Global Journal of Computer Science and Technology: A Hardware & Computation
Volume 17 Issue 1 Version 1.0 Year 2017
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Scheduling Techniques for Operating Systems for Medical and IoT Devices: A Review

By Vipan Kakkar
Shri Mata Vaishno Devi University

Abstract- Software and Hardware synthesis are the major subtasks in the implementation of hardware/software systems. Increasing trend is to build SoCs/NoC/Embedded System for Implantable Medical Devices (IMD) and Internet of Things (IoT) devices, which includes multiple Microprocessors and Signal Processors, allowing designing complex hardware and software systems, yet flexible with respect to the delivered performance and executed application. An important technique, which affect the macroscopic system implementation characteristics is the scheduling of hardware operations, program instructions and software processes. This paper presents a survey of the various scheduling strategies in process scheduling. Process Scheduling has to take into account the real-time constraints. Processes are characterized by their timing constraints, periodicity, precedence and data dependency, pre-emptivity, priority etc. The affect of these characteristics on scheduling decisions has been described in this paper.

Keywords: *process scheduling, hardware software synthesis, implantable medical devices (IMD), internet of things (IoT) devices, dynamic voltage and frequency scaling (DVFS), multiprocessor, fault tolerant scheduling.*

GJCST-A Classification: *J.3*

Strictly as per the compliance and regulations of:

Scheduling Techniques for Operating Systems for Medical and IoT Devices: A Review

Vipan Kakkar

Abstract- Software and Hardware synthesis are the major subtasks in the implementation of hardware/software systems. Increasing trend is to build SoCs/NoC/Embedded System for Implantable Medical Devices (IMD) and Internet of Things (IoT) devices, which includes multiple Microprocessors and Signal Processors, allowing designing complex hardware and software systems, yet flexible with respect to the delivered performance and executed application. An important technique, which affect the macroscopic system implementation characteristics is the scheduling of hardware operations, program instructions and software processes. This paper presents a survey of the various scheduling strategies in process scheduling. Process Scheduling has to take into account the real-time constraints. Processes are characterized by their timing constraints, periodicity, precedence and data dependency, pre-emptivity, priority etc. The affect of these characteristics on scheduling decisions has been described in this paper.

Keywords: process scheduling, hardware software synthesis, implantable medical devices (IMD), internet of things (IoT) devices, dynamic voltage and frequency scaling (DVFS), multiprocessor, fault tolerant scheduling.

I. Introduction

The scheduling problem in portable and mobile systems has many facets [1] [2]. Scheduling algorithms have been developed in both the operation research and computer science community, with different models and objectives. The techniques that are applicable today to the design of hardware and software systems draw ideas from both communities.

Generally speaking, hardware and software scheduling problems differ not just in the formulation but in their overall goals. Nevertheless, some hardware scheduling algorithms are based on techniques used in the software domain, and some recent system-level process scheduling methods have leveraged ideas in hardware sequencing.

Scheduling can be loosely defined as assigning an execution start time to each task in a set, where tasks are linked by some relations (e.g., dependencies, priorities, etc.). The tasks can be elementary operations (like hardware operations or computer instructions) or can be an ensemble of elementary operations (like software programs). The tasks can be periodic or aperiodic, and task execution may be subject to real time constraints or not.

Scheduling under timing constraints is common for hardware circuits, and for software applications in embedded control systems. Task execution requires the use of resources, which can be limited in number, thus causing the serialization of some task execution. Most scheduling problems are computationally intractable, and thus their solutions are often based on heuristic techniques. Scheduling algorithms as applied to design of operating systems are explained below.

Scheduling in High-Level Synthesis (HLS) is an optimization problem [3]. The different entities that should be optimized here are speed, cost (area or resources) and power consumption. By making use of these entities, scheduling problems can be listed as (i) time constrained scheduling (ii) resource constrained scheduling (iii) feasible constrained scheduling and (iv) power constrained scheduling. There are also other factors that are important in evaluating designs such as pin limitations, package selection, testability, variety of latches, library of cells, clock skew etc. These are not discussed here.

II. Scheduling in Different Operating Systems

Process scheduling is the problem of determining when processes execute and includes handling synchronization and mutual exclusion problem [3]. Algorithms for process scheduling are important constituents of operating systems and run-time schedulers.

The model of the scheduling problem is more general than the one previously considered. Processes have a coarser granularity and their overall execution time may not be known. Processes may maintain a separate context through local storage and associated control information. Scheduling objectives may also vary. In a multitasking operating system, scheduling primarily addresses increasing processor utilization and reducing response time. On the other hand, scheduling in real-time operating systems (RTOS) primarily addresses the satisfaction of timing constraints.

First consider the scheduling without real-time constraints. The scheduling objective involves usually

Author: Shri Mata Vaishno Devi University, Katra.
e-mail: vipan.kakar@smvdu.ac.in

variety of goals, such as maximizing CPU utilization and throughput as well as minimizing response time. Scheduling algorithms may be complex, but they are often rooted on simple procedures [97] such as shortest job first (SJF) or round robin (RR). The SJF is a priority-based algorithm that schedules processes according to their priorities, where the shorter the process length (or, more precisely, its CPU burst length) the higher the priority as shown in Fig. 1. This technique arranges the processes with the least burst time in head of the queue and longest burst time in tail of the queue. This requires advanced knowledge or estimations about the time required for a process to complete. This algorithm would give the minimum average time for a given set of processes, their (CPU-burst) lengths were known exactly. In practice, predictive formulas are used. Processes in a SJF may allow preempting other processes to avoid starvation.

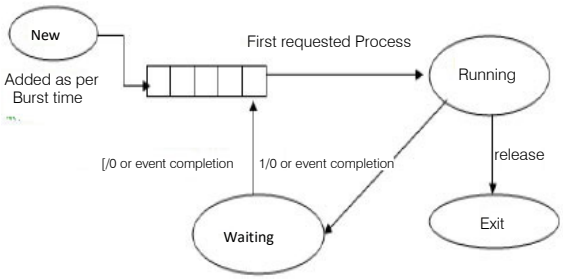

Fig. 1: Shortest Job First (SJF) Scheduling

The round robin scheduling algorithm as shown in Fig. 2, uses a circular queue and it schedules the processes around the queue for a time interval up to a predefined quantum. The queue is implemented as a first-in/first-out (FIFO) queue and new processes are added at the tail of the queue. The scheduler pops the queue and sets a timer. If the popped process terminates before the timer, the scheduler pops the queue again. Otherwise it performs a context switch by interrupting the process, saving the state, and starting the next process on the FIFO.

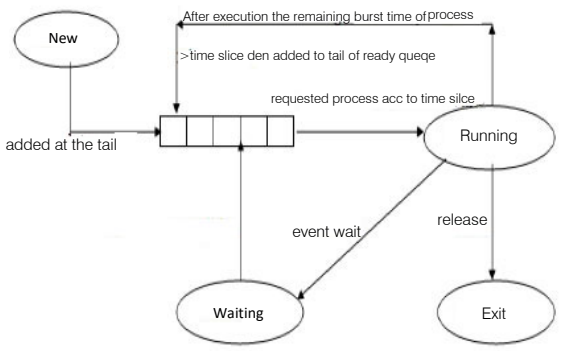

Fig. 2: Round Robin (RR) Scheduling

Different goals and algorithms characterize process scheduling in real-time operating system. Schedules may or may not exist that satisfy the given timing constraints. In general, the primary goal is to schedule the tasks such that all deadlines are met: in case of success (failure) a secondary goal is maximizing earliness (minimizing tardiness) of task completion. An important issue is predictability of the scheduler, i.e., the level of confidence that the scheduler meets the constraints.

The different paradigms for process scheduling in RTOS can be grouped as static or dynamic. In the former case, a schedule ability analysis is performed before run time, even though task execution can be determined at run time based on priorities. In the latter case, feasibility is checked at run time. In either case, processes may be considered periodic or aperiodic. Most algorithms assume periodic tasks and tasks are converted into periodic tasks when they are not originally so.

Rate monotonic (RM) analysis is one of the most celebrated algorithms for scheduling periodic processes on a single processor. RM is a priority-driven preemptive algorithm. Processes are statically scheduled with priorities that are higher for processes with higher invocation rate, hence the name. Liu and Lay land showed that this schedule is optimum in the sense that no other fixed priority scheduler can schedule a set of processes, which cannot be scheduled by RM. The optimality of RM is valid under some restrictive assumptions, e.g., neglecting context-switch time. Nevertheless, RM analysis has been the basis for more elaborate scheduling algorithms. Deadline Monotonic (DM) executes at any time instant the instance of the ready task with the shortest deadline, first. If two or more tasks have the same deadline, then DM randomly selects one for execution next. DM becomes equivalent to the RM algorithm when the deadlines of tasks are equal to their period [95].

Process scheduling plays an important role in the design of mixed hardware/software systems, because it handles the synchronization of the tasks executing in both the hardware and software components. For this reason, it is currently a subject of intensive research. A description on process scheduling is given in the next chapter.

III. Process Scheduling

This section presents various process scheduling algorithms available in the literature. Section 3.1 gives an overview of Real-time system and its characteristics have been given in section 3.2. Definition and the terminology used in process scheduling are given in section 3.3. Section 3.4 details various approaches taken for real-time scheduling. Various scheduling schemes have been compared.

Also, many references have been suggested for every scheduling scheme for an interested reader to get more details.

a) Real-time System

Real-time systems are broadly classified into soft real-time systems and hard real-time systems. In soft real-time systems, the tasks have either soft deadlines or do not have deadlines at all. Scheduler performs task scheduling as fast as possible. If the task with soft deadline finishes late, it does not lead to serious problems, but results in degraded system performance. On the other hand, in hard real-time systems, tasks have timing constraints and if these timing constraints are not met, the outcomes may be fatal. Missing the deadline of critical tasks leads to system malfunction or breakdown. Therefore, scheduling algorithm employed for task scheduling in a hard real-time system has to work satisfactorily and ensure that every task completes before its deadline. In practice, hard real-time systems invariably have many soft real-time jobs and vice versa.

