Journal of Computer Science

Original Research Paper

Feasibility Analysis of Non-Preemptive Periodic Systems from Infeasibility Perspective

Nasro Min-Allah

College of Computer Science and Information Technology,
Imam Abdulrahman Bin Faisal University, P.O. Box 1982, Dammam, Saudi Arabia

Article history
Received: 08-04-2019
Revised: 18-06-2019
Accepted: 7-07-2019

Email: nabdullatief@iau.edu.sa

Abstract: Due to simple implementation, non-preemptive scheduling has the advantage over preemptive counterpart when it comes to deployment of real-time systems. Accordingly, many feasibility techniques have been established to answer schedulability of the task set for non-preemptive case. The time complexity of exact condition for non-preemptive under dynamic priority assignment is of pseud-polynomial nature. Recently, efforts are made to decrease the computation cost of existing exact solutions. However, the time complexity class remains the same. In such systems, feasibility is tested by starting with highest priority task and test continues in that order until the last task is analyzed in the set. Normally, higher priority tasks rarely miss the deadline and hence, when a system is determined infeasible, it is mainly because of the low priority tasks as these tasks are assigned low priorities and can claim CPU time only when there is no pending higher priority task in the queue. In this study, we propose a mechanism that reduces the computation cost of feasibility for non-preemptive earliest deadline first scheduling algorithm by testing the infeasibility of the system in reverse priority order. In worst case, the proposed technique is not inferior for a system with low utilization that scans a task set from feasibility perspectives. On the other hand, our test exhibits better performance when the system infeasibility is tested for the system demanding higher CPU utilization. Our experimental results show that the overall computation cost, especially for the larger task sets with higher CPU demands, is significantly reduced with the proposed solution by evaluating a system from infeasibility perspective.

Keywords: Operating Systems, Scheduling, Non-Preemptive Scheduling, Real-Time Systems, Feasibility Analysis

Introduction

Task preemption play a decisive role in the construction of real-time systems and deeply influences the overall system utilization. In scheduling theory, a task is called non-preemptive if it runs to its completion once it has been given the processor. In the preemptive counterpart, however, processor time can be allocated to another waiting task based on some scheduling strategy and hence low priority tasks cannot block higher priority tasks. Though promising from utilization point of view, preemptive scheduling are complex due to context switching, pipeline, cache-related costs etc. In contrast, non-preemptive systems also known as co-operative scheduling manage all the tasks fairly and offer simple implementation, however, analysis of non-preemptive systems under dynamic scheduling is more complicated than the preemptive counterpart.

A number of scheduling polices exist today in literature (Baruah *et al*., 2003; Liu and Layland, 1993; Swaminathan and Chakrabarty, 2005; Liu, 2000), especially tailored for performance metrics such as minimize response time, higher throughput, completion time and higher predictability etc (Baruah *et al*., 2003; Liu and Layland, 1993; Swaminathan and Chakrabarty, 2005; Liu, 2000; Krishna and Shin, 1997; Davis and Burns, 2011; George *et al*., 1996; Guan *et al*., 2008; Sha *et al*., 2004; Min-Allah, 2019; Audsley *et al*., 1995; Min-Allah *et al*., 2010; 2013; Burns *et al*., 2012; Jeffay *et al*., 1991; Jejurikar and Gupta, 2005;

Alrashed *et al*., 2017; Nasri and Kargahi, 2014; Nasri, 2017; Min-Allah *et al*., 2019; Khan and Min-Allah, 2011; Min-Allah *et al*., 2012). Among these matrices, predictability is of the highest importance for the success of a real-time systems (Swaminathan and Chakrabarty, 2005), however, in general system some other criteria might be of more interest such as throughput.

Real-time systems ensure the deadlines associate with individual tasks are met and missing even a single deadline can result in catastrophic consequences. Thus, missing deadline by single task/process makes the entire system infeasible. Unlike non-real-time systems, where statistical or experimental evaluations can be used, real-time system must be proven not to fail under any possible circumstances and hence the validity of each task is tested mathematically. For this matter, tasks are assigned priorities such that the decision for allocating processor is based on the task priority.

