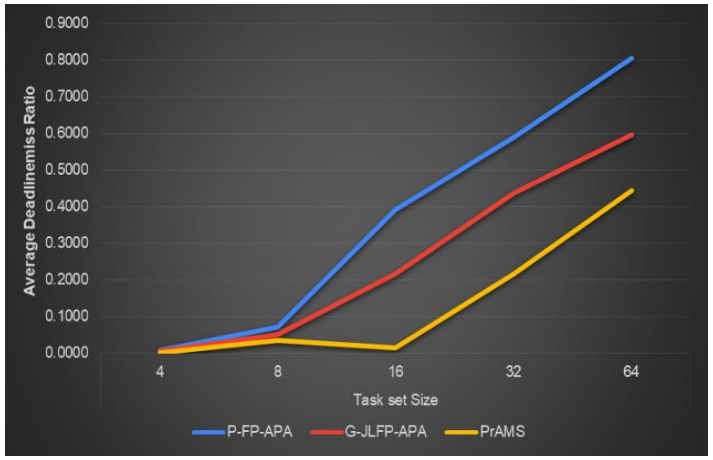

Figure 6.6: Average Deadline Miss Ratio on Two Processors

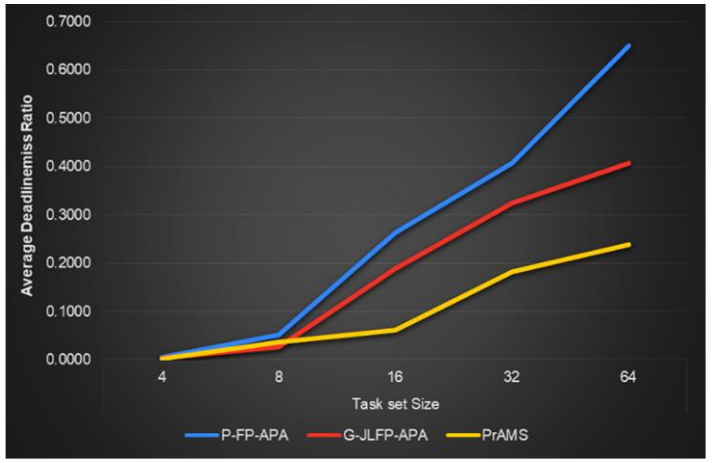

Figure 6.7: Average Deadline Miss Ratio on Four Processors

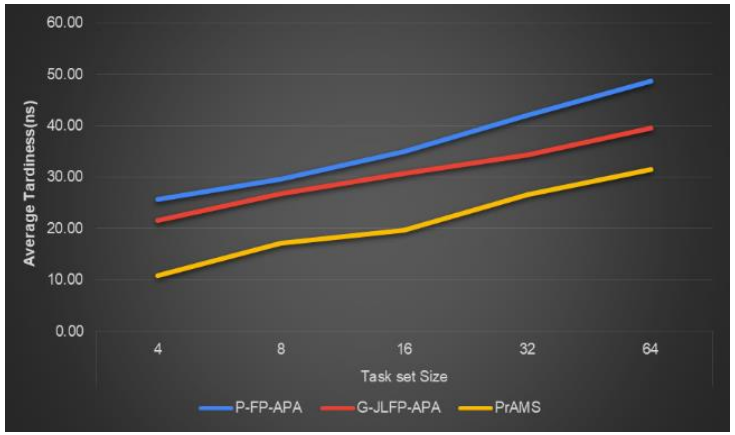

Figure 6.8: Average Tardiness on Two Processors

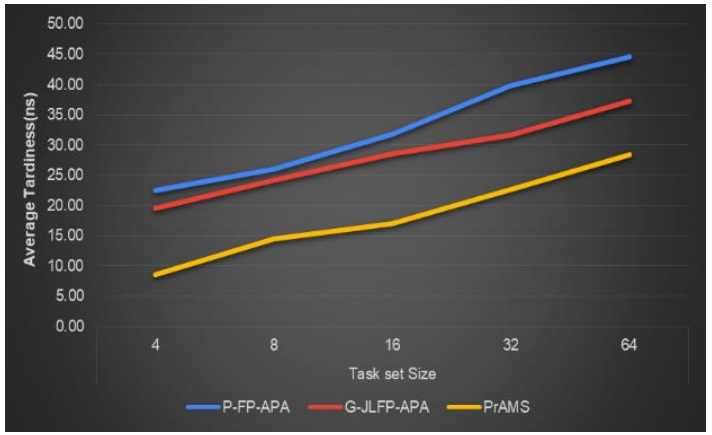

Figure 6.9: Average Tardiness on Four Processors

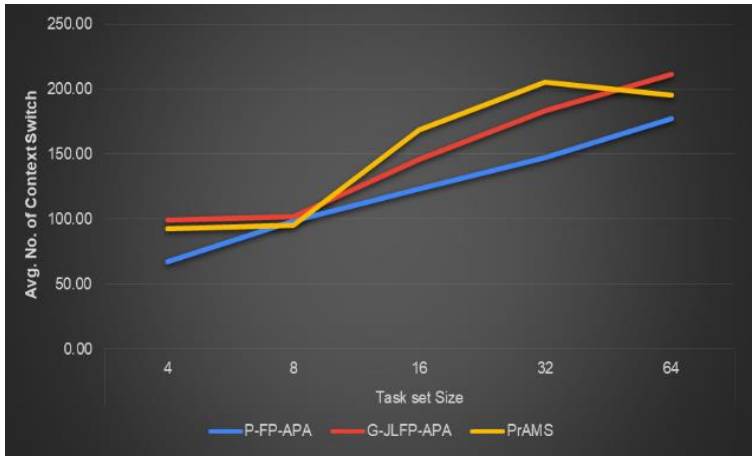

Figure 6.10: Average No. of Context Switch on Two Processors

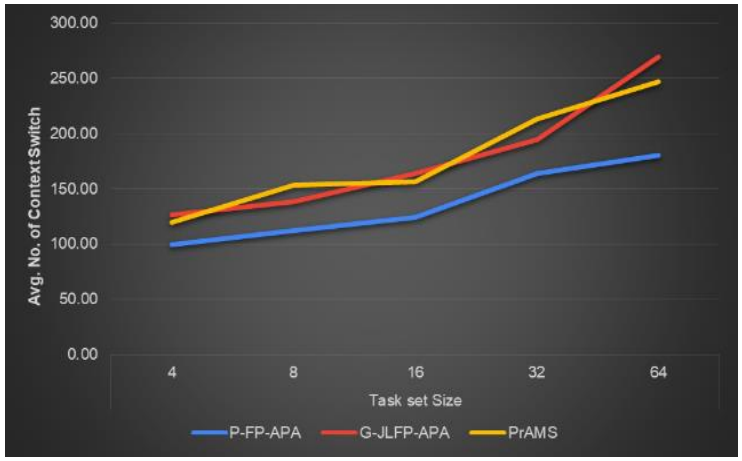

Figure 6.11: Average No. of Context Switch on Four Processors

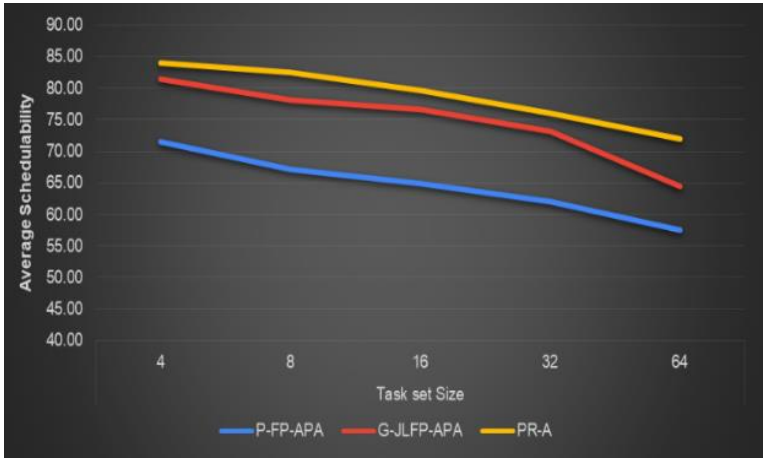

Figure 6.12: Average Schedulability on Two Processors

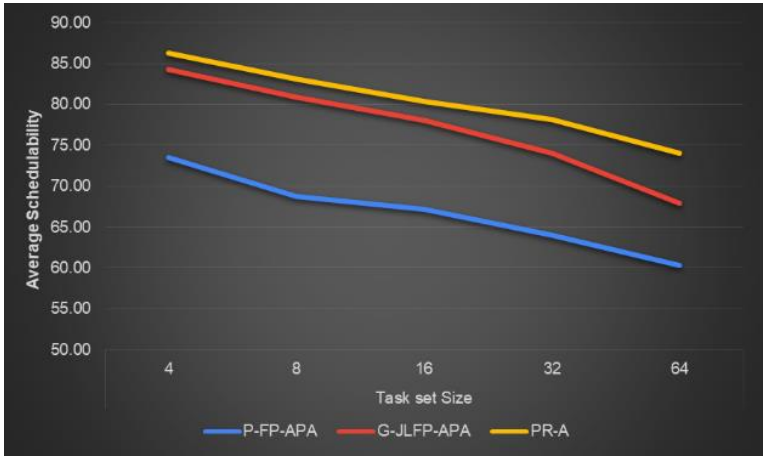

Figure 6.13: Average Schedulability on Four Processors

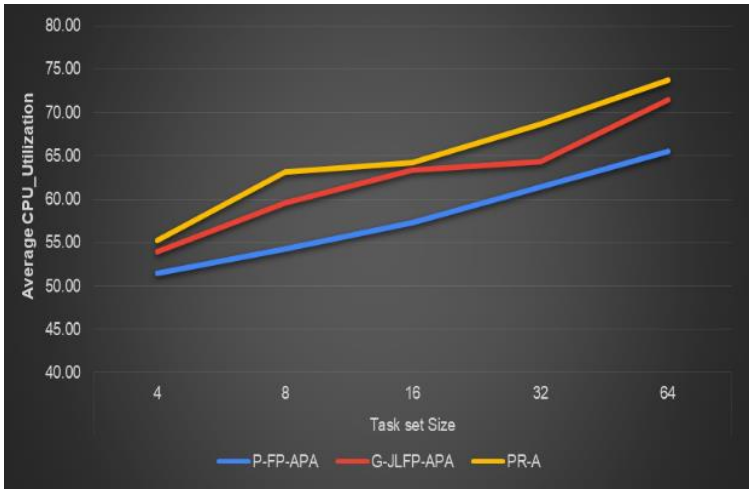

Figure 6.14: Average CPU_Utilization on Two Processors

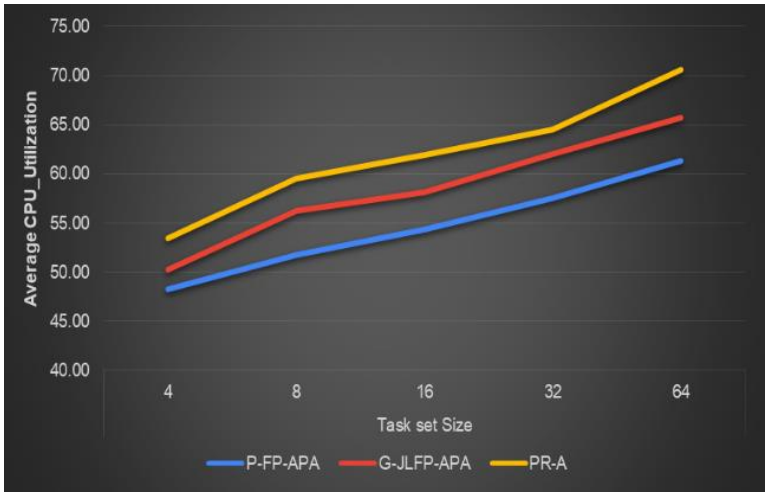

Figure 6.15: Average CPU_Utilization on Four Processors

6.4.3. Results of All Schedulers in All Test cases for implementing processor affinity based multiprocessor scheduler with flexible migration (PrAMS) with the average of 100 Task sets

