International Journal of Computer Applications (0975 – 8887)
Volume 131 – No.17, December2015

A Survey on Scheduling Approaches for Hard Real-Time Systems

Mehrin Rouhifar
Computer Engineering Department
Central Tehran Branch – Islamic Azad University
Tehran - Iran

Reza Ravanmehr
Computer Engineering Department
Central Tehran Branch – Islamic Azad University
Tehran - Iran

ABSTRACT

In this paper, main scheduling algorithms for hard real-time systems (RTSs) have been investigated that include both uni and multi processors schemes. It provides the summary of schedulability analysis and well-known attributes. This paper composed of two parts; first part surveyed the basic hard RTS scheduling algorithms that guarantee the on-time completion of the tasks. Second part contains the different heuristic and partitioned approaches for some specific factors of real-time systems such as energy consumption, dependability, performance, scheduling feasibility and utilization of memory resource. Finally, the analysis and evaluation of the mentioned methods are shown based on the schedulability of task sets and efficiency.

Keywords

Real-time system, Hard RTS, Scheduling, Schedulability.

1. INTRODUCTION

Real-time systems are computational systems that must perform their operations correctly, considering the predefined time constraints [1], [2]. Processes in RTSs, refer to tasks that each of them has specific characteristics as Deadline, Execution time and Release time. The real-time system has requirements in terms of the deadline, so depending on the consequences of deadline missing, a real-time task can be classified to three categories of hard, soft or firm. In case of hard real-time task if a deadline missed, it will lead to complete failure of system and occur the catastrophic consequences. In the soft real-time system a deadline can be missed, while lead to no complete failure, only decrease system performance. Also, a real-time task is firm because if the deadline is missed, producing result will be not useful and cause no damage of system [3].

Today, hard real-time systems are extensively used in many various application fields including; automotive electronics, avionics, space systems, medical systems, household automation, and robotics [2, 4]. Scheduling algorithms in RTSs have special importance to ensure the desired and predictable behavior of system. In multiprocessor systems, tasks scheduling and processors management on distinct processors are as important issues in designing of scheduling algorithms. One of the generic methods is global or hierarchical scheduling in which a processor has played the role of manager and divides the tasks between processors based on the general parameters. Then, each processor can independently perform the scheduling according to the internal approaches. Therefore, selecting an appropriate algorithm has very effects on the RTS behavior and hence exist many types of them. Also, other important factors in real-time scheduling methods include issues such as response time, average of scheduling rate, energy consumption and fault tolerance.

Increasing the applications of embedded real-time systems in modern life and need to the more processing, has been pervasive the use of multi-core processors. Enhancing the processing power, be increased the energy consumption. As a result, the effective use of energy will be more important. On the other hand, due to variety of tasks and different requirements of each task, how to schedule the distinct tasks on processors will has greater importance. Gradually, real-time applications become the main role in this field. Therefore, the scheduling algorithms should simultaneously handle the time constraints of tasks and energy consumption.

Dynamic and static consumed power is the source of energy consumption for processors. Dynamic consumed power derives from the mode change of processor internal circuits to perform the tasks and leakage current is the source of static consumed power. The dynamic consumed power is an important part of the power consumption of the processor in micron technology. In recent years, the number of cores, density and temperature are enhanced, so static consumed power is greatly increased. Therefore, both dynamic and static consumed power parameters should be considered for optimizing the energy consumption.

DVFS (Dynamic Voltage and Frequency Scaling) and DPM (Dynamic Power Management) are two general techniques to reduce the energy consumption in processors. DVFS and DPM techniques were provided to reduce the dynamic consumed power and static consumed power respectively. DVFS method dynamically adjusts the processor operating voltage and frequency level to reduce the consumed power, while does not consider to reducing the static consumed power. DPM method also, finds the idle time between the processor tasks and it varies the processor mode from idle to sleep to reduce the static consumed power. Since, switching the processor mode has the energy and delay headers so, mode variation must be perform when idle time be more than a certain threshold. Several papers reduced the energy consumption using each of two above methods or hybrid of them. The effects of two methods are unequal and have distinct results in different conditions.

The fault tolerant system is a system that with existence of the hardware and software faults, is continued still to perform its services. In designing this type of system, mechanisms should be used that ensure the expected services accuracy, even in the presence of fault. According to the real-time nature of many fault tolerant systems, it is clear that provided mechanisms for this purpose will not be consistent with the scheduling constraints of real-time applications. It should be noted, PB (Primary Backup) and TMR (Triple Module Redundancy) are two generic methods for fault tolerance in real-time tasks scheduling of multiprocessor environments [5].

The remaining of the paper is organized as follows: Section 2 describes the basic methods of scheduling for hard real-time systems including uniprocessor and multiprocessor. Section 3 presents the Quality of Service (QoS) factors and also challenges and respective issues. In Section 4, hard real-time scheduling algorithms have been investigated in some specific factors such as energy consumption, reliability (fault tolerance), partitioned and memory centric methods. The results of analysis and comparison of the mentioned methods describe in section 5. Finally, section 6 concludes the approaches with schedulability analysis and QoSs.