Clearly, scheduling pure soft real-time tasks is a trivial job and scheduling hard real- time tasks is quite complex. In the remainder of this paper, scheduling in hard real-time systems is considered only. It is good to note that task scheduling is among the most important and critical services real-time operating system should provide. Task scheduling in hard real-time can be static or dynamic as will be seen in this paper.

b) Characteristics of the Real-Time Tasks

i. *Timing Constraints*

Their timing constraints, precedence constraints and resource requirements typically characterize real-time tasks. Real-time tasks should have the information about their timing constraints so that they can be scheduled and managed efficiently. Various timing parameters used to characterize the hard real-time tasks are given below:

Deadline: Deadline of a request for a task is defined to be the time of the next request for a task. This is the time by which the task must finish.

Response time: The response time of the task is the time span between the request and the end of the response to that request.

Arrival time or Release time (r): It is the time at which a task is invoked in the system. However, in many real time systems, we do not know the exact instant r_i at which the task J_i will be released. We only know, r_i is in the range $[r_i^- r_i^+]$, that is, r_i can be as early as r_i^- and as late as r_i^+. This range of r is sometimes called as release time jitter or simply jitter.

Relative Deadline: Relative deadline is the maximum allowable response time of the job.

Ready time: It is the earliest time at which the task can begin execution. Obviously, the ready time of a task is equal to or greater than the arrival time.

Execution time: It is the amount of time required to complete the execution of a task when it executes alone and has all the resources it requires. The actual amount of time taken may however differ for many reasons. The actual execution for a task is known only after it finishes. Hence, the execution time is mentioned as minimum and maximum execution times. Knowing the maximum execution time is enough for determining whether the task meets its deadline. Therefore, in many hard real time systems, the execution time specifically means its maximum execution time. In hard real-time systems, tasks can be periodic, sporadic or aperiodic in nature.

Slack time: Time difference between execution time and the deadline

ii. *Periodic Task Model*

The periodic task model is a well-known deterministic workload model. With its various extensions, the model characterizes accurately many traditional hard real-time applications. Many scheduling algorithms based on this model have good performance and well-understood behavior. In this model, each computation and data transmission that is repeated at regular or semi regular intervals in order to provide a function of the system on a continuing basis is modeled as a periodic task. Specifically, each periodic task, denoted by T_i is a sequence of jobs. The period p_i of the periodic task T_i is the minimum length of all time intervals between release times of consecutive jobs in T_i. Its *execution time* is the maximum execution time of all the jobs in it. We use e_i to denote the execution time of the periodic task T_i, as well as that of all the jobs in it. At all times, the period and execution time of every periodic task in the system are known.

iii. *Aperiodic and Sporadic Tasks*

Aperiodic and sporadic tasks are used to characterize the external events to the real- time system. Aperiodic and sporadic tasks are the streams of aperiodic and sporadic jobs respectively. The release times for aperiodic and sporadic tasks are not known a priori.

Real-time system has to respond to the external events while it is executing some other tasks. Real-time system executes certain routines in response to the external events. These routines or tasks to be executed in response to an external event may have soft or hard timing constraints. If the task has soft deadline or no deadline, we call it as an aperiodic task. Since the aperiodic tasks have soft deadline, we want that the real-time system to be

responsive in a sense that it completes the job as soon as possible. Although late response is annoying, it is tolerable, so the need is to optimize the responsiveness of the system for aperiodic tasks, but never at the expense of the hard real-time tasks whose deadlines must be met at all times.

If the task has hard real-time constraints, it has to meet its deadline. Failure in meeting deadlines lead to catastrophic results. Task of recovering from transient fault in time, for example, should complete before system goes down. The jobs that execute in response to these events have hard deadlines. Tasks containing jobs that are released at random time instants and have hard deadlines are sporadic tasks. Sporadic jobs may arrive at any instant, even immediately after each other. Moreover their execution times may vary widely, and their deadlines are arbitrary. In general, it is impossible for some sporadic jobs to meet their deadlines no matter what algorithm we use to schedule them. The only alternatives are (1) to reject the sporadic jobs that cannot complete in time or (2) to accept all sporadic jobs and allow some of them to complete them. Primary concern for sporadic tasks is to ensure that their deadlines are always met; minimizing their response times is of secondary concern.

iv. *Precedence Constraints and Data Dependency*

Jobs are said to be independent of each other if they can execute in any order without affecting the end result. In practice, however, jobs wait for the control and data inputs from other jobs and hence cannot execute independently. Therefore, control and data dependencies constrain the order in which the jobs can execute. Presence of dependency complicates the job scheduling, especially on a multiprocessor system.

v. *Functional Parameters*

Though scheduling and resource access-control decisions are generally taken without considering the functional characteristics of the task, several functional parameters do affect these decisions. Therefore, task workload model should explicitly mention the relevant functional parameters. Following functional parameters are generally described in the task workload model:

a. *Preemptive Jobs*

Preemption of the task is provided in the real-time systems to suspend the execution of the current job for giving processor to a higher priority or urgent task. However, some jobs need to be executed from start to finish without interruption to avoid errors in the system and to keep the switching overheads to a minimum. Such jobs are said to be non-preemptive.

b. *Priority of Jobs*

Priority of the job is the measure of the criticality or importance of the job with respect to other jobs in the system. Higher the priority, the larger its importance. Tasks scheduling algorithm decisions are mainly based on the priority of the tasks and hence the priority assignment to the task is very important. As we will see, scheduling algorithms uses static and dynamic priority assignment schemes for assigning priority to the tasks. Assigning priorities to the tasks so that all tasks meet their deadline is a difficult problem and usually some sort of heuristic is employed.

c. *Energy Aware Scheduling*

The trend in the industry towards Dynamic Power Management (DPM), where hardware technologies for dynamic frequency scaling (DVS) and dynamic voltage scaling (DVS) are being used to reduce the power consumption of individual processing elements (PE) at run-time. However, crucial to the success of this approach is a presence of intelligent software that adjusts the system performance level to maximize energy savings while still meeting application real-time deadlines.

Moreover, another trend is to build SoCs/ NoC/ Embedded System for Implantable Medical Devices (IMD) and Internet of Things (IoT) devices, which includes multiple PEs (Microprocessors+DSPs), allowing designing complex systems, yet flexible with respect to the delivered performance and executed application. The energy management of multi-PE SoCs should manage several elements with shared resources, each running their own OS, and a plurality of both real-time and non real-time applications.

Therefore, there is a need to directly address the energy problem. Intelligent energy management has impact on the hardware as well as on the software architecture of system, both implementing an infrastructure for energy management.

The objective of this energy-aware scheduling is to design a Generic Adaptive Power optimized design, which can be used in IoT and IMD devices. Its main purpose is to enable intelligent as well as adaptive power management, including the ability to make dynamic changes to the voltages and frequencies being applied to these devices. Peng *et.al* (2010) presented a novel wireless integrated power management design for biomedical telemetry systems. They designed a model such that it draws ultra-low standby current [30]. Gaurav *et.al* (2008) evaluated the effectiveness of power management using DVFS from a system level energy savings perspective [100]. However, simple policies they justified their work using benchmarks ranging from memory intensive workloads to CPU intensive workloads.

In order to introduce intelligence in any system, different learning techniques have been developed so far such as TD-learning and Q-learning, which are two powerful in terms of saving power. The "wake-up" operation after sleep mode creates a significant power-draw from the battery supply (energy overhead). To deal with this issue Siyu *et.al* (2012) proposed a hybrid power supply using continuous Q-Learning and Discrete Q-Learning for reinforcement learning respectively [101] with good improvement in efficiency.

Umair and Bernard (2012) proposed a novel, model-free RL (reinforcement learning) Technique for the power management of a portable traffic monitoring system having the computer hardware which is the major contributor to the entire power consumption. Unlike the previous works they have proposed to use Timeout policy for RL in sleep as well as idle state [102]. They used MLANN (Multi-layer artificial neural network) for the workload estimation as shown in Fig. 3. In addition to this they used multiple state update in idle as well as sleep modes to increase the convergence speed of the algorithm. Their work proves that using Timeout policy in idle as well as sleep state is more efficient than using Timeout in idle state and N-policy in sleep state.

Although the DPM techniques effectively reduce the power consumption, they do not provide an optimal policy to extend the battery service lifetime of the battery. Maryam *et.al* (2013) proposed a power management policy claiming to extend the battery service lifetime by 35% compared to previous methods [103] as shown in Fig. 4. They have presented a model-free reinforcement learning technique used to define the optimal battery threshold value for a closed loop policy and used the same to specify the system active mode. Their power manager automatically adjusts the power management policy by learning the optimal timeout value. It performs power management in an "event-driven" and "continuous-time" manner. Their algorithm has a fast convergence rate and has less reliance on the Markovian property.