In this case the variable no. of processors (m) and the variable taskset size (n) has been taken into consideration. The taskset size is kept random with the range from m+1 to 4m (where the m is no. of processors and n is taskset size).

The test has been taken with variable No. of Processors (m=8, 12, 16) and variable taskset size (n=10, 12, 14, 16, 20, 24, 28, 32, 48, 52, 56, 60).

As shown in the Table-6.3, proposed approach performs better concerning all other schedulers (P-FP-APA, G-JLFP-APA) in all cases. In few cases the average no. of context switches are more than global due to flexible migration policy but increases overall schedulability, maximize the CPU_Utilization, decreases deadline miss ratio and tardiness.

Table 6.3: Results of All Schedulers in All Test cases with the average of 100 task sets with random task set size (n= m+1 to 4m)) and more number of processors(m=8,12,16)

		Case 1 (m=8)				Case 2 (m=12)				Case 3 (m=16)			
Parameter	**Approach**	**n=10**	**n=12**	**n=14**	**n=16**	**n=20**	**n=24**	**n=28**	**n=32**	**n=48**	**n=52**	**n=56**	**n=60**
Average Deadline Miss Ratio	**P-FP-APA**	0.0945	0.1449	0.2048	0.2661	0.2863	0.4011	0.5342	0.6739	0.6143	0.6835	0.7172	0.7339
	G-JLFP-APA	0.0696	0.1229	0.1841	0.2538	0.2444	0.3745	0.4856	0.6033	0.545	0.6418	0.6755	0.6919