2. HARD REAL-TIME SYSTEM SCHEDULING APPROACHES

Scheduling schemes in RTS are important to the desired behavior of system be predictable. Scheduling algorithm is a set of rules that determines how to manage a RTS by the scheduler. In other words, it specifies the task queuing and allocates time to processor. In addition, many tasks are duplicated and are performed once in each period which it called periodic. In Off-line scheduling algorithms, be made the decisions about the scheduling the system start to execute and the scheduler has the complete information from all the tasks. So, the tasks execute with a predetermined order whereas, on-line scheduling algorithms create a time table for tasks and schedule them in execution during. It should be note the scheduling in terms of the tasks priorities are categorized into the static and dynamic algorithms. In static algorithms, be allocated a fixed priority to the tasks before they begin to execute but the order of executing the tasks be determined during execution in dynamic algorithms. On the other hand, scheduling algorithms are classified to two types, preemptive and non-preemptive. In case of preemptive algorithms, when the task is executed on a processor, if another task arrives with higher priority, it will stop whereas in non-preemptive algorithms, the task’s execution no stop until it completed and finished. So according to all of the above, real-time scheduling can be classified as shown in Figure 1.

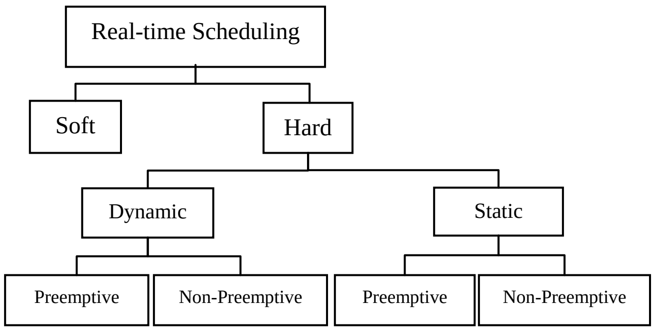

Figure1: Taxonomy of real-time scheduling algorithms

Selecting the appropriate algorithm for scheduling depends on the type of system that whether it is uniprocessor, multiprocessor or distributed. The uniprocessor system executes only a single process and it switches between the different processes. The multiprocessor system could be a multi-core or multiprocessor which it handles the several separate uniprocessor independently. The nodes are independent in a distributed system, while they greatly interact with each other in multiprocessors [3].

2.1 Scheduling Algorithms for Uniprocessor Systems

Scheduling algorithm for uniprocessor systems must ensure to allocate the enough time to all the system tasks at certain points of time that they can meet their deadline as far as possible. Figure 2 shows the classification of different kinds of algorithms.

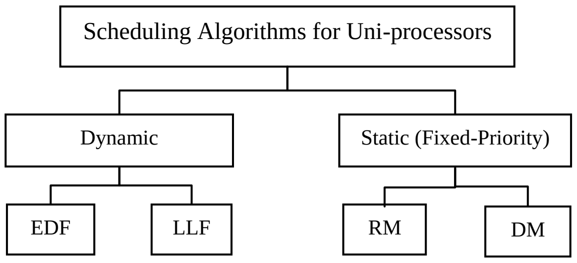

Figure 2: Scheduling algorithms for uniprocessor systems

2.1.1 Static Algorithms

Some of the most important static-priority scheduling algorithms are described as follows:

- RM (Rate Monotonic)

RM is a preemptive and static priority scheduling algorithm on uniprocessor systems. In this algorithm, the tasks with shorter periods have higher priority for execution because if the demand rate be more, the period would be shorter and the priority would increase. Therefore, it is used in periodic tasks. RM algorithm has been considered with following assumptions:

1) The periodic tasks have fixed runtime and they are ready to be executed at the beginning of each period T.
2) The implicit deadline of tasks (D) represents the end of period namely $D=T$.
3) The tasks are independent and do not block each other.
4) The scheduling overhead is assumed to zero due to time of context switching and exchanging.

- DM (Deadline Monotonic)

There is another scheduling algorithm called DM and is similar to RM with difference that $D \leq T$ (constrained deadline). So, RM can be considered as a specific case of DM which the deadline determines task priority. Consequently, a task with the shorter deadline will be executed at higher priority.