The main classes of real-time scheduling are (i) fixed priority (Min-Allah, 2019; Audsley *et al*., 1995; Min-Allah *et al*., 2010; 2013) and (ii) dynamic priority (Liu and Layland, 1993; Liu, 2000; Krishna and Shin, 1997; Burns *et al*., 2012). Fixed priority scheduling results in more predictability however the processor utilization is compromised as a tradeoff for respecting timing constraints. Infixed priority systems, a task is assigned priority that remains unique and not interchange-able with any other task. On the other hand, task priority may change at execution time for the tasks and processor is assigned to the task with shortest deadline. Task priorities are broken arbitrarily when two or more tasks share the same deadline. The priority assignment ensures that CPU is always running higher priority task and it is always known in case of fixed priority system that which task will miss the deadline. While the priority order changes at run time in dynamic system and hence CPU will be running a task that has highest priority at a given time, say deadline of the running task is the earliest among the other tasks in the queue. Irrespective of fixed or dynamic priority, a system is declared infeasible when at least one task misses the deadline. As compared to fixed priority systems, the advantage associated with dynamic priority is the higher performance and hence dynamic priority systems is discussed in this study.

To ensure that the deadlines associated with task are fully respected, feasibility analysis has to run before the processor is allocated to a running task (Liu and Layland, 1993; Jeffay *et al*., 1991; Audsley *et al*., 1995; Guan *et al*., 2008; Nasri and Kargahi, 2014; Nasri, 2017). Due to nature of tasks scheduling in preemptive and non-preemptive cases, different types of feasibility tests have been applied to know the schedulability of a task set. Literature shows that preemptive scheduling results in more context switching and considerable processor time is wasted on such operations (Liu, 2000; Krishna and Shin, 1997; Davis and Burns, 2011; George *et al*., 1996; Guan *et al*., 2008; Sha *et al*., 2004), while non-preemptive provide simpler implementation and can be used in commercial operating systems (Jejurikar and Gupta, 2005; Alrashed *et al*., 2017; Nasri and Kargahi, 2014; Nasri, 2017; Min-Allah *et al*., 2019; Khan and Min-Allah, 2011). For the feasibility analysis of non-preemptive system under dynamic priority assignment the existing feasibility test are expensive from computation cost point of view.

As per existing scheduling algorithms, Earliest Deadline First (EDF) (Liu and Layland, 1993) is considered the optimal one among workload conserving scheduling algorithms for preemptive real-time systems. In such cases, EDF can schedule any task set as long as utilization demand is less or equal to 100%. In other words, no other scheduling algorithm can provide higher system utilization than EDF for the preemptive case. However, this bound becomes irrelevant when task set has to be scheduled non-preemptively. The feasibility analysis of non-preemptive case depends not only on the utilization but also on the schedulability analysis of each task and hence computation cost increases significantly.

In this study, we propose testing the schedulability of a real-time system from the infeasibility perspective. By in-feasibility, we mean to determine if the task set in unschedulable with non-preemptive EDF scheduling on a single CPU system instead of answering if the task set is EDF schedulable under non-preemptive EDF. When task set schedulability is of interest, testing schedulability of individual tasks starting from the highest priority tasks is an established approach, while lowest priority task should be evaluated first when they task set has to be analyzed from infeasibility perspective. We test schedulability of lowest priority task first and if the task is found schedulable, we test the schedulability of lower priority task till we reach the task with highest priority. Authors in (Alrashed *et al*., 2017) derived that it is suffix to check the schedulability of the lowest priority task in a special case on non-preemptive EDF and hence out of scope of this paper. In this study extend the work done in (Jejurikar and Gupta, 2005) by employing the lowest priority first approach. For the system when all tasks are schedulable non-preemptively with EDF, our test exhibits the same complexity as the testing feasibility by starting with highest priority task and so on. On the other side, for the infeasible task sets, our technique determines system infeasibility much early by testing system feasibility in the opposite direction.

The remaining of this paper is organized as follows. Section 2 presents the related work and background of our task model. Section 3 establishes a novel task-scheduling mechanism for non-preemptive tasks that are

of periodic nature. The results for in-feasibility analysis are shown in Section 4 and the effectiveness of our technique is evaluated accordingly. Finally, we conclude the paper in Section 5.

