Hindawi Publishing Corporation
Modelling and Simulation in Engineering
Volume 2014, Article ID 705929, 11 pages
http://dx.doi.org/10.1155/2014/705929

Research Article

Utilization Bound Scheduling Analysis for Nonpreemptive Uniprocessor Architecture Using UML-RT

S. Ewins Pon Pushpa[1] and Manamalli Devasigamani[2]

[1] *Department of Electronics Engineering, Madras Institute of Technology, Anna University, Chennai 600 044, India*
[2] *Department of Instrumentation Engineering, Madras Institute of Technology, Anna University, Chennai 600 044, India*

Correspondence should be addressed to S. Ewins Pon Pushpa; ewinspon2000@yahoo.co.in

Received 28 June 2013; Revised 4 October 2013; Accepted 18 October 2013; Published 13 February 2014

Academic Editor: Zeki Ayag

The key for adopting the utilization-based schedulability test is to derive the utilization bound. Given the computation times, this paper proposes two utilization bound algorithms to derive interrelease times for nonpreemptive periodic tasks, using a new priority scheme, "Rate Monotonic Algorithm-Shortest Job First." The obtained task set possesses the advantage of Rate Monotonic Algorithm and Shortest Job First priority scheme. Further, the task set is tested for schedulability, by first deriving a general schedulability condition from "problem window" analysis and, a necessary and sufficient schedulability condition for a task to be scheduled, at any release time are also derived. As a technical contribution, success ratio and effective processor utilization are analyzed for our proposed utilization bound algorithms on a uniprocessor architecture modeled using UML-RT.

1. Introduction

Tasks in a real-time system must be executed within its deadline to produce functionally correct results in a timely manner. This implies that the timing requirements of tasks, submitted to the real-time system, are known [1]. Each task in a real-time system is scheduled depending on the characteristic of the scheduling algorithm. One of the most widely used uniprocessor scheduling algorithm for periodic real-time tasks is the Rate Monotonic Algorithm (RMA), proposed by Lehoczky et al. in [2]. Let $(\tau = \tau_1, \tau_2, \tau_3, \ldots, \tau_m)$ denote the task set of "m" tasks in a periodic system. For each task τ_i, task utilization U_i be the ratio of computation time to its interrelease time: $U_i = C_i/T_i$. Then, system load $U(\tau)$ of a periodic task system be the sum of all utilization of the tasks in τ : $U(\tau) = \sum_{\tau_i \in \tau} U_i$. Rate Monotonic Algorithm (RMA) assigns priority—smaller the period, higher the priority, and each task has its own unique priority [3]. According to [4], set of "m" independent preemptive periodic tasks are schedulable using RMA if and only if the system load $U(\tau)$ is given by the following:

$$U(\tau) \leq m\left(2^{1/m} - 1\right). \tag{1}$$

Later, a minimum utilization bound of RMA for "m" number of preemptive tasks on single processor was found to be in [5] given by the following:

$$U(\tau) \leq (m-1)\left(2^{1/(m-1)} + \frac{2}{p} - 1\right), \tag{2}$$

where p is ratio between any two periods, which is less than 2.

There are two types of scheduling strategies: nonpreemptive scheduling and preemptive scheduling. In nonpreemptive strategy, tasks started to execute, would not preempt in an occurrence of higher priority tasks. A higher priority tasks will preempt any lower priority tasks in preemptive scheduling. However, nonpreemptive scheduling is being preferred because they are easier to implement and have lower run-time overhead than preemptive scheduling [6].

There is a growing interest in using the object paradigm for developing real-time software [7]. The dominance of software in real-time embedded systems design caused interest in methodologies with widely accepted notations in the software community, such as the Unified Modeling Language (UML). The Unified Modeling Language (UML)

is a modeling language for capturing the specification and the implementation using the graphical notations in software community [4]. This approach describes the specification graphically, promotes correctness of the specification, and generates the test scenarios for the implementation automatically. This paper contributes an object-oriented platform-based design, for uniprocessor architecture using UML-RT to analyze the schedulability for the proposed utilization bound algorithms based on a new priority scheme "Rate Monotonic-Shortest Job First" (RM-SJ).

In Section 2, related works to the proposed method is discussed. Section 3 describes the system model using UML-RT and notations used. Condition for schedulability of a task and general schedulability test are discussed in Sections 4 and 5, respectively. Proposed priority scheme and utilization bound algorithms are discussed in Section 6. A necessary and sufficient schedulability condition for a task to be scheduled, at any release time, is derived in Section 7. Section 8 deals with performance evaluation for the proposed algorithms. Finally, Section 9 concludes with results analyzed.

2. Related Works

For single-processor scheduling, Jeffay et al. [6] presented an exact schedulability test of pseudopolynomial time-complexity for a periodic or sporadic task set under nonpreemptive EDF scheduling on a single-processor. George et al. [8] established exact schedulability tests for both nonpreemptive EDF and nonpreemptive fixed-priority scheduling on a single processor with pseudopolynomial time-complexity and addressed general task models in which relative deadlines and periods are not necessarily related. Baruah et al. [3] addressed schedulability analysis for nonpreemptive recurring tasks, which is the general form of nonpreemptive sporadic tasks, and showed that the nonpreemptive schedulability analysis problem can be reduced to a polynomial number of preemptive schedulability analysis problems.

The admission control is an algorithm that depends on the scheduling policy and ensures that only tasks that will meet their deadlines are accepted into the system [9]. Accordingly, general bound is derived and used for admission control of RMS-based systems that only requires information about the number of tasks to be admitted in the system. If more information about the tasks characteristics is analyzed, a better admission control can be obtained. For example, the exact characterization described by Lehoczky et al. in [2] uses computation time (C_i) and interrelease time (T_i) values for admission control. In [10], Burchard et al. used knowledge about the periods of the tasks to obtain better admission control for RMS tasks. Clearly, using more information increases the complexity of the admission control since there are more variables to take into consideration during admission control. In [11], Lu et al. derived a schedulability bound for nonpreemptive Rate Monotonic scheduling by using period ratio of the tasks to provide a guarantee that tasks will meet their deadlines. When uniprocessor priority scheme is considered, Liu and Layland [4] showed that Rate Monotonic priority ordering (RMPO) is optimal for independent synchronous periodic tasks. Leung and Whitehead [12] showed that deadline monotonic priority ordering (DMPO) is optimal for independent synchronous tasks with constrained deadlines. Audsley [13] devised an optimal priority assignment (OPA) algorithm that solved the problem of priority assignment for asynchronous tasksets, and for tasks with arbitrary deadlines.

