Research Article

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

S. Ewins Pon Pushpa1 and Manamalli Devasigamani2

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

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 \( \tau_i \), 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 \( \tau \): \( U(\tau) = \sum_{\tau_i \in \tau} U_i \). 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:

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

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

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:

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

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 nonpreemptive 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 tasks, and for tasks with arbitrary deadlines.

In [1], Guan et al. addressed 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, $(τ = τ_1, τ_2, τ_3, \ldots, τ_m)$ where, there are "m" tasks, each task $τ_k = (C_k, T_k)$ 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 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.

<table>
<thead>
<tr>
<th>Notations used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$</td>
<td>A task set</td>
</tr>
<tr>
<td>$m$</td>
<td>Total number of tasks in $\tau$</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>Highest priority task</td>
</tr>
<tr>
<td>$\tau_m$</td>
<td>Lowest priority task</td>
</tr>
<tr>
<td>$\tau_k$</td>
<td>A task in task set</td>
</tr>
<tr>
<td>$C_k$</td>
<td>Computation time of $\tau_k$</td>
</tr>
<tr>
<td>$T_k$</td>
<td>Interrelease time or period of $\tau_k$</td>
</tr>
<tr>
<td>$\phi_k$</td>
<td>Problem area of the task $\tau_k$, analyzed for schedulability. A necessary condition for the deadline miss to occur for $\tau_k$ is that the worst-case work load, $W(k)$ in the problem area $\phi_k$ by all other tasks in the task set $\tau$ except $\tau_k$ is no less than $\phi_k$.</td>
</tr>
<tr>
<td>$C_{sk}$</td>
<td>Computation time lesser than $C_k$</td>
</tr>
<tr>
<td>$C$</td>
<td>Nonzero computation time</td>
</tr>
<tr>
<td>$U_k = C_k/T_k$</td>
<td>Task utilization of $\tau_k$</td>
</tr>
<tr>
<td>$U(\tau) = \sum_{i\in\tau} U_i$</td>
<td>System load</td>
</tr>
<tr>
<td>hp(k)</td>
<td>Set of all higher priority tasks of $\tau_k$</td>
</tr>
<tr>
<td>lp(k)</td>
<td>Set of all lower priority tasks of $\tau_k$</td>
</tr>
<tr>
<td>$C_{hp}(k)$</td>
<td>Set of all computation times of $hp(k)$</td>
</tr>
<tr>
<td>$C_{lp}(k)$</td>
<td>Set of all computation times of $lp(k)$</td>
</tr>
<tr>
<td>$C_m$</td>
<td>Maximum computation time in the subset of $C_{lp}(k)$</td>
</tr>
<tr>
<td>$hc(k)$</td>
<td>Summation of computation times of all higher priorities of $\tau_k$</td>
</tr>
<tr>
<td>$hc(k)$</td>
<td>Summation of computation times of all equal and higher priorities of $\tau_k$</td>
</tr>
<tr>
<td>$lc(k)$</td>
<td>Summation of computation times of lower priorities of $\tau_k$</td>
</tr>
<tr>
<td>$lec(k)$</td>
<td>Summation of computation times of equal and lower priorities of $\tau_k$</td>
</tr>
<tr>
<td>$C_{hp}(k)$</td>
<td>Set of all computation times of $hp(k)$</td>
</tr>
<tr>
<td>$C_{lp}(k)$</td>
<td>Set of all computation times of $lp(k)$</td>
</tr>
<tr>
<td>$U_{hp}(k)$</td>
<td>Summation of all the task utilization of $hp(k)$</td>
</tr>
<tr>
<td>$U_{lp}(k)$</td>
<td>Summation of equal and all task utilization of $lp(k)$</td>
</tr>
<tr>
<td>$r_n(k)$</td>
<td>$n$th release time of $\tau_k$</td>
</tr>
<tr>
<td>$r_{n+1}(k)$</td>
<td>$(n+1)$th release time of $\tau_k$ and deadline for the task released at $r_n(k)$</td>
</tr>
<tr>
<td>$l_f(k)$</td>
<td>Latest feasible start time for $\tau_k$ released at $r_n(k)$, to start execution in order to meet its deadline.</td>
</tr>
<tr>
<td>$N_{l_f}(k)$</td>
<td>Number of intermediate tasks in $\phi_k$</td>
</tr>
<tr>
<td>$w$</td>
<td>Work done by initial job, intermediate job or final job in $\phi_k$</td>
</tr>
<tr>
<td>$W r(k)$</td>
<td>Worst-case latency of $\tau_k$. It is the maximum time lapse for a task to start execute.</td>
</tr>
<tr>
<td>$W(k)$</td>
<td>Total work done by other tasks in the problem area of $\phi_k$</td>
</tr>
</tbody>
</table>

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.
4. Condition for Schedulability of a Task

A necessary condition for deadline miss to occur for \( \tau_k \), is that the worst-case work load, \( \dot{W}(k) \) by all other tasks in the task set \( \tau \) except \( \tau_k \) is no less than \( \varphi_k \). \( l_n(k) \) is the latest feasible start time for \( \tau_k \), 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, \( \varphi_k \) as shown in Figure 2. The work load in \( \varphi_k \) can be categorized into three parts as shown in Figure 3.