Related Work and Problem Formulations

As the complexity of smart home solution grows, many devices are expected to support tasks that demands timely responses. While simple tasks can be supported with tiny sensors with no computation capability, complex tasks need devices offering desired computation capabilities. Such embedded devices generally need a basic operating system at least to run a few tasks successfully. However, due to memory and power constraints, it is always preferred to incorporate a lightweight operating system in such devices. Again, the functionality of an operating system depends on the target application and accordingly a scheduling policy should be decided before deploying onto the real system. In a simple embedded system, implementing one scheduling approach might suffix but for basic operations but for flexible systems the scheduler should support multiple schemes that can be configured per application needs. The role of scheduling algorithm is of particular interest to the real-time system where timings requirements are of great interest.

Real time system can be categorized into multiple levels but the two major types are soft and hard real-time systems. In soft real-time system, meeting the task deadline is desirable while its becomes inevitable for the hard real-time system. Consequently scheduling algorithms are selected based on the nature of targeted systems. Real-time system theory is rich in this aspect and a number of alternatives are available (Liu and Layland, 1993; Swaminathan and Chakrabarty, 2005) of the fundamental fixed and dynamic scheduling algorithms (Liu and Layland, 1993) our work is applicable to independent tasks system where there is no precedence among tasks. The task set is periodic and all tasks are ready for execution simultaneously at time t = 0. A task τ_i may finish early that its worst case completion time and each independent task createsa sequence of jobs where each job arrived at integer multiple of task period p_i. Theexecution requirement of a task is given by c_i. There are n number of tasks and to assign a unique priority to a task, the number of discrete priority levels that are supported bythe hardware is the same as task number. Each task must have a priority that my change at run time.

The utilization of a task is denoted by u_i and it is the ration of execution time to the task period. The task set is represented by τ and total system utilization is given by $u_{tot}\sum_{i=1}^{n}\left(c_i / p_i\right)$. Our target system is uni-processor system and run at uniform speed which is the maximum speed in this case. The results established in this study can be potentially applied to the CPUs running on multiple discrete levels but such systems are out of the scope of this work. The system never becomes idle when there is a pending task and a task needs to be executed till completion as soon as the priority system can assigns it the CPU. In the beginning all tasks are assumed to arrive at t = 0. This assumption allows us to mimic the worst case of combination of tasks release times also known as critical instant (Liu and Layland, 1993).

To schedule a task set, various scheduling schemes have been implement such as First-come First-served (FCFS) (FIFO), Last-in First-Out (LIFO), Shortest Job First (SJF), Round Robin (RR), Rate-Monotonic (RM), Earliest Deadline First (EDF) etc. (Baruah *et al*., 2003; Liu and Layland, 1993; Swaminathan and Chakrabarty, 2005). Thesetechnique are designed for keeping some perforans criterion in mind such as CPU utilization, throughput, waiting time, response time and predictability etc. The applicability of these metrics depends on the system that is under study. For instance, in real-time systems predictability is everything i.e., task deadline must be respected irrespective of the system load.

The aforementioned scheduling schemes can be classified into two main types: (i) general scheduling for handling any task queue and (ii) priority specific mainly designed for real-time systems such as RR and EDF etc. EDF is dynamic priority schemes that support both preemptive and non-preemptive systems. EDF is an optimal technique for real-time systems in the sense that if any other scheduling approach can successfully schedule a non-preemptive task set then EDF will never fail. Various techniques are available for feasibility analysis of preemptive (Baruah *et al*., 2003; Liu and Layland, 1993; Guan *et al*., 2008; Sha *et al*., 2008; Min-Allah, 2019; Audsley *et al*., 1995; Min-Allah *et al*., 2010; 2013) and non-preemptive case (Jeffay *et al*., 1991; Jejurikar and Gupta, 2005; Guan *et al*., 2008). For dynamic scheduling, the classic test for preemptive real-time systems was derived in (Liu and Layland, 1993).

Theorem 2.1. Liu and Layland (1993)

Under a non-preemptive EDF scheduling policy, a periodic task set is schedulable on a uni-processor system iff:

$$\sum_{i=1}^{n}\frac{c_i}{p_i}\leq 1 \tag{1}$$

The test given in Theorem 2.1 is pretty straight forward and guarantees schedulability of any task set, as long as utilization is less than or equal to 100%,