In [1], Guan et al. addresse schedulability test conditions for non-preemptive-fixed priority (NP-FP), building upon the "problem window analysis" for preemptive scheduling for multiprocessors. First, linear-time general schedulability test condition that works not only on NP-FP, but also any other work-conserving nonpreemptive scheduling algorithm is derived; then an analysis is done on a test condition of quadratic time-complexity for NP-FP, which has significant performance improvement comparing to the first one. In [14] Bertogna et al. derived a general method for schedulability conditions for multiprocessor real-time systems in a "problem window" and then it is tightened and analyzed for two algorithms, namely, Earliest Deadline First (EDF) and Fixed Priority (FP).

This concept of "problem window" analysis done in multiprocessors is generalized on uniprocessor environment for which task schedulability condition and schedulability at any release time for a periodic task are derived. Further two utilization bound algorithms are proposed using a new priority assignment scheme "Rate Monotonic-Shortest Job" (RM-SJ) and task schedulability is analyzed.

3. System Model Using UML-RT and Notations

Consider a work conserving nonpreemptive uniprocessor real-time system. A task set, $(\tau = \tau_1, \tau_2, \tau_3, \ldots, \tau_m)$ where, there are "m" tasks, each task $\tau_k = (C_k, T_k)$ is characterized by two parameters—computation time C_k and interrelease time or period T_k, with an interpretation that tasks are generated in every integer multiples of interrelease time to execute C_k units before the next integer interrelease time T. All tasks generated are independent and are executed on a nonpreemptive uniprocessor architecture modelled using UML-RT. The notations used are described in Table 1.

3.1. UML-RT. UML-RT is a modeling tool for real-time, developed by the Rational Corporation. UML-RT uses Unified Modeling Language (UML) to express the original ROOM (Real-Time Object-Oriented Modelling) concepts and extensions. It includes constructs for modelling both behaviour and structural concepts of event-driven real-time systems. Rational Rose Real time is a software development environment tailored to the demands of real time software. It is used to create models of the software system based on the UML constructs, to generate code, compile, and then run and debug the application. Rational Rose Real time can be used through all phases of the software development lifecycle, through design, implementation, test, and final deployment. UML-RT profile introduces a set of common scheduling annotations which are fairly sufficient to perform schedulability analysis [15]. Using Rational Rose Real time developers

Table 1: Notations and description.

Notations used	Description
τ	A task set
m	Total number of tasks in τ
τ_1	Highest priority task
τ_m	Lowest priority task
τ_k	A task in task set
C_k	Computation time of τ_k
T_k	Interrelease time or period of τ_k
$\varphi_k = T_k - C_k$	Problem area of the task τ_k, analyzed for schedulability. A necessary condition for the deadline miss to occur for τ_k is that the worst-case work load, $\dot{W}(k)$ in the problem area φ_k by all other tasks in the task set τ except τ_k is no less than φ_k.
Cs_k	Computation time lesser than C_k
C	Nonzero computation time
$U_k = C_k/T_k$	Task utilization of τ_k
$U(\tau) = \sum_{\tau_i \in \tau} U_i$	System load
hp(k)	Set of all higher priority tasks of τ_k
lp(k)	Set of all lower priority tasks of τ_k
$C_{\text{hp}}(k)$	Set of all computation times of hp (k)
$C_{\text{lp}}(k)$	Set of all computation times of lp (k)
C_m	Maximum computation time in the subset of $C_{\text{lp}}(k)$
hc(k)	Summation of computation times of all higher priorities of τ_k
hec(k)	Summation of computation times of all equal and higher priorities of τ_k
lc(k)	Summation of computation times of lower priorities of τ_k
lec(k)	Summation of computation times of equal and lower priorities of τ_k
$C_{\text{hp}}(k)$	Set of all computation times of hp (k)
$C_{\text{lp}}(k)$	Set of all computation times of lp (k)
$U_{\text{hp}}(k)$	Summation of all the task utilization of hp (k)
$U_{\text{elp}}(k)$	Summation of equal and all task utilization of lp (k)
$r_n(k)$	nth release time of τ_k
$r_{n+1}(k)$	$(n+1)$th release time of τ_k and deadline for the task released at $r_n(k)$
$l_n(k)$	Latest feasible start time for τ_k released at $r_n(k)$, to start execution in order to meet its deadline.
$N_{\tau_i}(k)$	Number of intermediate tasks in φ_k
w	Work done by initial job, intermediate job or final job in φ_k
$W\tau(k)$	Worst-case latency of τ_k. It is the maximum time lapse for a task to start execute.
$\dot{W}(k)$	Total work done by other tasks in the problem area of φ_k

can execute, test, and debug models at the modelling language level using visual observation tools.

In UML-RT the four principle constructs for modelling structure are capsules, ports, protocols, and connectors [5].

3.1.1. Capsules. The fundamental modelling constructs of UML-RT are capsules. Capsules are distributed architectural active objects that interact with other capsules exclusively through one or more ports. The behavior of a capsule is modelled in the state transition diagram that can process (send and receive) messages via their ports, while its (hierarchical) structure is modelled in the capsule structure diagram. The structure diagram for the proposed model using UML-RT is shown in Figure 1.

3.1.2. Ports. Through ports, messages are sent and received to and from capsule instances. Ports connected to a state machine of a capsule (end port) can handle messages sent to them. Capsules and its associated input and output ports are shown in Table 2.

3.1.3. Protocols. Protocols define a set of messages exchanged between a set of capsules. Important timer ports and messages used in capsules are shown in Table 3.

3.1.4. Connectors. Connectors capture the key communication relationships between capsule roles. They interconnect capsule roles that have similar public interfaces through ports.