Some extensions have been performed on RM to increase its performance. So that, when the tasks share its resources, we can also use the RM. In order to prevent the simultaneous use from the shared resources, is used a technique called semaphore. In which case, when the task arrives to the critical section, it will lock and after the task exiting is released. The critical section is a part of the code for access to a shared source. Using the semaphores may have problems such as

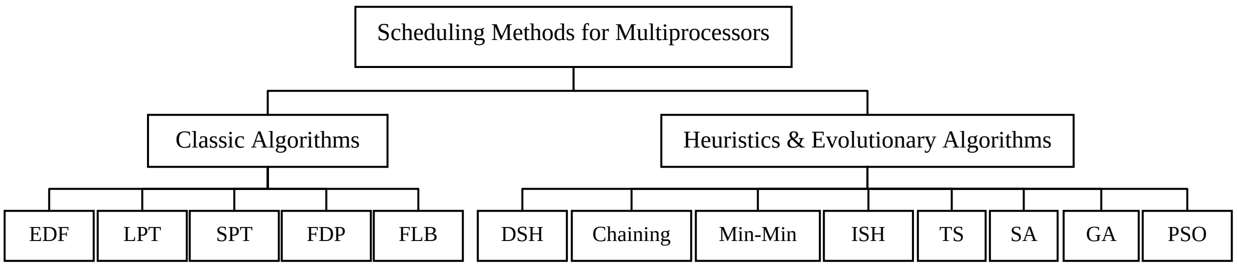
Scheduling Methods for Multiprocessors
Classic Algorithms
Heuristics & Evolutionary Algorithms
EDF
LPT
SPT
FDP
FLB
DSH
Chaining
Min-Min
ISH
TS
SA
GA
PSO

Figure3: Scheduling algorithms for multiprocessor systems

blocking. It occurs when a task will not execute by tasks with lower priority. To solve this problem, there are two methods as follows:

- PIP (Priority inheritance protocol)
  If a task blocks the task with higher priority, dynamically task priority will change.

- PCP (Priority ceiling protocol)
  This protocol has a semaphore that be allocated as the priority ceiling. Hence, it prevents the deadlock occurrence [6].

2.1.2 Dynamic Algorithms

Here there are two main dynamic-priority scheduling algorithms that include:

- EDF (Earliest Deadline First)

EDF is a dynamic priority scheduling algorithm that determines the task priority in terms of deadline. So, the higher priority would allocate to a task which is close to the end of its deadline. All the conditions and assumptions of RM algorithm is also valid for EDF except $D = T$. Furthermore, it is a preemptive method and has the capability of access to full efficiency of system.

- LLF (least Laxity First)

The task priority of LLF algorithm has been determined based on laxity, therefore the task priority will be higher that it had less laxity. The laxity is the time interval that a task is relaxed to execute.

Thus, if there are two tasks with the same laxity time, they will constantly preempt each other and will stop another execution. As a result, very context switching will create. But, if we ignore its derived cost, LLF like EDF, will is an optimal dynamic scheduling algorithm.

2.2 Scheduling Algorithms for Multiprocessor Systems

Multiprocessor scheduling is an attempt to answer the two problems of allocation and priority.

- Allocation problem: It determines which processor should execute the task and includes the following:

- No Migration: The each task is allocated to a processor and the any migration is not allowed.

- Task Level Migration: A task jobs execute on different processors, but it should be noted that a job is only executable on a single processor.

- Job Level Migration: A single job can be moved and execute on various processors but, it cannot be executed in parallel on different processors.

- Priority problem: It determines the tasks in what order should execute and has the following types:
  - Fixed Task Priority: Every task has a fixed priority for all of its jobs.
  - Fixed Job Priority: A task jobs have the different priorities, whereas each job has a fixed priority.
  - Dynamic Priority: each job may have the various priorities at different times.

Microprocessors are the most important factor for power of modern computers and their performance increase exponentially every year because of two main reasons; First, according to Moore's Law, the speed of transistors increase which it has a direct impact onto the performance of processors that have been made the transistors [7]. Second, increasing in microprocessor performance is more than Moore's Law estimation because the designers by controlling the increasing transistors onto modern chips could reach the more parallelism in compared with software techniques.

The new solution of microprocessors design is "multiprocessor chips" which in fact, are the equivalent of multi-core processors. The multiprocessor chip represents a set of the uniprocessors on a single chip, so that generally have the performance similar to a team. Indeed, instead of filling the chip space with single large processor, it uses the multiple small cores. Multi-core systems scheduling is a NP (Non-deterministic Polynomial time) Problem [8], [9]. Whereas, scheduling for this kind of systems with a view of improving the energy consumption is an NP-Hard problem [10]. Different algorithms have been proposed for multiprocessor scheduling. Principally, despite the presence of various proposed solutions, achieving a fully optimized solution with a lot of tasks almost is impossible while each of the algorithms are trying to obtain a near-optimal solution. These algorithms can be classified in the following, which is also shown schematically in Figure 3.

- Classic Algorithms: In this category, there are some algorithms such as EDF (Earliest Deadline First), LPT (Longest Processing Time), SPT (Shortest Processing Time), FDP (Fast Critical Path), FLB (Fast Load Balancing). The most of these methods, exclusively aren't in order to scheduling of multi-core systems but are useable about them. The above-mentioned algorithms often achieve the answers with much less time-complexity but no access to the optimal solution.