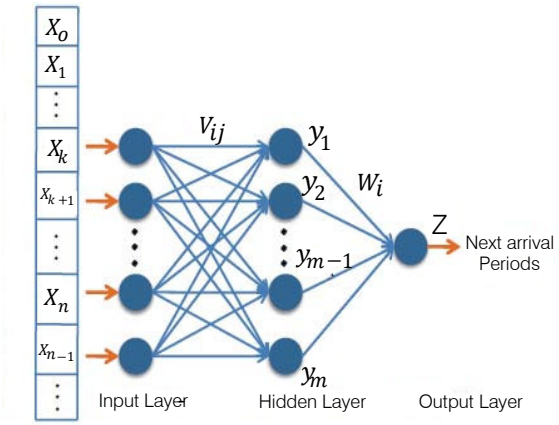

Fig. 3: ML-ANN based Workload Estimator

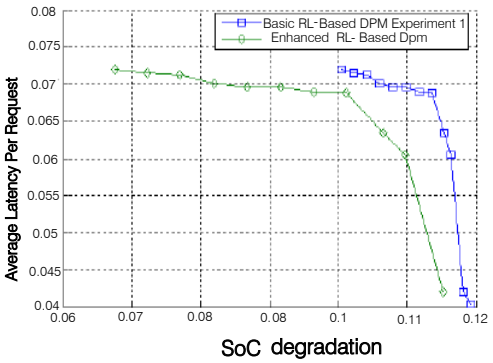

Fig. 4: Model-free Reinforcement Learing based Energy saving

M. Triki *et.al* (2015) proposed a novel, online, as well as adaptive RL based hierarchical approach to directly schedule the service request traffic that reaches the power managed components through SFC [104], using the technique is robust and has a faster convergence rate, the authors performed continuous time and event driven power management using the same. They were able to achieve a maximum energy saving of almost 63% during testing.

Based on the literature survey it is seen that a lot of work has been done in DPM for portable systems. Various low power design techniques have been used at circuit level to manage power consumption in IMDs in [18][20][27]. However no or very less work has been done in Power Management in IMDs at architectural level. Hence, there is a scope to work in this area.

c) Process Scheduling Techniques

Process scheduling involves allocating the tasks (ready for execution) to the available hardware resources. As the available hardware resources are often less in number than the tasks, tasks compete for it and the winner is scheduled for execution. Optimal task scheduling algorithm is a one that always keeps the available hardware resources occupied with tasks. The basic goal of any scheduling algorithm is to maximize the processor utilization. If the processor utilization is equal to or less than 1, then the schedule is said to be feasible.

The complexity of the scheduling algorithm increases when many tasks are to be scheduled on a large number of processing elements. In such systems, complexity of the scheduling algorithm decides the overall system performance.

Scheduling the tasks on more than one processor is a NP-complete problem and no optimal solution exists for such a system. Therefore, heuristics are applied.

i. *Terminology used in Scheduling*

a. *Scheduler*

Scheduler is a module that schedules tasks using some scheduling algorithms and resource access-control protocols.

b. *Schedule*

By schedule it means assignment of the jobs to the available processors as per the guidelines from the scheduler.

c. *Feasible Schedule*

A feasible schedule is a one that schedules the set of tasks meeting their deadlines. The feasible schedule is represented by timed labeled transition system.

d. *Optimal Scheduling or Scheduler*

A scheduling algorithm or scheduler (static or dynamic) is said to be optimal if it always constructs a feasible schedule for every task that has feasible schedule.

A static scheduling algorithm is said to be optimal if, for any set of tasks, it always produces the feasible schedule whenever any other algorithm can do so.

A dynamic scheduling algorithm is said to be optimal if it always produces a feasible schedule whenever a static scheduling algorithm with complete prior knowledge of all the possible tasks can do so.

An aperiodic scheduling algorithm is optimal if it minimizes wither the response time of the aperiodic job or the average response time of all the aperiodic jobs for a given task set.

An algorithm for scheduling sporadic jobs is optimal if it accepts each sporadic job newly offered to the system and schedules the job to complete in time if and only if the new job can be correctly scheduled.

e. *Static Scheduling Algorithm*

A scheduling algorithm is said to be static if priorities are assigned to tasks once and for all. A static priority algorithm is said to be fixed-priority scheduling algorithm also.

f. *Dynamic Scheduling Algorithm*

A scheduling algorithm is said to be dynamic if priorities of tasks might change from request to request.

g. *Mixed Scheduling Algorithm*

A scheduling algorithm is said to be mixed scheduling algorithm if the priorities of some of the tasks are fixed yet the priorities of the remaining tasks vary from request to request.

ii. *Definition of Scheduling Problem*

Task Scheduling involves determining the schedule, for a set of given tasks, such that the timing constraints, precedence constraints and resource requirements for the tasks are met and to compute the schedule if it is found to exist.

Real-time system has a mix of periodic and non-periodic (aperiodic and sporadic) tasks. Out of which periodic and sporadic tasks have hard deadlines to follow while aperiodic tasks have soft deadlines. The basic aim of any scheduling algorithm or scheme is to model these task characteristics with various changing parameters. Therefore, scheduling scheme should provide following things:

1. Assumptions made for the tasks.
2. Scheduling of non-periodic tasks that include soft aperiodic and hard sporadic tasks.
3. Schedulability test and analysis.
4. Performance analysis.

a. *Schedulability Analysis*

Its required to analyze schedulability to determine whether a set of tasks meets its timing constraints.

One way to analyze schedulability is to compute the worst case response time (WCRT) of each task as proposed in Balarin, L. Lavagno, Murthy and Vincentelli [2]. A task's WCRT is the maximum possible length of an interval that begins with the task being enabled and ends with the task completing its execution. It includes both the task's runtime and interference from other tasks. The WCRT concept is useful regardless of the scheduling approach.

However, finding WCRT for a real life embedded system is a difficult task due to the presence of varying parameters like runtimes, dependency between tasks, and non- periodic events in the environment.

b. *Performance Analysis of Scheduling Algorithms*

Performance analysis of scheduling algorithm is required to find out its effectiveness in scheduling the set of tasks. The most often used measure of the performance is the ability of the scheduling algorithm to find out the feasible schedule for a set of tasks provided such a schedule exists. Schedulable utilization and fast response time to urgent tasks are also used as main performance measures. Other commonly used performance measures include maximum and average tardiness, lateness, and response time and the miss, loss, and invalid rates. Generally, only the relevant performance measures are used in the performance analysis of a particular scheduling algorithm. This depends on the task characteristics and the environment.

d) Approaches Taken to Real-Time Scheduling

The approaches taken to real-time scheduling can be broadly classified into three categories: clock-driven scheduling, round robin scheduling and priority-driven scheduling. Priority driven scheduling can be further classified into fixed and dynamic priority scheduling. The scheduling scheme may support either preemptive or non-preemptive scheduling etc. The scheduling algorithms found in the literature target the topic of scheduling the hybrid of real-time

periodic and non-periodic (aperiodic and sporadic) tasks with hard or soft deadlines respectively. In literature the work of scheduling covers specific cases of uniprocessor, multiprocessor and distributed systems (with identical or heterogeneous processors). Each scheduling algorithm assumes certain task characteristics. Some assumptions are often made for the real-time task [9] that may include:

The real time tasks with hard deadlines are periodic.

The tasks are independent i.e. the tasks release time does not depend on the initiation or completion of other tasks.

Run-time for each task remain constant; run-time here means the time taken by the processor to execute the task.

Any non-periodic (aperiodic and sporadic) tasks are special cases; they are initialization and failure-recovery routines; and do not have hard deadlines. All parameters of the periodic jobs are known a priori. In particular, variations in the inter-release times in any periodic job are negligibly small.

Different scheduling algorithms try to relax one or more of the above assumptions so as to make the task model more realistic. The way the aperiodic and sporadic tasks are scheduled distinguishes various scheduling schemes.

i. *Static and Dynamic Task Scheduling*

Task scheduling in hard real-time system can be either static or dynamic. In static task scheduling, the schedule for the tasks is prepared offline and requires complete prior knowledge of the task characteristics. In dynamic task scheduling, on the other hand, tasks are accepted for scheduling during run-time (if a feasible schedule is obtained). If the tasks' characteristics are well known and doesn't vary, static scheduling schemes always produce feasible schedule. We can use complex static scheduling scheme, as schedule is computed offline. However, static scheduling schemes are inflexible and cannot adapt to changing environment. The schedule needs to be recomputed if the system is reconfigured. In contrast, dynamic schemes have high run-time cost as the schedule is found on the fly. However, they are flexible and can easily adapt to the changes in the environment.

ii. *Preemptive vs. Non-preemptive Scheduling*

Most of the scheduling algorithms assume that the tasks are preemptive. However, non-preemptive scheduling of a set of periodic and sporadic tasks on a uniprocessor is important for variety of reasons such as:

In many practical real-time scheduling problems such as I/O scheduling, properties of device hardware and software either make preemption impossible or prohibitively expensive.

Non-preemptive scheduling algorithms are easier to implement than preemptive algorithms, and can exhibit dramatically lower overhead at runtime.

The overhead of preemptive algorithms is more difficult to characterize and predict than that of non-preemptive algorithms. Since scheduling overhead is often ignored in scheduling models, an implementation f a non-preemptive scheduler will be closer to the formal model than an implementation of a preemptive scheduler.

Non-preemptive scheduling on a uniprocessor naturally guarantees exclusive access to shared resources and data, thus eliminating both the needs for synchronization and its associated overhead.

The problem of scheduling all tasks without preemption forms the theoretical basis for more general tasking models that include shared resources.

Jeffay et al. [17] focus on scheduling a set of periodic or sporadic tasks on a uniprocessor without preemption and without inserted idle time. The paper gives necessary and sufficient set of conditions C for a set of periodic or sporadic tasks to be schedulable for arbitrary release time of the tasks. They have shown that a set of periodic or sporadic tasks that satisfy C can be scheduled with an earlier-deadline-first (EDF) scheduling algorithm. For a set of sporadic tasks with specified release times conditions C are necessary and sufficient for schedulability. However, for sets of periodic tasks with specified release times, conditions C are sufficient but not necessary.

iii. *Clock-driven Scheduling*

In clock-driven scheduling, the jobs are scheduled by the scheduler at specific time instants. These time instants are chosen a priori before the system starts execution. The timing instants may or may not be at regular intervals. All the parameters of hard real-time jobs should be fixed and known before hand. In other words, the clock driven scheduling is possible for a system that is by and large deterministic.