preemptively. It is worth highlighting that Inequality 1 denotes the maximum theocratical limit on system utilization that can be achieved be achieved under a simple uni-processor system. How-ever, non-preemptive system presents new challenge of letting a running task continue till completion. Inequality 1 suggests that higher utilization is obtained for the preemptive case, while such system utilization has to be traded for the sake of predictability incase of non-preemptive systems. For instance, there are situations where the utilization bound drops to zero for non-preemptive scheduling e.g., assuming two tasks in the task set are $\tau_1 = (c_1,p_1)$ and $\tau_2 = (p_1,p_2)$, where c_1 and p_2 is arbitrary small and large, re-spectively. Hence, $\sum_{i=1}^{2} c_i / p_i \to 0$. In contrast to satisfying only condition of utilization under preemptive scheduling, EDF test has two conditions to satisfy for non-preemptive case i.e., (i) utilization and (ii) passing schedulability test at all scheduling points in a set. Authors in (Jeffay *et al*., 1991) presented the most celebrated results for the non-preemptive periodic systems given as.

Theorem 2.2. Jeffay et al. (1991)

A periodic task set that is sorted in increasing order of the task period can only be scheduled feasibly under a non-preemptive EDF scheduling policy iff:

$$\sum_{i=0}^{n} \frac{c_i}{p_i} \le 1,$$
$$\forall_i,(1 < i \le n), \forall t,(p_1 \le t \le p_i): c_i + \sum_{k=1}^{i-1} \left\lfloor \frac{t}{p_k} \right\rfloor c_k \le t \tag{2}$$

Theorem 2.2 shows that the feasibility of τ_i needs to be checked in ($p_1<t< p_i$). Before we present our main contribution, we provide a recent work that solve the issue of ($p_1<t< p_i$) (open interval starting from p_1 up to p_i) in Theorem 2.2 by restricting the feasibility analysis to a subset of points in (Jejurikar and Gupta, 2005) for obtaining power efficient scheduling. According to (Jejurikar and Gupta, 2005), a set of periodic tasks that is sorted in increasing task period order can be scheduled non-preemptively under EDF scheduling policy, when:

$$\sum_{i=0}^{n} \frac{c_i}{p_i} \le 1,$$
$$\forall i,(1 < i \le n), \forall t,(p_1 \le t \le p_i): \frac{1}{\eta}\left(c_i + \sum_{k=1}^{i-1} \left\lfloor \frac{t}{p_k} \right\rfloor c_k \right) \le t, \tag{3}$$

where, $t \in S_i = lp_j; \forall_j,(j = 1,...,i); \forall l,\left(l = 1,\cdots \left\lfloor p_i / p_j \right\rfloor\right)$. See (Jejurikar and Gupta, 2005) for more details.

The work done in (Jejurikar and Gupta, 2005) provides the foundation to lower the system speed on the fly for energy gains in non-preemptive real-time systems under dynamic scheduling mecha-nism. The approach discussed in (Jejurikar and Gupta, 2005) determines the suitable system speed such that the deadlines of the tasks are never compromised. In rest of the study, we assume $\eta = 1$ in Inequality 3 as our system is running at only speed level (full speed). Since the feasibility starts with higher priority task in Inequality 2, we denote this expression with Highest Priority First EDF (HPF-EDF) for the EDF class. Recently, an improved tests was pro-posed in (Min-Allah, 2019) by exploiting the probability of satisfying task schedulability at larger points, however, the approach applies to fixed priority systems. In (Nasri and Kargahi, 2014), authors discussed feasibility test for non-preemptive case at scheduling points that are common to tasks. The test derived in (Nasri and Kargahi, 2014) is aligned with traditional EDF feasibility techniques as it follows the highest priority first order when analyzing the feasibility of the entire task set according to non-preemptive scheduling algorithm. This work is focused on the lowest priority first order and hence uses (Jeffay *et al*., 1991) for comparison purposes.

Determining Schedulability with Lowest Priority Task First

As a close counter part, fixed priority system handle a task set preemptively by assigned fixed priorities to task that remains fixed throughout. Under fixed priority class, Rate-Monotonic (RM) (Liu and Layland, 1993) scheduling algorithm is highly celebrated result. The total workload due a task τ_i is calculated as $W_i(t) = c_i \sum_{j=1}^{i-1} \left\lceil \frac{t}{p_j} \right\rceil c_j$ and hence τ_i is schedulable if and only if $\left(W_i(t) \le t\right)$ at any pint in time $t \in L_i$ where $L_i = \left\{ ap_b \mid b = 1,\ldots,i; a = 1,\ldots,\left\lfloor p_i / p_b \right\rfloor \right\}$. It is to be noted that under RM, a task is considered schedulable according to RM priority assignment on uni-processor device when this expression holds even at a single scheduling point as the task is guaranteed to get the required time by that time. However, with Inequality 3, the cumulative work load must be satisfied at all candidate points to make a task EDF schedulable under non-preemptive scheduling strategy. Traditionally, feasibility tests are performed by testing schedulability of individual task, starting from the task with shortest period/deadline and continue in that order with HPF-EDF.