**Initial Job.** The contribution of task \( \tau_i \), in problem area \( \varphi_k \), is when computation of task \( \tau_i \) 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 \( \tau_i \) started in \( \varphi_k \) and having deadline in the problem area earlier to \( l_n(k) \).

**Final Job.** The contribution of task \( \tau_i \), in problem area \( \varphi_k \), is when computation of task \( \tau_i \) started in \( \varphi_k \) 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].

For \( r(k) \) to be scheduled,

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

where

\[
\dot{W}(k) = \sum_{hp(k), lp(k)} w \left( \text{initial jobs + intermediate jobs + final jobs} \right), \tag{4}
\]

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 \( \tau_1 \) 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 \( \tau_k \) may miss its deadline. We recall that \( C_m \) is the maximum computation time in the subset of \( C_{lp}(1) \). For a nonpreemptive architecture, any \( \tau_j \in hp(1) \) 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 \( \tau_j \), which started to execute \( C_m \) at \( r_{n-1}(j) \), is still executing beyond \( I_p(1) \) causing the task \( \tau_1 \) to miss deadline.

Therefore, if \( \varphi_1 \geq C_m \), then \( \tau_1 \) will be schedulable. \( \square \)

**Theorem 2.** A task \( \tau_k \) is schedulable by a work conserving nonpreemptive scheduling algorithm if

\[
1 - \frac{C_m}{\varphi_k} + U_{dp}(k) \geq U(\tau).
\]

**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 \( \tau_k \) 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 \( \tau_k \) will be

\[
\varphi_k \geq C_m + \sum_{hp(k)} w \text{ (intermediate jobs)},
\]

\[
\varphi_k \geq C_m + \sum_{\tau \in hp(k)} \frac{\varphi_k}{T_i} C_i.
\]

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,

\[
\varphi_k \geq C_m + \varphi_k U_{hp}(k),
\]

\[
\varphi_k \geq C_m + \varphi_k \left( U(\tau) - U_{dp}(k) \right),
\]

\[
\varphi_k - C_m \geq \varphi_k \left( U(\tau) - U_{dp}(k) \right),
\]

dividing by \( \varphi_k \),

\[
1 - \frac{C_m}{\varphi_k} + U_{dp}(k) \geq U(\tau).
\]

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

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

\[
\varphi_k \geq \varphi_k U_{hp}(k)
\]

\( C_m \) is neglected in (12), because schedulability condition for \( \tau_m \) to execute \( C_m \) is analyzed. \( \square \)

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 “\( m - 1 \)” tasks, by deriving the highest priority task release time \( T_j = \text{hec}(m - 1) \). Schedulability of each task is analyzed using the general schedulability test. Later \( T_j \) of tasks are varied to find an optimized utilization bound in \( 0.9 < U(\tau) \leq 1 \) to schedule \( \tau \) 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 \( W_r \) 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 \( r_k(k) \) is derived. Let \( \varphi_k \) be the problem area of \( \tau_k \) shown in Figure 5, and \( l_p(k) \) is the latest feasible start time to execute \( C_k \), for \( \tau_k \) 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.** \( hp(k), lp(k) \) released before \( r_k(k) \).

**Case B.** \( hp(k) \), \( lp(k) \) released in \([r_n(k), l_p(k)]\).

7.1. **Case A: \( hp(k), lp(k) \) Released before \( r_k(k) \)**

7.1.1. **\( hp(k) \) Released before \( r_k(k) \)**

Figure 5 shows \( \tau_j \in hp(k) \) released at \( r_{n-1}(j) \) before \( r_k(k) \):

\[
N_{\tau_j}(k) = 0;
\]

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

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

\[
(1) \text{ Initial job will be present if, } r_{n-1}(j) + W_r(j) + C_j - r_n(k) \leq C_j
\]

\[
(1) r_{n-1}(j) + W_r(j) + C_j - r_n(k) = C_j, \text{ then initial job will be } C_j;
\]
Algorithm 1

1. Computation times are randomly generated and arranged according to shortest job first priority scheme.
2. Initial job release order is as per priority scheme.
3. Release order will not be altered.
4. Computation times are random numbers arranged in ascending order.
5. Jobs are assigned to tasks.
6. Start task execution.
7. If task execution time exceeds the deadline, the job is considered failed.
8. Terminate the algorithm.