To keep the information ready for the scheduler, the schedule for the jobs is computed off-line and is stored in the form of a table for use at run-time. Each entry in this table gives time instant at which a scheduling decision is made. Scheduler makes use of a timer. Upon receiving a timer interrupt, the scheduler sets the timer to expire at the next decision instant (from the table entry). When the timer expires again, scheduler repeats this operation.

iv. *Weighted Round Robin Scheduling*

The round robin approach is commonly used for scheduling time-shared applications. When jobs are scheduled on a round robin basis, every job joins a

First-in-first-out (FIFO) queue when it becomes ready for execution. The job at the head of the queue executes for at most one time slice. If the job does not complete by the end of the time slice, it is preempted and placed at the end of the queue to wait for its next turn. When there are n ready jobs in the queue, each job gets one time slice every n time slices, that is every round. In essence, each job gets 1/nth share of the processor when there are n jobs ready for execution. The problem with round robin scheduling is that it provides poor service to urgent tasks. It is possible that even the most urgent task needs to wait for all other tasks to execute before it gets its turn. Thus to satisfy the timing constraints a very fast processing unit may be necessary, which may not be available. Then round robin may not produce the feasible schedule.

Therefore, weighted round robin scheduling scheme is used. It builds basic round robin scheme. Rather than giving all the ready jobs equal shares of the processor, different jobs may be given different weights. Here, the weight of a job refers to the fraction of processor time allocated to the job. By adjusting the weight of the jobs, we can speed up or retard the progress of each job toward its completion.

If round robin scheme is used to schedule precedence constrained jobs; the response time of a chain of jobs can be unduly large. For this reason, the weighted round robin approach is not suitable for scheduling such jobs. On the other hand, a successor job may be able to incrementally consume what a predecessor produces. In this case, weighted round robin scheduling is a reasonable approach, since a job and its successors can execute concurrently in a pipelined fashion.

v. *Priority Driven Scheduling*

The term priority-driven algorithms refer to a large class of scheduling algorithms that never leave any processor idle intentionally. Priority driven algorithms assign priorities to the tasks either statically or dynamically. Scheduling decisions are taken when events such as releases and completions of jobs occur and hence priority-driven algorithms are also known as event-driven. As any scheduling decision time, the jobs with the highest priority are scheduled and executed on the available processors.

Compared with the clock-driven approach, the priority-driven scheduling approach has many advantages. Many well-known priority-driven algorithms use very simple priority assignments, and for these algorithms, the run-time overhead due to maintaining a priority queue of ready jobs can be made very small. A clock-driven scheduler requires the information on the release times and execution times of the jobs a priori in order to decide when to schedule them. In contrast, a priority-driven scheduler does not require most of this information, making it much better suited for applications with varying time and resource requirements.

Despite its merits, the priority-driven approach has not been widely used in hard real- time systems, especially safety-critical systems, until recently. The major reason is that the timing behavior of a priority-driven system is non-deterministic when job parameters vary. Consequently, it is difficult to validate that the deadlines of all jobs scheduled in a priority-driven manner indeed meet their deadlines when the job parameters vary.

vi. *Static or Fixed Priority Scheduling Algorithms*

One way of building hard real-time systems is from a number of periodic and sporadic tasks and a common way of scheduling such tasks is by using a static priority pre- emptive scheduler; at runtime the highest priority runnable job is executed. Rate-Monotonic scheduling scheme proposed by Liu and Layland [9] and Deadline- Monotonic scheme proposed by Leung [62] are used to assign static priorities to the real-time jobs. In this section, both these scheduling schemes are explained and how they are used to schedule periodic and non-periodic jobs is covered.

a. *Rate Monotonic Priority Assignment*

Liu and Layland [9] in 1973 proposed a fixed priority scheduling scheme known as Rate Monotonic Scheduling. In rate monotonic priority assignment, priorities are assigned to tasks according to their request rates, independent of their runtimes. Specifically, tasks with higher request rates will have higher priorities. They also derived a schedulability analysis that determines if a given task set will always meet all deadlines under all possible release conditions. However, original rate monotonic scheme had several restrictions:

All tasks are independent to each other and they cannot interact.
All tasks are periodic.
No task can block waiting for an external event.
All tasks share a common release time (critical instant).
All tasks have a deadline equal to their period.

Liu & Layland's work has had a wide impact on research in real-time computing and embedded systems. However, every assumption of their model is violated to some extent in the design of embedded systems.

Tasks are rarely independent and generally events in the environment or execution of other tasks invoke them. In many systems, request for tasks do not arrive at regular periods. Only some constraints on the request rate are known. In many low-cost

embedded systems preemption cost is not affordable due to context switch overhead. In addition, tasks' runtime is almost never constant. It may vary with different input patterns as well as with the state of the task.

Because of all the above real life issues, research community has come up with more realistic models in which some of the assumptions of Liu and Layland have been relaxed.

The first assumption that tasks cannot interact has been removed by Sha et al. [31]. Sha also provided a test to incorporate processes that synchronize using semaphores in [47]. Sha [31] addresses the issue of priority inversion (if synchronization primitives like semaphores, monitors and ada task model [47] are directly applied). Two priority inheritance protocols called the basic priority inheritance protocol and Priority Ceiling Protocol (PCP) have been presented. This protocols also shown to avoid deadlocks. Baker [15] proposes a Stack Resource Policy (SRP) which is a resource allocation policy that permits processes with different priorities to share a single runtime stack. SRP is a refinement of PCP [31], which strictly bound priority inversion and permits simple schedulability analysis. The related work on this topic can also be found in [12, 48, 49].

Sha [61] reported work that includes test to allow aperiodic processes to be included in the theory.

Rajkumar [58] used external blocking (i.e. when a task is blocked awaiting an external event) with the Rate Monotonic approach to model the operation of a multiprocessor priority ceiling protocol [12] and provided schedulability analysis to bound its effects.

The restriction that tasks are assumed to share a common critical instant has been relaxed by Audsley [57].

Leung [62] suggested a Deadline-Monotonic priority assignment that removed the constraint that the deadline and period of a process must be equal. Audsley et al. [7] provided schedulability test for the scheme proposed by Leung.

b. *Deadline Monotonic Priority Assignment*

In deadline-monotonic scheduling theory, processes to be scheduled are characterized by the following relation:

Computation time <= deadline <= period

Deadline monotonic priority assignment is similar in concept to rate-monotonic priority assignment. Priorities assigned to processes are inversely proportional to the length of the deadline [62]. Thus, the process with the shortest deadline is assigned the highest priority and the longest deadline process is assigned to lowest priority. This priority assignment defaults to a rate-monotonic assignment when period = deadline.

Deadline monotonic priority assignment is shown to be optimal static priority scheme [62]. The implication of this is that if any static priority scheduling algorithm can schedule a process set where process deadlines are unequal to their periods an algorithm using deadline-monotonic priority ordering for processes will also schedule that process set.

Audsley et al. [7] also showed that since deadline-monotonic scheme guarantees that computation time is less than or equal to deadline, it is possible to schedule sporadic tasks within the existing periodic framework. They also discussed problems involved for guaranteeing deadlines of sporadic processes using sporadic servers within the rate-monotonic scheduling framework.

c. *Related Work*

Lehoczky [14] considers the problem of fixed priority scheduling of periodic tasks *with arbitrary deadlines* and an exact schedulability criterion has been developed. A worst case bound for the case of rate-monotonic scheduling is developed generalizing the original bounds of Liu and Layland in that the tasks are allowed to have deadlines $D = \Delta T$ for any $\Delta > 0$. The bounds show that when one additional period ($\Delta = 2$) is given to tasks to complete their computation requirement, the worst case schedulable utilization increases from 0.693 to 0.811. Also, average schedulable utilization is shown to have increased from 0.88 to over 0.95 that often goes to 1.00.

Audsley et al. [20] have given exact schedulability analysis for real-time systems scheduled at runtime with static priority preemptive scheme. Exact analysis of sporadic tasks is given and analysis extended to include release jitter. Schedulability analysis to predict worst case response times for a set of periodic and sporadic tasks under any given priority assignment and scheduled by a static priority preemptive scheduler can be found in [20].

Lehoczky et al [63] provides an exact characterization and stochastic analysis for a randomly generated set of periodic tasks scheduled by rate-monotonic algorithm.

Shih et al. [65] presents modified rate-monotonic algorithm for scheduling periodic jobs with deferred deadlines. The deadline of the request in any period of a job with deferred deadline is some time instant after the end of the period. The paper describes a semi-static priority-driven algorithm for scheduling periodic jobs with deferred deadlines: each job is assigned two priorities, the higher one of the old request and the lower one for the current request. The optimal schedulability analysis and the applications where the algorithm will be useful are also discussed.

Predictive periodic and non-periodic algorithms are given by Singh [64]. A predictive preemptive scheduling algorithm avoids unnecessary preemption while a non-preemptive algorithm is predictive in a sense that it looks for future task arrival times and schedules them non-preemptively.

Recent work on scheduling has focussed on scheduling of flexible applications (or imprecise computation). The work in [28,30,38-46,54] provides sufficient material for the interest reader.

d. *Scheduling Non-Periodic Tasks in Fixed Priority Real-Time Systems*

Till now, the focus was only on the scheduling of periodic tasks. In practice, real- time systems comprise of a hybrid of hard periodic jobs and soft/hard aperiodic jobs. The mixed scheduling problem is important, because many real-time systems have substantial aperiodic task workloads.