The idea of determining task set feasibility with lowest priority task has been discussed in (Min-Allah *et al*., 2013) for the preemptive scheduling for static priority systems. However, for the non-preemptive systems, to the best of our knowledge, the feasibility analysis of non-preemptive case with lowest priority first approach has

not been discussed in the literature. Authors in (Min-Allah *et al*., 2013) answered the in-feasibility problem of preemptive systems for fixed prior-ity tasks using HPF approach. The work done in (Min-Allah *et al*., 2013) is more appealing when the system infeasible, otherwise all *n* tasks have to be analyzed for the feasible system. On the other hand, for non-preemptive case, the set of scheduling points is vital for the analysis of non-preemptive EDF schedulability. In this way, the complexity of non-preemptive EDF test is much higher than the preemptive counter part. Each task in our system is assigned a priority that can change at run time due to the nature of EDF. The highest priority task has the shorted deadline. The CPU is also released in favor of highest priority task when task are competing for the CPU at at any time *t*. The likelihood of missing a deadline by task is higher when the task has lower priority and hence lowest priority task is very likely to miss its respective deadline. In priority based scheduling of tasks/processes, a lower priority task/process can only be allocated CPU slots when is no higher priority tasl/process in the system queue to be scheduled on a uni-processor system.

Assuming a point t_1 where a task τ_i is schedulable, a lower priority task τ_j may alsobe feasible at the same point in time. However, vice versa is not true as the cumulative demand increases with lower priority tasks. Since the schedulability of a task is tested at a set of scheduling pints and the following relationships exist among sets of points. Unlike the fixed priority scenario, where any *t* that satisfies the commutative workload is enough to declare task schedulability, all points in set S_i must satisfies the workload incase of non-preemptive EDF. This requirement makes each and every point in S_i schedulability must be analyzed respectively at all individual points. It can be seen tha a set S_{i+1}, consist of set S_i as well as the additional points constituted due to p_{i+1}. It is very likely that if a system is infeasible, it due to lower priority tasks they can only run when system is idle and system will be never idle in case of infeasible system under our assumptions made in Section 2. It is worth mentioning that EDF test depends only on the utilization bound in preemptive case, while in non-preemptive case, in addition to utilization bound it also needs check feasibility of a task at all candidates scheduling points. We now establish a connection between two consecutive tasks from feasibility perspective.

Theorem 3.1

Under non-preceptive EDF, a task τ_{i+1} is always infeasible when τ_i is unschedulable on a uni-processor system.

Proof

Since there are common points in set S_i and S_{i+1}, we compare the workload of τ_i and τ_{i+1} on same points $t' : t' \in S_i \bigcup S_{i+1}.\tau_{i+1}$. As $\sum_{i=0}^{n} \frac{c_i}{p_i} \leq 1$ is obvious term for both task, we compare the addition part which is due to non-preemptive component of the scheduling theory i.e., $\forall t, \left(p_1 \leq t \leq p_i\right) : \left(c_i + \sum_{k=1}^{i-1} \left\lfloor \frac{t}{p_k} \right\rfloor c_k \leq t.\right.$ This comparison straight way shows that when τ_{i+1} is feasible at a point t' while τ_{i+1} is unschedulable at same point t', then it shows $c_{i+1} \leq 0$ which is not possible as τ_{i+1} can not have negative workload. This completes the proof.

The above theorem advocates that when a scheduling point is unable to satisfy schedulability of a higher priority task then it can not accommodate the workload due to a low priority task as well. This is understandable as the workload increases with low priority task as CPU can be assigned to such tasks when there is no pending higher priority task in the run queue. We extend the above formulation to independent task set consisting of *n* tasks to be scheduled non-preemptively under EDF. Our proposed solution test task schedulability in reverse order starting with lowest priority first task such that any un-schedulable task encountered in the process determines in in-feasibility of the entire task set. When determining the feasibility of the task set, the following theorem applies to non-preemptive EDF case.