	PrAMS	0.0226	0.0847	0.1447	0.216	0.1476	0.2876	0.3823	0.4866	0.4233	0.5422	0.6243	0.6458
Average Tardiness	**P-FP-APA**	19.51	21.49	23.53	24.45	22.42	26.1	27.52	29.56	31.7	34.35	34.78	35.71
	G-JLFP-APA	11.36	15.38	19.72	20.07	20.43	21.32	21.86	24.56	27.46	29.54	30.33	32.15
	PrAMS	8.51	10.33	13.5	14.58	14.64	15.49	17.67	19.38	23.47	25.37	28.61	31.53
Average No. of Context Switch	**P-FP-APA**	59.48	71.45	83.54	87.6	112.32	139.74	154.52	172.58	263.79	272.91	297.62	321.66
	G-JLFP-APA	67.56	83.53	99.23	107.73	178.62	189.29	199.69	219.77	368.23	382.59	397.92	412.43
	PrAMS	71.24	81.52	91.6	99.38	156.35	167.25	180.54	203.31	364.58	389.2	407.11	414.6
Average Schedulability	**P-FP-APA**	82.02	79.81	77.6	76.575	82.44	78.5	78.45	77.66	76.676	75.416	75.232	74.5
	G-JLFP-APA	86.34	83.855	81.37	80.025	87.72	81.55	80.64	79.62	81.281	80.449	79.42	79.302
	PrAMS	89.622	87.382	84.685	84.446	89.51	86.37	84.45	82.4	84.658	83.444	82.446	81.49
Average CPU Utilization	**P-FP-APA**	59.28	59.44	60.35	61.38	59.57	62.41	65.45	67.77	62.45	64.43	65.59	66.64
	G-JLFP-APA	64.37	64.5	65.43	67.34	63.57	65.56	66.61	68.5	67.56	68.53	69.32	71.61
	PrAMS	66.32	67.39	68.32	70.39	65.54	69.49	70.42	71.87	68.59	70.5	71.68	72.45