Aperiodic job and sporadic job scheduling algorithms are solutions to the following problems:

1. Sporadic job scheduler decides whether to accept or reject the newly arrived sporadic job depending on its execution time and the deadline. If it accepts a job, it schedules a job such that all other hard deadline periodic tasks and previously accepted sporadic tasks meet their deadlines. Here the problem lies in determining how to do acceptance test and how to schedule accepte$_{d\ sp}$orad$_{ic\ j}$obs.
2. Aperiodic job scheduler tries to complete each aperiodic job as early as possible. The problem with this scheduler is to do so without causing other hard periodic and sporadic tasks to miss their deadline. Obviously, average response time is a measure of performance of these schedulers.

Within the framework of fixed priority preemptive scheduling, a number of approaches have been developed for scheduling mixed task sets. The simplest and perhaps least effective of these is background scheduling of aperiodic tasks. In background scheduling, soft deadline tasks are executed at a lower priority than any hard deadlines tasks. Clearly, this method always produces correct schedules and is simple to implement. However, the execution of aperiodic jobs may be delayed and their response times prolonged unnecessarily. An obvious way to make the response times of aperiodic jobs as short as possible is to make their execution interrupt driven.

Whenever an aperiodic job arrives, the execution of periodic tasks is interrupted, and the aperiodic job is executed. However, if aperiodic tasks always execute as fast as possible, periodic tasks may miss some deadlines. Another approach for scheduling aperiodic tasks is to use a periodic task that looks for the ready aperiodic tasks in an aperiodic task queue. Such a periodic task is called as polling server. A polling server has a fixed priority level (usually the highest) and an execution capacity. The capacity of the server is calculated off-line and is normally set to the maximum possible, such that the hard task set, including server, is schedulable. At run-time, the polling server is released periodically and its capacity is used to service soft real-time tasks. Once this capacity has been exhausted, execution is suspended until it can be replenished at the server's next release. The polling server will usually significantly improve the response times of soft tasks over background processing. However, if the ready soft tasks exceed the capacity of the server, then some of them will have to wait until its next release, leading to potentially long response times. Conversely, no soft tasks may be ready when the server is released, wasting its high priority capacity.

This drawback is avoided by the Priority Exchange, Deferrable server [60, 67, 68] and Sporadic servers [61,68] algorithms. These are all based on similar principles to the polling server. However, they are able to preserve capacity if no soft tasks are pending when they are released. Due to this property, they are termed as "bandwidth preserving algorithms". These three algorithms differ in the ways in which the capacity of the server is preserved and replenished and in the schedulability analysis needed to determine their maximum capacity.

In general, all three offer improved responsiveness over the polling approach. However, there are still disadvantages with these more complex server algorithms. They are unable to make use of slack time that may be present due to the often favorable phasing of periodic tasks. Further, they tend to degrade to providing essentially the same performance as the polling server at high loads. The deferrable and sporadic servers are also unable to reclaim spare capacity gained, when for example, hard tasks require less than their worst case execution time. This spare capacity termed gain time, can however be reclaimed by the extended priority exchange algorithm [69].

Chetto [66] and Lehoczky [18] proposed the slack stealing algorithm. This algorithm uses the strategy to make use of the available slack times of periodic and sporadic jobs to complete aperiodic jobs. The slack stealing algorithm suffers from none of the above disadvantages. It is optimal in the sense that it minimizes the response times of soft aperiodic tasks amongst all algorithms that meet all hard periodic task deadlines. The slack stealer services aperiodic

requests by making any spare processing time available as soon as possible. In doing so, it effectively steals slack from the hard deadline periodic tasks.

In [22], Davis et al. presents new analysis that allows the slack available on hard deadline periodic and hard deadline sporadic tasks to be calculated. The analysis caters for tasks that have release time jitter, synchronization, stochastic execution times and arbitrary deadlines. Further extension to the basic slack stealing work can be found in [21,25].

vii. *Dynamic Priority Scheduling Algorithms: EDF, LST*

Now the turn comes to the study of dynamic scheduling algorithms that we call the deadline driven scheduling algorithm. As said earlier, processor utilization increases by use of the dynamic scheduling schemes. In this section, the dynamic priority assignment scheduling schemes used in the literature is studied.

a. *Earlier- Deadline- First (EDF) Scheduling Algorithm*

Liu and Layland [9], proposed an Earlier-Deadline-First EDF scheduling scheme. Using this algorithm, priorities are assigned to tasks according to the deadlines of their current requests. Specifically, a task will be assigned the highest priority if the deadline of its current request is the nearest, and will be assigned the lowest priority if the deadline of its current request is the furthest. Such a method of assigning priorities to the tasks is a dynamic one, in contrast to a static assignment in which priorities of tasks do not change with time. Schedulability analysis to determine whether a given task set can be scheduled by EDF is given in [9]. An EDF algorithm is optimal for scheduling preemptive jobs on one processor. However, it is non- optimal when jobs are non-preemptive or when there is more than one processor [96].

b. *Least- Slack- Time- First (LST) Scheduling Algorithm*

Another well-known dynamic-priority algorithm is the Least-Slack-Time-First (LST) [48] algorithm. At time t, slack of a job whose remaining execution time is x and whose deadline is d is equal to d - t - x. The LST scheduling algorithm checks the slacks of all the ready jobs each time a new job is released and orders the new job and the existing jobs on the basis of their slacks: the smaller the slack, the higher the priority. Like EDF, LST algorithm is also optimal for scheduling preemptive periodic jobs [95] on one processor but non-optimal for scheduling non-preemptive jobs or multiprocessor scheduling.

c. *Scheduling Non-Periodic Tasks in Dynamic Priority Systems*

As dynamic priority-driven scheduling schemes makes a better processor utilization, many approaches have been reported in the literature that cover the problem of scheduling the soft / hard aperiodic jobs in the dynamic priority-driven framework. Chetto and Chetto [66] studied the localization and duration of idle times and proposed an algorithm for scheduling hard aperiodic tasks. Chetto's algorithm requires that the periodic task deadlines be equal to their periods, and assumes that when any hard aperiodic task arrives and is required to run, all the aperiodic tasks previously accepted have completed their execution. Schwan and Zhou [70] relax the above assumptions and propose a joint algorithm in which every task, whether periodic or aperiodic, is subject to an acceptance test upon arrival.

Work has been carried out for dynamic priority versions of deferrable server, sporadic servers and other bandwidth preserving algorithms, as is found in the fixed priority schemes. Three server mechanisms under EDF have been proposed by Ghazalie and Baker [68]. The authors describe a dynamic version of the known Deferrable and Sporadic servers [61], called Deadline Deferrable server and Deadline Sporadic Server respectively. Then, the later is extended to obtain a simpler algorithm called Deadline Exchange Server. Later, Spuri and Buttazzo [72,73], presented five new online algorithms for servicing soft aperiodic tasks scheduled using EDF. They presented following algorithms:

1. Dynamic Priority Exchange, an extension of previous work under RM.
2. A new bandwidth-preserving algorithm called as Total Bandwidth Server.
3. Earliest-Deadline-Last (EDL) Server.
4. Improved Priority Exchange with less runtime overhead and
5. Dynamic Sporadic Server (DSS) Algorithm.

Spuri et al in [29], extended the Total Bandwidth Sever algorithm to handle hard aperiodic tasks and to deal with overload situations. Total Bandwidth approach was further expanded toward optimality by Buttazzo and Sensini [51,74]. They provided a general method for assigning deadlines to soft aperiodic requests.

Homayoun et al [56] combine the EDF algorithm for scheduling periodic tasks with the deferrable server for servicing aperiodic tasks. An online algorithm for scheduling sporadic tasks with shared resources in hard real-time systems has been presented in [75]. Jeffay [75] describes a method, the Dynamic Deadline Modification (DDM) protocol, for scheduling sporadic tasks with shared

resources under the Earliest Deadline First (EDF) scheduling algorithm. Baker [15] proposed a general resource access protocol, the Stack Resource Policy (SRP), which can be used under fixed as well as dynamic priority assignments. Group Priority Earliest Deadline First (GPEDF) performs schedulability test prior to grouping a particular job. In the GPEDF, jobs with short execution time are executed first in the group, which leaves more time for other jobs to execute. This allows more jobs to be completed, the response is reduced. [96].

In [71], Caccomo et al extended the analysis to deal with dynamic deadline modifications, in order to use the tunable Total Bandwidth server [51,74], for improving aperiodic responsiveness in the presence of resource constraints.

Kim et al [76-78], discuss two scheduling algorithms known as Alternative Priority Scheduling (APS) and Critical Task Indication (CTI) algorithms.

Buttazzo [50] proposes a variant of earliest deadline first scheduling algorithm which exploits skips to minimize the response time of aperiodic requests in a firm real-time system.

viii. *Scheduling in Multiprocessor Systems*

a. *Introduction*

Thus far we have seen about the scheduling algorithms without considering the case where the real-time system has more than one processor. A multiprocessor system is classified into the shared-memory and distributed-memory systems. A shared-memory multiprocessor model is a centralized system as the processors are located at a single point in the system and the inter-processor communication cost is negligible compared to the processor execution cost. The distributed-memory multiprocessor model, also known as distributed system, is one in which the processors are distributed at different points in the system and the inter-processor communication cost is not negligible compared to the processor execution cost. A local area network is an example of such system.

Scheduling scheme for multiprocessor systems has to provide solutions for the problems that arise in the multiprocessor environments. Firstly, task assignment is an important problem in multiprocessor systems. Most hard real-time systems built to date are static, that is jobs or tasks are partitioned and statically bound to processors. The task assignment problem is concerned with how to partition the system of tasks and passive resources into modules and how to assign the modules to processors. Second problem is the inter-processor synchronization. Some kind of synchronization protocol is needed to ensure that precedence constraints of jobs on different processors are always satisfied. Finally, in a distributed real-time system, tasks may arrive unevenly at the nodes (processors) in the system and / or processing power may vary from node to node, thus getting some nodes temporarily overloaded while leaving others idle or under-loaded. Many load sharing (LS) algorithms have been proposed in the literature to counter this problem.

