Improving Dual Priority Scheduling

by

Mayur Jadhav

(Under the direction of Shelby H.Funk)

Abstract

A real-time system is a highly time constrained system and computes within restricted time intervals. These systems tend to have multiple time-critical tasks that must be carefully scheduled. Real-time systems can be used in many industries including healthcare, aircraft control, etc., where a single time derailment could create life threatening consequences.

Real-time systems commonly need to execute the same jobs repeatedly. We call these repeated jobs periodic tasks. A common scheduling paradigm, called fixed priority, assigns priorities to each task and executes all jobs generated by a task at its assigned priority. A simple variation of fixed priority is dual priority scheduling. In dual priority scheduling, tasks have a low priority, a high priority, and a promotion time. Each task executes in lower priority level untill it reaches its promotion time. Once it reaches the promotion time it starts executing at its higher priority level. This thesis focuses on improving the dual priority multiprocessor scheduling algorithm to make more task sets meet their deadlines. Our approach to increase feasibility is by increasing the priority promotion times. The increase in priority promotion time reduces the interference of the tasks that miss their deadlines in Fixed Priority schedules or Standard Dual Priority schedules. Resulting comparisons and evaluations derived from the experiments demonstrates that this approach can make many,

but not all, of the task sets meet their deadlines. In fact for some scenarios 100 percent of the Fixed Priority unschedulable task sets we tested are schedulable using our Modified Dual Priority algorithm.

INDEX WORDS: Real-Time Systems, Real-Time Multiprocessor Scheduling, Dual Priority Scheduling

Improving Dual Priority Scheduling

by

Mayur Jadhav

B.E., University of Mumbai, India, 2012

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2014

©2014

Mayur Jadhav

All Rights Reserved

Improving Dual Priority Scheduling

by

Mayur Jadhav

Professor:	Shelby H.Funk
Committee:	E.Rodney Canfield Hamid Arabnia

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2014

Improving Dual Priority Scheduling

Mayur Jadhav

April 24, 2014

Acknowledgments

I would like to express my deepest appreciation to my advisor Dr.Shelby Funk for guiding and advising me during my Masters program. I would also like to thank my committee members Dr.Hamid Arabnia and Dr.Rodney Canfield for their help in academics. My sincere thanks also goes to Chiahsun Ho for helping during my Masters thesis.

A special thanks to my family. Words cannot express how grateful I am to my parents and my sister for all the sacrifices that they have made. In the end, I would like to thank all my friends for supporting me.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

A real-time system is a system which computes within a real-time constraint. It has to respond within a specific time interval to avoid any severe or life threatening consequences. The time constraint describes the behavior of the system. Real-time systems are used in a wide range of industries like medicine, automobiles, health, defense, etc. Some of the applications of real-time systems are as follows: monitoring the chemical plant conditions, automated car, aircraft control, airline reservation system, laser printer, missile system, cell phones and several others [15].

When we talk about these applications we wonder how time constraints really affect these real-time systems. To answer this question we will elaborate on one of the applications. For example,when monitoring a chemical plant the real-time system periodically checks the reading of certain metrics like pressure, temperature, etc. [15]. Based on these readings the system changes the parameters of the chemical control periodically. If the system reads one value at a different time than others, it might cause a miscalculation and create harmful situations. This example emphasizes how time constraints are important in a real-time system.

Some of the characteristics of real-time systems are time constrained, embedded in nature,

reactive, task critical, custom hardware, safety critical, concurrent, stable, and reliable. Real-time systems use deadlines to infer time constraints. Based on the gravity of the consequences of a deadline miss in a real-time system, the system is further categorized to hard real-time systems (HRT) and soft real-time systems (SRT).

In hard real-time systems jobs must be completed prior to their deadlines. The system will always fail in HRT if the tasks do not execute or produce results as per the given time bounds. For example consider an automated car running on the road. If it does not sense the signals or the cars ahead of it, a potential fatal accident is bound to happen. Applications having hard real-time constraints are safety critical. From the above applications mentioned aircraft control, monitoring chemical power plant and missile system are hard real-time systems.

In soft real-time systems, some deadline misses are allowed. The system tries to reduce the penalties to avoid deadline misses. For example, it may try to minimize frame losses during online streaming. This does affect the system performance but the consequences are not life threatening. The applications having soft real-time constraints are not safety critical. From the above applications mentioned laser printer and cell phones are soft real-time systems.

Scheduling is the heart of a real-time system. Much of the research in real-time system is based on scheduling algorithms. Many real-time systems are composed of tasks, which repeatedly perform some job. There are two types of tasks: periodic and sporadic. In periodic tasks, the interval is the same between each occurrence of the job of the task. In sporadic tasks, the interval may or may not be equal between occurrence of the task, but there is a minimum time between consecutive releases. Most of the ongoing research uses periodic or sporadic tasks sets to derive scheduling algorithms based on different needs for real-time system applications.

As multi-core processors become more prevalent, more real-time systems are being designed for multiprocessors. Multi-cores can run several tasks concurrently (one task per

processor). To take advantage of this feature new scheduling algorithms are being proposed. Multiprocessor scheduling proposes two types of scheduling: global scheduling and partitioned scheduling.

In global multiprocessor scheduling all the task sets are stored in a single global queue and assigned to execute by a global scheduler. The global scheduler assigns jobs to the processors, these algorithms also permit migration of tasks over processors. In partitioned multiprocessor scheduling the task set is subdivided into subsets. Each task subset has a local scheduler which assigns the tasks jobs to their allocated processor.

Multiprocessor real-time scheduling is of two types: offline scheduling and online scheduling. A scheduling algorithm is offline if all the decisions are made before the execution starts. All the details of the system are known like the job arrival times, idle time of the processors, etc. In an online scheduling algorithm the decisions of executing the jobs are made while the system is running. There is no prior knowledge of the job arrival times, etc. These scheduling algorithms can be preemptive or non-preemptive in nature.

This thesis will contribute in enhancing the dual priority scheduling algorithm [6] for multiprocessors. The dual priority multiprocessor scheduling algorithm by Davis et.al. [6] is modified to make more task sets meet their deadlines. Experimental results demonstrate the method we propose is successful much of the time. In some cases, our modification of the dual priority algorithm makes 100 percent of task sets meet their deadlines.