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

For

\[
(r_n(j) + Wτ(τ) + C_j < l_n(k))
\]

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

\[
(r_n(j) + Wτ(τ) + C_j \geq l_n(k))
\]

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 $r_j \in hp(k)$ released $[r_n(k), l_n(k)]$ will contribute intermediate job, if $r_n(j) + Wτ(τ) + C_j < l_n(k)$, while $r_j$ is pending.

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

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

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

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

(ii) $r_{n-1}(j) + Wτ(τ) + C_j < C_j$,

$$r_{n-1}(j) + Wτ(τ) + C_j < C_j$$

therefore, the initial job will be $C_j$.

(2) Intermediate job will be present if, $r_{n-1}(j) + Wτ(τ) + C_j = C_j$.

For the above condition,

$$r_{n-1}(j) + Wτ(τ) + C_j - r_n(k) > C_j > 0$$

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

7.2.1. hp(k) Released before $r_n(k)$.

Figure 6 shows $r_j \in hp(k)$ released at $r_{n-1}(j)$ before $r_n(k)$. Maximum contribution of any hp(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 $r_j \in hp(k)$ released $[r_n(k), l_n(k)]$ will contribute intermediate job, if $r_n(j) + Wτ(τ) + C_j < l_n(k)$, while $r_j$ 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

$$r_n(j) + Wτ(τ) + C_j < l_n(k)$$

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

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

$$r_n(j) + Wτ(τ) + C_j \geq l_n(k)$$

Proof. Consider Figure 7. By contradiction, if $r_n(j) + Wτ(τ) + C_j < l_n(k)$, then by Lemma 4, this condition possesses intermediate job. Therefore, $r_n(j) + Wτ(τ) + C_j \geq l_n(k)$ will contribute final job (see Algorithm 3).
Step 1. Arrange computation times according to shortest job first,
\[ C_1 \leq C_2 \leq C_3 \leq \cdots \leq C_k \leq \cdots \leq C_m \]
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,
\[ W \tau (1) = C_m \]
\[ W \tau (k) = h_c (k) + C_m \]
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,
\[ U(T) = \sum_{\tau \in \tau} U_i \] 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 \( W \tau(j) + C_j - C \geq C_j \) then, final job will be \( C_j \)
else final job is \( Cs_j = W \tau(j) + C_j - C \)

Algorithm 3

Proof. Any \( lp(k) \) released in \( [r_n(k), l_n(k)] \), as shown in Figure 7, will have no contribution in \( \varphi_k \) because any \( lp(k) \) is executed. So, \( lp(k) \) released in \( [r_n(k), l_n(k)] \) will contribute no workload in \( \varphi_k \). \( \square \)

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:
\[ C_i = \{2 \leq 3 \leq 4 \leq 5 \leq 6 \leq 7\} . \] (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:
\[ \varphi_k \geq \varphi_k U_{lp} (k) , \] (18)
\[ 29 \geq 28.898. \]

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 \( \tau \) 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 \( [2, 19] \); \( T_i \) 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: $\tau_j \in h_p(k)$ released at $r_{n-1}(j)$ before $r_n(k)$.

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

Figure 7: $\tau_j \in l_p(k)$ 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 $T_i$.

<table>
<thead>
<tr>
<th>$C_i$</th>
<th>Interrelease time, $T_i$ analyzed by work conserving utilization bound algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>34</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
</tr>
<tr>
<td>$U(\tau)$</td>
<td>0.92024</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>System load</th>
<th>Success ratio</th>
<th>Effective processor utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9964</td>
<td>96.02</td>
<td>92.83</td>
</tr>
<tr>
<td>0.9919</td>
<td>98.35</td>
<td>96.66</td>
</tr>
<tr>
<td>0.9871</td>
<td>98.78</td>
<td>96.66</td>
</tr>
<tr>
<td>0.962</td>
<td>97.97</td>
<td>95.27</td>
</tr>
<tr>
<td>0.986</td>
<td>98.13</td>
<td>95.85</td>
</tr>
<tr>
<td>0.9827</td>
<td>98.35</td>
<td>95.5</td>
</tr>
<tr>
<td>0.9784</td>
<td>99.52</td>
<td>97.05</td>
</tr>
<tr>
<td><strong>0.977</strong></td>
<td><strong>100</strong></td>
<td><strong>97.5</strong></td>
</tr>
<tr>
<td>0.9749</td>
<td>100</td>
<td>96.72</td>
</tr>
<tr>
<td>0.9676</td>
<td>100</td>
<td>96.16</td>
</tr>
<tr>
<td>0.9605</td>
<td>100</td>
<td>95.83</td>
</tr>
<tr>
<td>0.9546</td>
<td>100</td>
<td>94</td>
</tr>
<tr>
<td>0.9202</td>
<td>100</td>
<td>91.16</td>
</tr>
</tbody>
</table>