Scheduling schemes for multiprocessor system has to take into account the following important factors: memory and resource utilization, deadlock avoidance, precedence constraints, and communication delay. Because of all these complicating factors, the development of appropriate scheduling schemes for multiprocessor real-time systems is problematic, it is known that optimal scheduling for multiprocessor systems is NP hard. It is therefore necessary to look for ways of simplifying the problem and algorithms that give adequate sub-optimal results.

b. *Scheduling Problem Definition for Multiprocessor Systems*

The problem of multiprocessor scheduling is to determine when and on which processor a given task executes. This can be done either statically or dynamically. In static algorithms, the assignment of tasks to processors and the time at which the tasks start execution are determined a priori. Static algorithms [19], [37] are often used to schedule periodic tasks with hard deadlines. The main advantage is that, if a solution is found, then one can be sure that all deadlines will be guaranteed. However, this approach is not applicable to aperiodic tasks whose characteristics are not known a priori. Scheduling such tasks in a multiprocessor real-time system requires dynamic scheduling algorithms. In dynamic scheduling [4], [53], when new tasks arrive, the scheduler dynamically determines the feasibility of scheduling these new tasks without jeopardizing the guarantees that have been provided for the previously scheduled tasks. Thus, for predictable executions, schedulability analysis must be done before a task's execution is begun.

Dynamic scheduling algorithms can be either distributed or centralized. In a distributed dynamic scheduling scheme, tasks arrive independently at each processor. When a task arrives at a processor, the local scheduler at the processor determines whether or not it can satisfy the constraints of the incoming task. The task is accepted if they can be satisfied, otherwise, the local scheduler tries to find another processor which can accept the task. In a centralized scheme, all the tasks arrive at a central processor called the *scheduler*, from where they are distributed to other processors in the system for execution.

c. *Inter-Processor Synchronization Protocols*

Synchronization protocol is a protocol that governs when the schedulers on different processors release the jobs of sibling subtasks. A synchronization protocol is said to be correct if it (1) never releases jobs in any first subtask before the end-to-end release times of the jobs and (2) never allows the violation of any precedence constraint among sibling subtasks. Four types of synchronization protocols are reported in the literature. Those are Greedy Synchronization Protocol, Phase Modification (PM) Protocol, Modified Phase-Modification (MPM) Protocol and the Release-Guard (RG) Protocol [8,80]. Rajkumar et al [12] extend the priority inheritance protocol for uniprocessors [31] to multiprocessors.

d. *Load Sharing Algorithms*

In load sharing scheme, if a node cannot guarantee a task or some of its existing guarantees are to be violated as a result of inserting a task into its schedule, it has to determine candidate receiving processor(s) for the task(s) to be transferred. Two issues need to be considered when choosing a receiving processor(s).

1. Minimization of the probability of transferring a task to an incapable node.
2. Avoidance of task collisions and / or excessive task transfers, and minimization of the possibility of a task's guarantee being violated due to future tighter-laxity task arrivals.

Most of the work concentrates on *1* and chooses the most desirable receiving processor based on the state information collected from periodic/aperiodic state broadcasts [87,88, 98] or state probing/bidding [89]. Moreover, implied in this work is the assumption of homogeneous workload distribution among nodes. This assumption does not always hold, because the distribution that governs task arrivals at different nodes may vary greatly over time and thus the workload distribution is not homogeneous among the nodes. Therefore both 1 and 2 above should be considered in guaranteeing tasks on a heterogeneous system.

Hou and Shin [81] propose a load-sharing algorithm for real-time applications, which takes into account the future task arrivals.

e. *Fault Tolerant Scheduling*

In many real-time systems, a fault tolerance is an important issue. A system is fault tolerant if it produces correct results even in the presence of faults. When a fault occurs, extra time is required during task execution to handle fault detection and recovery. For real-time systems in particular, it is essential that the extra time be considered and accounted for prior to execution. Methods explicitly developed for fault tolerance in real-time systems must take into consideration the number and type of faults, and ensure that the timing constraints are not violated.

In a multiprocessor system fault tolerance can be provided by scheduling multiple copies of tasks on different processors [81,82] and the high-performance computation power from multiple cores on the platforms [99]. A primary / backup (PB) approach and triple modular redundancy (TMR) approach are two basic approaches that allow multiple copies of task to be scheduled on different processors [83]. One or more of these copies can be run to ensure that the task completes before its deadline. In TMR, multiple copies are usually run to achieve error checking by comparing results after completion. In PB approach, if correct results are generated from the primary task, the backup task is activated. Ghost et al [84] study techniques for providing fault tolerance for non-preemptive, aperiodic, dynamic real-time tasks using the PB approach. Maode et al [85] proposed a strategy called as task reassignment fault tolerance (TRFT) scheduling scheme. The basic idea in [85] is that when a fault appears in the system, it means that a node has no capability to handle tasks and it can not accept other tasks any more. The tasks that have been assigned to it not successfully done should be reassigned to other node which is ready to accept new batch of tasks. Liberto et al [86], focus on global scheduling where tasks can migrate across processors. Two varieties of global multiprocessor scheduling schemes, frame- based scheduling and periodic scheduling, are discussed.

In the frame-based scheduling model, an aperiodic task set is scheduled to create a template (frame), and that schedule may be executed periodically. In the periodic model, each task in the set has a separate period, and is executed with no explicitly predetermined schedule.

f. *Related Work*

Tasks can be statically bounded to a processor i.e. once tasks are allocated to processors; each processor runs the same set of tasks. Each task thus runs on its host processor. Dhall and Liu [79] have shown that the rate monotonic algorithm, which performs well on uniprocessors, behaves poorly for multiprocessor with dynamic binding. They considered the problem of assigning a set of independent periodic tasks to a minimal number of processors. They proposed two heuristic algorithms, called the Rate-monotonic-First-Fit (RMFF) and Rate-Monotonic-Next-Fit (RMNF) algorithms respectively. They showed that in the worst-case, the assignment produced by the RMFF algorithm uses no more than 2.33 times the optimal number of processors, while RMNF uses no more than 2.67

times. Davari and Dhall [90] considered another variation of the heuristic, called First-Fit-Decreasing-Utilization-Factor (FFDUF) algorithm, which improves the worst-case performance to 2 times the optimal number of processors. Davari and Dhall then devised an on-line algorithm, called Next-Fit-M algorithm [91] which has a worst-case performance ratio of 2.2838.

Baruah et al [92,93] devised new dynamic-priority schemes that result in optimal multiprocessor schedulers for hard real-time periodic tasks. Authors [92] proved that any task set whose combined weights is at most m can be scheduled in a pfair manner on m processors, and presented a scheduling algorithm that would achieve this. In [93], they provided a more efficient algorithm.

Kwon et al. proposed an optimal algorithm for parallelizing and scheduling a task set with multiple parallelization options on multiple processor systems [10]. The algorithm presented in [10] is a global strategy while our proposed algorithm is a partitioning strategy.

IV. Summary and Conclusion

Different goals and algorithms characterize process scheduling in real-time operating system. Schedules may or may not exist that satisfy the given timing constraints. In general, the primary goal is to schedule the tasks such that all deadlines are met: in case of success (failure) a secondary goal is maximizing earliness (minimizing tardiness) of task completion. An important issue is predictability of the scheduler, i.e., the level of confidence that the scheduler meets the constraints.

In this section, various scheduling schemes and their schedulability tests have been given. Recent work in process scheduling for multiprocessor and distributed systems is also covered.

The scheduling problem for the design of hardware/software systems is explained in this report. Here it has defined the scheduling in the scenario of embedded systems. Generally speaking, hardware and software scheduling problems differ not just in the formulation but in their overall goals. Nevertheless, some hardware scheduling algorithms are based on techniques used in the software domain, and some recent system-level process scheduling methods have leveraged ideas in hardware sequencing. Scheduling algorithms as applied to design of hardware, compilers, and operating systems were explained in chapters 2, 3 and 4 respectively.

Various process scheduling algorithms have been described. Process Scheduling has to take into account the real-time constraints. Processes are characterized by their timing constraints, periodicity, precedence and data dependency, pre-emptivity, priority etc. The way in which these characteristics affect scheduling decisions has been described.

Broadly, the approaches taken to real-time task scheduling are classified into three categories: clock-driven scheduling, round-robin scheduling and priority-driven scheduling. Priority driven scheduling can be further classified into fixed and dynamic priority scheduling. Also, scheduling schemes are differentiated as preemptive and non-preemptive scheduling scheme. The scheduling algorithms found in the literature target the topic of scheduling the hybrid of real-time periodic and non- periodic (aperiodic and sporadic) tasks with hard or soft deadlines respectively. In literature the work of scheduling covers specific cases of uniprocessor, multiprocessor and distributed systems (with identical or heterogeneous processors).

Clock-driven scheduler schedules the jobs at specific and pre-defined time instants. So, clock-driven scheduling is possible for a system that is by and large deterministic. In round robin scheduling, every process gets its share of the processor (depending on its weight or priority) when there are n jobs ready for execution. Round robin scheduling is very simple to implement but is not suitable for the jobs with precedence constraints. Moreover, it may require a very fast processing unit to satisfy timing constraints. Priority-driven scheduling algorithms are mostly used because they never leave any processor idle intentionally and therefore often results into better processor utilization. Priorities to the tasks can be assigned statically or dynamically. Rate Monotonic (RM) and Deadline Monotonic (DM) scheduling schemes are static priority scheduling schemes and Earlier-Deadline-First (EDF) and Least-Slack-Time- First (LST) are the examples of dynamic priority scheduling schemes.