The remainder of this thesis is organized as: Chapter 2 presents some conceptual definitions and specifications of related terms. Chapter 3 explains the related multiprocessor scheduling algorithms and the research done over decades on the same. Chapter 4 describes the modifications and execution process. Chapter 5 shows our experimental results. Finally, Chapter 6 gives some concluding remarks and avenues for future work.

Chapter 2

MODEL AND DEFINITIONS

The notations mentioned in this section are used throughout the thesis. We consider executing a task set $\tau = T_1, ...T_n$ on m processors. A task T_i is a piece of work to be done repeatedly and is described by the tuple (p_i, e_i) where the task releases jobs p_i time units apart and e_i denotes the execution time. The k^{th} job of task T_i denoted by $T_{i,k}$ is released at time $a_{i,k}$ where $a_{i,k} = (k-1) \cdot p_i$ is the arrival time, e_i is the execution time and $d_{i,k} = k \cdot p_i$ is the deadline of the k^{th} job of task T_i. This job must be allowed to execute for e_i time units during the interval $[a_{i,k}, d_{i,k}]$. Each task is assigned a utilization denoted by u_i, which is calculated as $u_i = e_i$ / p_i. This is the fraction of processing time the task will require.

Scheduling algorithms can be categorized as fixed priority scheduling algorithms or dynamic priority scheduling algorithms. In fixed priority scheduling at any time the task with highest priority runs. In dynamic priority scheduling priorities can change during execution time. One variation of fixed priority scheduling is dual priority scheduling, initially introduced by Davis and Wellings [6]. This algorithm improves aperiodic tasks response time and also guarantees that HRT tasks meet their deadlines. An extended version of dual priority algorithm has three priority ranges: lower, middle and upper. The HRT tasks are assigned two fixed priorities; one in the lower range and one in the upper range. There is one middle

Table 2.1: Notations

Symbol	Description
T_i	Task number i
e_i	Worst case execution time of T_i
R_i	Worst case response time of T_i
D_i	Deadline of T_i
λ_i	Promotion time of T_i
$\tau_{i,low}$	Low priority assigned to task T_i
$\tau_{i,high}$	High priority assigned to task T_i
p_i	Period of T_i
a_i	Arrival time of job of task T_i
n	Number of tasks
m	Number of processors
u_i	Utilization of task T_i

priority range. All non-real-time jobs, these jobs have no deadlines and they are executed in the middle priority range in first in first out (FIFO) order.

T_i's worst-case response time, is the maximum amount of time that can elapse between the release and completion of any job of T_i. We compute R_i by adding the execution time e_i and the worst case intereference I_i. λ_i is the corresponding promotion time i.e. difference of period p_i and worst case response time R_i of the task T_i. In priority scheduling there are different levels of priorities. They are lower priority and higher priority. In fixed priority scheduling the priorities assigned to the task are fixed that is lower priority τ_{low} or higher priority τ_{high}. In dual priority scheduling the priorities assigned are also lower priority τ_{low} and higher priority τ_{high} but the priorities may change during execution of the task. Let $\tau_{i,low}$ be the low priority assigned to the task T_i and $\tau_{i,high}$ be the high priority assigned to the task T_i.

We use certain concepts and terms in the algorithms. We will define the terms lag, fluid schedule and time slots. A schedule is said to be a fluid schedule when at all times t, each task T_i has executed for $u_i \cdot t$ time units. Fluid schedules are theoretical and are used for

analysis only. A task's lag at time t is defined as the difference between its actual scheduled time during $[0, t]$ and ideal scheduled time. Time slots are the time intervals defined by a start time, end time and a duration. Scheduling decisions might be made at the beginning of time slots which can be of unit length intervals or intervals between consecutive deadlines of the tasks. A schedule is said to be a valid schedule when all task deadlines are met. A task set is said to be feasible if a valid schedule can be found. A scheduling algorithm is said to be optimal if the algorithm produces a valid schedule for all feasible task sets.

Chapter 3

BACKGROUND AND RELATED WORK

This chapter discusses background material used in developing our results for enhanced multiprocessor dual priority scheduling [6]. We first discuss multiprocessor scheduling results and then discuss dual priority scheduling [6].

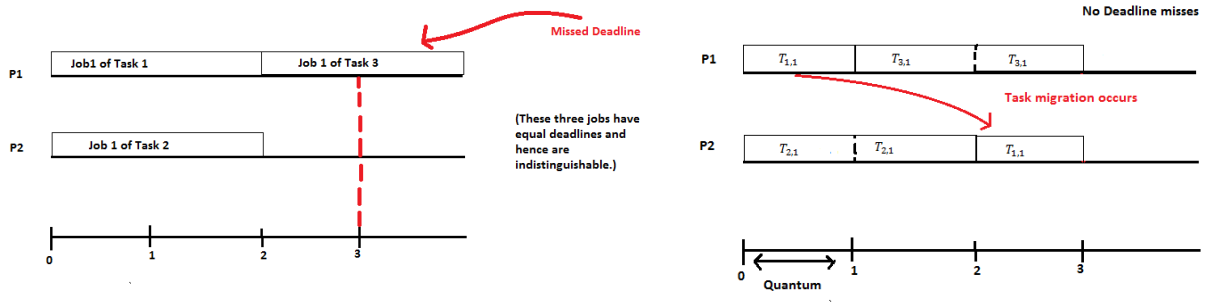
3.1 Multiprocessor scheduling algorithms

Many applications are created for multiprocessor environment. Some algorithms based on time based strategies include P-Fair [3], DP-Wrap [13], EKG [20], LLREF [11], LRETL [7] and based on global scheduling include Dual Priority [6], PNPDP [10], U-EDF [16], RUN EDF [14], EDF [1] and Fixed Priority [2]. None of the algorithms we consider allow for task parallelism–i.e., a task can only be executing on one processor at a time.

3.1.1 P-Fair Scheduling

The P-Fair scheduling algorithm [3] is optimal, provides low output jitter and works in coexistence with real-time and non-real-time tasks. The P-Fair algorithm [3] schedules tasks whose execution times, periods and deadlines are all integer values. Recall that a fluid schedule executes for $u_i \cdot t$ time units at all times t for each task T_i.