- Heuristics and Evolutionary Algorithms: They are included DSH (Duplication Scheduling Heuristic), Chaining, Min-Min, ISH (Insertion Scheduling Heuristic), Tabu Search (TS), Simulated annealing (SA), Genetic algorithms (GAs) and PSO (Particle Swarm Optimization). These algorithms have been widely using for multiprocessor systems scheduling.

Evolutionary algorithms provide a better solution spending the more runtime compared to other algorithms.

Multiprocessor systems scheduling are classified into three categories as follows:

- Heterogeneous: In these systems, processors are different. So, execution rate of each task depends on both the processor and task.
- Homogeneous: The processors are identical in this kind of systems. Thus, the execution rate is equal onto all of the processors.
- Uniform: The execution rate of each task only depends on the processor speed. Consequently, it is clear a processor with more speed will execute the tasks faster.

It should be mentioned to solve the scheduling problem of multiprocessor systems, the task model usually are considered as an independent task set that they are no periodic. The hard real-time tasks must be completed before their deadlines expire. The multiprocessor environments comprise of m processors or cores. This paper consider a task model [1] as a set $T = \{\tau_1, \tau_2, ..., \tau_n\}$ where is composed of *n* independent tasks and symbol *i* represents the task number. Every task is defined by the characteristics $(T_i, C_i, D_i,)$ that T_i , C_i and D_i notations denote the period, runtime and relative deadline respectively.

2.2.1 Scheduling of homogeneous multiprocessor systems

In regard to daily increasing development of multiprocessor systems in the last decade, generally the scheduling methods of homogeneous multiprocessor systems is classified to three categories of global, partitioned and hybrid [4]. The taxonomy of scheduling methods for homogeneous systems is shown in Figure 4.

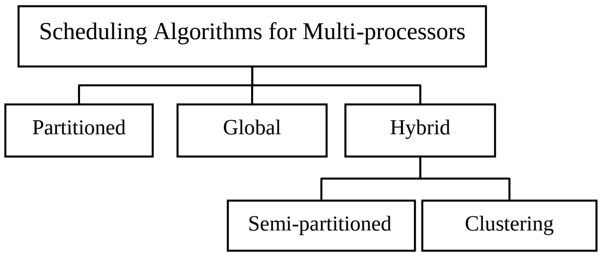

Figure 4: Scheduling methods for homogeneous multiprocessor systems

- Global scheduling algorithmsThese algorithms put the ready tasks in a sorted queue based on their priority. The highest priority task puts at first of queue. Then, it is selected by scheduler and would execute on any processors. Also, the task will migrate to the other processor, if necessary.

- Partitioned scheduling algorithmsIn this method, each task is allocated to a processor and it will only execute on the same processor exclusively. Instead of having a general queue, be used a separate execution queue for each processor.

- Hybrid scheduling algorithmsAccording to the hardware architecture, the overhead cost of general scheduling algorithms potentially is very high. In mentioned method, the jobs can potentially be migrated from one processor to another processor, which may lead to excessive communication loads and a large amount of cache is missed. As a result, WCET (Worst Case Execution Time) will increase when, does not exists in partitioned mode. Hybrid algorithm is a combination of global and partitioned approaches and it contains two methods of the semi-partitioned and clustering. Semi-partitioned is an approach that is applied in partitioned systems in order to use the sliced spare capacity and it divides a few tasks between two processors. Clustering case is also a partitioned method where in, clusters are composed a few of fast processors which have been allocated to tasks.

3. QUALITY OF SERVICE FACTORS

One of the QoSs for scheduling algorithms is to maximize the number of tasks that will complete before its deadline. Other QoS factors include the schedulability, efficiency, reliability, on-time completion of tasks, rejection ratio, performance and data quality. The guarantee of QoS for heterogeneous systems and clusters has become a very important requirement in today's real-time systems. Generally, data quality in the soft RTS and reliability in hard RTS systems has particular importance because the algorithms with requirement of reliability can tolerant the failures and improve the system reliability [11].

3.1 Challenges and Issues

Due to the increasing demand of the multiprocessor systems, they have the high capability of efficiency and reliability as a powerful computing solution in the hard real-time systems. Scheduling of multiprocessor systems still is as one of the challenges in the computer engineering field. The main problem of the real-time tasks scheduling in multiprocessor systems is to determine a task from the task set to execute and also determining a processor which should be executed the task on it. Other issues include the following:

- Restrictions of processor usage
- Ineffective tests of schedulability
- Considering the overhead (Cost)
- Limited task models for multiprocessor systems
- Limited policies of access to shared resources (Resource Allocation)

4. HARD REAL-TIME SCHEDULING ALGORITHMS AND APPROACHES FOR SPECIFIC APPLICATIONS

In this section, we review a number of recent methods and researches related to the area of scheduling of hard RT tasks in multiprocessor environment. In some of them, scheduling has been investigated from different aspects. Also, the issues such as reliability, energy consumption and etc have been considered for their implementation.