Scheduling scheme for multiprocessor systems has to provide solutions for the problems that arise in multiprocessor environments. The problems that need to be tackled by the multiprocessor scheduling schemes are: task assignment to a processor, Synchronization protocol, load-balancing etc. Also, scheduling scheme for multiprocessor system has to take into account the following important factors: memory and resource utilization, deadlock avoidance, precedence constraints, and communication delay. Because of these conflicting requirements, development of scheduling scheme for multiprocessor system is difficult.

References Références Referencias

1. G.D. Micheli and R.K. Gupta, Hardware/Software Codesign, Proceedings of the IEEE, vol.85, no.3, pp. 349-365, March, 1997
2. D.D. Gajski and F. Vahid, Specification and Design of Embedded Hardware/Software Systems, IEEE

Design and Test of Computers, vol.11, no.3, pp.44-54, 1994

3. F. Balarin, L. Lavango, P. Murthy, and A.S.-Vincentelli, Scheduling for Embedded Real-Time Systems, IEEE Design and Test of Computers, vol.12, no.1, Jan.-March, 1998
4. K. Ramamritham, J. A. Stankovic, P. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems", COINS Technical Report 89-37, April 13, 1989.
5. O. Plata, F. F. Rivera, "Dynamic Scheduling on Distributed- Memory Multiprocessors", University of Malaga, Technical Report No: UMA-DAC-95/12, June 1995.
6. K. Ramamritham, J. A. Stankovic, "Scheduling algorithms and operating systems support for Real-Time Systems", University of Massachusetts.
7. N. C. Audsley A. Burns M. F. Richardson 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. 127 – 132, 1991.
8. Bettati, R., "End-to-end scheduling to meet deadlines in distributed systems," Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana- Champaign, 1994.
9. Liu and Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real- Time Environment", 1973.
10. Philip J. Christopher, Apostolos Dollas, "Knowledge Based Process Scheduling on Symmetric Multiprocessors", Proceedings of the IEEE, Nov. 1991.
11. Babak Hamidzadeh, Lau Ying Kit, David J. Lilja, "Dynamic Task Scheduling Using Online Optimization", IEEE Transactions on parallel and distributed systems, vol. 11, no. 11, Nov. 2000.
12. Ragunathan Rajkumar, Lui Sha, John P. Lehoczky, "Real-Time Synchronization Protocols for Multiprocessors", Proceedings of the 1988 Real-Time Systems Symposium, 1988.
13. Kwang S. Hong, Joseph Y-T Leung, "On-line Scheduling of Real-Time Tasks", IEEE Transactions on Computers 41, 1998.
14. John P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", Proceedings of the Real-Time Systems Symposium, pp. 201-209, 1990.
15. T. P. Baker, "A Stack-Based Resource Allocation Policy for Real-Time Processes", Proceedings of IEEE Real-Time Systems Symposium, 1990.
16. Chao-Ju Hou, Kang G. Shin, "Load Sharing with Consideration of Future Task Arrivals in Heterogeneous Distributed Real-Time Systems", IEEE Trans. Computers, 43(9): 1076-90, 1991.
17. Kevin Jeffay, Donald F. Stanat, Charles U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", Proceedings of the 12th IEEE Symposium on Real-Time Systems, pp. 129-139,1991.
18. John P. Lehoczky, Sandra Ramos-Thuel, "An Optimal Algorithm for Scheduling soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems", Proceedings 13th Real- Time Systems Symposium, pp. 110-123, 1992.
19. K. Ramamritham, "Allocation and Scheduling of Precedence-Related Periodic Tasks," IEEE Trans. Parallel and Distributed Systems, vol. 6, no. 4, pp. 412-420, Apr. 1995.
20. N. Audsley, A. Burns, M. Richardson, K. Tindell, A.J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling", Software Engineering Journal, 1993.
21. Sandra Ramos-Thuel, John P. Lehoczky, "On-line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems", Proceedings of 14th Real-Time Systems Symposium, pp. 160-171, 1993.
22. R. I. Davis, K. W. Tindell, A. Burns, "Scheduling Slack Time in Fixed Priority Pre-emptive Systems", Proceedings of Real-Time Systems Symposium, 1993.
23. A. Burns, A. J. Wellings, "Dual Priority Assignment: A Practical Method for Increasing Processor Utilization", Real-Time Systems Symposium, 1993.
24. Jun Sun, Riccardo Bettati, Jane W. S. Liu, "An End-to-End Approach to Schedule Tasks with Shared Resources in Multiprocessor Systems", Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software, 1994.
25. Sandra R. Thuel, John P. Lehoczky, "Algorithm for Scheduling Hard Aperiodic Tasks in Fixed-Priority Systems using Slack Stealing", IEEE Real-Time Systems Symposium, pp. 22-33, IEEE Computer Society Press, 1994.
26. Too-Seng Tia, Jane W. S. Liu, "Task and Resource Assignment in Distributed Real-Time Systems", Parallel and Distributed Real-Time Systems, 1994.
27. Robert Davis, Andy Wellings, "Dual Priority Scheduling", IEEE Real-Time Systems Symposium, 1995.
28. Sanjay K. Baruah, "Fairness in periodic real-time scheduling", Proceedings of 16th Real-Time Systems Symposium, 1995.
29. Marco Spuri, Fiorgio Buttazzo, Fabrizio Sensini, "Robust Aperiodic Scheduling under Dynamic Priority Systems", IEEE Real-Time Systems Symposium, 1995.

30. Haken Aydin, Rami Melhem, Daniel Mosse, Pedro Mejia-Alvarez, "Optimal Reward-Based Scheduling of Periodic Real-Time Tasks", IEEE Transactions on Computers, Vol. 50. No. 2, Feb. 2001.
31. Sha L., R. Rajkumar, J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Transactions on Computer, Vol. 39, No. 9, Sep. 1999.
32. Satoshi Fujita, Hui Zhou, "Multiprocessor Scheduling Problem with Probabilistic Execution Costs", IEEE 2000.
33. J.M. Lopez, M. Garcia, J.L. Diaz, D.F. Garcia, "Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems", Proceedings of the 12th Euromicro Conference on Real-Time Systems (EUROMICRO-RTS 2000), 2000.
34. Sanjay K. Baruah, "Scheduling Periodic Tasks on Uniform Multiprocessors", Proceedings of the 12th Euromicro Conference on Real-Time Systems (EUROMICRO- RTS 2000), 2000.
35. Shu-Ling Lee, Chao-Tung Yang, Shian-Shyong Tseng, Chang-Jiun Tsai, "A Cost Effective Scheduling with Load Balancing for Multiprocessor Systems", IEEE 2000.
36. Babak Hamidzadeh, Yacine Atif, "Dynamic Scheduling of Real-Time Tasks, by Assignment", IEEE Concurrency 1998.
37. J. Xu and L. Parnas, "Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations," IEEE Trans. Software Eng., vol. 16, no. 3, pp. 360-369, Mar. 1990.
38. G. Bernat and A. Burns, ™Combining (n, m) Hard Deadlines and Dual Priority Scheduling, Proc. 18th IEEE Real-Time Systems Symp. pp. 46-57, Dec. 1997.
39. A. Bertossi and L.V. Mancini, ™Scheduling Algorithms for Fault-Tolerance in Hard-Real-Time Systems, Real-Time Systems, vol. 7, no. 3, pp. 229-245, 1994.
40. J. -Y. Chung, J.W.-S. Liu, and K. -J. Lin, ™Scheduling Periodic Jobs that Allow Imprecise Results, IEEE Trans. Computers, vol. 19, no. 9, pp. 1156-1173, Sept. 1990.
41. W. Feng and J.W.-S. Liu, ™Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines, IEEE Trans. Software Eng., vol. 23, no. 2, pp. 93-106, Feb. 1997.
42. M. Hamdaoui and P. Ramanathan, ™A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines, IEEE Trans. Computers, vol. 44, no. 12, pp. 1443-1451, Dec. 1995.
43. J.K. Dey, J. Kurose, D. Towsley, C.M. Krishna, and M. Girkar, ™Efficient on- line Processor Scheduling for a Class of IRIS Increasing Reward with Increasing Service) Real-Time Tasks, Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems, pp. 217-228, May 1993.
44. K. -J. Lin, S. Natarajan, and J.W.-S. Liu, ™Imprecise Results: Utilizing Partial Computations in Real-Time Systems, Proc. Eighth IEEE Real-Time Systems Symp. pp. 210-217, Dec. 1987.
45. J.W.-S. Liu, K. -J. Lin, W. -K. Shih, A.C.-S. Yu, C. Chung, J. Yao, and W. Zhao, ™Algorithms for Scheduling Imprecise Computations, Computer, vol. 24, no. 5, pp. 58-68, May 1991.
46. W. -K. Shih, J.W.-S. Liu, and J. -Y. Chung, ™Algorithms for Scheduling Imprecise Computations with Timing Constraints, SIAM J. Computing, vol. 20, no. 3, pp. 537-552, July 1991.
47. Sha L., J. B. Goodenough, "Real-Time Scheduling Theory and Ada", IEEE Computer, 1990.
48. Mok, A. K., "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment", Ph.D. Thesis, MIT, 1983.
49. Chen, M.I., and K. J. Lin, "Dynamic priority ceiling: A concurrency control protocol for real-time systems", Real-Time System Journal, vol 2, no. 4, pp. 325-346, Dec. 1990.
50. Giorgio C. Buttazzo "Minimizing Aperiodic Response Times in a Firm Real-Time Environment", IEEE Transactions On Software Engineering, Vol. 25, No. 1, January/February 1999
51. Giorgio C. Buttazzo, F. Sensini, "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environment", Proceedings of 3rd IEEE Conference on Engineering of Complex Computer Systems (ICECCS'97), pp. 39-48, 1997.
52. M.L. Dertouzos and A.K. Mok, "Multiprocessor On-Line Scheduling of Hard Real-Time Tasks," IEEE Trans. Software Eng., vol. 15, no. 12, pp. 1,497-1,506, Dec. 1989.
53. M. Silly-Chetto, "Dynamic Acceptance of Aperiodic Tasks with Periodic Tasks Under Resource Sharing Constraints", IEEE Proc. On Software Engg. , Vol 146, No. 2, Apr. 1999.
54. Dey, J. K., J. Kurose, and D. Towsley, "On-line scheduling policies for a class of IRIS (Increasing Reward with Increasing Service) real-time tasks," IEEE Transactions on Computers, vol. 45, no.7, July 1996.
55. Gutierrez, J. C. P., J. J. G. Garcia, and M. G. Harbour, "On the schedulability analysis for distributed hard real-time systems," Proceedings of Euromicro Workshop on Real-Time Systems, pp. 136–143, June 1997.
56. Homayoun, N., and P. Ramanathan, "Dynamic priority scheduling of periodic and aperiodic tasks