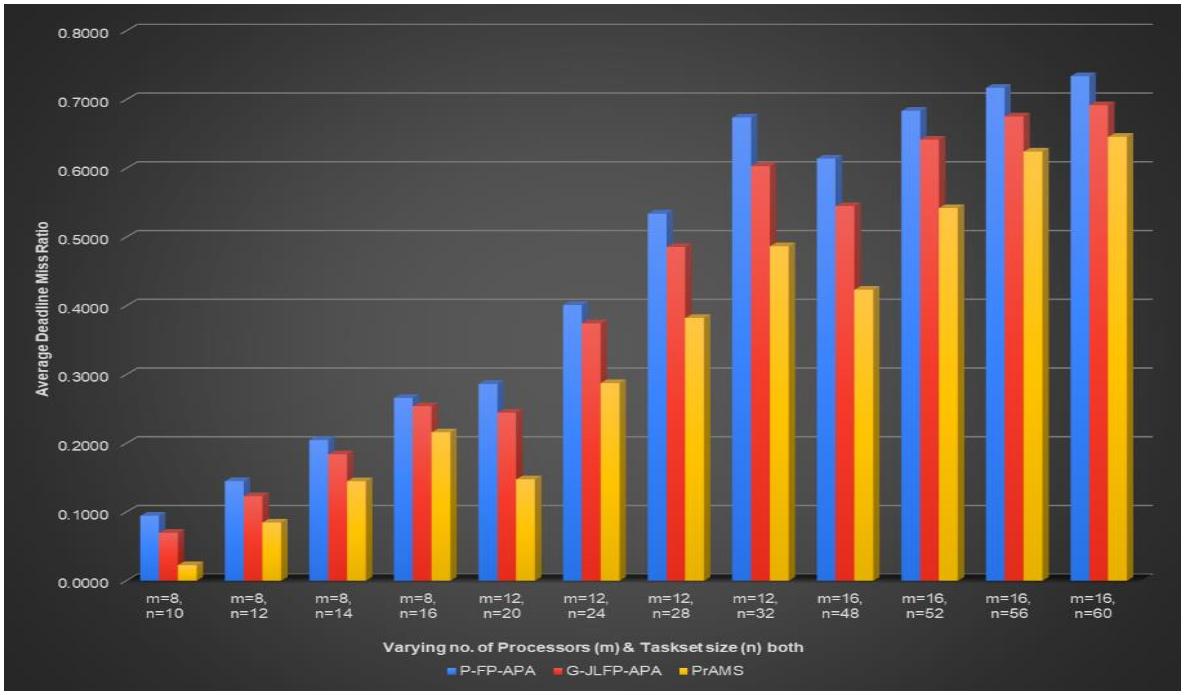

Figure 6.16: Average Deadline Miss Ratio with variable No. of Processors and Taskset size