In P-Fair [3], decisions are made at each integer time value. Lag is defined as the difference between the actual scheduled time duration and the fluid scheduled time duration in a given time interval. Each job is divided into subjobs with unit length of execution time. Subjobs are assigned pseudo-deadlines. P-Fair execute jobs with earliest pseudo-deadline first. The schedule is valid if and only if all tasks have a lag value of 0 at every deadline that ensure the tasks lag values are above -1 and below 1. Based on this condition closed form expressions can be derived that computes the earliest and latest slots for each integer length subjob of each job of T_i. The reason why P-Fair [3] meets its deadline is that the lag of each task lies between -1 and 1 exclusive and lag belongs to $\mathbb{Z}$ at each deadline because the fluid schedule and the actual schedule have both executed T_i for an integer amount of time at end of its deadline.



(a) Fixed priority schedule

(b) P-Fair schedule

Figure 3.1: P-Fair Scheduling Source:The Case for Multiprocessor Fair Scheduling

For example, consider three tasks all with period equal to 3 and execution time equal to 2. Figure 3.1(a) illustrates fixed priority schedule and Figure 3.1(b) P-Fair schedule [3].

In fixed priority scheduling there is a deadline miss. To avoid deadline misses in P-Fair scheduling [3] each task is allocated a time slot i.e. quantum. Subjob $T_{1,1}$ of TaskT_1 executes one quantum time on processor 1 and one quantum time on processor 2–i.e., there is task migration from processor 1 to processor 2. All the tasks meet their deadlines.The pseudo deadline of subjob $T_{2,1}$ of task T_2 is 3 and of subjob $T_{3,1}$ of task T_3 is 2 because the subjob $T_{3,1}$ of task T_3 requires 3 units of time to complete 2 units of execution.

3.1.2 DP-Wrap

P-Fair [3] creates a scheduling event which computes over a discrete time quantum for a set of running tasks. P-Fair [3] has a very strict rule which says that the work should be completed within 1 unit of its fluid curve rate for each quantum time. A fluid curve rate is the amount of ideal work completed from the jobs release time to the jobs deadline.

Deadline Partitioning(DP) is a technique of partitioning time into slices. Time slice boundaries are determined by task deadlines. A time slice is the period of time which a process is allowed to run in a preemptive multitasking system. There are two aspects of deadline partitioning, the jobs are allocated a workload for the time slice and share the same deadline within each slice. DP-fair algorithm [13] is said to be DP-Correct if,

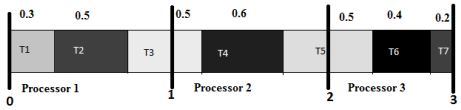
- First, jobs are allocated local workloads for each time slice in proportion to their utilization
- Secondly, the scheduler executes all the jobs by the end of the time slice

With these two conditions, we can conclude that any DP-Correct scheduler is optimal. It is a global scheduling algorithm where tasks are periodic with deadlines less than or equal to periods. Scheduling within a time slice is done by considering jobs laxities i.e. remaining time to end of time slice minus remaining local execution.

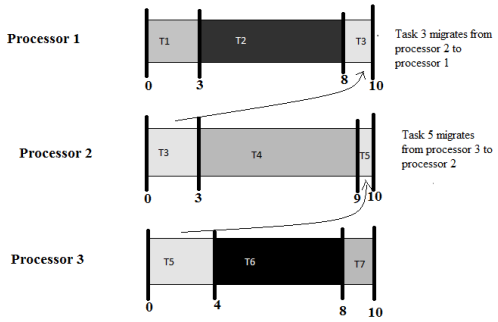
DP-Fair [13] algorithm follows the following rules:

- First, if the job has 0 laxity, it will execute it right away
- Secondly, job should not execute if its local execution time is zero
- Third, the processor should not idle for more than the system's slack time, which is $(m - U) \cdot L$ where m is the number of processors, U is the total task utilization and L is the length of the time slice

DP-Wrap [13] is a simple DP-Fair scheduling algorithm [13]. DP-Wrap [13] satisfies the above rules of DP-Fair scheduling. In this algorithm tasks which begin running at a time slice on one processor ends on another processor at the end of the time slice. This is valid as long as the utilization of the task is not more than 1.



(a) Seven tasks with utilizations shown above. These are lined up in arbitrary order, then split at length 1 intervals.



(b) Each processor runs its task set over a length 10 time slice. Jobs sliced in (a) are seen migrating in (b).

Figure 3.2: Source:DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling

For example, figure 3.2(a) consists of 7 tasks and 3 processors. Consider the task T_1(10,3),T_2(20,10), T_3(10,5), T_4(15,9), T_5(12,6), T_6(15,6) and T_7(20,4). We divide the task set into time slices. The first time slice has length 10, which is the smallest task period. The tasks execute on each processor with a fixed utilization time for each. Task T_1 executes for 3 time units and T_2 executes for 5 time units on processor 1. Task 3 from processor 2 migrates to processor1 and executes as shown in figure 3.2(b). The remaining execution of task T_3 is

executed on processor 2 and task T_4 executes for 6 units time on processor 2. T_5 migrates from processor 3 to processor 2 and tasks T_6 and T_7 complete their execution on processor 3. At the end of each time slice, all the tasks have a lag equal to 0. Because each deadline coincides with a time slice boundary, this algorithm is optimal and all the tasks meet their deadline.

3.2 Response Time Analysis

The time demand analysis (TDA) was first presented by Panday et al. [17] to determine schedulability for larger utilization values than 0.693. To determine the schedulability we use time demand analysis (TDA), which recursively determines the worst case response time of a task. It also determines that a periodic task is schedulable with deadlines less than periods using fixed priority scheduling.In a uniprocessor fixed priority scheduling, the worst-case response time is said to happen during a critical instant. A critical instant for a task T_i is a time where all the higher priority tasks are released at the same as T_i.

Worst case system demand of task $T_1, ...T_i$ in an interval of length x is denoted by $W_i(x)$. The maximum system demand for fixed priority scheduling on uniprocessor in the interval is,

$$W_i(x) = e_i + \sum_{k=1}^{i-1} \lceil \frac{x}{p_k} \rceil \cdot e_k$$

A task T_i is said to meet all its deadline if the worst case system demand $W_i(x)$ is less than the interval of length x.

Bertogna et al. [4] introduced worst case scenario for a level- k busy period, which is the largest interval during which the processor is executing tasks at higher priority. For multiprocessors, the critical instant theorem no longer applies. Therefore we consider three types of jobs in the level-k busy period.