in hard real-time systems," Real-Time Systems Journal, vol 6, no. 2, pp. 207–232, 1994.
57. Audsley N. C., "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", Report YCS 164, Dept. of Comp. Sci., University of York, Dec. 91.
58. Rajkumar R, "Real-Time Synchronization Protocols for Shared Memory Multiprocessors", IEEE Proc. on Dist. Computing systems, Jun 1990.
59. Joseph M. and Pandya P., "Finding response times in a real-time systems", Computing Journal, 1986.
60. Lehoczky J. P., Sha L., stronides J. K., "Enhancing aperiodic responsiveness in hard real-time environment", IEEE Proc. Real-Time systems Symp. Dec 1987.
61. Sprunt, B, L. Sha, and J. P. Lehoczky, "Aperiodic task scheduling for hard real- time systems," Real-Time Systems Journal, vol 1, no. 1, pp. 27–60, 1989.
62. Leung, J. Y. T., and J. Whitehead, 'On the complexity of fixed-priority scheduling of periodic real-time tasks," Performance Evaluation, vol. 2, pp. 37–250, 1982.
63. Lehoczky J. P., Sha L., Ding Y., "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", Proceedings of the 10th IEEE Symposium on Real-Time Systems, pp. 166-171, 1989.
64. Singh H., "Scheduling Techniques for Real-Time Application Consisting of Periodic Task Sets", IEEE 1994.
65. Shih W. K., Liu J. W. S., Liu C. L., "Modified Rate-Monotonic Algorithm for Scheduling Periodic Jobs with Deferred Deadlines", IEEE Transactions on Software Engineering, Vol 19, No. 12, Dec 1993.
66. Chetto, H., and M. Chetto, "Some results of the earliest deadline scheduling algorithm," IEEE Transactions on Software Engineering, vol. 15, no. 10, pp. 1261–1269, October 1989.
67. Strosnider, Lehoczky, and L.Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," IEEE Transactions on Computers, vol 44, issue 1, Jan 1995.
68. Ghazalie, T. M., and T. P. Baker, "Aperiodic servers in deadline scheduling environment," Real-Time Systems Journal, vol. 9, no. 1, pp. 31–68, 1995.
69. Sprunt B, Lehoczky J. P. and Sha L., "Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm", Proceedings IEEE Real-Time Systems Symposium, Dec 1988.
70. K. Schawan and H. Zhou, "Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads," IEEE Trans. Software Eng., vol. 18, pp. 736-747, Aug. 1992.
71. Caccamo M., Lipari G., Buttazzo G., "Sharing Resources among Periodic and Aperiodic Tasks with Dynamic Deadlines", Proceedings of the 20th IEEE Real-Time Systems Symposium.
72. M. Spuri, and G.C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling", Proc. IEEE Real-Time Systems Symp. 1994.
73. M. Spuri, and G.C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems", Journal on Real-Time Systems, vol 10, no. 2, 1996.
74. Buttazzo, G.C.; Sensini, F., "Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments", IEEE Transactions on, Volume: 48 Issue: 10 , Oct. 1999, Page(s): 1035 –1052
75. Jeffay K., "Scheduling sporadic tasks with shared resources in hard real-time systems", Proceedings of the 13th IEEE Real-Time Systems Symposium, Phoenix, AZ, December 1992, pp. 89-99.
76. Kim H. I., Lee S. Y., Lee J.W., "A soft Aperiodic Task Scheduling Algorithm in Dynamic Priority Systems", Proceedings of the 2nd International Workshop on Real- Time Computing Systems and Applications, 1995.
77. Kim H. I., Lee S. Y., Lee J. W., "Scheduling of hard Aperiodic Requests in Dynamic Priority Systems", IEEE 1995.
78. Kim H. I., Lee S. Y., Lee J. W., "Alternative Priority Scheduling in Dynamic Priority Systems", Proceedings of the 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'96), 1996.
79. Dhall S. K., Liu C. L., "On a Real-Time Scheduling Problem", Operations Research, vol 26. no. 1, 1978.
80. Sun J, Liu J., "Synchronization Protocols in Distributed Real-Time Systems", 16th International Conference on Distributed Computing Systems, 1996.
81. Liestman A. L., Campbell R. H. 'A Fault-Tolerant Scheduling Problem", IEEE Trans. Software Engg. vol 12, no 11, 1988.
82. Oh Y., Son S., "Multiprocessor Support for Real-Time Fault Tolerant Scheduling", Proc. IEEE 1991 Workshop Architectural Aspects of Real-Time Systems, 1991.
83. Pradhan D. K., "Fault-Tolerant Computing: Theory and Techniques", Englewood Cliffs, N.J: Prentice Hall, 1986.
84. Ghosh S., Melhem R., Mosse D., "Fault-Tolerance Through Scheduling of Aperiodic

Tasks on Hard Real-Time Multiprocessor Systems", IEEE Trans. On Parallel and Distributed Systems, vol. 8, no. 3, 1997.
85. Maode M., Babak H., "A Fault-tolerant Strategy for Real-Time Task Scheduling on Multiprocessor System", Proceedings of the 1996 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '96), 1996.
86. Liberto F., Lauzac S., Melhem R., Mosse D., "Fault Tolerant Real-Time Global Scheduling on Multiprocessors", IEEE Proceedings of the 11th Euromicro Conference on Real-Time Systems, 1999.
87. Shin K. G., Hou C. -J., "Design and evaluation of effective load sharing in distributed real-time systems", Proc. IEEE Symp. On Parallel and Distributed Processing, 1991.
88. Shin K. G., Chang Y. -C., "Load Sharing in distributed real-time systems with state change broadcasts", IEEE Trans. On Computers, vol C-38, no. 8, 1989.
89. Eager D. L., Lazowska E. D., Zahorjan J., "Adaptive load sharing in homogeneous distributed systems", IEEE Trans. On Software Engineering, vol. SE-12, no. 5, 1986.
90. Davari S., Dhall S., "On a Real-Time Task Allocation Problem", Proc. of 19th Annual International Conference on System Sciences, 1986
91. S. Davari and S.K. Dhall, An On Line Algorithm for Real-Time Tasks Allocation, IEEE Real-Time Systems Symposium, 1986
92. S.K. Baruah, N.K. Cohen, C.G. Plaxton, D.A. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation", ACM Symposium on Theory of Computing, 1994.
93. S. Baruah, J. Gehrke, and C. G. Plaxton. "Fast scheduling of periodic tasks on multiple resources", Proceedings of the 9th International Parallel Processing Symposium, pages 280-288, April 1995.
94. Ghosh S., Melhem R., Mosse D., "Fault-Tolerant Scheduling on a Hard Real- Time Multiprocessor System", International Parallel Processing Symp., 1994.
95. W. Li, K. Kavi, and R. Akl, "A non-preemptive scheduling algorithm for soft real-time systems", Computers and Electrical Engineering, vol.33, no. 1, pp. 12–29, 2007.
96. Li, Q. & Ba, W, "A group priority earliest deadline first scheduling algorithm", Frontiers of Computer Science October 2012, Volume 6, Issue 5, pp 560–567
97. Arshjot Kaur, Supriya Kinger, International Journal of Computer Science and Information Technologies, Vol. 5 (4) , 2014, 4886-4890
98. J. Kwon, K.-W. Kim, S. Paik, J. Lee, and C.-G. Lee. Multicore scheduling of parallel real-time tasks with multiple parallelization options. In IEEE 21st Real-Time and Embedded Technology and Applications Symposium (RTAS), pages 232–244, April 2015.
99. L. Zeng, P. Huang, and L. Thiele. Towards the design of fault-tolerant mixed-criticality systems on multicores. In Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES '16, ACM. pages 6:1–6:10, New York, NY, USA, 2016.
100. G. Dhiman, K. K. Pusukuri and T. Rosing, "Analysis of Dynamic voltage scaling for system level energy management," in Proc. UNISEX Workshop Power Aware Comput. Syst, 2008.
101. S. Yue, D. Zhu, Y. Wang and M. Pedram, "Reinforcement Learning Based Dynamic Power Management with a Hybrid Power Supply," Computer Design (ICCD), 2012 IEEE 30th International Conference, pp. 81-86, 2012.
102. U. khan and B. Rinner, "A reinforcement learning framework for dynamic power management of a portable, multi-camera traffic monitoring system," in Green Computing and Communications (GreenCom), IEEE, 2012, pp. 557-564.
103. M. Trikil, A. C. Ammari, Y. Wang and M. Pedram, "Reinforcement Learning-Based Dynamic Power Management of a Battery-Powered System Supplying Multiple Active Modes.," European Modelling Symposium (EMS),, pp. 437-442, 2013.
104. M. Triki, Y. Wang, A. Ammari and M. Pedram, "Hierarchical power management of a system with autonomously power-managed components using reinforcement learning.," Elsevier, 2015.