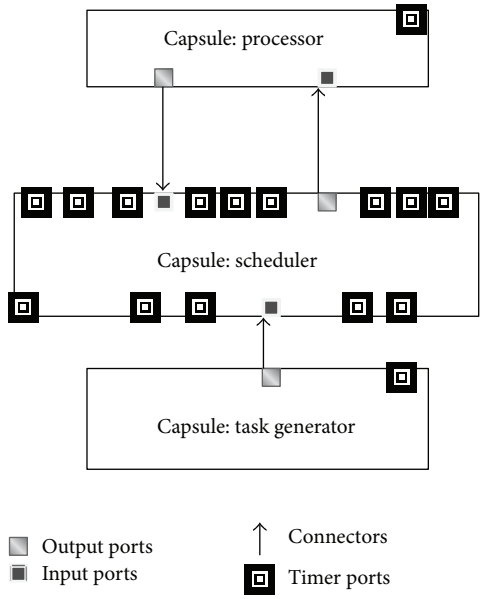

Figure 1: Structure diagram for the model proposed in UML-RT.

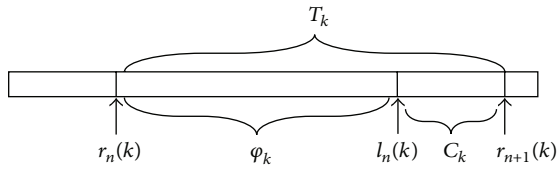

Figure 2: Problem area φ_k, for the task τ_k.

Table 2: Capsules and associated ports.

Capsules	Input ports	Output ports
Task generator	—	Gout
Scheduler	Sin	Sout
Processor	Pin1	Pout1

4. Condition for Schedulability of a Task

A necessary condition for deadline miss to occur for τ_k, is that the worst-case work load, $\dot{W}(k)$ by all other tasks in the task set τ except τ_k is no less than φ_k. $l_n(k)$ is the latest feasible start time for τ_k, to start execution in order to meet its deadline as shown in Figure 2.

The aim is to compute the individual contribution of worst-case workload by other tasks, that is, $\tau_i \in \tau$, in the problem area, φ_k as shown in Figure 2. The work load in φ_k can be categorized into three parts as shown in Figure 3.

Initial Job. The contribution of task τ_i, in problem area φ_k, is when computation of task τ_i started earlier or at $r_n(k)$ and having deadline in the problem area.

Intermediate Job, $N_{\tau_i}(k)$. The computation of a task τ_i started in φ_k and having deadline in the problem area earlier to $l_n(k)$.

Final Job. The contribution of task τ_i in problem area φ_k is when computation of task τ_i started in φ_k and having deadline later or at $l_n(k)$. Assume that all tasks contribute maximum in the problem area. This is a pessimistic but safe approximation to count the workload of a task in the problem area [3].

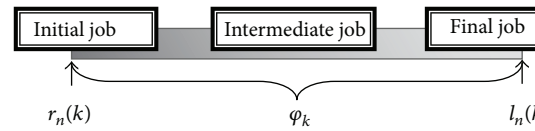

Figure 3: Work done by other tasks in φ_k.

Table 3: Capsules used timer ports and messages.

Capsules	Timer ports	Protocols
Task generator	initial	trigger
Scheduler	Task A	Pa
	Task B	Pb
	Task C	Pc
	Task D	Pd
	Task E	Pe
	Task F	Pf
	Task G	Pg
	Task H	Ph
	Task I	Pi
	Task J	Pj
	Task K	Pk
	Task L	Pl
	Task M	Pm
	Task N	Pn
Processor	exetask1	pfree1

For $\tau(k)$ to be scheduled,

$$\dot{W}(k) \leq \varphi_k, \tag{3}$$

where

$$\dot{W}(k) = \sum_{\mathrm{hp}(k),\,\mathrm{lp}(k)} w\,(\text{initial jobs} + \text{intermediate jobs} + \text{final jobs}). \tag{4}$$

5. General Schedulability Test

The entire task set is schedulable only when each task is schedulable. The computation timings of tasks are arranged, $C_1 \leq C_2 \leq C_3 \leq \cdots \leq C_k \leq \cdots \leq C_m$.

Theorem 1. *Highest priority task τ_1 is schedulable by a work conserving nonpreemptive scheduling algorithm if*

$$\varphi_1 \geq C_m. \tag{5}$$

Proof. By contradiction if, $\varphi_1 < C_m$, then the highest priority task τ_k may miss its deadline. We recall that C_m is the maximum computation time in the subset of $C_{\mathrm{lp}}(1)$. For a nonpreemptive architecture, any $\tau_j \in \mathrm{hp}(1)$ will not be preempted, if it starts to execute before the release time of any

higher priority task. Consider that $j = m$. From Figure 4, it is shown that τ_j, which started to execute C_m at $r_{n-1}(j)$, is still executing beyond $l_n(1)$ causing the task τ_1 to miss deadline.

Therefore, if $\varphi_1 \geq C_m$, then τ_1 will be schedulable. □

Theorem 2. *A task τ_k is schedulable by a work conserving nonpreemptive scheduling algorithm if*

$$1 - \frac{C_m}{\varphi_k} + U_{elp}(k) \geq U(\tau). \tag{6}$$

Proof. In [3], an algorithm is work conserving, if there are no idle processors, when a ready task is waiting. Schedulability condition for a task τ_k is given by (3) and (4). The final job contribution is bounded in the second term RHS of (7). Therefore, from (4) the general schedulability for τ_k will be

$$\varphi_k \geq C_m + \sum_{\mathrm{hp}(k)} w(\text{intermediate jobs}), \tag{7}$$

$$\varphi_k \geq C_m + \sum_{\tau_i \in \mathrm{hp}(k)} \frac{\varphi_k}{T_i} C_i. \tag{8}$$

Equation (8) indicates pessimistic and sufficient condition for a task to be schedulable. A task will be scheduled in the task set if (8) is satisfied. Therefore,

$$\varphi_k \geq C_m + \varphi_k U_{\mathrm{hp}}(k), \tag{9}$$

$$\begin{aligned} \varphi_k &\geq C_m + \varphi_k \left(U(\tau) - U_{\mathrm{elp}}(k)\right), \\ \varphi_k - C_m &\geq \varphi_k \left(U(\tau) - U_{\mathrm{elp}}(k)\right), \end{aligned} \tag{10}$$

dividing by φ_k,

$$1 - \frac{C_m}{\varphi_k} + U_{\mathrm{elp}}(k) \geq U(\tau). \tag{11}$$

Equation (11) is the condition for a task τ_k to be schedulable.