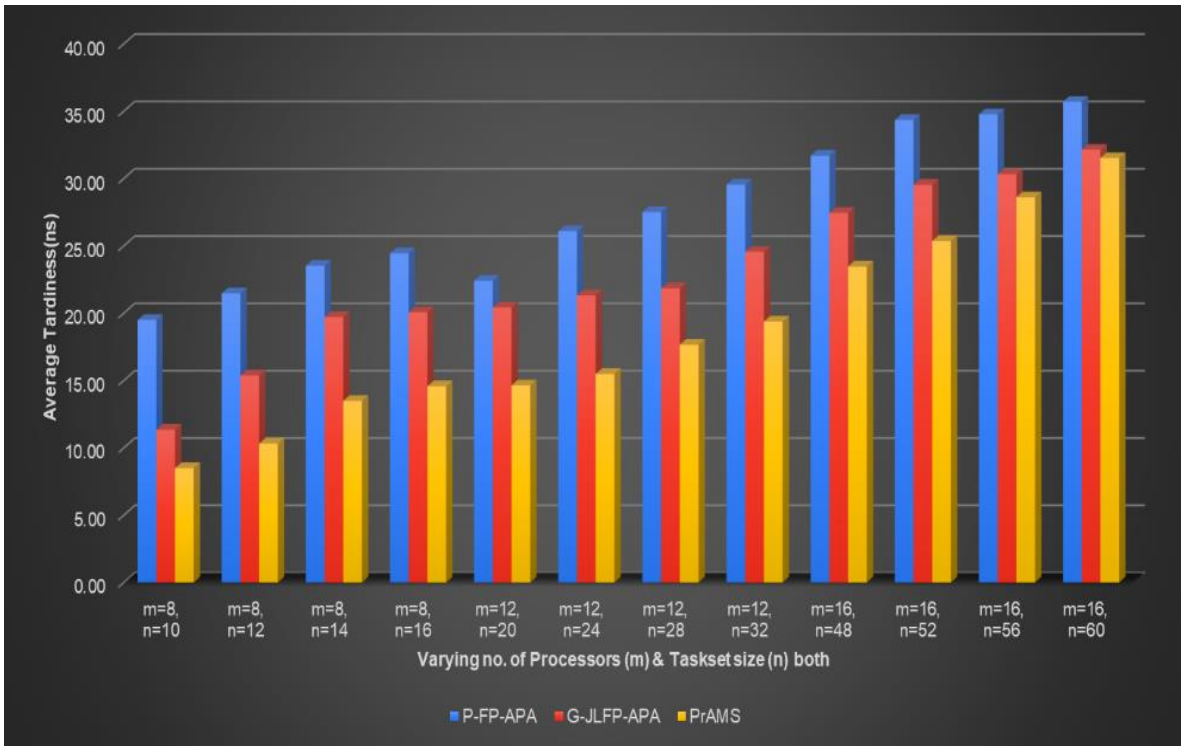

Figure 6.17: Average Tardiness with variable No. of Processors and Taskset size

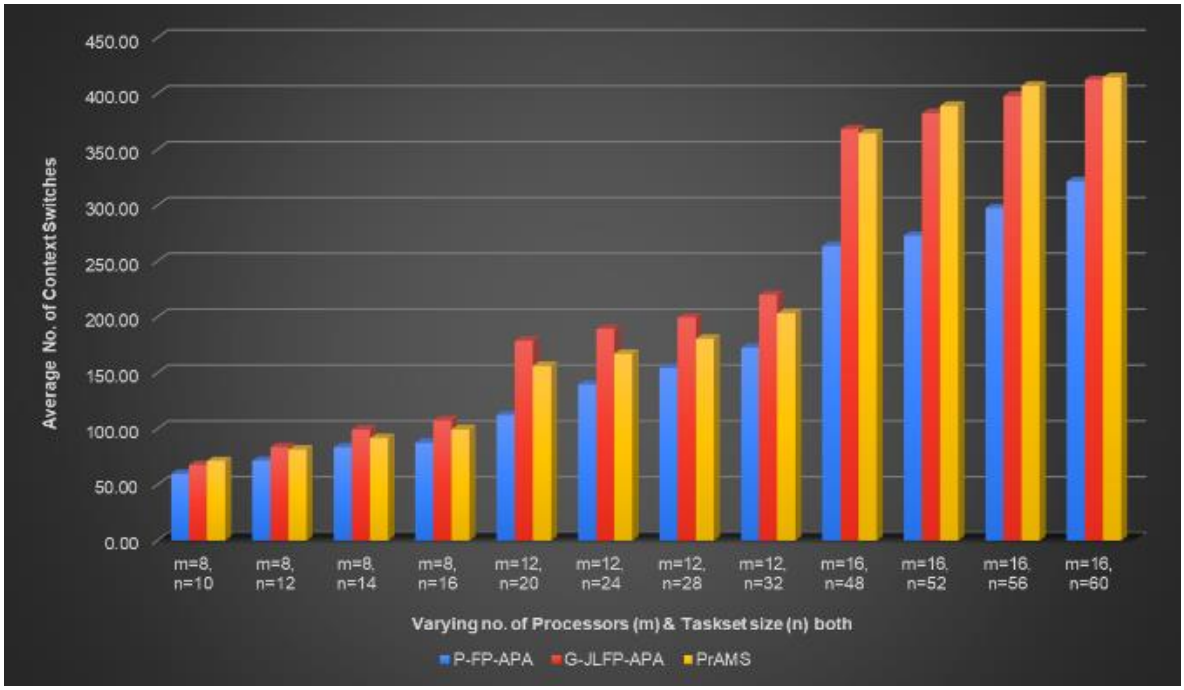

Figure 6.18: Average No. of Context Switches with variable No. of Processors and Taskset size

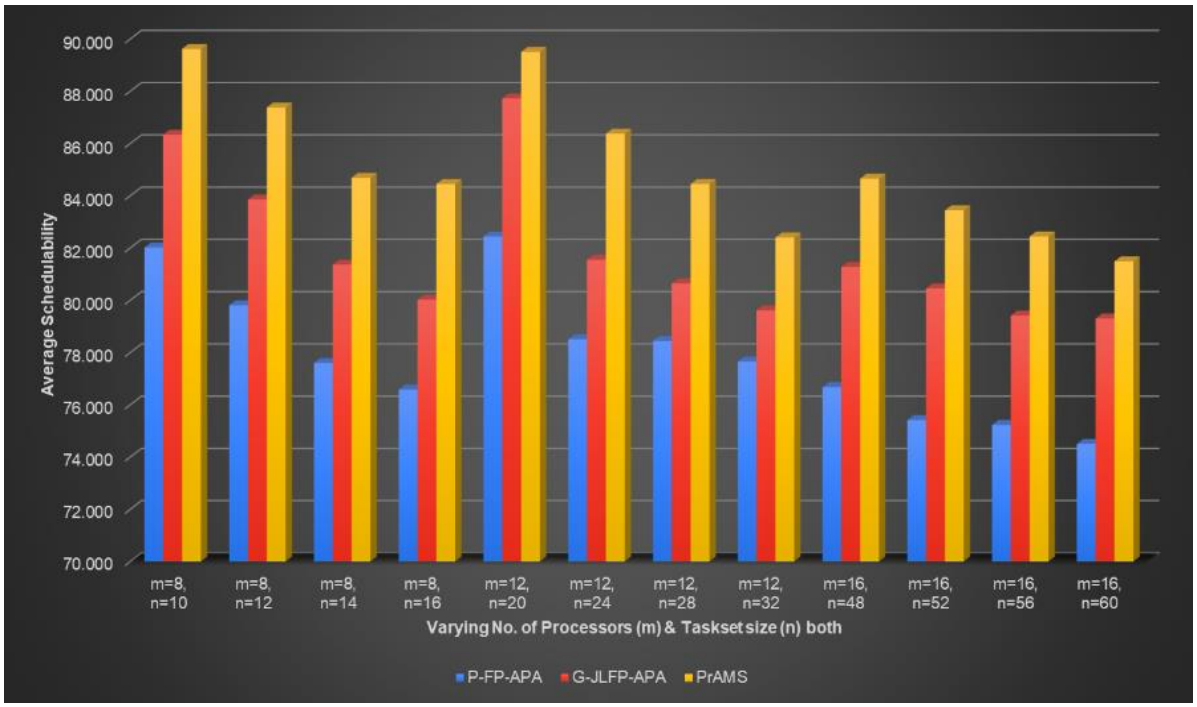

Figure 6.19: Average Schedulability with variable No. of Processors and Taskset size