Theorem 3.2

Under non-preemptive case, τ is feasible with EDF if:

$$\sum_{i=0}^{n} \frac{c_i}{p_i} \leq 1,$$
$$\forall_i, \left(n \geq i < 1\right)\right), \forall t, \left(p_1 \leq t \leq p_i\right) : c_i + \sum_{k=1}^{i-1} \left\lfloor \frac{t}{p_k} \right\rfloor c_k \leq t, \tag{4}$$

where, $t \in S_1 = lp_j; \forall j, (j = 1, \ldots, i); \forall l, \left(l = 1, \ldots, \lfloor p_i / p_j \rfloor\right)$.

Proof

The proof follows directly from the Theorem 3.1.

It is clear from Theorem 3.2 that feasibility is tested for the periodic tasks set starting with task τ_n. When τ_n is infeasibly, the test stops as one schedulable task makes the overall system infeasible. Incase, τ_n is schedulable under EDF, the schedulability of next test τ_{n-1} is determined and so on. Theorem 3.2 has the advantage of determining system in-feasibility much faster than the existing counter part (Jejurikar and Gupta, 2005) as lower priority task are prone to deadline miss. It should be noted that our test just explores another dimension of feasibility analysis in the domain of non-preemptive

dynamic scheduling and the timings parameters associated with a task are kept intact. When the system is declared feasible, the scheduling of the task will follow the EDF policy where highest priority tasks will be executed first followed by the next priority and so on in the decreasing priority order. Incase the task set is EDF feasible under non-preemptive case, the computational cost of our test is the same as Theorem 2.2. This is due to the fact that our solution starts with lowest priority task τ_n and continue till τ_1, the highest priority task and hence behavethe same. However, when the task set in infeasible according to non-preemptive EDF, the test finds task set infeasibility way faster than the highest priority first counterpart. We represent our work by Non-Preemptive Lowest Priority First EDF (NP-LPF) while Theorem 2.2 by Non-Preemptive Highest Priority First EDF (NP-HPF) and study their behavior in the following section.

Experimental Results and Analysis

Experimentation is done in Matlab and uniform distribution of 70% utilization for individual tasks has been used while creating task parameters. As a second step for creating tasks set, the corresponding task utilization is distributed among task such that the system utilization does not exceed the overall limit. Task periods and computation demands are also obtained using uniform distribution. First in Fig. 1, we kept utilization up to 80% as experiments show that below this point, almost all the task set are feasible non-preemptively under EDF and hence both techniques share the same trend. Since the system is under utilized in Fig. 1, behavior of both techniques is almost the same. But as there is a good chance that a few lower priority tasks can miss the deadline at such utilization due to non-preemptive constraints and as highlighted in previous section, there are low priority tasks. This arrangement results in early conclusion for infeasible task set by NP-LPF and hence results are slightly better as compared to NP-HPF. Due to non-preemptive nature, some tasks start missing the deadlines at 80% utilization, especially the low priority tasks and hence this situation favors our approach to conclude system infeasibility faster. A similar study for preemptive RM has been made in (Min-Allah *et al*., 2013) which provides the foundation for the current work. Since non-preemptive case is more pessimistic as compared to preemptive scheduling and it is very likely that randomly generated tasks will start missing deadlines much early. NP-HPF takes all task in decreasing order and test schedulability at all scheduling points and hence take more time before feasibility is determined. NP-HPF uses the highest priority first approach and continue till last task in the set when the task set is feasible or can stop whenever an unschedulable task is determined in the process. NP-HPF might need to analyse all tasks when system utilization is low as the overall task set can be schedulable with lower utilization. Thus, both NP-HPF and NP-LPF show similar behavior with slight improvement in favor of NP-LPF which is due to presence of no unschedulable tasks, especially when task size increases.

Figure 2 shows the behavior of both approaches when system utilization is increased to 90% which is very high system utilization for non-preemptive scheduling. At such high utilization, there is a fair chance that majority of the tasks are infeasible according to non-preemptive EDF scheduling. In Fig. 2, NP-HPF has to test significant number of higher priority tasks before it reached the unschedulable one.