Sufficient schedulability condition for least priority task from (9),

$$\varphi_k \geq \varphi_k U_{\mathrm{hp}}(k) \tag{12}$$

C_m, is neglected in (12), because schedulability condition for τ_m to execute C_m is analyzed. □

6. Proposed Utilization Bound Algorithms

6.1. Rate Monotonic Algorithm-Shortest Job First (RMA-SJF). In shortest job first (SJF), priority is given to shortest computation time, independent of interrelease time of the tasks, thus minimizing the waiting time and improving the throughput [6]. Rate monotonic algorithm (RMA) is of great practical importance because priority is given to tasks with lesser release time. This paper proposes two utilization bound algorithms to derive a task set which satisfies both RMA-SJF priority scheme; that is, higher priority tasks will have lesser computation time and lesser interrelease time.

When a set of computation times are given, our aim is to design a work conserving task system, to utilize the available processing capacity. To achieve maximum processing capacity, system load must be equal to unity for a uniprocessor system [8]. In [3], an algorithm is work conserving, if processor is not idle, when a ready task is waiting. Therefore, behavior of the system developed is strongly dependent upon the periods of the task comprising the task set [16].

6.2. Method 1: Work Conserving Utilization Bound. Initially the algorithm aims to schedule "$m - a$" tasks, by deriving the highest priority task release time $T_1 = \mathrm{hec}(m - 1)$. Schedulability of each task is analyzed using the general schedulability test. Later T_i of tasks are varied to find an optimized utilization bound in $0.9 < U(\tau) \leq 1$ to schedule "m" tasks utilizing the processor effectively (see Algorithm 1).

6.3. Method 2: Worst-Case Utilization Bound. In method 2, the worst-case latency of each task is considered to compute its interrelease time. Worst-case response time ($W\tau$) of each task is found at critical instant, where all tasks are released instantaneously (see Algorithm 2).

7. Sufficient Schedulability Condition at Any Release Time

Sufficient schedulability condition for $\tau_k \in \tau$ in its release time $r_n(k)$ is derived. Let φ_k be the problem area of τ_k shown in Figure 5, and $l_n(k)$ is the latest feasible start time to execute C_k, for τ_k to be schedulable.

Consider the worst-case latency analyzed in Section 6.3. The contributions of worst-case work load of other tasks are analyzed under two cases.

Case A. $\mathrm{hp}(k)$, $\mathrm{lp}(k)$ released before $r_n(k)$.

Case B. $\mathrm{hp}(k)$, $\mathrm{lp}(k)$ released in $[r_n(k), l_n(k)]$.

7.1. Case A: hp(k), lp(k) Released before $r_n(k)$

7.1.1. hp(k) Released before $r_n(k)$. Figure 5 shows $\tau_j \in \mathrm{hp}(k)$ released at $r_{n-1}(j)$ before $r_n(k)$:

$$N_{\tau_i}(k) = 0; \tag{13}$$

Lemma 3. *Task $\tau_j \in hp(k)$ released before $r_n(k)$, will contribute initial job or intermediate job, if $r_n(k) < r_{n-1}(j) + W\tau(j) + C_j < l_n(k)$.*

Proof. To prove by contradiction, if $r_{n-1}(j) + W\tau(j) + C_j < r_n(k)$, then this task τ_j does not contribute any workload in work area; therefore, for this condition there is no initial or intermediate job. Therefore, $r_n(k) < r_{n-1}(j) + W\tau(j) + C_j < l_n(k)$ will contribute initial job or intermediate job.

(1) Initial job will be present if, $r_{n-1}(j) + W\tau(j) + C_j - r_n(k) \leq C_j$

(i) $r_{n-1}(j) + W\tau(j) + C_j - r_n(k) = C_j$, then initial job will be C_j;

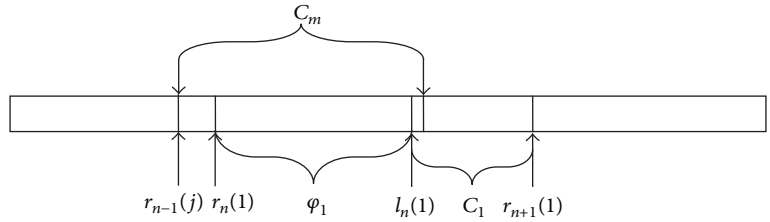

Figure 4: C_m starts to execute before $r_n(1)$ and still executing beyond $l_n(1)$.

Step 1. Computation times are randomly generated and arranged according to shortest job first priority scheme.
$C_1 \le C_2 \le C_3 \le \cdots \le C_k \le \cdots \le C_m$
Initialize $a = 1$;
Step 2. $\text{hec}(m-a) = C(1) + C(2) + \cdots + C(m-a)$
$\text{hec}(m) = C(1) + C(2) + \cdots + C(m)$
Step 3. $l_\delta = \text{hec}(m-a), l_\delta < \text{hec}(m)$
Step 4. assign $T_1 = l_\delta$;
for $i = 2$ to m
do $T_i = T_{i-1} + C_{i-1}$
end for
Step 5. Find $U(\tau) = \sum_{\tau_i \in \tau} U_i$ where $U_i = C_i/T_i$
Step 6. if $0.9 < U(\tau) \le 1$;
Step 6.1. Vary T_i, such that processor is effectively utilized to schedule "m" tasks in $U(\tau)$. Therefore $U(\tau)$ is work conserving utilization bound.
Step 6.2. exit;
Step 7. if $U(\tau) < 0.9$; $l_\delta - -$; go to Step 4.
Step 8. if $U(\tau) > 1$; $l_\delta + +$; go to Step 4.