Houben and Halan [12] have proposed a dynamic scheduling algorithm of energy aware for hard real-time systems that is based on EDF scheme. As previously mentioned, DVFS is one of the energy management techniques that it adjusts dynamically the frequency and voltage in the scheduling algorithms. Therefore, lead to decrease the idle time of processor and increases the performance. In proposed algorithm, separate modes change based on time. Since the processor modes are specified by need of voltage, frequency and performance values, so the energy consumption of processors is controlled separately. It is assumed the task set T is consists of n periodic tasks in the form $T_i=(P_i, r_i, a_i)$. Symbol T_i determines the ith task and its parameters specifies the period, execution time and response time of ith task respectively in fastest mode. The proposed method is for an uniprocessor platform where the separate performance modes of a processor are determined by of $M=\{M_1, M_2, ..., M_N\}$. So that, $M_i=(V_i, F_i, P_i)$ and the parameters of V_i, F_i and P_i represent the voltage, frequency and performance of the ith mode. The different modes arranged in terms of their frequencies in downtrend.

In order to dynamic scheduling, EDF algorithm is developed by considering all of the constraints of hard real-time systems. Hence, energy consumption decreased because it has a direct relationship with consumed power. For this purpose, the task set T divided to the two subsets of R and S. The R is a set of the "ready tasks" to execute and S is a copy of "shadow tasks". Each task of subset R by finishing its execution enters to the subset S and the other tasks that is executed only part of them, depending on the request be rescheduled again. Subset S also includes the tasks that not ready for execution but waiting for activation.

Samal, Mall and Tripathy in [13] have proposed a heuristic approach named GFTS (Genetic Algorithm Based Fault-Tolerant Scheduler) for fault tolerant scheduling of aperiodic and hard real-time tasks on multiprocessor systems using genetic algorithm (GA). The primary-backup (PB) is one of the conventional methods for scheduling of fault tolerant which is used to ensure the real-time tasks meet their deadline but also it is not without fault. In studied paper, an optimal scheduling algorithm is proposed based on GA and is combined with information about the scheduling of real-time tasks to provide fault tolerance in multiprocessor environments. It uses PBFTS (Primary Backup Fault Tolerant Scheduling) with novel form GA. GFTS acts better than previous methods for fault tolerant scheduling with the primary-backup in terms of system efficiency and performance. In PBFTS systems, two same copies of a task are scheduled on distinct processors, without time overlapping and the backup copy is executed only while the primary copy of task be failed (detecting through acceptance test). Dynamic scheduling can classified to centralized and distributed.

In [13] is used the task model which is presented by Ghosh et al. [14]. It is assumed the multiprocessor system based on hard real-time tasks consists of m same processors that are connected through a shared medium. Also, the tasks are considered as aperiodic and non-preemptive. Each task T_i is shown as $\langle a_i, r_i, c_i, D_i\rangle$ where a_i is the arrival time of ith task, r_i its ready time, c_i its worst case computation time and D_i its relative deadline. The scheduler is designed as centralized for real-time tasks in multiprocessor systems environment. All the tasks arrive at the queue of central processor namely scheduler. Then, dispatcher distributes them to other processors to execute. The processors have equal computing capability and are connected via a shared memory. The communication between scheduler and processors accomplish with dispatch queues. Every processor contains a distinct dispatch queue. The scheduler and other processes are executed in parallel.

Mottaghi and Zarandi [15] presented the dynamic scheduling algorithm called DFTS (Dynamic Fault Tolerant Scheduling) for real-time tasks in multi-core processors by considering the tolerance for transient faults of single and multiple. So, the tasks are scheduled according to three issues; First released tasks at present, Second available cores at present, and Third number of faults and their incidence.

In DFTS scheme, released tasks are classified as critical or non-critical in terms of threshold value θ. For this purpose, is proposed a parameter namely "task criticality" that determines the task type according to task utilization and the time at which resources are allocated the tasks by scheduler. Then, according to task utilization be dynamically choose an appropriate recovery method to tolerate the most number of multiple faults. In addition to, scheduling is performed for two important goals following: increasing scheduling possibility and reducing the runtime of the total tasks. Non-critical tasks are scheduled only on a single core and use the checkpointing method with rollback recovery. While critical tasks are repeated on separate cores to increase the finish probability of the tasks before their deadline expire despite the presence of fault. In DFTS, the task set $T=\{\tau_1, \tau_2, ..., \tau_n\}$ be composed of n independent real-time tasks which are sporadic and non-preemptive. Every task τ_i is represented by a tuple (C_i, T_i, D_i) where C_i is WCET of task in a no fault condition, T_i is the period and D_i is the relative deadline of the ith task. Also, multi-core platform is assumed as a set of M homogeneous cores in the form of $P=\{P_1, P_2, ..., P_M\}$ so that, each of the tasks can be executed on every core of processor. Next parameter is task utilization, U_i $(0 \leq U_i \leq 1)$ which is defined as $U_i = \frac{C_i}{D_i}$. Therefore, total utilization for an application is shown with notation U that is equal to sum of the all tasks utilization in application, namely $U = \sum_{i=1}^{n} U_i$. DFTS is a hybrid approach based on hardware and time redundancies. Then, scheduler in terms of above mentioned three factors chooses an appropriate method to tolerate faults that include: available hardware resources, task utilization and the number of expected faults.