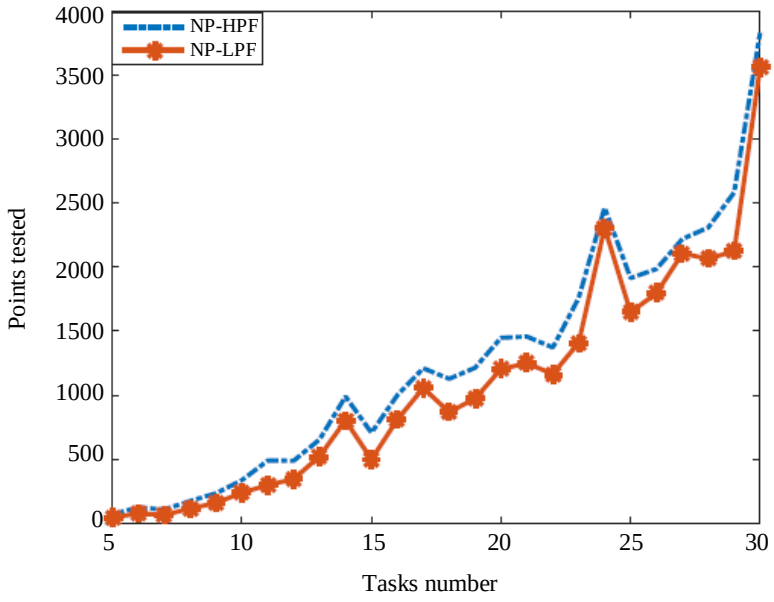

Fig. 1: Number of scheduling points needed for determining systemfeasibility at 80% system utilization

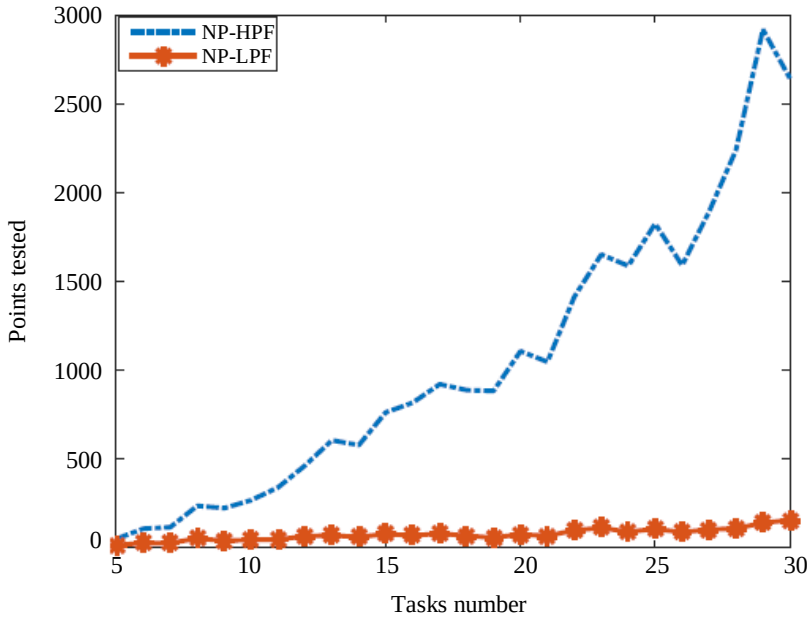

Fig. 2: Number of scheduling points needed for determining system feasibility at 90% system utilizationis

NP-LPF, on the other hand, only takes the lowest priority task but the computation cost is mainly due to he largest set of scheduling pints as S_n is the superset of all scheduling points sets constituted due to all higher priority tasks. Our solution has the advantage of determining system infeasibility much early and does not need to test schedulability of all tasks. It can be seen that with lower utilization, our system behaves like NP-HPF but suppresses NP-HPF when system utilization is higher which is the intended purpose of Theorem 3.2.

Conclusion

The effect of testing task feasibility of non-preemptive systems with lowest priority approach was analyzed and results were established using priority first approach. Under various system utilizations, comparisons were made with lowest priority first approach. The results were more promising when EDF feasibility was tested for a task set with higher utilization. It was also showed that the task infeasibility was due to the deadline miss of the task with largest task period and not because of the smallest one. It was concluded that starting feasibility with highest priority task is superior to the highest priority first approach, especially when system utilization is high. As a future work, infeasibility of dependent task sets under dynamic priority scheduling class should be considered in addition to the hybrid test that can combine both inexact and exact feasibility conditions.

Acknowledgment

The author expresses since thanks to the generous funding from the Deanship of Scientific Research, Imam Abdulrahman Bin Faisal University IAU), under project numbered 2019-359-CSIT. Thanks to Farman Ullah Jan and other colleagues at IAU for their helpful discussions on initial draft of this work. Special thanks to the reviewers for their thoughtful comments.