Algorithm 1

(ii) $r_{n-1}(j) + W\tau(j) + C_j - r_n(k) < C_j$, then $r_{n-1}(j) + W\tau(j) + C_j - r_n(k) = Cs_j < C_j$; therefore, the initial job will be Cs_j.

(2) Intermediate job will be present if, $r_{n-1}(j) + W\tau(j) + C_j - r_n(k) > C_j$.

For the above condition,

$$r_{n-1}(j) + W\tau(j) + C_j - r_n(k) - C_j = C > 0. \quad (14)$$

Recalling that all tasks contribute maximum in the problem area, τ_i is assumed to start execute in φ_k and has deadline in problem area. So, $r_{n-1}(j) + W\tau(j) + C_j - r_n(k) > C_j$ will contribute intermediate job, $N_{\tau_i}(k)$. □

7.1.2. lp(k) Released before $r_n(k)$. Figure 6 shows $\tau_j \in \text{lp}(k)$ released at $r_{n-1}(j)$ before $r_n(k)$. Maximum contribution of any $\text{lp}(k)$ will be the task having maximum computation time as initial job released at $r_{n-1}(j)$ and started to execute before $r_n(k)$. Therefore, initial job = C_m.

7.2. Case B: hp(k), lp(k) Released in $[r_n(k), l_n(k)]$

7.2.1. hp(k) Released $[r_n(k), l_n(k)]$

Lemma 4. *If $\tau_j \in hp(k)$ released $[r_n(k), l_n(k)]$ will contribute intermediate job, if $r_n(j) + W\tau(j) + C_j < l_n(k)$, while τ_j is pending.*

Proof. Recalling that, contribution of the task started to execute in problem area and finishes before $l_n(k)$ is intermediate job.

For

$$(r_n(j) + T(j) = r_{n+1}(j), r_{n+1}(j) < l_n(k), n++) \quad \{N_{\tau_i}(k) + +;\}. \quad (15)$$

The above equation shows that there are $N_{\tau_i}(k)$ intermediate jobs contributed by, $\tau_i \in \text{hp}(k)$ in the problem area while τ_k is pending. □

Lemma 5. *If $\tau_j \in hp(k)$ released $[r_n(k), l_n(k)]$ will contribute final job, if*

$$r_n(j) + W\tau(j) + C_j \ge l_n(k). \quad (16)$$

Proof. Consider Figure 7. By contradiction, if $r_n(j) + W\tau(j) + C_j < l_n(k)$, then by Lemma 4, this condition possesses intermediate job. Therefore, $r_n(j) + W\tau(j) + C_j \ge l_n(k)$ will contribute final job (see Algorithm 3). □

7.2.2. lp(k) Released in $[r_n(k), l_n(k)]$

Lemma 6. *If $\tau_j \in lp(k)$ released $[r_n(k), l_n(k)]$ will have no contribution in φ_k for scheduling τ_k.*

Step 1. Arrange computation times according to shortest job first,
$C_1 \le C_2 \le C_3 \le \cdots \le C_k \le \cdots \le C_m$
Step 2. Worst case response time of tasks analyzed. It is the maximum time lapse for a task to start to execute. So it is known that,
$W\tau(1) = C_m$
∘
$W\tau(k) = \mathrm{hc}(k) + C_m$
∘
$W\tau(m) = \mathrm{hc}(m)$
Step 3. $T_1 = W\tau(1) + C_1$;
Step 4. for $i = 2$ to m do
$T_i = W\tau(i)$
end for
Step 5. Worst case utilization bound, $U(\tau) = \sum_{\tau_i \in \tau} U_i$ where $U_i = C_i/T_i$
Step 6. Check schedulability of "m" tasks by general schedulability test and find success ratio and effective processor utilization.
Step 7. exit.

Algorithm 2

(i) According to the definition of final job,
if $r_n(j) + W\tau(j) + C_j = l_n(k)$, then
$r_n(j) + W\tau(j) - l_n(k) = |C_j|$ therefore the final job will be C_j
else $r_n(j) + W\tau(j) + C_j > l_n(k)$, then
$r_n(j) + W\tau(j) + C_j - l_n(k) = C > 0$;
if $W\tau(j) + C_j - C \ge C_j$ then, final job will be C_j
else final job is $Cs_j = W\tau(j) + C_j - C$

Algorithm 3

Proof. Any $\mathrm{lp}(k)$ released in $[r_n(k), l_n(k)]$, as shown in Figure 7, will have no contribution in φ_k because any $\mathrm{lp}(k)$ is executed. So, $\mathrm{lp}(k)$ released in $[r_n(k), l_n(k)]$ will contribute no workload in φ_k. □

8. Performance Evaluation

Example 7. Given the computation time C_i = (6, 5, 7, 3, 4, 2) the aim is to find optimum interrelease time for each task to achieve 100% success ratio by utilizing the processor effectively by work conserving utilization bound algorithm.

First computation times are arranged in nondecreasing order:

$$C_i = \{2 \le 3 \le 4 \le 5 \le 6 \le 7\}. \tag{17}$$

The various system load values by varying T_i are shown in Table 4. All timings are in seconds.

Maximum utilization bound was found to be 0.9969 for the task set given below: $\tau = \{2/18, 3/20, 4/23, 5/27, 6/33, 7/36\}$. When schedulability of $\tau(6)$ was analyzed, $K = 6$; $C_6 = 7$; $\varphi_6 = 36 - 7 = 29$; $U(\tau) = 0.9964$. Lowest priority task $\tau(6)$will be schedulable if (12) is satisfied:

$$\varphi_k \ge \varphi_k U_{\mathrm{hp}}(k), \qquad 29 \ge 28.898. \tag{18}$$

Though the condition is satisfied, $\tau(6)$ is unschedulable practically because of runtime of the algorithm. The runtime of the algorithm for six tasks is found to be 0.37599 seconds. In order to compensate the runtime, period is incremented and the problem is analyzed practically.

For the task set, $U(\tau) = \{2/18, 3/20, 4/23, 5/27, 6/33, 7/40\}$, utilization bound $U(\tau) = 0.977$, the task set τ is found schedulable using processor effectively as shown in Figure 8.