Wiese and et al. [16] proposed a partitioned EDF scheduling approach on the unrelated multiprocessor platforms. PTAS (polynomial-time approximation scheme) is an algorithm to solve the certain types of optimization problems. For partitioning is assumed a set of n implicit deadline sporadic task must be partitioned on m unrelated processors in multiprocessor system containing κ different types of processors. For this purpose, first, task system be converted to other system that its partitioning is easier. Then, big and small tasks are separated and the patterns of big tasks indicate the feasible partitioning of a task system on the considered platform. Since, there are many different patterns for big tasks that their complexity is only polynomial, so it is possible a part of the polynomial time be expended to testing each of patterns. The feasible partitioning using differential solution obtained as follows: first a bipartite graph is created from fractional solution, next a differential matching is defined on the mentioned graph that is corresponds to tasks differential allocation on the identical processors. Then, using Brikhoff algorithm determine the matching kind that is integer or fractional. Finally, the partitioning of task system is specified by an integer matching.

Table 1: Schedulability analysis of basic hard RTS algorithms

	Priority	Condition	Utilization bound test	Response time Analysis	Complexity	Optimal
RM	Fixed	$D_i = T_i$	$U = \sum_{i=1}^{n} \frac{C_i}{T_i} <= n * (2^{1/n} - 1)$	$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$ $if \quad R_i \le T_i$	$O(\log n)$	Yes
		$D_i <= T_i$			Pseudo-Polynomial	No
DM	Fixed	$D_i <= T_i$	$U = \sum_{i=1}^{n} \frac{C_i}{D_i} <= n * (2^{1/n} - 1)$	$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$ $if \quad R_i \le D_i$	Pseudo-Polynomial $O(nN)$	Yes
EDF	Dynamic	$D_i = T_i$	$U = \sum_{i=1}^{n} \frac{C_i}{T_i} <= 1$	$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$ $if \quad R_i \le D_i$	$O(n)$	Yes
		if $D_i <= T_i$	is more complicated		Pseudo-Polynomial	

For hard real-time scheduling on multi-core platforms, [17] has proposed a memory-centric approach to improve hard real-time utilization in which firstly, core isolation is performed via the coarse-grained (high-level) and memory schedules with Time Division Multiple Access (TDMA), and secondly, when memory access is scheduled by high level, the scheduling policy of every core will increase the priority of memory computations compared with the computations that are performed only by CPU. For memory-centric real-time scheduling using PREM (PRedictable Execution Model), is assumed the hard real-time system is scheduled on partitioned multi-core platform which tasks are statically allocated to the fixed priority cores. Every task is periodically activated with limited deadline. There are no shared sources between cores, except main memory that is used by each deadline. Memory-centric technique schedules the simultaneous accesses to main memory from various sources. The used approach for this purpose is based on high level TDMA [18]. An iteration of the coarse-grained memory scheduling is made offline so that slots with fixed size allocated to cores of processor. Each core of processor, exclusively accesses to main memory during the slot is given to its TDMA memory.

In memory-centric scheduling based on PREM model, the every task has a code which is divided into a collection of scheduling intervals, so that they are sequentially run at execution time. Scheduling intervals are categorized to the compatible and predictable intervals. In order to schedule under PREM is needed to the predictable intervals be compiled. Therefore, this kind interval (predictable) is divided into two phases including memory phase and execution phase. At the initial of memory phase, to accomplish a collection of fetch and replacement operations of cache line, CPU should be accessed to main memory. All of the cache lines at the end of this phase should be accessible in cache of last level. So in execution phase, the task accomplishes the effective computations without missing the cache of last level.

5. EVALUATION AND COMPARISON

In this section of paper, implementation details of RTS scheduling algorithms in different approaches are investigated and the results of analysis are described in two subsections.

5.1 Basic Scheduling Algorithms

In first subsection, a number of the basic scheduling algorithms such as RM, DM and EDF have been analyzed. They are used for scheduling of periodic tasks in hard real-time environment. The schedulability of these algorithms be checked by utilization bound test (sufficient test) or response time analysis (precise test). Therefore, they have been compared from different aspects. The obtained results of their analysis have been shown in Table 1.