- body: jobs have arrival times and deadlines in the level-k busy period
- carry-in: jobs have arrival times before the level-k busy period and deadlines in the level-k busy period
- carry-out: jobs have arrival times in the level-k busy period and deadlines after the level-k busy period

Further, Guan et al [9] extended Bertogna et al. [4] to more accurately compute fixed priority response times for multiprocessors. Like [4] there are two types of interference for a level-k busy period x, carry-in I_k^{CI} and no-carry-in I_k^{NC} (where no-carry-in includes both body and carry-out jobs). They proved that there can be at most $m-1$ carry-in jobs. With this in mind, we let $\mathbb{Z}_k \leq \tau \cdot \tau$ be the set of all partitions of $T_1, ..., T_{k-1}$ such that $\forall(\tau_k^{NC}, \tau_k^{CI}) \in \mathbb{Z}$, $|\tau^{CI}| \leq m-1$.

Further, they determined the total interference of all higher priority jobs caused during the $level-k$ busy period denote by $\Omega_k(x)$.

$$\Omega_k(x) = \max_{(\tau_k^{NC}, \tau_k^{CI}) \in \mathbb{Z}} (\sum I_k^{NC}(x) + \sum I_k^{CI}(x))$$

Using the upper bound $\Omega_k(x)$ of the total interference to a task T_k during the level-k busy period of length x, RTA is determined for task T_k. The level-k busy period begins at time t, the response time analysis is the minimal value x such that,

$$x = \lfloor \frac{\Omega_k(x)}{m} \rfloor + e_k$$

The solution obtained determines the upper bounds of the task T_k's response time. Hence we can conclude that if the fixed point x is greater than the period p_k, the task set may be unschedulable. Because the results of this analysis are pessimistic the test is sufficient only–i.e., fixed priority-schedulable task sets may fail this test.

3.3 Dual Priority Scheduling

Davis et al. [6] presented dual priority uniprocessor scheduling algorithm. This algorithm was designed to improve non-real-time jobs response times and also guarantee that HRT tasks meet their deadlines. In this algorithm, there are three priority ranges: lower, middle and upper. The HRT periodic tasks are assigned two fixed priorities; one in the lower range and one in the upper range. There is one middle priority range and all non-real-time jobs execute at the middle priority range in first in first out(FIFO) order.

Each periodic task T_i is assigned two fixed priorities $\tau_{i,low}$ and $\tau_{i,high}$ and a promotion time λ_i. The promotion time is equal to deadline minus the worst case response time. Initially jobs generated by each task T_i are assigned lower priority $\tau_{i,low}$. If the job does not finish its execution before the promotion time it is promoted to higher priority $\tau_{i,high}$. All jobs are executed in a priority driven manner– i.e., at all times t, the highest priority job is executing.

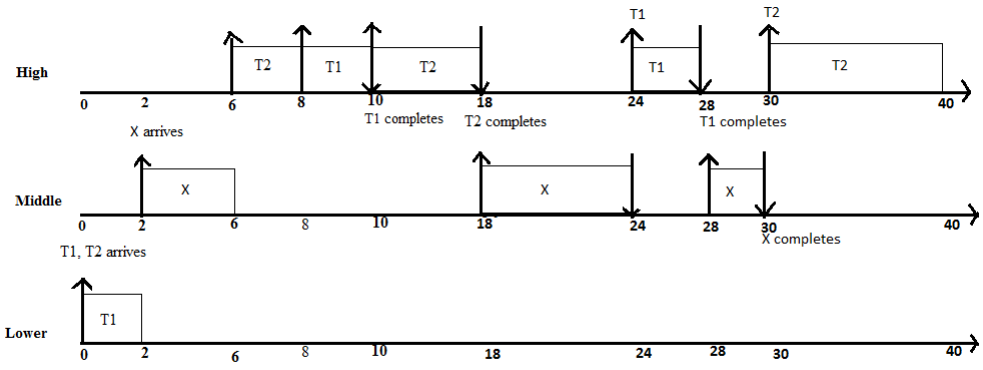


Figure 3.3: Uniprocessor Dual Priority Schedule- Source: Reducing scheduling overheads in multiprocessor real time scheduling

Dual Priority scheduling [6] was originally developed for uniprocessors and extended for multiprocessors. Figure 3.3 illustrates a uniprocessor Dual Priority schedule [6].Consider two tasks T_1(16,4), T_2(24,10) and one non-real-time job X(36,12). The promotion time of task T_1 is λ_1(8) and of task T_2 is λ_2(6). Task T_1 and T_2 arrive at time 0. Task T_1 has

higher priority than task T_2 so it starts executing. Non real-time job X arrives at time 2 and preempts task T_1 and starts executing. Task T_2 reaches it promotion time and preempts job X at time 6 and executes for 2 time units. At time 8 task T_1 reaches its promotion time and preempts task T_2 and completes its execution. At time 10 task T_2 resumes and completes its execution. At time 24 task T_1 executes at high priority level and completes its execution. At time 28 job X resumes and completes its execution.

Dual priority [6] was originally designed for uniprocessor scheduling to improve non-real-time response times and was extended in a variety of ways. Jejurkar et al. [12] extended the uniprocessor dual priority scheduling approach to reduce power consumption. Gopalakrishnan et al. [8] determined utilization bounds for EDF and RM scheduling with deferred preemptions. Bril et al. [5] explore worst-case response time analysis of uniprocessor fixed priority systems.

3.3.1 PNPDP (Partially Non Preemptive Dual Priority)

The Partially Non Preemptive Dual Priority scheduling algorithm [10] extends the dual priority [6] paradigm with some modifications. In dual priority [6], the HRT tasks are assigned two priorities, one lower priority level and one higher priority level. Like in Dual Priority [6], when a job is released the task will execute in lower priority level. Once it reaches the promotion time it is promoted to higher priority level. Unlike Dual priority [6], there are no non-real-time jobs in PNPDP [10] scheduling. There are several differences between standard dual priority and PNPDP [10].

- First, lower priority jobs do not initiate preemptions
- Secondly, promotion times are deferred when tasks execute at lower priority

The promoted task can only initiate preemptions. Consider a task T_i which is promoted at time t. It can only initiate preemption if and only if $\tau_{i,high}$ is higher than the current

priority task T_j.