(i) When $U(\tau) < 0.977$, all tasks are schedulable, using less processor capacity.

(ii) When $U(\tau) > 0.977$ task set is found unschedulable because of runtime of algorithm.

When $U(\tau) = 0.977$ task set is schedulable using processor capacity effectively with 97.5% as showed in Table 5. For the system load $U(\tau) = 0.977$ the effective processor utilization is found to be 97.5.

Extensive simulations are conducted in UML-RT, for the worst-case utilization bound and work conserving unity utilization bound and their performance is compared. The task parameters settings are as follows: for each task computation timings are randomly distributed in {2, 19}; Ti is calculated according to the two algorithms proposed. Maximum of 14 tasks are generated, to compare the success ratio and effective processor utilization. Figure 9 shows the comparison

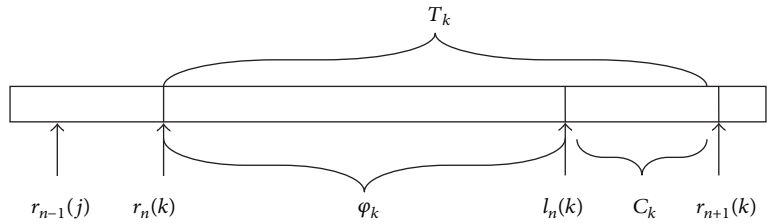

FIGURE 5: $\tau_j \in \text{hp}(k)$ released at $r_{n-1}(j)$ before $r_n(k)$.

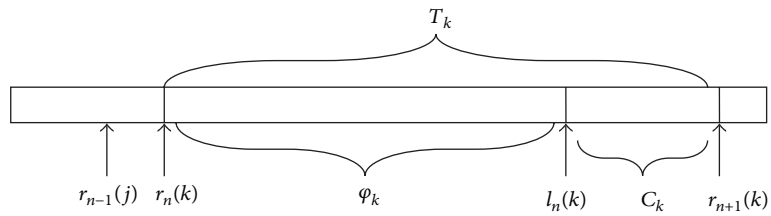

FIGURE 6: $\tau_j \in \text{lp}(k)$ released in $r_{n-1}(j)$ before $r_n(k)$.

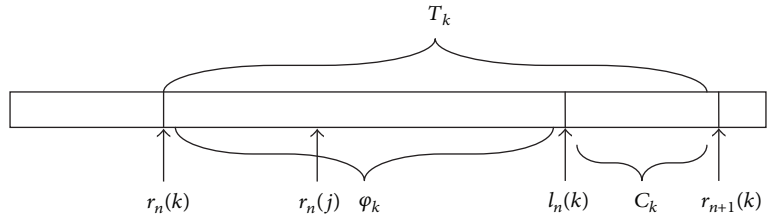

FIGURE 7: $\tau_j \in \text{lp}(k)$ released in $[r_n(k), l_n(k)]$.

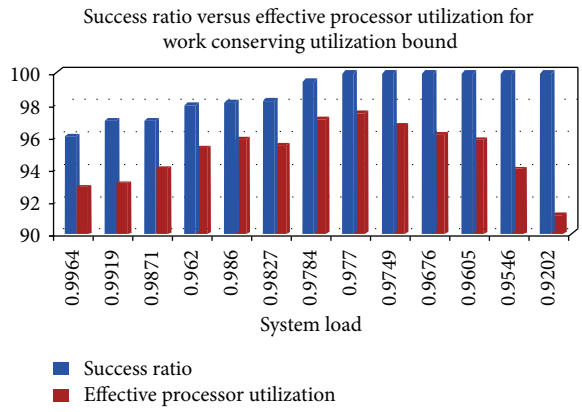

FIGURE 8: Success ratio and effective processor utilization analyzed for example, task set using UML-RT.

TABLE 4: Analysis of work conserving utilization bound algorithm for various set of T_i.

C_i	Interrelease time, T_i analyzed by work conserving utilization bound algorithm										
2	20	19	18	18	18	18	18	18	18	18	18
3	22	21	21	20	20	20	20	20	20	20	20
4	25	24	24	24	23	23	23	23	23	23	23
5	29	28	28	28	28	27	27	27	27	27	27
6	34	33	33	33	33	33	33	32	32	32	32
7	40	39	39	39	39	40	39	40	39	38	37
$U(\tau)$	**0.92024**	**0.95466**	**0.96051**	**0.96766**	**0.9749**	**0.97702**	**0.98152**	**0.98271**	**0.9872**	**0.99192**	**0.9969**

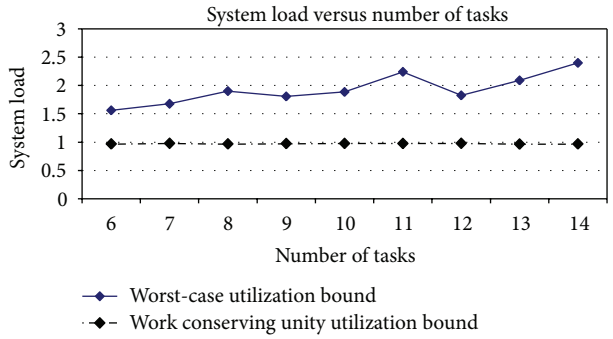

Figure 9: Comparison of utilization bound of two algorithms proposed.

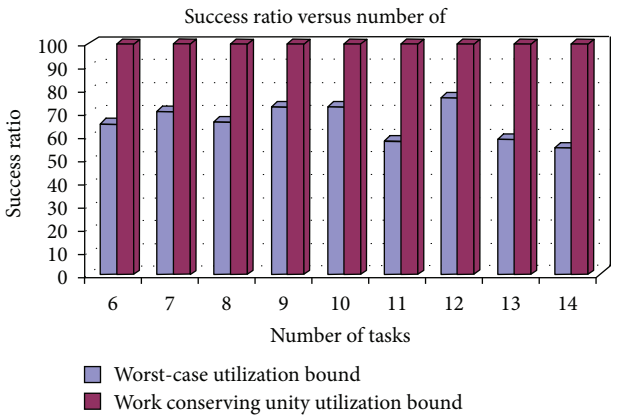

Figure 10: Success ratio versus number of tasks.