As mentioned in Sec. 2.2, is assumed the task set $T=\{\tau_1, \tau_2, \ldots, \tau_n\}$ is composed of n periodic tasks. The task τ_i is represented as (T_i, C_i, D_i) where its parameters determine the period, execution time and relative deadline respectively. Moreover, R_i is the worst-case response time of task τ_i, U_i is the task utilization and N is the number of iterations in the inner loop.

5.2 Specific Scheduling Algorithms

In [12] runtime complexity of energy aware and EDF based scheduling algorithm is $O(n \log n)$ if a task is ready for execution. For the implementation of this method PEARL90 [19] has been used which has the most advanced real-time capabilities. The experimental results shown it can guarantee the constraints of hard RTSs as well.

Table 2: QoSs analysis and platform comparison for some hard RTS specific applications

	Platform	QoS1	QoS2	QoS3	QoS4
Energy-Aware [12]	Uniprocessor	Timeliness guarantee	Decreasing energy consumption	Improving performance	
	Dynamic				
	Periodic tasks				
GFTS [13]	Multiprocessor	Fault tolerance	Improving schedulability	Improving performance (even under fault condition)	Decreasing rejection ratio
	Dynamic				
	Aperiodic tasks				
	Non-preemptive				
DFTS [15]	Multi-core homogenous	Fault tolerance	Improving schedulability	Reducing time overhead	hardware & timing constraints meet
	Dynamic				
	Sporadic tasks				
	Non-preemptive				
Partitioned EDF [16]	Unrelated multiprocessor	Polynomial – time	Highly intractable	Good performance	
	Sporadic tasks				
Memory-centric [17]	Multi-core	Reducing time overhead	Improving schedulability	Improving core utilization	Improving utilization of memory source
	Preemptive				

In order to simulate of [13] is used the computing systems based on Intel core i5 processor $(650@3.2GHz)$ and Matlab platform. Simulation accomplished by generating the random task sets with different sizes on different numbers of processors and also producing random fault data for every task set containing failed processors and time that processor failed. GFTS algorithm is executed with task set of size 10 on 4 processors as input and initial population size is 100 random individuals which task set in both cases under no fault condition and fault condition is scheduled. The time complexity of proposed method in [13] is equal to $T(n) = P \times N^2 \times M^2 \times T$ in which notations of P, N, M and T indicate the population size, number of tasks, number of processors and latest deadline respectively. In regard to simulation results, GFTS improved the rejection ratio over 50% and at around 25% fitness value of GA in comparison with TFTS method.

DFTS method in [15] has been simulated by using a simulator of task scheduler in C++. In the multi-core processors, the inputs of simulator contain the number of cores, the rate of fault-arrival, checkpoint cost parameters and checkpoint recovery of tasks which are simulator's input. During every execution round, application is scheduled on a multi-core processor as a collection of non-scheduled tasks. In order to increase the fault tolerance in [15], was used the approach that is a combination of two traditional fault tolerant methods called task replication for hardware redundancy and also checkpointing with rollback recovery for time-based redundancy. Experiments are accomplished using Intel® core ™ i7-2670QM processor with 4 GB RAM. For evaluate the DFTS feasibility rate is considered the parameters such as the checkpoint savings rate (φ), checkpoint recovery cost (μ) with fixed fault rate ($\lambda = 0.01$).

[17] was simulated based on the benchmarks of EEMBC (Embedded Microprocessor Benchmark Consortium). The experiments of memory-centric scheduling performed on Intel Q6700 processor. In order to configure the system as embedded, CPU frequency set to 1GHz to memory bandwidth reached 1.8 Gbytes/s. The Q6700 processor is composed of four processing cores, so that each pair of cores is shared with a second level cache. In [18] two scheduling frameworks of memory-centric and connection-based is considered, their performance under different configurations are compared. Simulation performed on a system with 8 cores, 10 tasks for per core and fixed interval length i.e. 1ms. Results show the task length in connection-based is 20% shorter than the memory-centric, because contention-based does not need to use the compiled tasks in terms of PREM. In contrast, utilization of a core in memory-centric scheduling has an improvement at around 20% compared with connection-based. This means the memory-centric technique increases the schedulability ratio sharply.

Here according to all of the above, the QoSs and characteristics of scheduling methods in some RTS specific applications have been determined which including the energy consumption, fault tolerance and utilization of memory resource. The summary of their analysis and also schedulability capability have been expressed in Table 2.

As can be seen, energy-aware scheduling algorithm based on EDF using DVFS technique decrease the processor idle time and improves the performance. GFTS is a fault tolerant scheduling algorithm in multiprocessor environment which combines GA with the information of tasks scheduling. In partitioned EDF was shown the sporadic tasks with implicit deadline should be partitioned in PTAS scheme. The feasible partitioning obtained using differential solution and so performance increase. Finally, the last case has proposed a new approach to schedule the multi-core platform in which, system modeled in two phase of memory and execution that does not detects the read and write operations from each other in main memory.