In PNPDP [10], Ho et al. [10] use Guan's WCRT analysis [9] to determine each task's priority promotion offset. Worst case response time (WCRT) is the sum of execution time e_i and worst-case interference I_i– i.e., $R_i = e_i + I_i$. The PNPDP [10] algorithm delays the promotion time when tasks execute at lower priority. Delaying promotion time does not cause a job to miss its deadline because WRCT must be reduced at least by the amount of time the task has already executed.

Table 3.1: Task Set

Task T_i	p_i	e_i	λ_i	R_i
T_0	4	2	2	2
T_1	7	3	4	3
T_2	10	4	3	7
T_3	25	5	6	19

Table 3.1 describes a task set to be scheduled on two processors. Figure 3.4 illustrates the PNPDP [10] schedule. Consider T_i to be the periodic task, $\tau_{i,high}$ to be the higher priority and $\tau_{i,low}$ to be the lower priority and λ_i to be the promotion time. At $t = 0$ task T_2 and T_3 are released.They begin executing in lower priority. At time $t = 1$ task T_1 and at time $t = 2$ task T_0 are released. Even though tasks T_0 and T_1 have higher priorities than tasks T_2 and T_3, they do not begin execution. Here at lower priority $\tau_{i,low}$ both the tasks T_2 and T_3 execute normally without preemption and they are promoted to higher priority once they reach their promotion time λ_i. Preemption only occurs at higher priority $\tau_{i,high}$. An example of PNPDP [10] schedule is as follows:

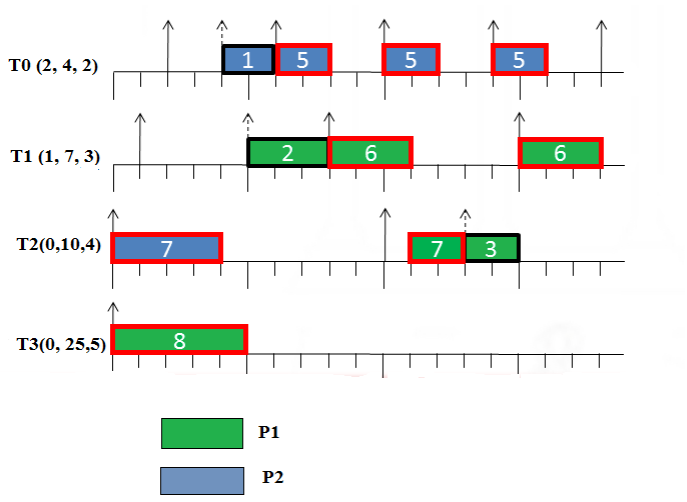


Figure 3.4: PNPDP schedule

This approach reduces the preemption and migration overhead. Ho et al. [10] also analytically proved that PNPDP [10] will never allow a deadline miss if the dual priority schedule is valid. Task sets that miss deadlines in fixed priority schedule may be schedulable in PNPDP [10] approach. Our research builds upon this idea by increasing the priority promotion times further.

Chapter 4

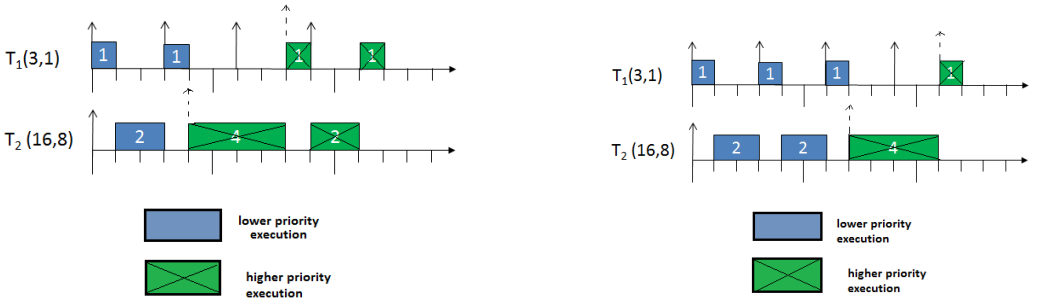
IMPROVING DUAL PRIORITY SCHEDULING

The motivation of this research is to calculate the amount of time tasks execute at their lower priority levels and increase priority promotion times. Task that spend less time in their promoted priority level will impose less interference on other tasks. This may allow some tasks to meet their deadlines even though they miss deadlines with the original priority promotion times. For this research, we assume that tasks high and low priorities are in the same order. For example, if $i < k$, then if T_i and T_k have not yet been promoted then T_i has higher priority than T_k (since tasks are indexed by high priority, T_i certainly is higher priority than T_k when promoted).

This thesis works along the lines of standard Dual Priority scheduling [6] as discussed in Chapter 3. In particular, the priority promotion times are increased as much as possible. Using Guan's response time calculation [9] promotion time λ_i is set to the difference of period p_i of task T_i and the worst case response time R_i– i.e., $\lambda_i = p_i$ - R_i. Therefore, these priority promotion times are computed assuming the tasks execute only at their higher priority levels.

Worst case response time R_i of task T_i is equal to summation of worst case execution

time e_i and worst case interference caused by the higher priority tasks–i.e., $R_i = e_i + I_i$. If task T_i executes before it is being promoted its remaining execution time will clearly be reduced. Hence if task T_i executes for ϵ_i time units then the promotion time can be delayed by at least ϵ_i. Also the interference I_i caused to task T_i may be reduced due to the reduced execution time. This reduces the worst case response time at the high priority level and helps tasks to meet their deadlines. For example, consider two tasks T_1(3,1) and T_2(16,8). R_1= 1 is worst case response time and the promotion time is λ_1=2 time units for task T_1. R_2= 12 is worst case response time and the promotion time is λ_2=4 time units for task T_2. Figure 4.1(a) shows the schedule before delaying the priority promotion time.



(a) Before delaying promotion time

(b) After delaying promotion time

Figure 4.1: Priority Promotion

After we recalculate the promotion times we get a larger priority promotion time for task T_2. The new worst case response time for task T_2 is R_2=9 and promotion time is λ_2=7 time units. Figure 4.1(b) shows the schedule after delaying the promotion time.

The above motivation is applied in the following way. Let e_i^{lp} denote the lower priority execution of task T_i. We can calculate e_i^{lp} if we know the minimum priority–i availability for an interval of length denoted λ_i. Priority–i available time is the time when no higher priority tasks are executing so task T_i can execute. Minimum priority–i availability is dual of maximum interference. In order to compute the minimum priority–i availability for a given interval λ_i we modify Guan's response time analysis as described below [9].