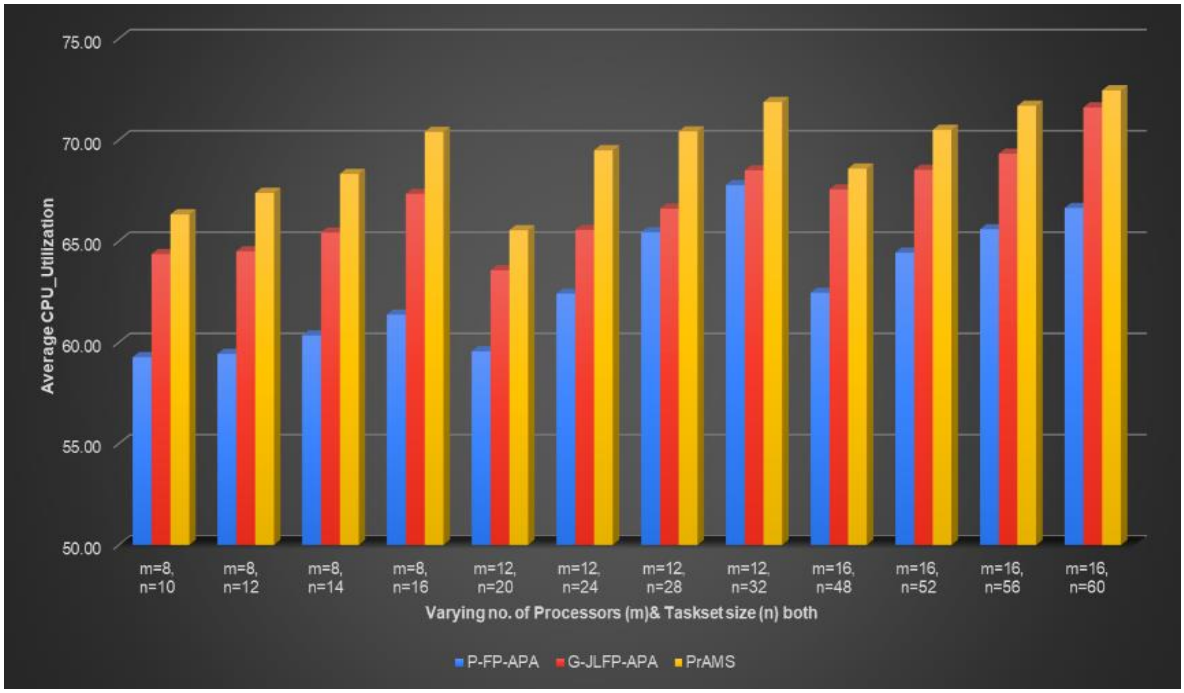

Figure 6.20 : Average CPU_Utilization with variable No. of Processors and Taskset size

7. Achievements with respect to objectives

Main objective of the proposed work is to implement efficient affinity based generalized scheduler for multiprocessor soft real-time system which should be able to increase schedulabilty and CPU_Utilization and reduce the deadline miss ratio and tardiness.

The proposed scheduler PrAMS along with three different approaches like processor affinity for processor selection, JLDP for process selection and flexible migration policy with priority reprioritization gives better performance for all four parameters (Schedulability, CPU_Utilization, Tardiness and Deadline miss ratio) in all cases but increases the number of context switches in few cases.

8. Conclusion

The generalized multi-processor real-time scheduler presented in this research primarily focuses on three different problems. First is the processor selection problem in a multiprocessor system which is based on affinity concept and processor affinity is kept static in this experiment. The second problem addressed is the process selection that is priority assignment for task execution order which is done with Job-Level-Dynamic-Priority (JLDP) approach. The third is to provide flexible migration policy with the task shifting by reprioritizing the task.

One of the key contributions of research work is it provides generalized affinity based scheduling algorithm for multiprocessor soft real-time system. Another key contribution is that it addresses the priority inversion problem perfectly which is the big problem in any real-time system. The proposed processor affinity based scheduler improves overall schedulability of the system by providing the flexible migration policy. The proposed scheduler provides consistent and optimum results for different performance measure parameters like less deadline miss ratio, less tardiness, Maximum CPU_Utilization and improved schedulability as compared to traditional approaches discussed in the literature.

- Scheduling tasks with proposed approach of task shifting improves schedulability and CPU_Utilization as compared to existing approaches.
- Scheduling tasks with proposed approach minimize the Deadline miss ratio as well as Tardiness as compared to traditional approaches.
- Proposed approach gives always more no. of context switches than partitioned approach but when we compare it with global then in some cases it has less no. of context switches than the Global approach and in some cases it is more than global approach.

The common observations based on result analysis are as following:

Parameter	**Observations for all Schedulers**
Deadline Miss Ratio	i. The deadline miss ratio increases when n-m increases as the number of tasks are higher than processors which causes misses more frequently.
	ii. The deadline miss ratio decreased if the number of processors are increased with the same taskset size that is performs better for n is lower and m is constant.
	iii. If m increases and n is kept constant then PrAMS' performance is getting improved (ratio gets lower) as we have more choices to migrate our tasks.
Tardiness	i. The Tardiness increases when n-m increases (more misses as tasks are higher than processors and if more no. of misses than time elapsed after deadline is more)
	ii. The Tardiness is decreasing if number of processors are increased with fixed taskset size (constant n) as we have more

		choices to migrate our tasks and due to that less deadline miss appears in the system which reduces the tardiness.
No. of Context Switch	i.	The number of context switch increases in both the cases either n or m any value increases as if n increases according to priority requirement it need to shift that's why increases and if m increases it gets more chance to migrate so it should increases
	ii.	PrAMS has always more no. of context switches than Partitioned approach and in some cases (like more no. of processors) it also has more than global approach or almost same as global (for less no. of processors).
	iii.	In any case partitioned approach has less no. of context switches than global approach and proposed approach.
Schedulability	i.	Schedulability decreases when n-m increases (more misses as tasks are higher than processors and if more no. of misses than lower the schedulability)
	ii.	The PrAMS performs better in all three cases; a) No. of processors fixed and variable taskset size b) No. of processors variable and constant taskset size c) Both variable.
	iii.	The schedulability is increasing when m increases and n is constant as we have more choice to migrate our tasks and less misses.
CPU_Utilization	i.	CPU_Utilization increases when n-m increases as more tasks will be executed on same no. of processors.
	ii.	CPU_Utilization decreases when m increases for constant n as the same number of tasks are being executed on more no. of processors.
	iii.	In all the cases global and PrAMS are having more CPU_Utilization than partitioned approach as in partitioned approach the task must be executed on predefined processor even though other processors free in our system.

	iv. The PrAMS performs better than global and partitioned approach in all the cases as it is possible to schedule more number of tasks for execution due to its flexible migration policy.