Ethics

All the experimental and theoretical results reported in this study were achieved by keeping in view the standard ethics practices.

References

Alrashed, S., J. Alhiyafi and S.A. Min-Allah, 2017. An efficient schedulability condition for non-preemptivereal-time systems at common scheduling points. J. Super Comput. 72: 4651-4661. DOI: 10.1007/s11227-016-1751-6

Audsley, N.C., A. Burns, R.I. Davis, K.W. Tindell and A.J. Wellings, 1995. Real-Time System Scheduling. In: Predictably Dependable Computing Systems, Randell, B., J.C. Laprie, H. Kopetz and B. Littlewood (Eds.), Springer, pp: 41-52.

Baruah, S., S. Funk and J. Goossens, 2003. Robustness results concerning EDF scheduling upon uniformmultiprocessors. IEEE Transact. Comput., 52: 1185-1195.

Burns, A., R.I. Davis, P. Wang and F. Zhang, 2012. Partitioned EDF scheduling for multiprocessors usinga C = D Scheme. Real-Time Syst. J., 48: 3-33.

Davis, R.I. and A. Burns, 2011. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surveys, 43: 1-44.

George, L., N. Riverre and M. Spuri, 1996. Preemptive and non-preemptive real-time uniprocessor scheduling. Research Report 2966, INRIA, France.

Guan, N., D. Qingxu, G. Zonghua, X. Wenyao and Y. Ge, 2008. Schedulability analysis of preemptive andnon-preemptive EDF on partial runtime-reconfigurable FPGAs. ACM Trans. Des. Autom. Electron. Syst., 13: 1-43.

Jeffay, K., D.F. Stanat and C.U. Martel, 1991. On non-preemptive scheduling of periodic and sporadic tasks. Real-Time Syst. Symposium, 1: 129-139.

Jejurikar, R. and R. Gupta, 2005. Energy aware non-preemptive scheduling for hard real-time systems. Procendings of the 17th of Euromicro Conference on Real-Time Systems, (RTS' 05), pp: 21-30.

Khan, S.U. and N. Min-Allah, 2011. A goal programming based energy efficient resource allocation in datacenters. J. Super Comput., 61: 502-519.

Krishna, C.M. and K.G. Shin, 1997. Real-Time Systems. 1st Edn., McGrawHill.

Liu, C.L. and J.W. Layland, 1973. Scheduling algorithms for multiprogramming in a hard real-time environ-ment. J. ACM, 20: 40-61

Liu, J.W.S., 2000. Real Time Systems. 1st Edn., Prentice Hall.

Min-Allah, N., 2019. Effect of ordered set on feasibility analysis of static priority system. J. Super Comput., 75: 475-487.

Min-Allah, N., M.B. Qureshi, S. Alrashed and F. Rana, 2019. Cost efficient resource alloca-tion for real-time tasks in embedded systems. Sustainable Cities Society.

Min-Allah, N., S.U. Khan, N. Ghani, J. Li and L. Wang *et al*., 2012. A comparative study of rate monotonicschedulability tests. J. Super Comput., 59: 1419-1430.

Min-Allah, N., S.U. Khan, X. Wang and A.Y. Zomaya, 2013. Lowest priority first based feasibility analysis ofreal-time systems. J. Parallel Distributed Comput., 73: 1066-1075.

Min-Allah, N., X. Jiansheng and W. Yongji, 2010. Utilization bound for periodic task set with composite-deadline. J. Comput. Electrical Eng., 6: 1101-1109.

Nasri, M. and M. Kargahi, 2014. Precautious-RM: A predictable non-preemptive scheduling algorithm forharmonic tasks. Real-Time Syst., 50: 548-584.

Nasri, M., 2017. On flexible and robust parameter assignment for periodic real-time components. ACM SIGBED Rev., 14: 8-15.

Sha, L., T. Abdelzaher, K. Erzen, A. Cervin and T. Baker *et al*., 2004. Real-time scheduling theory: A historical perspective. Real-Time Syst., 28: 101-155

Swaminathan, V. and K. Chakrabarty, 2005. Pruning-based, energy-optimal, deterministic I/O device sched-uling for hard real-time systems. ACM Trans. Embedded Comput. Syst. 4: 141-167.