Guan's [9] work computes T_i's higher priority interference Ω_i by considering only tasks T_1,...T_{i-1}. By contrast, for Dual Priority [6] scheduling, lower priority tasks may also interfere with T_i before it is promoted. For this reason, we modified Guan's [9] approach to compute the interference of all tasks as follows

- For $j < i$, (higher priority tasks) each job of T_i interferes for e_j time units
- For $j > i$, (lower priority tasks) each job of T_j interferes for e_j^{hp} time units

We modify Guan's approach iteratively to get larger priority promotion times so that tasks meet their deadlines.

Let $MinAvail_i(\lambda_i)$ denote the minimum priority–i availability over an interval of length λ_i and let $\Omega_i(\lambda_i)$denote the maximum interference of tasks $T_1, ..., T_{i-1}$ over the same interval length. Then,

$$MinAvail_i(\lambda_i) = ComputationalTime - MaximumInterference$$
$$MinAvail_i(\lambda_i) = m \cdot \lambda_i - \Omega_i(\lambda_i)$$

To compute the minimum priority–i availability over the interval λ_i we need to know the interference during that interval length. $\Omega_i(\lambda_i)$ is the interference which is calculated over the interval λ_i. This value is completed pessimistically–i.e., the actual interference may be smaller than the value $\Omega_i(\lambda_i)$ that we compute using our modified response time analysis. Therefore the minimum priority–i availability is,

$$MinAvail_i(\lambda_i) \geq m \cdot \lambda_i - \Omega_i(\lambda_i)$$

In the worst case scenario, available time occurs on all processors simultaneously. For example, figure 4.2 illustrates an example with $m = 4$ processors and the intereference $\Omega_i(\lambda_i)$=28, over an interval of length 10. If the 28 units of interference occur on all 4 processors simultaneously, that leaves only 3 time units for T_i to execute. If the higher

priority interference was distributed differently, T_i would be able to execute for more time. With this in mind, we can compute the minimum amount of time, T_i will execute at the lower priority level as follows:

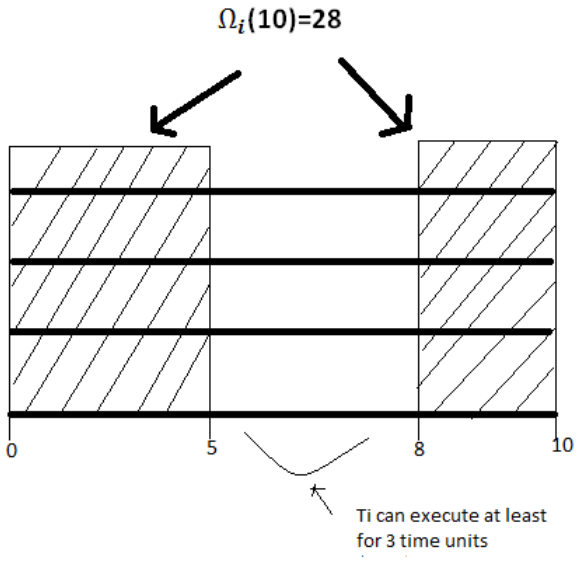


Figure 4.2: Worst case scenario when available time occurs on all processors simultaneously

$$MinAvail_i(\lambda_i) = \frac{m \cdot \lambda_i - \Omega_i(\lambda_i)}{m}$$

Therefore the available time will be 3 time units.

$$e_i^{lp} = \frac{MinAvail_i(\lambda_i)}{m}$$

$$e_i^{hp} = e_i - e_i^{lp}$$

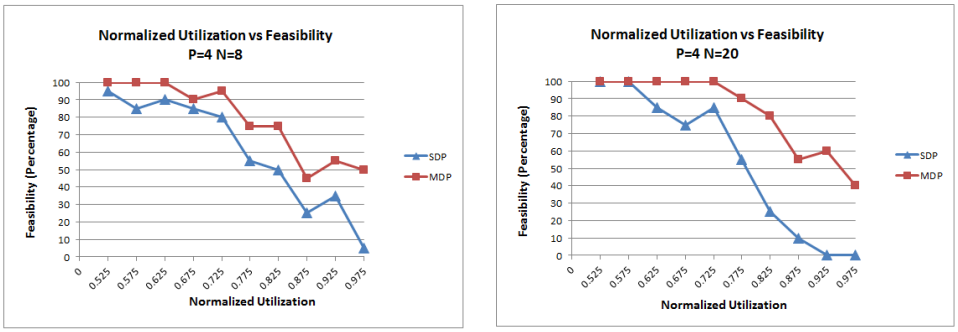
Next chapter demonstrates that these steps can greatly improve schedulability for many task sets.

Chapter 5

EXPERIMENTS

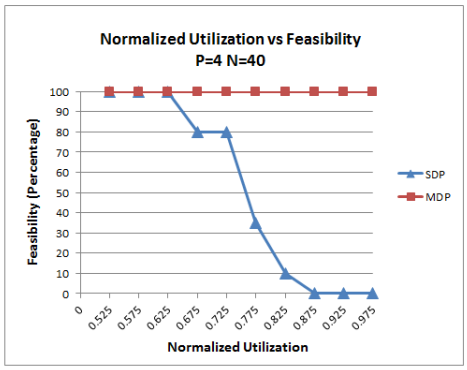
We use the randfixedsum method [19] to generate utilization values. This method has four inputs: n, a, b, s. It generates n random values lying in the interval $[a, b]$, subject to the condition that their sum be equal to s. The distribution of values is uniform in the sense that it has the conditional probability distribution of a uniform distribution in the range $[a, b]$ given that the sum is s. We generate the random utilization values by giving the total utilization s and the task utilization in the range $[a = 0, b = 1]$. We then generate task sets with these utilization values by randomly assigning periods to the tasks and compute the associated executions times ($e_i = p_i \cdot u_i$). As we are interested in evaluating how effectively our method improves schedulability, we reject all task sets that are fixed priority schedulable. We consider the OPA fixed priority assignment [18]. We generate 20 OPA-unschedulable task sets per scenario.We consider the following scenarios:

- Number of processors m=4, 8 and 16.
- Number of tasks $n = 2m, 5m, 10m, 15m$
- Total utilization $u = 0.525m, 0.575m, 0.625m, 0.675m, 0.725m, 0.775, 0.825m, 0.925m, 0.975m$. (Normalized utilization $= \frac{u}{m} = 0.525,\ 0.575,\ 0.625,\ 0.675,\ 0.725,\ 0.775,\ 0.825,\ 0.875,$

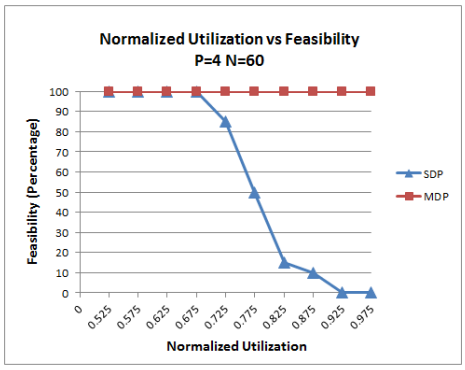


(a) 8 tasks per task set

(b) 20 tasks per task set



(c) 40 tasks per task set



(d) 60 tasks per task set

Figure 5.1: Normalized utilization vs Percentage of feasible task sets for Processors=4 and Number of tasks= 8, 20, 40,60

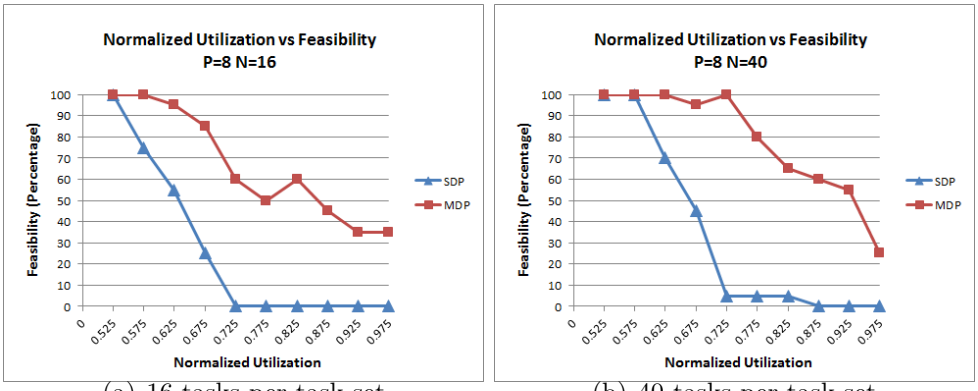
0.925, 0.975)

Therefore, we generated $20 \cdot 3 \cdot 4 \cdot 10 = 2400$ fixed priority infeasible task sets and simulated their schedules with both the Standard Dual Priority(SDP) [6] and our Modified Dual Priority(MDP) algorithms.

Figure 5.1 compares the percentage of feasible task sets containing 8, 20, 40 and 60 tasks and executing on 4 processors. The X Axis denotes the normalized utilization values for the task sets. The graph illustrates a comparison between the Standard Dual Priority Algorithm (SDP) [6]and the Modified Dual Priority Algorithm (MDP). The graphs shows that for lower normalized utilization values the percentage of feasibility is high that (75 percent), as the

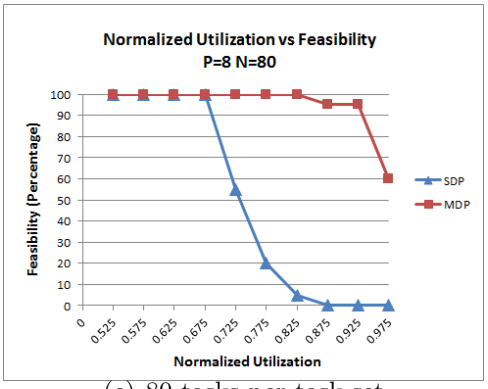
normalized utilization values increases the feasibility decreases to 0 percent for SDP [6]. While in MDP the feasibilty percentage rate is higher for lower normalized utilization values as well as higher normalized utilization values as compared to SDP [6].

We also observe that as the number of tasks increases the feasibility percentage increases in MDP. It shows that there is 100 percent feasibility for a larger number of tasks. We can conclude that, as the number of tasks increase their utilization values become smaller leading to smaller execution times with larger periods. Therefore, increasing promotion times has greater impact allowing the infeasible tasks with small execution to meet their deadlines.