Table 5: Success ratio and effective processor utilization for various loads analyzed.

System load	Success ratio	Effective processor utilization
0.9964	96.02	92.83
0.9919	98.5	96.66
0.9871	98.78	96.66
0.962	97.97	95.27
0.986	98.13	95.85
0.9827	98.35	95.5
0.9784	99.52	97.05
0.977	**100**	**97.5**
0.9749	100	96.72
0.9676	100	96.16
0.9605	100	95.83
0.9546	100	94
0.9202	100	91.16

of utilization bound derived for the two algorithms proposed. From Figure 10, the success ratio for work conserving unity utilization bound is high compared to worst-case utilization bound. Using the worst-case workload of task the utilization

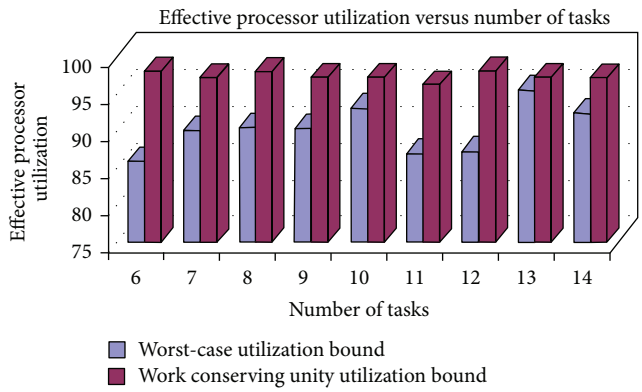

Figure 11: Comparison of Effective processor utilization of the algorithms proposed.

bound or the system load derived is heavily loaded for Worst-Case Utilization Bound Algorithm. Effective processor utilized by the proposed algorithm is shown in Figure 11.

Advantage of Work Conserving Utilization Bound

(i) Processor can be effectively utilized to schedule task set.

Example 8. To find whether $\tau(5)$ is schedulable for the task set having $U(\tau) = 0.9827$ as shown in Table 4 by general schedulability test.

Solution. For $\tau(5)$ to be schedulable, (8) is sufficient condition:

$$\begin{gathered} K = 5; \quad C_5 = 6; \quad C_m = 7; \\ \varphi_5 = 32 - 6 = 26; \qquad U(\tau) = 0.9827, \\ \varphi_k \geq C_m + \sum_{\tau_i \in \mathrm{hp}(k)} \frac{\varphi_k}{T_i} C_i, \\ \varphi_5 \geq C_m + \sum_{\tau_i \in \mathrm{hp}(k)} \frac{\varphi_5}{T_i} C_i, \\ 26 \geq 7 + 16.125, \\ 26 \geq 23.125. \end{gathered} \tag{19}$$

Therefore, $\tau(5)$ is schedulable by general schedulability test.

When schedulability of $\tau(6)$ was analyzed, $K = 6$; $C_6 = 7$; $\varphi_6 = 40 - 7 = 33$; $U(\tau) = 0.9827$. Lowest priority task $\tau(6)$ will be schedulable if (12) is satisfied:

$$\begin{gathered} \varphi_k \geq \varphi_k U_{\mathrm{hp}}(k), \\ 33 \geq 22.6544. \end{gathered} \tag{20}$$

Similarly, all the tasks are tested for schedulability and practically found schedulable using UML-RT.

(ii) To find whether $\tau(5)$ in the task set $\tau = \{(2, 11), (2, 13), (3, 15), (3, 18), (4, 21), (4, 25)\}$ is schedulable in its 54th release.

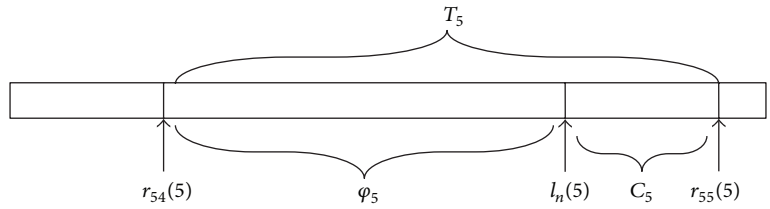

Figure 12: Problem area φ_5, for nonpreemptive uniprocessor scheduling.

Here $k = 5$, 5th task $\dot{W}(k) \leq \varphi_k$ for $\tau_n(k)$ to be scheduled, where

$$\dot{W}(k) = \sum_{\mathrm{hp}(k),\mathrm{lp}(k)} w(\text{initial jobs} + \text{intermediate jobs} + \text{final jobs}) \tag{21}$$

Case 1. 54th release time of the task $\tau(5)$, $r_{54}(5) = 1134$ and $r_{55}(5) = 1155$.

$C_5 = 4$; therefore, $l_{54}(5) = 1155 - 4 = 1151$ and $\varphi_5 = 17$. Figure 12 shows the problem area of $\phi(5)$.

(a) Finding initial job or intermediate job contribution of τ_1

$$r_{54}(5) \,\mathrm{MOD}\, T_1 = 1,$$
$$r_{n-1}(1) = 1134 - 1 = 1133, \tag{22}$$
$$r_n(1) = 1144$$

$W\tau(1) = \max[C_{\mathrm{lp}}(1)]$; therefore, $W\tau(1) = 4$. By Lemma 3, task $\tau_j \in \mathrm{hp}(k)$ released before $r_n(k)$, will contribute initial job or intermediate job, if $r_n(k) < r_{n-1}(j) + W\tau(1) + C_j < l_n(k)$

$$r_{n-1}(1) + W\tau(1) + C_1 = 1133 + 4 + 2 = 1139 \tag{23}$$

here, $r_{54}(5) < r_{n-1}(1) + W\tau(1) + C_1 < l_{54}(5) W\tau(j)$.

τ_1 will contribute initial job if, $r_{n-1}(j) + W\tau(j) + C_j - r_n(k) \leq C_j$, accordingly for $r_{n-1}(1) + W\tau(1) + C_1 - r_{54}(5) \leq C_1$, the condition is not satisfied; therefore, it possesses only intermediate job.