9. Publications

[1]. Ms. Jayna Donga, Dr. M. S. Holia, (October, 2018). “The Comparative Analysis of Scheduling Approaches in Real-Time Systems” published in UGC approved International Journal of Technical Innovation in Modern Engineering & Science (IJTIMES), Impact Factor: 5.22 (SJIF-2017), [ISSN: e-ISSN: 2455-2585]-(UGC Approved).

[2]. Ms. Jayna Donga, Dr. M. S. Holia, (December, 2019). “An Analysis of scheduling algorithms in Real-Time operating system” published in Lecture Notes in Networks and Systems 98 (LNNS) with Inventive Computation Technologies (ISSN: 2367-3370), Springer Series. (pp. 374-381). (SCOPUS, INSPEC, WTI Frankfurt eG, zbMATH, SCImago)

[3]. Ms. Jayna Donga, Dr. M. S. Holia, Dr. N. M. Patel (April, 2021). "The Classification and Comparative study of Real-Time Task Scheduling Algorithms based on various parameters" published in Algorithms for Intelligent Systems (AIS) (ISSN: 2524-7565), Springer Series. (pp.239-246).

[4]. Ms. Jayna Donga, Dr. M. S. Holia (December, 2020). "Processor Affinity based Multiprocessor Scheduling algorithm for Soft Real-Time System” published in Solid State Technology (ISSN: 0038-111X), volume-63, Issue-5. (Google Scholar, SCOPUS, Ei Compendex)

[5]. Ms. Jayna Donga, Dr. M. S. Holia (December, 2020). “A Hybrid Multiprocessor Scheduler for Soft Real-Time System with Processor Affinity Concept” paper presented in International Conference on Applied Mathematics, Modeling and Simulation in Engineering 2021 (AMSE-2021) during September 15-16,2021 at STANLEY College of Engineering, Hyderabad, India and it is going to be published by IOP SCIENCE (JPCS) [doi:10.1088/I Online ISSN: 1742-6596 /Print ISSN: 1742-6588], Volume 2089-(Scopus indexed).

10. Patents

[1]. Jayna Donga, Dr. Mehfuza S. Holia, Dr. Vatsal H. Shah. " PrAMS for Scheduling in Real-Time Operating System using LITMUSRT " Indian Patent, Application No: 202121019254 A, Published date: : 02 July 2021.

11. References

[1]. G. C. Buttazzo. Hard real-time computing systems: predictable scheduling algorithms and applications, volume 24. Springer, 2011.

[2]. Giorgio C. Buttazzo. Hard Real-time Computing System: Predictable Scheduling Algorithms and Apps. (Real-Time Sys. Series). Springer-Verlag TELOS, 2004.

[3]. Baruah S. K., Mok A. K., Rosier L. E., "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor." In proceedings of the IEEE Real-Time System Symposium, pp. 182-190, 1990.

[4]. Leung J.Y. and Whitehead, J., 1982. "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks, Performance Evaluation" number 2, pp. 237-250, 1982.

[5]. N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard Real-Time Scheduling: The Deadline Monotonic Approach", Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software, pp. 133-137, May 1991.

[6]. K. Kotecha and A. Shah,"ACO based dynamic scheduling algorithm for real-time operating system", Sent to AIPR-08, Florida, 2008.

[7]. Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel", O'Reilly Online Catalogue, October 2000.

[8]. V. Singh, K. Vidhani, P. Ganeshpurkar and S. Khatri, "Analytical Study on Scheduling Algorithms for Real Time Static System", Proceedings of the International Conference on Recent Advances in Communication, VLSI & Embedded Systems, pp. 179-182, December 2014.

[9]. Hermann Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications", Springer, second edition.

[10]. A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, 8th ed., Wiley, 2012.

[11]. Su, H. and Zhu, D. (2013). An elastic mixed-criticality task model and its scheduling algorithm. In Proceedings of the 2013 Design, Automation Test in Europe Conference

Exhibition, pages 147–152.

[12]. N.C. Audsley, “Deadline Monotonic Scheduling”, Technical Report, Department of Computer Science, University of York, September 1990.

[13]. Leung J.Y. and Whitehead, J., 1982. “On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks, Performance Evaluation” number 2, pp. 237-250, 1982.

[14]. B. Brandenburg, J. Calandrino, and J. Anderson. On the scalability of real- time scheduling algorithms on multicore platforms: A case study. In Real-Time Systems Symposium, 2008, pages 157–169, 30 2008-dec. 3 2008.

[15]. Gujarati A, Cerqueira F, Brandenburg BB (2015) Schedulability analysis of the linux push and pull scheduler with arbitrary processor affinities. In: Proceedings of the 25th Euromicro Conference on Real-Time Systems, ECRTS’13, pp 69–79.

[16]. 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.

[17]. Habibah Ismail(&) and Dayang N.A. Jawawi, “A Hybrid Multiprocessor Scheduling Approach for Weakly Hard Real-Time Tasks”, Published in asiasim 2017, Part II, CCIS 752, pp. 666–678, Springer 2017.

[18]. Gujarati A, Cerqueira F, Brandenburg BB, Multiprocessor Real-Time Scheduling with Arbitrary Processor Affinities: From Practice to Theory. In: Proceedings of the 25th Euromicro Conference on Real-Time Systems, ECRTS’16, pp 69–79, 2016.