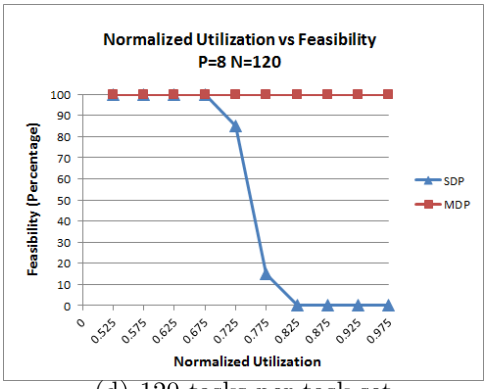


(a) 16 tasks per task set

(b) 40 tasks per task set



(c) 80 tasks per task set

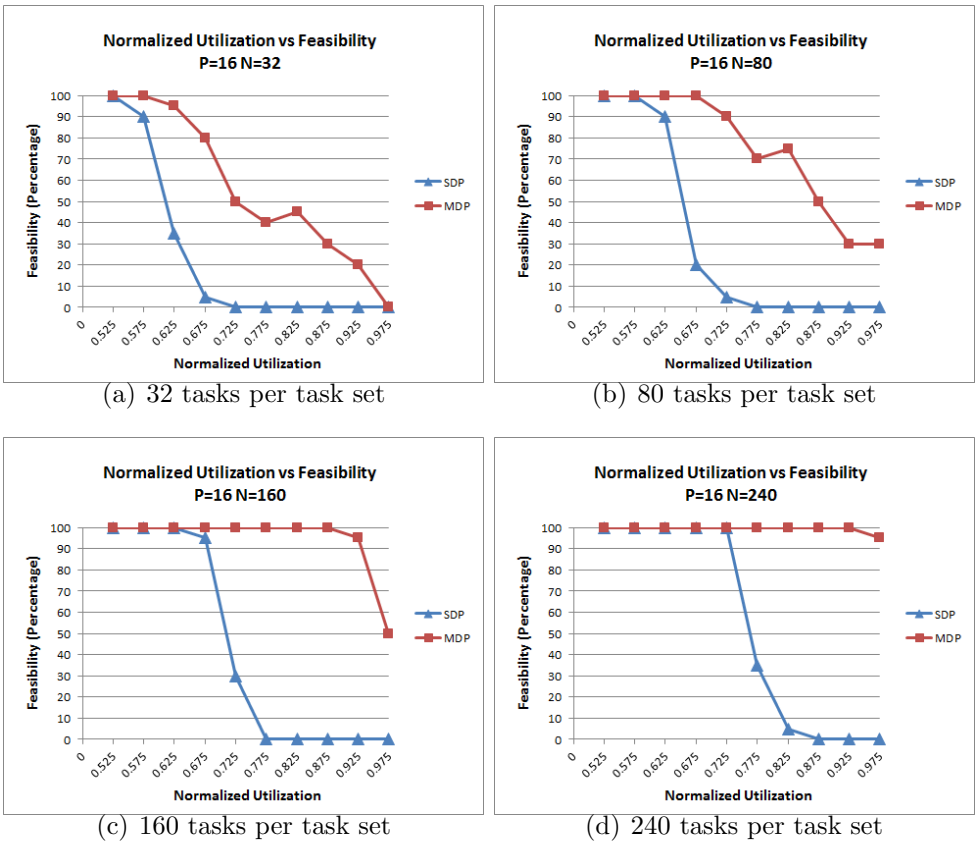


(d) 120 tasks per task set

Figure 5.2: Normalized utilization vs Percentage of feasible task sets for Processors=8 and Number of tasks= 16, 40, 80, 120

Figure 5.2 compares the percentage of feasible task sets for 16, 40, 80 and 120 tasks executing on 8 processors. In comparison with 4 processors, the results with 8 processors

are similar but different. The feasibility percentage has decreased with 8 processors than 4 processors for MDP. The reason for this decrease is that Guan's response time analysis [9] is pessimistic. This leads to unfavorable worst case response time in turn affecting the priority promotion times.



(a) 32 tasks per task set

(b) 80 tasks per task set

(c) 160 tasks per task set

(d) 240 tasks per task set

Figure 5.3: Normalized utilization vs Percentage of feasible task sets for Processors=16 and Number of tasks= 32, 80, 160, 240

Figure 5.3 compares the percentage of feasible task sets for 32, 80, 160 and 240 tasks executing on 16 processors. We also conclude the same observations as we did with 4 and 8 processors. The difference observed is also same when compared with 4 and 8 processors, though there are even more unschedulable tasks sets as the number of processors increases.

Hence we can conclude that the MDP gives much better feasibility rate than SDP [6]. Also with increase in number of processors for MDP the feasibility decreases in comparison

due the Guan's pessimistic response time analysis.

Chapter 6

CONCLUSION

The use of real-time systems in the form of embedded systems has grown rapidly. Most of the real-time systems are multiprocessor systems and are widely used for computation. Some applications of the real-time multiprocessor systems include aircraft, smart vehicles, etc.

Dual priority [6] is one of the online multiprocessor scheduling algorithms. This dissertation modifies the standard Dual priority scheduling algorithm [6]. We use Guan's response time analysis [9] to calculate the worst case response time and recompute the priority promotion times. We run the task sets with higher priority promotion times to check feasibility.

The results showed that with higher promotion times we can increase feasibility. Most of the task sets that are infeasible in standard dual priority were feasible in our modified dual priority. The results show that modified dual priority works much better than the standard dual priorities in all scenarios.

The future work of this thesis would be to apply the concept to PNPDP (Partially Non-Preemptive Dual Priority) [10] multiprocessor scheduling algorithm and also consider the different lower priority orderings.

REFERENCES

[1] B. Andersson, J. Marinho, and S. Petters. Global-edf scheduling of multimode real-time systems considering mode independent tasks. In *IEEE Real-Time Systems (ECRTS)*, pages 205–214, July 2011.

[2] N. Audsley, A. Burns, D. Robert, W. Ken, and A. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. In *Springer Real-Time Systems (RTS)*, pages 173–198, May 1995.

[3] S. Baruah. Fairness in periodic real time scheduling. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 200–209, December 1995.

[4] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetic multiprocessor platforms. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 149–160. 28th IEEE International, 2007.

[5] R. Bril and W. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *IEEE Real-Time Systems (ECRTS)*. IEEE Computer Society Washington, DC, USA, 2007.

[6] R. Davis and A. Wellings. Dual priority scheduling. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 100–109, Washington, DC, USA, December 1995. IEEE Computer Society.

[7] S. Funk and V. Nadadur. LRE-TL: An optimal multiprocessor scheduling algorithm for sporadic task sets. In *IEEE Real-Time and Network Systems (RTNS)*, pages 159–168, December 2009.

[8] R. Gopalakrishnan, G. Parulkar, and M.Gurudatta. Bringing real-time scheduling theory and practice closer for multimedia computing. *SIGMETRICS Performance Evaluation Review*, 24(1):1–12, 1996.

[9] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 387–397, December 2009.

[10] C. Ho. *Reducing Scheduling Overheads In Multiprocessor Real-time Scheduling*. PhD thesis, University of Georgia, 2013.

[11] C. Hyeonjoong, B. Ravindran, and I. Douglas. An optimal real-time scheduling algorithm for multiprocessors. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 101–110, December 2006.

[12] R. Jejurikar and R.Gupta. Procrastination scheduling in fixed priority real-time systems", year = 2004. In *LCTES '04: Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 57–66, New York, NY, USA.

[13] G. Levin, S. Funk, and S. Brandt. DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 3–13, July 2010.

[14] G. Lima, E. Massa, P. Regnier, G. Levin, and S. Brandt. RUN:optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 104–115, November 2011.

[15] R. Mall. *Real time systems theory and practice*. Prentice Hall, 2009.

[16] D. Milojevic, J. Goossens, V. Berten, and V. Neils. U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks. In *IEEE Real-Time Systems (ECRTS)*, pages 13–23, July 2012.

[17] M.Joseph and P. Panday. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *The Computer Journal*, 29(5):390–395, October 1986.

[18] I. Robert, Davis, and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems Symposium, IEEE International*, 0:398–409, 2009.

[19] R.Stafford and P. Emberson. Techniques for the synthesis of multiprocessor task sets. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2010.

[20] E. Tovar and B. Andersson. Multiprocessor scheduling with few preemptions. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 322–334, November 2006.