τ_1 will contribute intermediate job if $r_{n-1}(j) + W\tau(j) + C_j - r_n(k) > C_j$.

Here,

$$r_{n-1}(1) + W\tau(1) + C_1 - r_{54}(5) > C_1. \tag{24}$$

$r_{n-1}(1) + W\tau(1) + C_1 - r_{54}(5) - C_1 = 3 > 0$. Therefore, computation of a task τ_1 started in φ_k and having deadline in the problem area, will contribute an intermediate job.

By Lemma 4, task $\tau_1 \in \mathrm{hp}(k)$ released $[r_{54}(5), l_{54}(5)]$ will contribute intermediate job, if

$$r_n(1) + W\tau(1) + C_1 < l_{54}(5) \tag{25}$$

$l_{54}(5) = 1151$, $r_n(1) = 1144$, release time is in the problem area.

$r_n(1) + W\tau(1) + C_1 < l_{54}(5)$ is satisfied because $r_n(1) + W\tau(1) + C_1 = 1150$ and $l_{54}(5) = 1151$. Therefore τ_1 does contribute one intermediate job in φ_1.

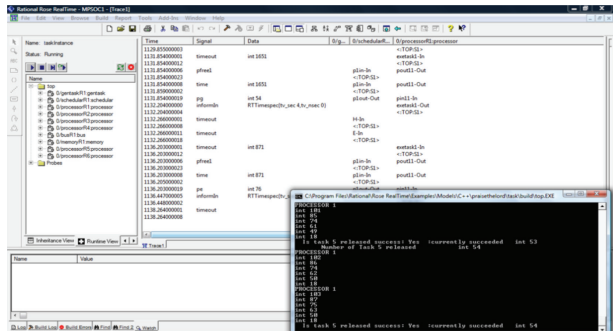

Figure 13: Simulation result in UML-RT.

By Lemma 4, if $\tau_1 \in \mathrm{hp}(5)$ released $[r_{54}(k), l_{54}(k)]$ will contribute final job, if

$$r_n(1) + W\tau(1) + C_1 \geq l_{54}(5). \tag{26}$$

$r_n(1) = 1144$, release time is in the problem area; therefore, τ_1 does not contribute final job in φ_1 because $r_n(1) + W\tau(1) + C_1 < l_{54}(5)$.

Therefore, the final job = 0.

So, the total worst-case workload contribution of τ_1 is $w(1) = 2(\text{intermediate job}) = 4$.

Likewise, the other task contribution on $\phi(1)$ is found out similarly. One has

$$w(2) = 2(\text{intermediate job}) + 1(\text{final job}) = 3,$$
$$w(3) = 2(\text{initial job}) + 3(\text{final job}) = 5,$$
$$w(4) = 3(\text{intermediate job}) = 3,$$
$$w(6) = 0,$$
$$\dot{W}(5) = \Sigma w(\text{initial jobs} + \text{intermediate jobs} + \text{final jobs}),$$
$$\mathrm{hp}(k), \quad \mathrm{lp}(k)$$
$$\dot{W}(5) = 4 + 3 + 5 + 3 = 15. \tag{27}$$

$\dot{W}(5) < \varphi_5$; therefore, $\tau_{54}(5)$ will be scheduled. Figure 13 shows the simulation result of $\tau_{54}(5)$ in UML-RT.

9. Conclusion

Two algorithms with a new priority assignment scheme (RMA-SJF) is proposed which possesses advantages of both rate monotonic and shortest job first, for a nonpreemptive periodic task set, on an uniprocessor architecture modelled using UML-RT. Further, a general schedulability test is derived using "problem window" analysis, to check for task schedulability. A sufficient schedulability condition to verify whether a periodic task is schedulable in any release time is also proposed. Task set generated is analyzed for the above schedulability test and condition with typical examples. The performances of the proposed algorithms are also analyzed with success ratio and effective processor utilization using UML-RT. In future aperiodic dependent nonpreemptive task with the proposed scheme on multiprocessor architecture modelled using UML-RT will be analyzed.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] N. Guan, W. Yi, Q. Deng, Z. Gu, and G. Yu, "Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling," *Journal of Systems Architecture*, vol. 57, no. 5, pp. 536–546, 2011.

[2] J. Lehoczky, L. Sha, and Y. Ding, "Rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proceedings of the Real-Time Systems Symposium*, pp. 166–171, December 1989.

[3] S. K. Baruah and S. Chakraborty, "Schedulability analysis of non-preemptive recurring real-time tasks," in *Proceedings of the 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS '06)*, pp. 108–116, 2006.

[4] C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 47–61, 1973.

[5] "Modeling language guide-rational rose real time".

[6] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *Proceedings of the 12th Real-Time Systems Symposium (RTSS '91)*, pp. 129–139, December 1991.

[7] Q. Zhu, R. Oishi, T. Hasegawa, and T. Nakata, "Integrating UML into SoC design process," in *Proceedings of the Design, Automation and Test in Europe (DATE '05)*, vol. 2, pp. 836–837, March 2005.

[8] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," Tech. Rep., INRIA, 1996.

[9] S. Lauzac, R. Melhem, and D. Mossé, "An improved rate-monotonic admission control and its applications," *IEEE Transactions on Computers*, vol. 52, no. 3, pp. 337–350, 2003.

[10] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1429–1442, 1995.

[11] W.-C. Lu, H.-W. Wei, and K.-J. Lin, "Rate monotonic schedulability conditions using relative period ratios," in *Proceedings of the12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '06)*, pp. 3–9, August 2006.

[12] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[13] N. C. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters*, vol. 79, no. 1, pp. 39–44, 2001.

[14] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 4, pp. 553–566, 2009.

[15] B. Kumar and J. Jasperneite, "UML profiles for modeling real-time communication protocols," *Journal of Object Technology*, vol. 9, no. 2, pp. 178–198, 2010.

[16] L. Kishor, D. Goyal, R. Singh, and P. Sharma, "Optimized scheduling algorithm," in *Proceedings of the IJCA International Conference on Computer Communication and Networks (CSI-COMNET '11)*, pp. 130–134, 2011.