[19]. Shiva Nejati1, Stefano Di Alesio1 , Mehrdad Sabetzadeh1,Lionel Briand1“Modeling and Analysis of CPU Usage in Safety-Critical Embedded Systems to Support Stress Testing”.

[20]. Baker TP, Baruah SK (2007) Schedulability analysis of multiprocessor sporadic task systems. In: Handbook of Realtime and Embedded Systems, CRC Press.

[21]. Kato S, Yamasaki N, Ishikawa Y (2009) Semi-partitioned scheduling of sporadic task systems on multiprocessors. In: Proceedings of the 21st Euromicro Conference on Real-Time Systems, ECRTS’09, pp 249 –258.

[22]. Baruah SK, Brandenburg BB (2013) Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities. In: Proceedings of the 34th IEEE Real-Time Systems Symposium, RTSS’13, pp 160–169.

[23]. Anderson JH, Bud V, Devi UC (2005) An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In: Proceedings of the 17th Euromicro Conference on Real-Time Systems, ECRTS'05, pp 199–208.

[24]. 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.

[25]. Bado B, George L, Courbin P, Goossens J (2012) A semi-partitioned approach for parallel real-time scheduling. In: Proceedings of the 20th International Conference on Real-Timeand Network Systems, RTNS'12, pp 151–160.

[26]. Easwaran A, Shin I, Lee I (2009) Optimal virtual cluster-based multiprocessor scheduling.Real-Time Systems 43(1):25–59.

[27]. A. Foong, J. Fung, and D. Newell, "An in-depth analysis of the impact of processor affinity on network performance," in Proceedings of the 12th IEEE International Conference on Networks, vol. 1, 2004, pp. 244 – 250.

[28]. C.L. LIU, JAMES W. LAYLAND "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", in Readings in Hardware/Software Co-Design(Elsevier), 2002.

[29]. U.M.C. Devi; J.H. Anderson,"Tardiness bounds under global EDF scheduling on a multiprocessor," presented in 26th IEEE International Real-Time Systems Symposium, IEEE Xplore-2006, Print ISSN: 1052-8725.

[30]. Buttazzo, G. C. and Stankovic, J. A. (1995). Adding robustness in dynamic preemptive scheduling. In Fussell,D. S. and Malek, M., editors, Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems, volume 297 of The Springer International Series in Engineering and Computer Science, pages 67–88. Springer US.

[31]. Jinkyu Lee, Member, IEEE, and Kang G. Shin, ''Preempt a Job or Not in EDF Scheduling of Uniprocessor Systems'', published in IEEE Transactions On Computers, Vol. 63, No. 5,pp-1197-1205, 2014.

[32]. Keerthanaa C, Poongothai M., "Improved Priority based Scheduling Algorithm for Real Time Embedded Systems", published in IEEE xplore for International Conference on Circuit, Power and Computing Technologies [ICCPCT], 2016.

[33]. Giorgio C. Buttazzo, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications", Springer, Third edition.

[34]. Hoon Sung Chwa, Jaebaek Seo, Jinkyu Lee, Insik Shin, "Optimal Real-Time Scheduling on Two-Type Heterogeneous Multicore Platforms", Published in 2015 IEEE Real-Time Systems Symposium.

[35]. Jeremy P. Erickson (2014). Managing Tardiness Bounds and Overload in Soft Real-Time Systems. PhD thesis, the University of North Carolina at Chapel Hill.

[36]. R. Kalpana, S. Keerthika, "An Efficient Non-Preemptive Algorithm for Soft Real-Time Systems using Domain Cluster–Group EDF", published in International Journal of Computer Applications (0975 – 8887), Volume 93 – No.20, May 2014.

[37]. Xi Chen ,Hongli Xu, Liusheng Huang"Online Scheduling Strategy to Minimize Penalty of Tardiness for Real-Time Tasks in Mobile Edge Computing Systems" (ICBDC)4th International Conference on Big Data and Computing,ACM Digital Library,May 2019.

[38]. Davis RI, Burns A (2011b) A survey of hard real-time scheduling for multiprocessor systems. ACM Computing Surveys 43(4):35:1–35:44.

[39]. Suzhen Lin (2005) Feedback-based task scheduling in real-time systems. PhD thesis, The Iowa State University.

[40]. Kyoung-Don Kang, Sang H. Son, Senior Member, IEEE, and John A. Stankovic, Fellow, IEEE, "Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases", IEEE Transactions on Knowledge And Data Engineering, Vol. 16, No. 7, July 2004.

[41]. Ga´lvez JJ, Ruiz PM, Skarmeta AFG (2010) Heuristics for scheduling on restricted identical machines. Tech. rep., University of Murcia, Spain.

[42]. Brandenburg B.B. and Anderson J.H., "A Comparison of the M-PCP, D-PCP, and FMLP on LITMUS". In proceedings of the International Conference on Principles of Distributed Systems, 2008.

[43]. Markatos E, LeBlanc T (1992) Using processor affinity in loop scheduling on shared-memory multiprocessors. In: Proceedings of Supercomputing'92, pp 104–113.

[44]. Salehi JD, Kurose JF, Towsley D (1995) Further results in affinity-based scheduling of parallel networking. University of Massachusetts, Amherst, MA.

[45]. Rajib Mall. (2007). Real-Time Systems Theory and Practice, Pearson, First edition.

[46]. Andrew S. Tanenbaum. (2003). Modern Operating Systems, Pearson Education Publication, Third Edition, ISBN: 978-81-317-2003-5.

[47]. Jane Liu. (2000). Real-Time Systems, Pearson.

[48]. C. Krishna and K. Shin. (2000). Real-Time Systems, McGraw-Hill.