6. CONCLUSION

This paper provides the analysis summary of schedulability and QoS factors for hard RTSs scheduling approaches. Achieved results have been compared and have been shown as tables. First part investigates the basic and well-known scheduling algorithms. They under conditions are optimal techniques to guarantee the tasks meet their deadlines. Then, in next part, has been surveyed the several scheduling approaches used in some hard RTS applications and their QoSs are analyzed.

The energy-aware scheduling algorithm based on EDF uses DVFS technique to save the energy consumption and decreasing the processor idle time improves the performance. GFTS method improves the task rejection ratio and average fitness value of schedule in comparison with traditional schemes TFTS and other GAs algorithms. Another dynamic approach called DFTS increase the tolerance of multi-core systems for multiple faults in which is used the proper technique of hardware or time redundancy. In partitioned EDF has been proposed the algorithm for unrelated processors to provide the solution possibility of the NP-Hard optimization problems that put in PTAS class with polynomial-time approximation. Moreover, it was shown the sporadic tasks with implicit deadline should be partitioned in PTAS scheme. Finally, memory-centric scheduling method considers the main memory as most important shared resource. Using TDMA model, the tasks are scheduled in terms of their access to memory. As a result, the utilization of memory source, cores and hard real-time tasks improved.

7. REFERENCES

[1] Buttazzo. G. C, “Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications”, Springer Publications 3rd Edition, 2011.

[2] Robert. I, Davis, R and Burns A, “A Review of Fixed Priority and EDF Scheduling for Hard Real-Time Uniprocessor Systems”, EWiLi’13, August 26–27, 2013, Toulouse, FRANCE.

[3] Lindh. F, Otnes. T, Wennerström. J, “Scheduling Algorithms for Real-Time Systems”.

[4] Davis, R. I. and Burns. A, “A Survey of Hard Real-Time Scheduling for Multiprocessor Systems”, ACM Computer Survey, Vol. 43, No.4, Article 35, 44 pages, 2011.

[5] Shamim Shiravi and Mostafa E. Salehi, “Fault Tolerant Task Scheduling Algorithm for Multicore Systems”, The 22nd Iranian Conference on Electrical Engineering (ICEE 2014), 2014, Shahid Beheshti University.

[6] Sha L., Rajkumar R. and Lehoczky J. P., "Priority Inheritance Protocols: An Approach to Real Time Synchronisation", IEEE Transactions on Computers 39(9), pp. 1175-1185, September 1990.

[7] G. E. Moore, "Cramming more components onto integrated circuits", Electronics, Vol. 38, No. 8, McGraw-Hill, 1965.

[8] F. Kong, W. Yi, and Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multicores" in Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1-6, 2011.

[9] W. Y. Lee, "Energy-efficient scheduling of periodic real-time tasks on lightly loaded multicore processors", Parallel and Distributed Systems, IEEE Transactions on, vol. 23, pp. 530-537, 2012.

[10] J.-J. Chen and T.-W. Kuo, "Energy-efficient scheduling of periodic real-time tasks over homogeneous multiprocessors", in the 2nd international workshop on power-aware real-time computing, pp. 30-35, 2005.

[11] K. Manudhane, A. Wadhe, “QoS-Aware Approaches to Real-Time task scheduling on Heterogeneous Clusters”, international Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 4, pp. 174−180, 2013.

[12] K. Houben and A. Halan, “An Energy-Aware Dynamic Scheduling Algorithm for Hard Real-Time Systems”, 3rd Mediterranean Conference on Embedded Computing, MECO – 2014, ACM, PP. 14-17.

[13] Abhaya K. Samal , R. Mall and C. Tripathy, “Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm”, Elsevier. Swarm and Evolutionary Computation, 2014.

[14] S. Ghosh, R. Melhem, D. Mossé, “Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems”, IEEE Trans. Parallel Distrib. Syst. 8 (3), pp. 272-284, 1997.

[15] Mohammad H. Mottaghi and Hamid R. Zarandi, “DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors”, Elsevier.Microprocessors and Microsystems, Vol. 38, pp:88–97, 2014.

[16] A. Wiese, V. Bonifaci and S. Baruah, “Partitioned EDF scheduling on a few types of unrelated multiprocessors”, Springer, Real-Time Syst, vol. 49, pp:219–238, 2013.

[17] G. Yao, R. Pellizzoni, S. Bak, E. Betti and M. Caccamo, “Memory-centric scheduling for multicore hard real-time systems”, Springer, Real-Time Syst, vol. 48, pp:681–715, 2012.

[18] J. Rosen, P. Eles, A. Andrei, Z. Peng, “Bus access optimization for predictable implementation of realtime applications on multiprocessor systems-on-chip”, In: Proceedings of the 28th IEEE real-time system symposium, 2007.

[19] German standard DIN 66243-2, “Programmiersparche PEARL90”, Beuth, 1998.