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 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:

\[
K = 5; \quad C_5 = 6; \quad C_m = 7; \quad \phi_5 = 32 - 6 = 26; \quad U(\tau) = 0.9827,
\]

\[
\phi_k \geq C_m + \sum_{\tau \in \text{hp}(k)} \frac{\phi_k}{T_{ij}},
\]

\[
\phi_5 \geq C_m + \sum_{\tau \in \text{hp}(k)} \frac{\phi_k}{T_{ij}},
\]

\[
26 \geq 7 + 16.125,
\]

\[
26 \geq 23.125.
\]

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

When schedulability of \( \tau(6) \) was analyzed, \( K = 6; C_6 = 7; \quad \phi_6 = 40 - 7 = 33; U(\tau) = 0.9827 \). Lowest priority task \( \tau(6) \) will be schedulable if (12) is satisfied:

\[
\phi_k \geq \phi_k U_{hp}(k),
\]

\[
33 \geq 22.6544.
\]

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 \( \tau = \{2, 11, 2, 13, 3, 15, 5, 3, 18, 4, 21, 21, 4, 25\} \) is schedulable in its 54th release.
Here $k = 5$, 5th task $W(k) \leq \phi_k$ for $r_n(k)$ to be scheduled, where

$$W(k) = \sum_{h_p(k), l_p(k)} w(\text{initial jobs} + \text{intermediate jobs}$$

+ final jobs)

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

$C_s = 4$; therefore, $l_s(5) = 1155 - 4 = 1151$ and $\phi_5 = 17$. Figure 12 shows the problem area of $\phi(5)$.

(a) Finding initial job or intermediate job contribution of $r_1$

$$r_{54}(5) \mod T_1 = 1,$$

$$r_{n-1}(1) = 1134 - 1 = 1133,$$

$$r_n(1) = 1144$$

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

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

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

$r_{n-1}(j) + W(5) + C_j - r_s(k) = C_j$, accordingly for $r_{n-1}(1) + W(1) + C_1 - r_{54}(5) \leq C_1$, the condition is not satisfied; therefore, it possesses only intermediate job.

$r_{n-1}(j) + W(5) + C_j - r_{54}(5) > C_j$.

Here,

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

$$r_{n-1}(1) + W(1) + C_1 - r_{54}(5) - C_1 = 3 > 0.$$  

Therefore, computation of a task $r_1$ started in $\phi_k$ and having deadline in the problem area, will contribute an intermediate job.

By Lemma 4, task $r_1 \in h_p(k)$ released $[r_{54}(5), l_{54}(5)]$ will contribute intermediate job, if

$$r_n(1) + W(1) + C_1 < l_{54}(5)$$

$l_{54}(5) = 1151$, $r_n(1) = 1144$, release time is in the problem area. $r_n(1) + W(1) + C_1 < l_{54}(5)$ is satisfied because $r_n(1) + W(1) + C_1 = 1150$ and $l_{54}(5) = 1151$. Therefore $r_1$ does contribute one intermediate job in $\phi_1$.

By Lemma 4, if $r_1 \in \text{hp}(5)$ released $[r_{54}(k), l_{54}(k)]$ will contribute final job, if

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

$$r_n(1) = 1144$$

release time in the problem area; therefore, $r_1$ does not contribute final job in $\phi_1$ because $r_n(1) + W(1) + C_1 < l_{54}(5)$.

Therefore, the final job = 0.

So, the total worst-case workload contribution of $r_1$ is $w(1) = 2$ (intermediate job) = 4.

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

$$W(2) = 2$$

(24)

$$W(3) = 2$$

(25)

$$W(4) = 3$$

(26)

$$W(5) = 4$$

(27)

$W(5) < \phi_5$; therefore, $r_{54}(5)$ will be scheduled. Figure 13 shows the simulation result of $r_{54}(5)$ in UML-RT.

![Figure 12: Problem area $\phi_5$, for nonpreemptive uniprocessor scheduling.](image1)

![Figure 13: Simulation result in UML-RT.](image2)
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 non-preemptive 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 non-preemptive 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

Submit your manuscripts at
http://www.hindawi.com